pyCGNS.PAT/Manual

Release 4.2.0

Marc Poinot

October 24, 2012

CONTENTS

The *PATtern* module provides the user with functions dedicated to *CGNS/Python* trees. The *PAT.cgnslib* module uses the *SIDS* compliant data structures, you can create, read, check, modify some *CGNS/Python* sub-trees related to a *SIDS* type. With this module you are working with a Python data structure, all function are using plain Python/Numpy objects. Thus, the *PAT* module is not required for your applications, as you can write your own function to handle these Python objects. The *PAT.cgnsutils* provides utility fonctions for raw *CGNS/Python* trees or nodes. The *PAT* defines also constant modules such as *PAT.cgnskeywords* for all *SIDS* names or constant strings, *PAT.cgnstypes* for the *SIDS* types descriptions (enumerates, allowed list of children...) and the *PAT.cgnserrors* with error codes and their messages.

A special module *PAT.SIDS* has all *CGNS/SIDS* patterns gathered as *PAT.cgnslib* calls. These patterns, used for creation only, are building in a recursive way the whole sub-tree for a given *SIDS* type.

CHAPTER

ONE

UTILITIES

The *CGNS.PAT.cgnsutils* has a large set of utility functions using the *CGNS/Python* nodes, sub-trees or trees as arguments, you can manipulate tree paths, links, values. Functions are not gathered into a class because we want them to proceed on standard *CGNS/Python* trees. Most functions have an optional error management, you can ask them to raise an exception or to return None. The *dienow* argument is set to *False* as default, which means a error would return a *None*. A *dienow* set to *True* raises an *error*. Some functions also have an optional legacy management, to take into account old CGNS/Python stuff. When set to *True*, the *CGNSTree_t* top node should not appear and is not inserted when needed. The weird CGNS/SIDS types such as *"int[IndexDimension]"* are used instead of CGNS/Python replacements. The *legacy* argument is set to *False* as default.

The list below gives an overview of publicly available functions.

- Node life cycle: `nodeCreate` - `nodeCopy` - `nodeDelete` -
- Check functions: `checkNode` - `checkRootNode` - `checkNodeType` - `checkNodeName` - `checkSameNode` - `checkDuplicatedName` - `checkPath` -
- Node true/false tests: `hasChildType` - `hasAncestorType` - `hasChildName` - `hasAncestorName` - `hasValue` - `hasValueDataType` - `hasValueFlags` -
- Data retrieval simple functions: `getNodeByPath` - `getValueByPath` - `getChildrenByPath` - `getTypeByPath`
- Data retrieval specialized functions: `getAllNodesByTypeSet` - `getNodeAllowedChildrenTypes` - `getNodeAllowedDataTypes` -
- Node value manipulation: `getValueShape` - `getValueDataType` - `hasValue` - `hasValueDataType` - `getValueByPath` -
- Path retrieval functions: `getPathFromNode` - `getPathFullTree` - `getPathByNameFilter` - `getPathByTypeFilter` -
- Path manipulation: `getPathToList` - `getPathAncestor` - `getPathLeaf` - `getPathNoRoot` - `getPathAsTypes` - `getPathNormalize` -

CGNS.PAT.cgnsutils.**nodeCreate**(*name*, *value*, *children*, *type*, *parent=None*, *dienow=False*)

Create a new node with and bind it to its parent:

```
import CGNS.PAT.cgnskeywords as CK

n=createNode('Base',numpy([3,3]),[],CK.CGNSBase_ts)
z=createNode('ReferenceState',None,[],CK.ReferenceState_ts,parent=n)
```

•Args:

•*name*: node name as a string

•*value*: node value as a numpy array

•*children*: list of node children

•*type*: CGNS type as a string

•*parent*: parent node where to insert the new node (default: None)

•*dienow*: If True raises an exception in case of problem (default: False)

•Return:

•The new node

•Remarks:

•If parent is None (default) node is orphan

•Full-checks the node with *checkNodeCompliant* only if *dienow* is True.

CGNS.PAT.cgnsutils.**nodeCopy**(*node*, *newname=None*)

Creates a new node sub-tree as a copy of the argument node sub-tree. A deep copy is performed on the node, including the values, which can lead to a very large memory use:

```
n1=getNodeByPath(T,'/Base/Zone1/ZoneGridConnectivity')
n2=getNodeByPath(T,'/Base/Zone1/ZoneGridConnectivity/Connect1')
n3=nodeCopy(n2,'Connect2')
nodeChild(n1,n3)
```

•Args:

•*node*: node to copy

•*name*: new node (copy) name

•Return:

•The new node

•Remarks:

•Full-checks the node with *checkNodeCompliant* only if *dienow* is True.

•The new node name is the same by default, thus user would have to check for potential duplicated name.

CGNS.PAT.cgnsutils.**nodeDelete**(*tree*, *node*, *legacy=False*)

Deletes a node from a tree:

```
import CGNS.PAT.cgnslib as CL

T =CL.newCGNSTree()
b1=CL.newBase(T,'Base',3,3)
z1=CL.newZone(b1,'Zone1',numpy.array([1,2,3]))
z2=CL.newZone(b1,'Zone2',numpy.array([1,2,3]))
print getPathFullTree(T)
# ['/CGNSLibraryVersion', '/Base', '/Base/Zone1', '/Base/Zone1/ZoneType', '/Base/Zone2', '/Ba
nodeDelete(T,z1)
print getPathFullTree(T)
# ['/CGNSLibraryVersion', '/Base', '/Base/Zone2', '/Base/Zone2/ZoneType']
```

•Args:

•*tree*: target tree where to find the node to remove

•*node*: node to remove (actual CGNS/Python node or node name as absolute path)

•Return:

•The tree argument (without the deleted node)

•Remarks:

•Uses `checkSameNode()`.

•The actual memory of the node only if no other reference to this node is found by Python.

CGNS.PAT.cgnsutils.**checkNodeName**(*node*, *dienow=False*)

Checks if the name is CGNS/Python compliant node name:

•Args:

•*node*: the CGNS/Python node to check

•Return:

•True if the name has a correct syntax

•Remarks:

•The function is the same as `checkName()` but with a node as arg instead of string

•see also `checkNodeCompliant()`

CGNS.PAT.cgnsutils.**checkName**(*name*, *dienow=False*)

Checks if the name is CGNS/Python compliant node name.

```
if (not checkName(name)):
   print 'Such name ',name,' not allowed'
```

•Args:

•*name*: the string to check

•Return:

•True if the name has a correct syntax

•Remarks:

•Type of name should be a Python string

•Name cannot be empty

•No '/' in the name

•A single '.' or '..' are not allowed

•A name with only ' ' is not allowed

•Raises *cgnsNameError* codes 22,23,24,25,29 if *dienow* is True

CGNS.PAT.cgnsutils.**setChild**(*parent*, *node*)

Adds a child node to the parent node children list:

```
n1=getNodeByPath(T,'/Base/Zone1/ZoneGridConnectivity')
n2=getNodeByPath(T,'/Base/Zone1/ZoneGridConnectivity/Connect1')
n3=nodeCopy(n2)
setChild(n1,n3)
```

•Args:

•*parent*: the parent node

•*node*: the child node to add to parent

•Return:

•The parent node

•Remarks:

•No check is performed on duplicated child name or any other validity.

CGNS.PAT.cgnsutils.**checkDuplicatedName**(*parent*, *name*, *dienow=False*)

Checks if the name is not already in the children list of the parent:

```
count=1
while (not checkDuplicatedName(node,'solution#%.3d'%count)): count+=1
```

•Args:

•*parent*: the parent node

•*name*: the child name to look for

•Return:

•True if the child *IS NOT* duplicated

•False if the child *IS* duplicated

•Remarks:

•Sorry about the legacy interface, True means not ok... (see `checkHasChildName()`)

•Raises *cgnsNameError* code 102 if *dienow* is True

CGNS.PAT.cgnsutils.**checkHasChildName**(*parent*, *name*, *dienow=False*)

Checks if the name is in the children list of the parent:

```
count=1
while (checkHasChildName(node,'solution#%.3d'%count)): count+=1
```

•Args:

•*parent*: the parent node

•*name*: the child name to look for

•Return:

•True if the child exists

•Remarks:

•Raises *cgnsNameError* code 102 if *dienow* is True

CGNS.PAT.cgnsutils.**checkNodeType**(*node*, *cgnstype=[ ]*, *dienow=False*)

Check the CGNS type of a node. The type can be a single value or a list of values. Each type to check is a string such as *CGNS.PAT.cgnskeywords.CGNSBase_ts* constant for example. If the list is empty, the check uses the list of all existing CGNS types:

```
import CGNS.PAT.cgnskeywords as CK

n=createNode('Base',numpy([3,3]),[],CK.CGNSBase_ts)
checkNodeType(n)
checkNodeType(n,['Zone_t',CK.CGNSBase_t])
```

•Args:

•*node*: the CGNS/Python node to check

•*cgnstype*: a list of strings with the types to check

•Return:

•True if the type is a CGNSType or a type in the argument list.

•None if the parent is None (*may change to have consistent return*)

•Remarks:

•raises *cgnsTypeError* codes 103,104,40 if *dienow* is True

CGNS.PAT.cgnsutils.**checkNode**(*node*, *dienow=False*)

Checks if a node is a compliant CGNS/Python node structure of list. The syntax for a compliant node structure is:

```
[<name:string>, <value:numpy>, <children:list-of-nodes>, <cgnstype:string>]
```

With the following exception: a *value* can be None.

The function checks the syntax of the node and the types of its contents, it doesn't perform sub checks such as *checkNodeName*, *checkNodeType*...

You should always check first the node structure, a function such as *checkNodeName* blindly access to the first item of the list and would raise an exception if the structure is not correct.

•Args:

•*node*: the CGNS/Python node to check

•Return:

•True if the node is ok

•Remarks:

•see also `checkNodeCompliant()`

•Raises *cgnsNodeError* codes 1,2,3,4,5 if *dienow* is True

CGNS.PAT.cgnsutils.**checkRootNode**(*node*, *legacy=False*, *dienow=False*)

Checks if a node is the CGNS/Python tree root node. If *legacy* is True, then *[None, None, [children], None]* is accepted as Root. If it is not True (default) then a CGNS/Python node of type *CGNSTree_t* is expected as root node. Children contains then the *CGNSLibraryVersion* and *CGNSBase* nodes as flat list. The *flat* pattern with a list containing *CGNSLibraryVersion* and zero or more *CGNSBase* nodes is not accepted. You should use a trick as the one above by giving this pattern as child of a fake node.

```
# flatpattern is a CGNS/Python list with a 'CGNSLibraryVersion' node
# and 'CGNSBase' nodes at the same level.
# we build a temporary children list

tmp=[None, None, [flatpattern], None]
```

```
if (checkRootNode(tmp,True)):
  # do smomething
```

•Args:

•*node*: the CGNS/Python node to check

•*legacy*: boolean, True means you accept non-CGNSTree_t node

•Return:

•True if the node is a root node

•Remarks:

•Raises *cgnsNodeError* codes 90,91,99 if *dienow* is True

CGNS.PAT.cgnsutils.**checkSameNode**(*nodeA*, *nodeB*, *dienow=False*)

Checks if two nodes have the same contents: same name, same CGNS/SIDS type, same number of children, same value type (None of numpy). The children are not parsed, the value itself is not checked (see `checkSameValue()`):

```
if (checkSameNode(nodeA,nodeB)):
  nodeB=copyNode(nodeB)
```

•Args:

•*node A*: CGNS/Python node

•*node B*: CGNS/Python node

•Return:

•True if the nodes are same

•Remarks:

•Raises *cgnsNodeError* code 30 if *dienow* is True

CGNS.PAT.cgnsutils.**checkSameValue**(*nodeA*, *nodeB*, *dienow=False*)

Checks if two nodes have the same value. There is no tolerance on actual array values when these are compared one to one. This could lead to time consuming operations for large arrays.

```
if (checkSameValue(nodeA,nodeB)):
  # do something
```

•Args:

•*node A*: CGNS/Python node

•*node B*: CGNS/Python node

•Return:

•True if the nodes have same value

•Remarks:

•Raises *cgnsNodeError* code 30 if *dienow* is True

CGNS.PAT.cgnsutils.**checkArray**(*a*, *dienow=False*)

Check if the array value of a node is a numpy array.

```
if (checkArray(node[1])):
  # do something
```

•Args:

•*a*: value to check

•Return:

•True if the array is suitable as CGNS/Python value

•Remarks:

•Raises error codes 109,170 if *dienow* is True

CGNS.PAT.cgnsutils.**checkNodeCompliant**(*node*, *parent*, *dienow=False*)

Performs all possible checks on a node. Can raise any of the exceptions related to node checks (`checkNodeName()`, `checkNodeType()`, `checkArray()`...).

```
if (checkNodeCompliant(node)):
  # do something
```

•Args:

•*node*: CGNS/Python node to check

•*parent*: CGNS/Python parent node to check

•Return:

•True if all controls are ok

•Remarks:

•Calls `checkNode()`,:py:func:*checkNodeName*, `checkDuplicatedName()`, `checkArray()`, `checkNodeType()`

CGNS.PAT.cgnsutils.**getValueShape**(*node*)

Returns the value data shape for a CGNS/Python node for **display purpose**. If the shape cannot be determined a - is returned.

```
print 'node data shape is %s'%getValueShape(node)
```

•Args:

•*node*: CGNS/Python node

•Return:

•A string with the shape

CGNS.PAT.cgnsutils.**getValueDataType**(*node*)

Returns the value data type for a CGNS/Python node for **display purpose**.

```
print 'node data type is %s'%getValueDataType(node)
```

•Args:

•*node*: CGNS/Python node

•Return:

•A string in [*C1*,'I4','I8','R4','R8','??']

•Remarks:

•*??* is returned if datatype is not one of [*C1*,'I4','I8','R4','R8']

CGNS.PAT.cgnsutils.**getNodeByPath**(*tree*, *path*)

Returns the CGNS/Python node with the argument path:

```
zbc=getNodeByPath(T,'/Base/Zone001/ZoneBC')
bc1=getNodeByPath(zbc,'wall01')
```

The path is compared as a string, you should provide the exact path if you have a sub-tree or a tree with its *CGNSTree* fake node. The following lines are not equivalent (sic!):

```
zbc=getNodeByPath(T,'/Base/Zone001/ZoneBC')
zbc=getNodeByPath(T,'/CGNSTree/Base/Zone001/ZoneBC')
```

•Args:

•*tree*: the target tree to parse

•*path*: a string representing an absolute or relative path

•Return:

•The CGNS/Python node matching the path

•Remark:

•Returns None if the path is not found

•No wildcards allowed (see `getPathByNameFilter()` and `getPathByNameFilter()`)

CGNS.PAT.cgnsutils.**getValueByPath**(*tree*, *path*)

Returns the value of a CGNS/Python node with the argument path:

```
import CGNS.PAT.cgnskeywords as CK

v=getNodeByPath(T,'/Base/Zone001/ZoneType')

if (v == CK.Structured_s): print 'Structured Zone Found'
```

•Args:

•*tree*: the target tree to parse

•*path*: a string representing an absolute or relative path

•Return:

•The CGNS/Python node value matching the path

•Remark:

•Returns None if the path is not found

•No wildcards allowed (see `getPathByNameFilter()` and `getPathByNameFilter()`)

CGNS.PAT.cgnsutils.**getChildrenByPath**(*tree*, *path*)

Returns the children list of a CGNS/Python node with the argument path.

```
import CGNS.PAT.cgnskeywords as CK

for bc in getChildrenByPath(T,'/Base/Zone01/ZoneBC'):
  if (bc[3] == CK.BC_ts):
    print 'BC found:', bc[0]
```

•Args:

•*tree*: the target tree to parse

•*path*: a string representing an absolute or relative path

•Return:

•The CGNS/Python node children list of node matching the path

•Remark:

•Returns None if the path is not found

•No wildcards allowed (see `getPathByNameFilter()` and `getPathByNameFilter()`)

CGNS.PAT.cgnsutils.**getNextChildSortByType**(*node, parent=None, criteria=['AdditionalExponents_t', 'AdditionalFamilyName_t', 'AdditionalUnits_t', 'ArbitraryGridMotionType_t', 'ArbitraryGridMotion_t', 'AreaType_t', 'Area_t', 'AverageInterfaceType_t', 'AverageInterface_t', 'Axisymmetry_t', 'BCDataSet_t', 'BCDataType_t', 'BCData_t', 'BCProperty_t', 'BCTypeSimple_t', 'BCType_t', 'BC_t', 'BaseIterativeData_t', 'CGNSBase_t', 'CGNSLibraryVersion_t', 'CGNSTree_t', 'ChemicalKineticsModelType_t', 'ChemicalKineticsModel_t', 'ConvergenceHistory_t', 'DataArray_t', 'DataClass_t', 'DataConversion_t', 'DataType_t', 'Descriptor_t', 'DiffusionModel_t', 'DimensionalExponents_t', 'DimensionalUnits_t', 'DiscreteData_t', 'EMConductivityModelType_t', 'EMConductivityModel_t', 'EMElectricFieldModelType_t', 'EMElectricFieldModel_t', 'EMMagneticFieldModelType_t', 'EMMagneticFieldModel_t', 'ElementType_t', 'Elements_t', 'EquationDimension_t', 'FamilyBC_t', 'FamilyName_t', 'Family_t', 'FlowEquationSet_t', 'FlowSolution_t', 'GasModelType_t', 'GasModel_t', 'GeometryEntity_t', 'GeometryFile_t', 'GeometryFormat_t', 'GeometryReference_t', 'GoverningEquationsType_t', 'GoverningEquations_t', 'Gravity_t', 'GridConnectivity1to1_t', 'GridConnectivityProperty_t', 'GridConnectivityType_t', 'GridConnectivity_t', 'GridCoordinates_t', 'GridLocation_t', 'IndexArray_t', 'IndexRange_t', 'IntIndexDimension_ts', 'IntegralData_t', 'InwardNormalIndex_t', 'InwardNormalList_t', 'Ordinal_t', 'OversetHoles_t', 'Periodic_t', 'PointSetType_t', 'ReferenceState_t', 'RigidGridMotionType_t', 'RigidGridMotion_t', 'Rind_t', 'RotatingCoordinates_t', 'SimulationType_t', 'ThermalConductivityModelType_t', 'ThermalConductivityModel_t', 'ThermalRelaxationModelType_t', 'ThermalRelaxationModel_t', 'Transform_t"', 'TurbulenceClosureType_t', 'TurbulenceClosure_t', 'TurbulenceModelType_t', 'TurbulenceModel_t', 'UserDefinedData_t', 'ViscosityModelType_t', 'ViscosityModel_t', 'WallFunctionType_t', 'WallFunction_t', 'ZoneBC_t', 'ZoneGridConnectivity_t', 'ZoneIterativeData_t', 'ZoneType_t', 'Zone_t']*)

Iterator, returns the children list of the argument CGNS/Python sorted using the CGNS type then the name. The *sortlist* gives an alternate sort list/dictionnary.

```
for child in getNextChildSortByType(node):
    print 'Next child:', child[0]

zonesort=[CGK.Elements_ts, CGK.Family_ts, CGK.ZoneType_ts]
for child in getNextChildSortByType(node,criteria=mysort):
    print 'Next child:', child[0]

mysort={CGK.Zone_t: zonesort}
for child in getNextChildSortByType(node,parent,mysort):
    print 'Next child:', child[0]
```

•Args:

•*node*: the target node

•*parent*: the parent node

•*criteria*: a list or a dictionnary used as the sort criteria

•Return:

•This is an iterator, it returns a CGNS/Python node

•Remark:

•The function is an iterator

•If criteria is a list of type, the sort order for the type is the list order. If it is a dictionnary, its keys are the parent types and the values are list of types.

CGNS.PAT.cgnsutils.**getTypeByPath**(*tree*, *path*)

Returns the CGNS type of a CGNS/Python node with the argument path.

```
import CGNS.PAT.cgnskeywords as CK

if (getTypeByPath(T,'/Base/Zone01/ZoneBC/'):
  if (bc[3] == CK.BC_ts):
    print 'BC found:', bc[0]
```

•Args:

•*tree*: the target tree to parse

•*path*: a string representing an absolute or relative path

•Return:

•The CGNS/Python node CGNS/SIDS type (string)

•Remark:

•Returns None if the path is not found

•No wildcards allowed (see `getPathByTypeFilter()` and `getPathByNameFilter()`)

CGNS.PAT.cgnsutils.**getPathByNameFilter**(*tree*, *filter=None*)

Returns a list of paths from T matching the filter. The filter is a regular expression used to match the path of **node names**:

```
import CGNS.PAT.cgnskeywords as CK

for path in filterPathByName(T,'/Base[0-1]/domain\..*/.*/.*/FamilyName'):
   print 'FamilyName ',path,' is ',path[2]
```

•Args:

•*tree*: the target tree to parse

•*filter*: a regular expresion for the complete path to math to

•Return:

•A list of paths (strings) matching the path pattern

•Remark:

•Returns empty list if no match

CGNS.PAT.cgnsutils.**getPathByTypeFilter**(*tree*, *filter=None*)

Returns a list of paths from T matching the filter. The filter is a regular expression used to match the path of **node types**:

```
import CGNS.PAT.cgnskeywords as CK

for path in filterPathByType(T,'/.*/.*/.*/BC_t'):
  for child in getChildrenByPath(T,path):
   if (child[3]==CK.FamilyName_t):
     print 'BC ',path,' belongs to ',child[2]
```

•Args:

•*tree*: the target tree to parse

•*filter*: a regular expression for the complete path to math to

•Return:

•A list of paths (strings) matching the types-path pattern

•Remark:

•Returns empty list if no match

CGNS.PAT.cgnsutils.**getNodeFromPath**(*path*, *node*)

Beware: this parse starts with children, not current node...

CGNS.PAT.cgnsutils.**getParentFromNode**(*tree*, *node*)

Returns the parent node of a node. If the node is root node, itself is returned:

```
parent=getParentFromNode(T,node)
```

•Args:

•*tree*: the CGNS/Python target tree to parse

•*node*: the child node

•Return:

•A list of paths (strings) matching the types-path pattern

•Remark:

•Returns itself if node is root

CGNS.PAT.cgnsutils.**getPathFromNode**(*tree*, *node*, *path=''*)

Returns the path from a node in a tree. The argument tree is parsed and a path is built-up until the node is found. The node object is compared to the tree nodes, if you have multiple references to the same node, the first found is used for the path:

```
# T is a compliant CGNS/Python tree

path=getPathFromNode(T,node)
getNodeByPath(T,getPathAncestor(path))
```

Args:

- *tree*: the target tree to parse
- *node*: the target node to find

Remark:

- Returns None if not found

CGNS.PAT.cgnsutils.**getAllNodesByTypeOrNameList**(*tree*, *typeornamelist*)

Returns a list of paths from the argument tree with nodes matching the list of types or names. The list you give is the list you would have if you pick the node type or the node name during the parse.

```
tnlist=['CGNSTree_t','Base#001','Zone_t']

for path in getAllNodesByTypeOrNameList(T,tnlist):
   node=getNodeByPath(T,path)
   # do something with node
```

Would return all the zones of the named base. See also `getAllNodesByTypeSet()` See also `getAllNodesByTypeList()`

•Args:

•*tree*: the start node of the CGNS tree to parse

•*typeornamelist*: the (ordered) list of types

•Return:

•a list of strings, each string is the path to a matching node

•Remark:

•the first comparison is performed on name, then on type. If you have a node name that matches a type, the node is included in the result.

CGNS.PAT.cgnsutils.**getAllNodesByTypeList**(*tree*, *typelist*)

Returns a list of paths from the argument tree with nodes matching the list of types. The list you give is the list you would have if you pick the node type during the parse.

```
tlist=['CGNSTree_t','CGNSBase_t','Zone_t']

for path in getAllNodesByTypeList(T,tlist):
   node=getNodeByPath(T,path)
   # do something with node
```

Would return all the zones of your tree. See also `getAllNodesByTypeSet()`

•Args:

•*tree*: the start node of the CGNS tree to parse

•*typelist*: the (ordered) list of types

•Return:

•a list of strings, each string is the path to a matching node

`CGNS.PAT.cgnsutils.`**`getPathFullTree`**`(`*tree*`)`

Returns the list of all possible node paths of a CGNS/Python tree.

```
for paths in getPathFullTree(T):
   # do something
```

•Args:

•*tree*: the CGNS/Python target tree to parse

•Return:

•A list of strings, each is a path

•Remark:

•Returns [] is tree empty or invalid

`CGNS.PAT.cgnsutils.`**`checkPath`**`(`*path*, *dienow=False*`)`

Checks the compliance of a path, which is basically a UNIX-like path with constraints on each node name.

checkPath('/Base/Zone/ZoneBC')

•Args:

•*path*: path to check (string)

•Return:

•True if the path is ok, False if a problem is found

`CGNS.PAT.cgnsutils.`**`hasSameRootPath`**`(`*pathroot*, *pathtocompare*`)`

Compares two paths:

```
>>>hasSameRootPath('/Base/Zone/ZoneBC','/Base/ZoneBC/BC#2/Data')
True
>>>hasSameRootPath('/Base/Zone/ZoneBC','/Base/ZoneBC#2')
False
```

•Args:

•*pathroot*: root path to compare

•*pathtocompare*: path which is supposed to have rootpath as substring

•Return:

•Ture if 'rootpath' is a prefix of 'pathtocompare'

•Remarks:

•Each node name is a token, see example below: the second example doesn't match as a path while it matches as a string.

CGNS.PAT.cgnsutils.**getPathToList**(*path*, *nofirst=False*, *noroot=True*)

Return the path as a list of node names:

```
>>>print getPathToList('/Base/Zone/ZoneBC')
['','Base','Zone','ZoneBC']
>>>print getPathToList('/Base/Zone/ZoneBC',True)
['Base','Zone','ZoneBC']
>>>print getPathToList('/')
[]
```

•Args:

•*path*: path string to split

•*nofirst*: Removes the first empty string that appears for absoulte paths (default: False)

•*noroot*: If true then removes the CGNS/HDF5 root if found (default: True)

•Return:

•The list of path elements as strings

•With '/' as argument, the function returns an empty list

•Remarks:

•The path is first processed by `getPathNormalize()` before its split

CGNS.PAT.cgnsutils.**getPathAncestor**(*path*, *level=1*)

Return the path of the node parent of the argument node path:

```
>>>print getPathAncestor('/Base/Zone/ZoneBC')
'/Base/Zone'
```

•Args:

•*path*: path string of the child node

•*level*: number of levels back from the child (default: 1 means the father of the node)

•Return:

•The ancestor path

•If the path is '/' its ancestor is None.

CGNS.PAT.cgnsutils.**getPathLeaf**(*path*)

Return the leaf node name of the path:

```
>>>print getPathLeaf('/Base/Zone/ZoneBC')
'ZoneBC'
```

•Args:

•*path*: path string of the child node

•Return:

•The leaf node name

•If the path is '/' the function returns ''

CGNS.PAT.cgnsutils.**getPathNoRoot**(*path*)

Return the path without the implementation node 'HDF5 Mother node' if detected as first element:

```
>>>print getPathNoRoot('/HDF5 Mother Node/Base/Zone/ZoneBC')
['Base','Zone','ZoneBC']
```

•Args:

•*path*: path string to check

•Return:

•The new path without *HDF5 Mother node* if found

•Remarks:

•The path is processed by `getPathNormalize()`

CGNS.PAT.cgnsutils.**getPathAsTypes**(*tree*, *path*)

Return the list of types corresponding to the argument path in the tree:

```
>>>getPathAsTypes(T,'/Base/Zone/ZoneBC')
['CGNSBase_t','Zone_t','ZoneBC_t']
```

•Args:

•*tree*: target tree

•*path*: path to parse in the tree

•Return:

•The list of CGNS types found

•*None* if the path is not found

CGNS.PAT.cgnsutils.**getPathNormalize**(*path*)

Return the same path as minimal string, removes *////* and */./* and other simplifiable UNIX-like path elements.

```
# a checkPath here would fail, because single or double dots are not
# allowed as a node name. But actually this is allowed as a
# path description
p=getPathNormalize('///Base/././//Zone/../Zone/./ZoneBC//.')

# would return '/Base/Zone/ZoneBC'
if (checkPath(p)):
   # do something
```

•Args:

•*path*: path string to simplify

•Return:

•The simplified path

•Remarks:

•Uses *os.path.normpath*

•Before its normalization a path can be **non-compliant**

CGNS.PAT.cgnsutils.**childNames**(*node*)

Gets the children names

```
for c in childNames(node):
  # do something
```

•Args:

•*node*: CGNS/Python node

•Return:

•List of children names

CGNS.PAT.cgnsutils.**getAllNodesByTypeSet**(*tree*, *typeset*)

Returns a list of paths from the argument tree with nodes matching one of the types in the list.

```
#  Would return all the zones and BCs of your tree.
tset=['BC_t','Zone_t']

for path in getAllNodesByTypeSet(T,tset):
   node=getNodeByPath(T,path)
   # do something
```

•Args:

•*tree*: the start node of the CGNS tree to parse

•*typeset*: the list of types

•Return:

•a list of strings, each string is the path to a matching node

•Remarks:

•See also `getAllNodesByTypeList()`

CGNS.PAT.cgnsutils.**getNodeAllowedChildrenTypes**(*pnode*, *node*)

Returns all allowed CGNS-types for the node. The parent is mandatory.

```
if (node[2] not in getNodeAllowedChildrenTypes(parent,node)):
   # do something
```

•Args:

•*pnode*: CGNS/Python parent node of second argument

•*node*: CGNS/Python node

•Return:

•A list of CGNS/SIDS types (strings)

CGNS.PAT.cgnsutils.**getNodeAllowedDataTypes**(*node*)

Returns a list of string with all allowed CGNS data types for the node.

```
if (getValueDataType(node) not in getNodeAllowedDataTypes(node)):
   # do something
```

•Args:

•*node*: CGNS/Python node

•Return:

•A list of CGNS/SIDS value data types (strings)

•see also `getValueDataType()`

CGNS.PAT.cgnsutils.**hasChildType**(*parent*, *ntype*)

checks if the parent node has a child with given type.

```
node=getNodeByPath(T,'/Base/Zone/BC'):
if (hasChildType(node, 'AdditionalFamily_t')):
   # do something
```

•Args:

•*parent*: CGNS/Python parent node

•*ntype*: CGNS/SIDS node type (string)

•Return:

•True if at least one child with such type is found

CGNS.PAT.cgnsutils.**hasChildName**(*parent*, *name*, *dienow=False*)

Checks if the name is in the children list of the parent:

```
node=hasChildName(parent,CGK.ZoneGridConnectivity_s)
if (node is None):
   node=CGL.newZoneGridConnectivity(parent)
```

•Args:

•*parent*: the parent node

•*name*: the child name to look for

•Return:

•the actual child node if the child exists

•None if the child is not found

•Remarks:

•Raises *cgnsNameError* code 102 if *dienow* is True

CHAPTER

TWO

THE PYTHONISH CGNS LIB

The so-called *CGNSlib* or *MLL* or *Mid-level* library, is set of functions for used to read/write/modify a set of nodes matching a CGNS/SIDS type. The Pythonish flavour of this library declares a set of functions with more or less the same interface but with Python values.

CGNS.PAT.cgnslib.**newCGNSTree**()

Top CGNS/Python tree node creation:

```
T=newCGNSTree()
```

•Return:

•The new *CGNSTree_t* node

•Remarks:

•You *should* keep the returned node in a variable or reference to it in any other way, this tree root is a Python object that would be garbagged if its reference count reaches zero.

•The *CGNSTree* node is a CGNS/Python node which has no existence in a disk HDF5 file.

•Children:

•`newCGNSBase()`

CGNS.PAT.cgnslib.**newCGNSBase**(*tree*, *name*, *ncell*, *nphys*)

CGNSBase node creation:

```
# The base is put in the 'T' children list
T=newCGNSTree()
newBase(T,'Box-1',3,3)

# No parent, you should fetch the new node using a variable
B=newBase(None,'Box-2',3,3)
```

•Args:

•*tree*: the parent node (<*node*> or *None*)

•*name*: base name (*string*)

•*cdim*: cell dimensions (*int*)

•*pdim*: physical dimensions (*int*)

•Return:

•The new *CGNSBase_t* node

•Remarks:

•If a parent is given, the new node is added to the parent children list.

•Children:

•`newZone()`

CGNS.PAT.cgnslib.**newDataClass**(*parent*, *value='UserDefined'*)
-DataClass node creation -DataClass

'newNode:N='*newDataClass*'(parent:N,value:A)'

If a parent is given, the new <node> is added to the parent children list. The value argument is a DataClass enumerate. No child allowed. Returns a new <node> representing a DataClass_t sub-tree.

CGNS.PAT.cgnslib.**newDescriptor**(*parent*, *name*, *value=''*)
-Descriptor node creation -Descriptor

'newNode:N='*newDescriptor*'(parent:N,name:S,text:A)'

No child allowed. Returns a new <node> representing a Descriptor_t sub-tree.

CGNS.PAT.cgnslib.**newDimensionalUnits**(*parent, value=['Meter', 'Kelvin', 'Second', 'Radian', 'Kilogram']*)
DimensionalUnits node creation:

```
'newNode:N='*newDimensionalUnits*'(parent:N,value=[CK.MassUnits,CK.LengthUnits,CK.TimeUnits,C
```

If a parent is given, the new <node> is added to the parent children list. new <node> is composed of a set of enumeration types : *MassUnits*,'LengthUnits', TimeUnits,TemperatureUnits,AngleUnits are required Returns a new <node> representing a DimensionalUnits_t sub-tree. chapter 4.3

CGNS.PAT.cgnslib.**newDimensionalExponents**(*parent*, *MassExponent=0*, *LengthExponent=0*, *TimeExponent=0*, *TemperatureExponent=0*, *AngleExponent=0*)
-DimensionalExponents node creation -DimensionalExponents:

```
'newNode:N='*newDimensionalExponents*'(parent:N,MassExponent:r,LengthExponent:r,TimeExponent:
```

If a parent is given, the new <node> is added to the parent children list. Returns a new <node> representing a DimensionalExponents_t sub-tree. chapter 4.4

CGNS.PAT.cgnslib.**newGridLocation**(*parent*, *value='CellCenter'*)
-GridLocation node creation -GridLocation:

```
'newNode:N='*newGridLocation*'(parent:N,value:CK.GridLocation)'
```

If a parent is given, the new <node> is added to the parent children list. Returns a new <node> representing a GridLocation_t sub-tree. chapter 4.5

CGNS.PAT.cgnslib.**newPointList**(*parent*, *name='PointList'*, *value=None*)
-PointList node creation -PointList

'newNode:N='*newPointList*'(parent:N,name:S,value:[])'

If a parent is given, the new <node> is added to the parent children list. Returns a new <node> representing a IndexArray_t sub-tree. chapter 4.6

CGNS.PAT.cgnslib.**newPointRange**(*parent*, *name='PointRange'*, *value=*[ ])
-PointRange node creation -PointRange

'newNode:N='*newPointRange*'(parent:N,name:S,value:[])'

If a parent is given, the new <node> is added to the parent children list. Returns a new <node> representing a IndexRange_t sub-tree. chapter 4.7

CGNS.PAT.cgnslib.**newRind**(*parent*, *value*)
-Rind node creation -Rind

'newNode:N='*newRind*'(parent:N,value=A)'

If a parent is given, the new <node> is added to the parent children list. Returns a new <node> representing a Rind_t sub-tree. chapter 4.8

CGNS.PAT.cgnslib.**newDataConversion**(*parent*, *ConversionScale=1.0*, *ConversionOffset=1.0*)
-DataConversion node creation -DataConversion

'newNode:N='*newDataConversion*'(parent:N,ConversionScale:r,ConversionOffset:r)'

If a parent is given, the new <node> is added to the parent children list. Returns a new <node> representing a DataConversion_t sub-tree. chapter 5.1.1

CGNS.PAT.cgnslib.**newSimulationType**(*parent*, *stype='NonTimeAccurate'*)
-SimulationType node creation -SimulationType

'newNode:N='*newSimulationType*'(parent:N,stype=CK.SimulationType)'

If a parent is given, the new <node> is added to the parent children list. Returns a new <node> representing a SimulationType_t sub-tree. chapter 6.2

CGNS.PAT.cgnslib.**newOrdinal**(*parent*, *value=0*)
-Ordinal node creation -Ordinal

'newNode:N='*newOrdinal*'(parent:N,value=i)'

If a parent is given, the new <node> is added to the parent children list. Returns a new <node> representing a Ordinal_t sub-tree. chapter 6.3

CGNS.PAT.cgnslib.**newZone**(*parent*, *name*, *zsize=None*, *ztype='Structured'*, *family=''*)
Zone node creation:

```
s=NPY.array([[10],[2],[0]],dtype='i')

T=newCGNSTree()
B=newBase(T,'Box-1',3,3)
Z=newZone(B,name,s,CK.Unstructured_s,'Wing')
```

•Args:

•*parent*: the parent node (<*node*> or *None*)

•*name*: zone name (*string*)

•*zsize*: array of dimensions (*numpy.ndarray*)

•*ztype*: zone type (*string*)

•*family*: zone family (*string*)

•Return:

•The new *Zone_t* node

•Remarks:

•The zone size has dimensions [IndexDimensions][3]

•Children:

•`newElements()`

CGNS.PAT.cgnslib.**newGridCoordinates**(*parent*, *name*)
-GridCoordinates node creation -Grid

'newNode:N='*newGridCoordinates*'(parent:N,name:S)'

Returns a new <node> representing a GridCoordinates_t sub-tree. If a parent is given, the new <node> is added to the parent children list.

CGNS.PAT.cgnslib.**newDataArray**(*parent*, *name*, *value=None*)
-DataArray node creation -Global

'newNode:N='*newDataArray*'(parent:N,name:S,value:A)'

Returns a new <node> representing a DataArray_t sub-tree. If a parent is given, the new <node> is added to the parent children list. chapter 5.1

CGNS.PAT.cgnslib.**newDiscreteData**(*parent*, *name*)
-DiscreteData node creation -DiscreteData

'newNode:N='*newDiscreteData*'(parent:N,name:S)'

If a parent is given, the new <node> is added to the parent children list. Returns a new <node> representing a DiscreteData_t sub-tree. If a parent is given, the new <node> is added to the parent children list. chapter 6.3

CGNS.PAT.cgnslib.**newElements**(*parent*, *name*, *etype='UserDefined'*, *econnectivity=None*, *erange=None*, *eboundary=0*)
Elements_t node creation:

```
quads=newElements(None,'QUADS',CGK.QUAD_4,quad_array,NPY.array(start,end))'
```

•Args:

•*parent*: the parent node (<*node*> or *None*)

•*name*: element node name (*string*)

•*etype*: the type of element (*string*)

•*econnectivity*: actual array of point connectivities (*numpy.ndarray*)

•*erange*: the first and last index of the connectivity (*numpy.ndarray*)

•*eboundary*: number of boundary elements (*int*)

•Return:

•The new *Elements_t* node

•Remarks:

•If a parent is given, the new node is added to the parent children list.

•The *elementsrange should* insure a unique and continuous index for all elements nodes in the same parent zone.

•Children:

•`newDescriptor()`

CGNS.PAT.cgnslib.**newBoundary**(*parent*, *bname*, *brange*, *btype='Null'*, *family=None*, *ptype='PointRange'*)
-BC node creation -BC

'newNode:N='*newBoundary*'(parent:N,bname:S,brange:[*i],btype:S)'

Returns a new <node> representing a BC_t sub-tree. If a parent is given, the new <node> is added to the parent children list. Parent should be Zone_t, returned node is parent. If the parent has already a child name ZoneBC then only the BC_t,IndexRange_t are created. chapter 9.3 Add IndexRange_t required

CGNS.PAT.cgnslib.**newBCDataSet**(*parent*, *name*, *valueType='Null'*)
-BCDataSet node creation -BCDataSet

'newNode:N='*newBCDataSet*'(parent:N,name:S,valueType:CK.BCTypeSimple)'

If a parent is given, the new <node> is added to the parent children list. Returns a new <node> representing a BCDataSet_t sub-tree. chapter 9.4 Add node BCTypeSimple is required

CGNS.PAT.cgnslib.**newBCData**(*parent*, *name*)
-BCData node creation -BCData

'newNode:N='*newBCData*'(parent:N,name:S)'

Returns a new <node> representing a BCData_t sub-tree. chapter 9.5

CGNS.PAT.cgnslib.**newBCProperty**(*parent*, *wallfunction='Null'*, *area='Null'*)
-BCProperty node creation -BCProperty

'newNode:N='*newBCProperty*'(parent:N)'

Returns a new <node> representing a BCProperty_t sub-tree. If a parent is given, the new <node> is added to the parent children list. chapter 9.6

CGNS.PAT.cgnslib.**newCoordinates**(*parent*, *name='GridCoordinates'*, *value=None*)
-GridCoordinates_t node creation with name GridCoordinates -Grid

'newNode:N='*newCoordinates*'(parent:N,name:S,value:A)'

Creates a new <node> representing a GridCoordinates_t sub-tree with the coordinate DataArray given as argument. This creates both the GridCoordinates_t with GridCoordinates name and DataArray_t with the argument name. Usually used to create the default grid. If the GridCoordinates_t with name GridCoordinates already exists then only the DataArray is created. If a parent is given, the new GridCoordinates_t <node> is added to the parent children list, in all cases the DataArray is child of GridCoordinates_t node. The returned node always is the DataArray_t node. chapter 7.1

CGNS.PAT.cgnslib.**newAxisymmetry**(*parent*, *refpoint=array([0., 0., 0.])*, *axisvector=array([0., 0., 0.])*)
-Axisymmetry node creation -Axisymmetry

'newNode:N='*newAxisymmetry*'(parent:N,refpoint:A,axisvector:A)'

refpoint,axisvector should be a real array. Returns a new <node> representing a CK.Axisymmetry_t sub-tree. chapter 7.5 Add DataArray AxisymmetryAxisVector,AxisymmetryReferencePoint are required

CGNS.PAT.cgnslib.**newRotatingCoordinates**(*parent*, *rotcenter=array([0., 0., 0.])*, *ratev=array([0., 0., 0.])*)
-RotatingCoordinates node creation -RotatingCoordinates

'newNode:N='*newRotatingCoordinates*'(parent:N,rotcenter=A,ratev=A)'

Returns a new <node> representing a RotatingCoordinates_t sub-tree. If a parent is given, the new <node> is added to the parent children list. rotcenter,ratev should be a real array. chapter 7.6 Add DataArray RotationRateVector,RotationCenter are required

CGNS.PAT.cgnslib.**newFlowSolution**(*parent*, *name='{FlowSolution}'*, *gridlocation=None*)
-Solution node creation -Solution

'newNode:N='*newSolution*'(parent:N,name:S,gridlocation:None)'

Returns a new <node> representing a FlowSolution_t sub-tree. chapter 7.7

CGNS.PAT.cgnslib.**newZoneGridConnectivity**(*parent*, *name='ZoneGridConnectivity'*)
-GridConnectivity node creation -Grid

'newNode:N='*newZoneGridConnectivity*'(parent:N,name:S)'

Creates a ZoneGridConnectivity_t sub-tree This sub-node is returned. If a parent is given, the new <node> is added to the parent children list, the parent should be a Zone_t. chapter 8.1

CGNS.PAT.cgnslib.**newGridConnectivity1to1**(*parent*, *name*, *dname*, *window*, *dwindow*, *trans*)

-GridConnectivity1to1 node creation -Grid

'newNode:N='*newGridConnectivity1to1*'(parent:N,name:S,dname:S,window:[i*],dwindow:[i*],trans:[i*])'

Creates a ZoneGridConnectivity1to1_t sub-tree. If a parent is given, the new <node> is added to the parent children list, the parent should be a Zone_t. The returned node is the GridConnectivity1to1_t chapter 8.2

CGNS.PAT.cgnslib.**newGridConnectivityProperty**(*parent*)

-GridConnectivityProperty node creation -GridConnectivityProperty

'newNode:N='*newGridConnectivityProperty*'(parent:N)'

Returns a new <node> representing a GridConnectivityProperty_t sub-tree. If a parent is given, the new <node> is added to the parent children list. chapter 8.5

CGNS.PAT.cgnslib.**newPeriodic**(*parent*, *rotcenter=array([0., 0., 0.])*, *ratev=array([0., 0., 0.])*, *trans=array([0., 0., 0.])*)

-Periodic node creation -Periodic

'newNode:N='*newPeriodic*'(parent:N,rotcenter=A,ratev=A,trans=A)'

Returns a new <node> representing a Periodic_t sub-tree. If a parent is given, the new <node> is added to the parent children list. If the parent has already a child name Periodic then only the RotationCenter,RotationAngle,Translation are created. rotcenter,ratev,trans should be a real array. chapter 8.5.1 Add DataArray RotationCenter,RotationAngle,Translation are required

CGNS.PAT.cgnslib.**newAverageInterface**(*parent*, *valueType='Null'*)

-AverageInterface node creation -AverageInterface

'newNode:N='*newAverageInterface*'(parent:N,valueType:CK.AverageInterfaceType)'

Returns a new <node> representing a AverageInterface_t sub-tree. If a parent is given, the new <node> is added to the parent children list. If the parent has already a child name AverageInterface then only the AverageInterfaceType is created. chapter 8.5.2

CGNS.PAT.cgnslib.**newOversetHoles**(*parent*, *name*, *hrange*)

-OversetHoles node creation -OversetHoles

'node:N='*newOversetHoles*'(parent:N,name:S,hrange:list)'

Creates a OversetHoles_t sub-tree. the parent should be a Zone_t. If a parent is given, the new <node> is added to the parent children list. chapter 8.6 Add PointList or List(PointRange) are required

CGNS.PAT.cgnslib.**newFlowEquationSet**(*parent*)

-FlowEquationSet node creation -FlowEquationSet

'newNode:N='*newFlowEquationSet*'(parent:N)'

If a parent is given, the new <node> is added to the parent children list. Returns a new <node> representing a CK.FlowEquationSet_t sub-tree. chapter 10.1

CGNS.PAT.cgnslib.**newGoverningEquations**(*parent*, *valueType='Euler'*)

-GoverningEquations node creation -GoverningEquations

'newNode:N='*newGoverningEquations*'(parent:N,valueType:CK.GoverningEquationsType)'

Returns a new <node> representing a CK.GoverningEquations_t sub-tree. If a parent is given, the new <node> is added to the parent children list. If the parent has already a child name GoverningEquations then only the GoverningEquationsType is created. chapter 10.2 Add node GoverningEquationsType is required

CGNS.PAT.cgnslib.**newGasModel**(*parent*, *valueType='Ideal'*)

-GasModel node creation -GasModel

'newNode:N='*newGasModel*'(parent:N,valueType:CK.GasModelType)'

Returns a new <node> representing a CK.GasModel_t sub-tree. If a parent is given, the new <node> is added to the parent children list. If the parent has already a child name GasModel then only the GasModelType is created. chapter 10.3 Add node GasModelType is required

CGNS.PAT.cgnslib.**newThermalConductivityModel** (*parent*, *valueType='SutherlandLaw'*)
-ThermalConductivityModel node creation -ThermalConductivityModel

'newNode:N='*newThermalConductivityModel*'(parent:N,valueType:CK.ThermalConductivityModelType)'

Returns a new <node> representing a CK.ThermalConductivityModel_t sub-tree. If a parent is given, the new <node> is added to the parent children list. If the parent has already a child name ThermalConductivityModel then only the ThermalConductivityModelType is created. chapter 10.5 Add node ThermalConductivityModelType is required

CGNS.PAT.cgnslib.**newViscosityModel** (*parent*, *valueType='SutherlandLaw'*)
-ViscosityModel node creation -ViscosityModel

'newNode:N='*newViscosityModel*'(parent:N,valueType:CK.ViscosityModelType)'

Returns a new <node> representing a CK.ViscosityModel_t sub-tree. If a parent is given, the new <node> is added to the parent children list. If the parent has already a child name ViscosityModel then only the ViscosityModelType is created. chapter 10.4 Add node ViscosityModelType is (r)

CGNS.PAT.cgnslib.**newTurbulenceClosure** (*parent*, *valueType='EddyViscosity'*)
-TurbulenceClosure node creation -TurbulenceClosure

'newNode:N='*newTurbulenceClosure*'(parent:N,valueType:CK.TurbulenceClosureType)' Returns a new <node> representing a CK.TurbulenceClosure_t sub-tree. If a parent is given, the new <node> is added to the parent children list. If the parent has already a child name TurbulenceClosure then only the ViscosityModelType is created. chapter 10.5 Add node TurbulenceClosureType is (r)

CGNS.PAT.cgnslib.**newTurbulenceModel** (*parent*, *valueType='OneEquation_SpalartAllmaras'*)
-TurbulenceModel node creation -TurbulenceModel

'newNode:N='*newTurbulenceModel*'(parent:N,valueType:CK.TurbulenceModelType)'

Returns a new <node> representing a CK.TurbulenceModel_t sub-tree. If a parent is given, the new <node> is added to the parent children list. If the parent has already a child name TurbulenceModel then only the TurbulenceModelType is created. chapter 10.6.2 Add node TurbulenceModelType is (r)

CGNS.PAT.cgnslib.**newThermalRelaxationModel** (*parent*, *valueType*)
-ThermalRelaxationModel node creation -ThermalRelaxationModel

'newNode:N='*newThermalRelaxationModel*'(parent:N,valueType:CK.ThermalRelaxationModelType)'

Returns a new <node> representing a CK.ThermalRelaxationModel_t sub-tree. If a parent is given, the new <node> is added to the parent children list. If the parent has already a child name ThermalRelaxationModel then only the ThermalRelaxationModelType is created. chapter 10.7 Add node ThermalRelaxationModelType is (r)

CGNS.PAT.cgnslib.**newChemicalKineticsModel** (*parent*, *valueType='Null'*)
-ChemicalKineticsModel node creation -ChemicalKineticsModel

'newNode:N='*newChemicalKineticsModel*'(parent:N,valueType:CK.ChemicalKineticsModelType)'

Returns a new <node> representing a CK.ChemicalKineticsModel_t sub-tree. If a parent is given, the new <node> is added to the parent children list. If the parent has already a child name ChemicalKineticsModel then only the ChemicalKineticsModelType is created. chapter 10.8 Add node ChemicalKineticsModelType is (r)

CGNS.PAT.cgnslib.**newEMElectricFieldModel** (*parent*, *valueType='UserDefined'*)
-EMElectricFieldModel node creation -EMElectricFieldModel

'newNode:N='*newEMElectricFieldModel*'(parent:N,valueType:CK.EMElectricFieldModelType)'

Returns a new <node> representing a CK.EMElectricFieldModel_t sub-tree. If a parent is given, the new <node> is added to the parent children list.

If the parent has already a child name EMElectricFieldModel then

only the EMElectricFieldModelType is created. chapter 10.9 Add node EMElectricFieldModel-Type is (r)

CGNS.PAT.cgnslib.**newEMMagneticFieldModel** (*parent*, *valueType='UserDefined'*)
-EMMagneticFieldModel node creation -EMMagneticFieldModel

'newNode:N='*newEMMagneticFieldModel*'(parent:N,valueType:CK.EMMagneticFieldModelType)'

Returns a new <node> representing a CK.EMMagneticFieldModel_t sub-tree. If a parent is given, the new <node> is added to the parent children list. If the parent has already a child name EMMagneticFieldModel_s then only the EMMagneticFieldModelType is created. chapter 10.9.2 Add node EMMagneticFieldModelType is (r)

CGNS.PAT.cgnslib.**newEMConductivityModel** (*parent*, *valueType='UserDefined'*)
-EMConductivityModel node creation -EMConductivityModel

'newNode:N='*newEMConductivityModel*'(parent:N,valueType:CK.EMConductivityModelType)'

Returns a new <node> representing a CK.EMConductivityModel_t sub-tree. If a parent is given, the new <node> is added to the parent children list. If the parent has already a child name EMConductivityModel then only the EMConductivityModelType is created. chapter 10.9.3 Add node EMConductivityModelType is (r)

CGNS.PAT.cgnslib.**newBaseIterativeData** (*parent*, *nsteps=0*, *itype='IterationValues'*)
-BaseIterativeData node creation -BaseIterativeData

'newNode:N='*newBaseIterativeData*'(parent:N,nsteps:I,itype:E)'

Returns a new <node> representing a BaseIterativeData_t sub-tree. If a parent is given, the new <node> is added to the parent children list. chapter 11.1.1 NumberOfSteps is required, TimeValues or IterationValues are required

CGNS.PAT.cgnslib.**newZoneIterativeData** (*parent*, *name*)
-ZoneIterativeData node creation -ZoneIterativeData

'newNode:N='*newZoneIterativeData*'(parent:N,name:S)'

Returns a new <node> representing a ZoneIterativeData_t sub-tree. If a parent is given, the new <node> is added to the parent children list. chapter 11.1.2

CGNS.PAT.cgnslib.**newRigidGridMotion** (*parent*, *name*, *valueType='Null'*, *vector=array([0., 0., 0.])*)
-RigidGridMotion node creation -RigidGridMotion

'newNode:N='*newRigidGridMotion*'(parent:N,name:S,valueType:CK.RigidGridMotionType,vector:A)'

If a parent is given, the new <node> is added to the parent children list. Returns a new <node> representing a CK.RigidGridMotion_t sub-tree. If the parent has already a child name RigidGridMotion then only the RigidGridMotionType is created and OriginLocation is created chapter 11.2 Add Node RigidGridMotionType and add DataArray OriginLocation are the only required

CGNS.PAT.cgnslib.**newReferenceState** (*parent*, *name='ReferenceState'*)
-ReferenceState node creation -ReferenceState

'newNode:N='*newReferenceState*'(parent:N,name:S)'

Returns a new <node> representing a ReferenceState_t sub-tree. If a parent is given, the new <node> is added to the parent children list. chapter 12.1

CGNS.PAT.cgnslib.**newConvergenceHistory** (*parent*, *name='GlobalConvergenceHistory'*, *iterations=0*)
-ConvergenceHistory node creation -ConvergenceHistory

'newNode:N='*newConvergenceHistory*'(parent:N,name:S,iterations:i)'

Returns a new <node> representing a ConvergenceHistory_t sub-tree. If a parent is given, the new <node> is added to the parent children list. chapter 12.3

CGNS.PAT.cgnslib.**newIntegralData**(*parent*, *name*)
-IntegralData node creation -IntegralData

'newNode:N='*newIntegralData*'(parent:N,name:S)'

Returns a new <node> representing a IntegralData_t sub-tree. If a parent is given, the new <node> is added to the parent children list. chapter 12.5

CGNS.PAT.cgnslib.**newFamily**(*parent*, *name*)
-Family node creation -Family

'newNode:N='*newFamily*'(parent:N,name:S)'

Returns a new <node> representing a Family_t sub-tree. If a parent is given, the new <node> is added to the parent children list. chapter 12.6

CGNS.PAT.cgnslib.**newGeometryReference**(*parent*, *name='{GeometryReference}'*, *valueType='UserDefined'*)
-GeometryReference node creation -GeometryReference

'newNode:N='*newGeometryReference*'(parent:N,name:S,valueType:CK.GeometryFormat)'

Returns a new <node> representing a CK.GeometryFormat_t sub-tree. If a parent is given, the new <node> is added to the parent children list. If the parent has already a child name CK.GeometryReference then only the .GeometryFormat is created chapter 12.7 Add node CK.GeometryFormat_t is (r) and GeometryFile_t definition not find but is required (CAD file)

CGNS.PAT.cgnslib.**newFamilyBC**(*parent*, *valueType='UserDefined'*)
-FamilyBC node creation -FamilyBC

'newNode:N='*newFamilyBC*'(parent:N,valueType:CK.BCTypeSimple/CK.BCTypeCompound)'

Returns a new <node> representing a CK.FamilyBC_t sub-tree. If a parent is given, the new <node> is added to the parent children list. If the parent has already a child name FamilyBC then only the BCType is created chapter 12.8 Add node BCType is required

CGNS.PAT.cgnslib.**newArbitraryGridMotion**(*parent*, *name*, *valuetype='Null'*)
Returns a **new node** representing a *ArbitraryGridMotionType_t*

Parameters

- **parent** – CGNS/Python node
- **name** – String
- **valuetype** – String (`CGNS.PAT.cgnskeywords.ArbitraryGridMotionType`)

If a *parent* is not `None`, the **new node** is added to the parent children list. If the *parent* has already a child with name `RigidGridMotion` then only the `RigidGridMotionType` is created.

CGNS.PAT.cgnslib.**newUserDefinedData**(*parent*, *name*)
-UserDefinedData node creation -UserDefinedData

'newNode:N='*newUserDefinedData*'(parent:N,name:S)'

Returns a new <node> representing a UserDefinedData_t sub-tree. If a parent is given, the new <node> is added to the parent children list. chapter 12.9

CGNS.PAT.cgnslib.**newGravity**(*parent*, *gvector=array([0., 0., 0.])*)
-Gravity node creation -Gravity

'newNode:N='*newGravity*'(parent:N,gvector:A)'

Returns a new <node> representing a Gravity_t sub-tree. If a parent is given, the new <node> is added to the parent children list. gvector should be a real array chapter 12.10 Add DataArray GravityVector is required

CHAPTER

THREE

SIDS PATTERNS

The patterns are importable modules, they create a complete *SIDS* sub-tree with default values. There is no way to customize the default values or the actual contents of the sub-tree. The pattern creates the mandatory as well as the optional nodes. Once created, the user has to modify the sub-tree using the *PAT.cgnsutils* or *PAT.cgnslib* functions.

Once the pattern module is imported, the actual pattern is referenced by the *data* variable:

```
import BaseIterativeData_t.data as mysubtree
```

The pattern is a *CGNS/Python* list and thus it should be copied before any modification:

```
import BaseIterativeData_t
import copy

t=BaseIterativeData_t.data

t1=copy.deepcopy(t)
t2=copy.deepcopy(t)
```

For example, you can use *PAT.cgnslib* to create a *BaseIterativeData_t* node with:

```
data=C.newBaseIterativeData(None)
```

This call create the unique *BaseIterativeData_t* node (or sub-tree which is the same in this case because we have only one node). The new node is returned, the *None* argument means we do not define a parent node, it is up to the user to add this new node in a existing children list.

Now we can use the *PAT.SIDS.BaseIterativeData_t* which creates the same *BaseIterativeData_t* node as before, but also create the whole *SIDS* sub-tree with default values, here is a snippet of this pattern:

```
import CGNS.PAT.cgnslib      as C
import CGNS.PAT.cgnskeywords as K

data=C.newBaseIterativeData(None)
C.newDataArray(data,K.NumberOfZones_s)
C.newDataArray(data,K.NumberOfFamilies_s)
C.newDataArray(data,K.ZonePointers_s)
C.newDataArray(data,K.FamilyPointers_s)
C.newDataArray(data,'{DataArray}')
C.newDataClass(data)
C.newDimensionalUnits(data)
C.newUserDefinedData(data,'{UserDefinedData}')
C.newDescriptor(data,'{Descriptor}')
```

You see all the mandatory and optional *SIDS* nodes are created, the user has to set his own values in the resulting sub-tree using the *PAT.cgnslib* or the *PAT.cgnsutils* functions.

CHAPTER

FOUR

CGNS KEYWORDS

Instead of generating a new doc from a file, the file itself is included here. The purpose of *cgnskeywords.py* is to declare all constants as Python variables. This leads to several advantages:

- You cannot make a typo on a name. For example, if you use "ZoneGridConnectivity" as a plain string you may mistype it as "Zonegridconnectivity" or "ZoneGridConectivity" and this may silently produce a bad CGNS tree.
- You can handle enumerate as lists. For example you have lists for units: MassUnits_l, LengthUnits_l, AllDimensionalUnits_l, AllUnits_l
- You can identify what is a CGNS reserved or recommended name or not.

```
#  -------------------------------------------------------------------------
#  pyCGNS.PAT - Python package for CFD General Notation System - PATternMaker
#  See license.txt file in the root directory of this Python module source
#  -------------------------------------------------------------------------
#  $Release: v4.0.1 $
#  -------------------------------------------------------------------------
"""
 TYPES, ENUMERATES, CONSTANTS, NAMES from CGNS/MLL

 Conventions:

 [1] A CGNS/SIDS string constant is postfixed with _s
 'ZoneType' is ZoneType_s

 [2] A CGNS/SIDS string constant naming a type has _ts
 'ZoneType_t' is ZoneType_ts

 [3] A list of possible values for a given type has _l
 ZoneType_l is [Null_s,UserDefined_s,Structured_s,Unstructured_s]
 which is same as ["Null","UserDefined","Structured","Unstructured"]
 List should be ordered wrt the actual enumerate

 [4] An enumerate mapping of a list of values is not prefixed
 ZoneType is {'Unstructured':3,'Null':0,'Structured':2,'UserDefined':1}

 [5] The reverse dictionnary of the previous one is postfixed with _
 ZoneType_ is {0:'Null',1:'UserDefined',2:'Structured',3:'Unstructured'}

 [6] The variables are declared with an integer value (not enumerates)
 wrt their position in the _l list, for example:
 (Null,UserDefined,Structured,Unstructured)=ZoneType_.keys()

 [7] The _t type names are reserved for Cython, enums are then used as int:
 ctypedef int DataType_t
 int cg_array_read_as(int A, DataType_t type, void *Data)

"""
```

```
# -----------------------------------------------------------------------------
import CGNS.pyCGNSconfig

def stringAsKeyDict(l):
  return dict(zip(l,range(len(l))))

def enumAsKeyDict(l):
  return dict(zip(range(len(l)),l))

# ----------------------------------------------------
# --- ADF-level Datatypes
#
adftypes=('C1','I4','I8','R4','R8','MT','LK')
(C1,I4,I8,R4,R8,MT,LK)=adftypes

# ----------------------------------------------------
# --- ADF-level Constants
#
ADF_DATA_TYPE_LENGTH     =   32
ADF_DATE_LENGTH          =   32
ADF_FILENAME_LENGTH      = 1024
ADF_FORMAT_LENGTH        =   20
ADF_LABEL_LENGTH         =   32
ADF_MAXIMUM_LINK_DEPTH   =  100
ADF_MAX_DIMENSIONS       =   12
ADF_MAX_ERROR_STR_LENGTH =   80
ADF_MAX_LINK_DATA_SIZE   = 4096
ADF_NAME_LENGTH          =   32
ADF_STATUS_LENGTH        =   32
ADF_VERSION_LENGTH       =   32

ADF_ROOT_NODE_NAME  = "HDF5 MotherNode"
ADF_ROOT_NODE_LABEL = "Root Node of HDF5 File"

CGNSHDF5ROOT_s = ADF_ROOT_NODE_NAME

# ---------------------------------------------------- (NOT SIDS)
# --- CGNS/Python mapping extensions
#
CGNSTree_ts           = 'CGNSTree_t'
CGNSTree_s            = 'CGNSTree'

# --- Type with weird (coming from outer space) names
#
Transform_ts          = 'Transform_t"'
DiffusionModel_ts     = 'DiffusionModel_t'
EquationDimension_ts  = 'EquationDimension_t'
InwardNormalIndex_ts  = 'InwardNormalIndex_t'
IntIndexDimension_ts  = 'IntIndexDimension_ts'

# --- Add legacy strings for translation tools
#
Transform_ts2         = '"int[IndexDimension]"'
DiffusionModel_ts2    = '"int[1+...+IndexDimension]"'
EquationDimension_ts2 = '"int"'
InwardNormalIndex_ts2 = '"int[IndexDimension]"'

weirdSIDStypes={
  Transform_ts2:         IntIndexDimension_ts,
  DiffusionModel_ts2:    DiffusionModel_ts,
  EquationDimension_ts2: EquationDimension_ts,
  InwardNormalIndex_ts2: IntIndexDimension_ts,
```

```
  }

weirdSIDStypes_={
  Transform_ts:         Transform_ts2,
  DiffusionModel_ts:    DiffusionModel_ts2,
  EquationDimension_ts: EquationDimension_ts2,
  InwardNormalIndex_ts: InwardNormalIndex_ts2,
  }

# ------------------------------------------------ (SIDS)
# SIDS
#
Null_s        = "Null"
UserDefined_s = "UserDefined"

# ------------------------------------------------
Kilogram_s  = "Kilogram"
Gram_s      = "Gram"
Slug_s      = "Slug"
PoundMass_s = "PoundMass"
MassUnits_l = [Null_s,UserDefined_s,
               Kilogram_s,Gram_s,Slug_s,PoundMass_s]

MassUnits   = stringAsKeyDict(MassUnits_l)
MassUnits_  = enumAsKeyDict(MassUnits_l)
(MassUnitsNull,MassUnitsUserDefined,
 Kilogram,Gram,Slug,PoundMass)=MassUnits_.keys()

# ------------------------------------------------
Meter_s       = "Meter"
Centimeter_s  = "Centimeter"
Millimeter_s  = "Millimeter"
Foot_s        = "Foot"
Inch_s        = "Inch"
LengthUnits_l = [Null_s,UserDefined_s,
                 Meter_s,Centimeter_s,Millimeter_s,Foot_s,Inch_s]

LengthUnits   = stringAsKeyDict(LengthUnits_l)
LengthUnits_  = enumAsKeyDict(LengthUnits_l)
(LengthUnitsNull,LengthUnitsUserDefined,
 Meter,Centimeter,Millimeter,Foot,Inch)=LengthUnits_.keys()

# ------------------------------------------------
Second_s    = "Second"
TimeUnits_l = [Null_s,UserDefined_s,Second_s]

TimeUnits   = stringAsKeyDict(TimeUnits_l)
TimeUnits_  = enumAsKeyDict(TimeUnits_l)
(TimeUnitsNull,TimeUnitsUserDefined,Seconds)=TimeUnits_.keys()

# ------------------------------------------------
Kelvin_s           = "Kelvin"
Celcius_s          = "Celcius"
Rankine_s          = "Rankine"
Fahrenheit_s       = "Fahrenheit"
TemperatureUnits_l = [Null_s,UserDefined_s,
                      Kelvin_s,Celcius_s,Rankine_s,Fahrenheit_s]

TemperatureUnits   = stringAsKeyDict(TemperatureUnits_l)
TemperatureUnits_  = enumAsKeyDict(TemperatureUnits_l)
(TemperatureUnitsNull,TemperatureUnitsUserDefined,
 Kelvin,Celcius,Rankine,Fahrenheit)=TemperatureUnits_.keys()
```

```
# ------------------------------------------------
Degree_s     = "Degree"
Radian_s     = "Radian"
AngleUnits_l = [Null_s,UserDefined_s,Degree_s,Radian_s]

AngleUnits   = stringAsKeyDict(AngleUnits_l)
AngleUnits_  = enumAsKeyDict(AngleUnits_l)
(AngleUnitsNull,AngleUnitsUserDefined,Degree,Radian)=AngleUnits_.keys()

# ------------------------------------------------
Ampere_s               = "Ampere"
Abampere_s             = "Abampere"
Statampere_s           = "Statampere"
Edison_s               = "Edison"
auCurrent_s            = "auCurrent"
ElectricCurrentUnits_l = [Null_s,UserDefined_s,
                          Ampere_s,Abampere_s,Statampere_s,
                          Edison_s,auCurrent_s]

ElectricCurrentUnits   = stringAsKeyDict(ElectricCurrentUnits_l)
ElectricCurrentUnits_  = enumAsKeyDict(ElectricCurrentUnits_l)
(ElectricCurrentUnitsNull,ElectricCurrentUnitsUserDefined,
 Ampere,Abampere,Statampere,
 Edison,auCurrent)=ElectricCurrentUnits_.keys()

# ------------------------------------------------
Mole_s                 = "Mole"
Entities_s             = "Entities"
StandardCubicFoot_s    = "StandardCubicFoot"
StandardCubicMeter_s   = "StandardCubicMeter"
SubstanceAmountUnits_l =[Null_s,UserDefined_s,
                         Mole_s,Entities_s,
                         StandardCubicFoot_s,StandardCubicMeter_s]

SubstanceAmountUnits   = stringAsKeyDict(SubstanceAmountUnits_l)
SubstanceAmountUnits_  = enumAsKeyDict(SubstanceAmountUnits_l)
(SubstanceAmountUnitsNull,SubstanceAmountUnitsUserDefined,
 Mole,Entities,
 StandardCubicFoot,StandardCubicMeter)=SubstanceAmountUnits_.keys()

# ------------------------------------------------
Candela_s = "Candela"
Candle_s  = "Candle"
Carcel_s  = "Carcel"
Hefner_s  = "Hefner"
Violle_s  = "Violle"
LuminousIntensityUnits_l = [Null_s,UserDefined_s,
                            Candela_s,Candle_s,Carcel_s,Hefner_s,Violle_s]

LuminousIntensityUnits   = stringAsKeyDict(LuminousIntensityUnits_l)
LuminousIntensityUnits_  = enumAsKeyDict(LuminousIntensityUnits_l)
(LuminousIntensityUnitsNull,LuminousIntensityUnitsUserDefined,
 Candela,Candle,Carcel,Hefner,Violle)=LuminousIntensityUnits_.keys()


# ------------------------------------------------
DimensionalUnits_s    = "DimensionalUnits"
AdditionalUnits_s     = "AdditionalUnits"
AdditionalExponents_s = "AdditionalExponents"

AllDimensionalUnits_l = TimeUnits_l+MassUnits_l+LengthUnits_l\
                        +TemperatureUnits_l+AngleUnits_l
AllAdditionalUnits_l  = LuminousIntensityUnits_l+SubstanceAmountUnits_l\
```

```
                        +ElectricCurrentUnits_l
AllUnits_l            = AllDimensionalUnits_l+AllAdditionalUnits_l

# ----------------------------------------------
Dimensional_s                    = "Dimensional"
NormalizedByDimensional_s        = "NormalizedByDimensional"
NormalizedByUnknownDimensional_s = "NormalizedByUnknownDimensional"
NondimensionalParameter_s        = "NondimensionalParameter"
DimensionlessConstant_s          = "DimensionlessConstant"
DataClass_l=[Dimensional_s,NormalizedByDimensional_s,
             NormalizedByUnknownDimensional_s,NondimensionalParameter_s,
             DimensionlessConstant_s,Null_s,UserDefined_s]

DataClass_ts = "DataClass_t"
DataClass_s  = "DataClass"

# ------------------------------------------------------
GridLocation_ts= "GridLocation_t"
GridLocation_s = "GridLocation"

Vertex_s       = "Vertex"
CellCenter_s   = "CellCenter"
FaceCenter_s   = "FaceCenter"
IFaceCenter_s  = "IFaceCenter"
JFaceCenter_s  = "JFaceCenter"
KFaceCenter_s  = "KFaceCenter"
EdgeCenter_s   = "EdgeCenter"

GridLocation_l = [Null_s,UserDefined_s,Vertex_s,CellCenter_s,FaceCenter_s,
                  IFaceCenter_s,JFaceCenter_s,KFaceCenter_s,
                  EdgeCenter_s]
GridLocation   = stringAsKeyDict(GridLocation_l)
GridLocation_  = enumAsKeyDict(GridLocation_l)
(Null,UserDefined,Vertex,CellCenter,FaceCenter,
 IFaceCenter,JFaceCenter,KFaceCenter,EdgeCenter)=GridLocation_.keys()

# ------------------------------------------------------------
PointSetType_ts = "PointSetType_t"

PointList_s       = "PointList"
PointListDonor_s  = "PointListDonor"
PointRange_s      = "PointRange"
PointRangeDonor_s = "PointRangeDonor"
ElementRange_s    = "ElementRange"
ElementList_s     = "ElementList"
CellListDonor_s   = "CellListDonor"

PointSetType_l = [Null_s,UserDefined_s,
                  PointList_s,PointListDonor_s,PointRange_s,PointRangeDonor_s,
                  ElementRange_s,ElementList_s,CellListDonor_s]
PointSetType  = stringAsKeyDict(PointSetType_l)
PointSetType_ = enumAsKeyDict(PointSetType_l)
(Null,UserDefined,PointList,PointListDonor,PointRange,PointRangeDonor,
 ElementRange,ElementList,CellListDonor)=PointSetType_.keys()

# ------------------------------------------------------------
BCDataType_ts = "BCDataType_t"
BCDataType_s  = "BCDataType"

DirichletData_s = "DirichletData"
NeumannData_s   = "NeumannData"
Dirichlet_s     = "Dirichlet"
Neumann_s       = "Neumann"
```

```
BCDataType_l=[Null_s,UserDefined_s,Dirichlet_s,Neumann_s]
BCDataType   = stringAsKeyDict(BCDataType_l)
BCDataType_  = enumAsKeyDict(BCDataType_l)
(BCDataTypeNull,BCDataTypeUserDefined,Dirichlet,Neumann)=BCDataType_.keys()

FullPotential_s               = "FullPotential"
Euler_s                       = "Euler"
NSLaminar_s                   = "NSLaminar"
NSTurbulent_s                 = "NSTurbulent"
NSLaminarIncompressible_s     = "NSLaminarIncompressible"
NSTurbulentIncompressible_s   = "NSTurbulentIncompressible"

Ideal_s                       = "Ideal"
VanderWaals_s                 = "VanderWaals"
Constant_s                    = "Constant"
PowerLaw_s                    = "PowerLaw"
SutherlandLaw_s               = "SutherlandLaw"
ConstantPrandtl_s             = "ConstantPrandtl"
EddyViscosity_s               = "EddyViscosity"
ReynoldsStress_s              = "ReynoldsStress"
Algebraic_s                   = "Algebraic"
BaldwinLomax_s                = "BaldwinLomax"
ReynoldsStressAlgebraic_s     = "ReynoldsStressAlgebraic"
Algebraic_BaldwinLomax_s      = "Algebraic_BaldwinLomax"
Algebraic_CebeciSmith_s       = "Algebraic_CebeciSmith"
HalfEquation_JohnsonKing_s    = "HalfEquation_JohnsonKing"
OneEquation_BaldwinBarth_s    = "OneEquation_BaldwinBarth"
OneEquation_SpalartAllmaras_s = "OneEquation_SpalartAllmaras"
TwoEquation_JonesLaunder_s    = "TwoEquation_JonesLaunder"
TwoEquation_MenterSST_s       = "TwoEquation_MenterSST"
TwoEquation_Wilcox_s          = "TwoEquation_Wilcox"
CaloricallyPerfect_s          = "CaloricallyPerfect"
ThermallyPerfect_s            = "ThermallyPerfect"
ConstantDensity_s             = "ConstantDensity"
RedlichKwong_s                = "RedlichKwong"
Frozen_s                      = "Frozen"
ThermalEquilib_s              = "ThermalEquilib"
ThermalNonequilib_s           = "ThermalNonequilib"
ChemicalEquilibCurveFit_s     = "ChemicalEquilibCurveFit"
ChemicalEquilibMinimization_s = "ChemicalEquilibMinimization"
ChemicalNonequilib_s          = "ChemicalNonequilib"
EMElectricField_s             = "EMElectricField"
EMMagneticField_s             = "EMMagneticField"
EMConductivity_s              = "EMConductivity"
Voltage_s                     = "Voltage"
Interpolated_s                = "Interpolated"
Equilibrium_LinRessler_s      = "Equilibrium_LinRessler"
Chemistry_LinRessler_s        = "Chemistry_LinRessler"

FamilySpecified_s             = "FamilySpecified"

# ----------------------------------------------
DataType_ts    = "DataType_t"
DataType_s     = "DataType"

Integer_s      = "Integer"
LongInteger_s  = "LongInteger"
RealSingle_s   = "RealSingle"
RealDouble_s   = "RealDouble"
Character_s    = "Character"

DataType_l = [Null_s,UserDefined_s,
```

```
              Integer_s,RealSingle_s,RealDouble_s,Character_s,LongInteger_s]
DataType    = stringAsKeyDict(DataType_l)
DataType_   = enumAsKeyDict(DataType_l)

(DataTypeNull,DataTypeUserDefined,\
 Integer,RealSingle,RealDouble,Character,LongInteger)=DataType_.keys()

# ----------------------------------------------------
GridConnectivityType_ts = "GridConnectivityType_t"
GridConnectivityType_s  = "GridConnectivityType"
GridConnectivity_ts     = "GridConnectivity_t"
ZoneGridConnectivity_ts = "ZoneGridConnectivity_t"
ZoneGridConnectivity_s  = "ZoneGridConnectivity"

Overset_s       = "Overset"
Abutting_s      = "Abutting"
Abutting1to1_s  = "Abutting1to1"

GridConnectivityType_l = [Null_s,UserDefined_s,
                          Overset_s,Abutting_s,Abutting1to1_s]
GridConnectivityType   = stringAsKeyDict(GridConnectivityType_l)
GridConnectivityType_  = enumAsKeyDict(GridConnectivityType_l)

(Null,UserDefined,
 Overset,Abutting,Abutting1to1)=GridConnectivityType_.keys()

# ----------------------------------------------------
ZoneType_ts     = "ZoneType_t"
ZoneType_s      = "ZoneType"
Zone_ts         = "Zone_t"

Structured_s    = "Structured"
Unstructured_s  = "Unstructured"

ZoneType_l      = [Null_s,UserDefined_s,Structured_s,Unstructured_s]
ZoneType        = stringAsKeyDict(ZoneType_l)
ZoneType_       = enumAsKeyDict(ZoneType_l)

(ZoneTypeNull,ZoneTypeUserdefined,Structured,Unstructured)=ZoneType_.keys()

# ----------------------------------------------------
SimulationType_ts = "SimulationType_t"
SimulationType_s  = "SimulationType"

TimeAccurate_s    = "TimeAccurate"
NonTimeAccurate_s = "NonTimeAccurate"

SimulationType_l  = [Null_s,UserDefined_s,TimeAccurate_s,NonTimeAccurate_s]
SimulationType    = stringAsKeyDict(SimulationType_l)
SimulationType_   = enumAsKeyDict(SimulationType_l)

(Null,UserDefined,TimeAccurate,NonTimeAccurate)=SimulationType_.keys()

# ----------------------------------------------------
ConstantRate_s        = "ConstantRate"
VariableRate_s        = "VariableRate"
NonDeformingGrid_s    = "NonDeformingGrid"
DeformingGrid_s       = "DeformingGrid"
RigidGridMotionType_l = [Null_s,ConstantRate_s,VariableRate_s,UserDefined_s]

RigidGridMotionType_s="RigidGridMotionType"
RigidGridMotionType_ts="RigidGridMotionType_t"
```

```
Generic_s                   = "Generic"
BleedArea_s                 = "BleedArea"
CaptureArea_s               = "CaptureArea"
AverageAll_s                = "AverageAll"
AverageCircumferential_s    = "AverageCircumferential"
AverageRadial_s             = "AverageRadial"
AverageI_s                  = "AverageI"
AverageJ_s                  = "AverageJ"
AverageK_s                  = "AverageK"
CGNSLibraryVersion_s        = "CGNSLibraryVersion"
CellDimension_s             = "CellDimension"
PhysicalDimension_s         = "PhysicalDimension"
GridCoordinates_s           = "GridCoordinates"
CoordinateNames_s           = "CoordinateNames"
CoordinateX_s               = "CoordinateX"
CoordinateY_s               = "CoordinateY"
CoordinateZ_s               = "CoordinateZ"
CoordinateR_s               = "CoordinateR"
CoordinateTheta_s           = "CoordinateTheta"
CoordinatePhi_s             = "CoordinatePhi"
CoordinateNormal_s          = "CoordinateNormal"
CoordinateTangential_s      = "CoordinateTangential"
CoordinateXi_s              = "CoordinateXi"
CoordinateEta_s             = "CoordinateEta"
CoordinateZeta_s            = "CoordinateZeta"
CoordinateTransform_s       = "CoordinateTransform"
InterpolantsDonor_s         = "InterpolantsDonor"
ElementConnectivity_s       = "ElementConnectivity"
ParentData_s                = "ParentData"
ParentElements_s            = "ParentElements"
ParentElementsPosition_s    = "ParentElementsPosition"
ElementSizeBoundary_s       = "ElementSizeBoundary"
VectorX_ps                  = "%sX"
VectorY_ps                  = "%sY"
VectorZ_ps                  = "%sZ"
VectorTheta_ps              = "%sTheta"
VectorPhi_ps                = "%sPhi"
VectorMagnitude_ps          = "%sMagnitude"
VectorNormal_ps             = "%sNormal"
VectorTangential_ps         = "%sTangential"
Potential_s                 = "Potential"
StreamFunction_s            = "StreamFunction"
Density_s                   = "Density"
Pressure_s                  = "Pressure"
Temperature_s               = "Temperature"
EnergyInternal_s            = "EnergyInternal"
Enthalpy_s                  = "Enthalpy"
Entropy_s                   = "Entropy"
EntropyApprox_s             = "EntropyApprox"
DensityStagnation_s         = "DensityStagnation"
PressureStagnation_s        = "PressureStagnation"
TemperatureStagnation_s     = "TemperatureStagnation"
EnergyStagnation_s          = "EnergyStagnation"
EnthalpyStagnation_s        = "EnthalpyStagnation"
EnergyStagnationDensity_s   = "EnergyStagnationDensity"
VelocityX_s                 = "VelocityX"
VelocityY_s                 = "VelocityY"
VelocityZ_s                 = "VelocityZ"
VelocityR_s                 = "VelocityR"
VelocityTheta_s             = "VelocityTheta"
VelocityPhi_s               = "VelocityPhi"
VelocityMagnitude_s         = "VelocityMagnitude"
VelocityNormal_s            = "VelocityNormal"
```

```
VelocityTangential_s          = "VelocityTangential"
VelocitySound_s               = "VelocitySound"
VelocitySoundStagnation_s     = "VelocitySoundStagnation"
MomentumX_s                   = "MomentumX"
MomentumY_s                   = "MomentumY"
MomentumZ_s                   = "MomentumZ"
MomentumMagnitude_s           = "MomentumMagnitude"
RotatingVelocityX_s           = "RotatingVelocityX"
RotatingVelocityY_s           = "RotatingVelocityY"
RotatingVelocityZ_s           = "RotatingVelocityZ"
RotatingMomentumX_s           = "RotatingMomentumX"
RotatingMomentumY_s           = "RotatingMomentumY"
RotatingMomentumZ_s           = "RotatingMomentumZ"
RotatingVelocityMagnitude_s   = "RotatingVelocityMagnitude"
RotatingPressureStagnation_s  = "RotatingPressureStagnation"
RotatingEnergyStagnation_s    = "RotatingEnergyStagnation"
RotatingEnergyStagnationDensity_s = "RotatingEnergyStagnationDensity"
RotatingEnthalpyStagnation_s  = "RotatingEnthalpyStagnation"
EnergyKinetic_s               = "EnergyKinetic"
PressureDynamic_s             = "PressureDynamic"
SoundIntensityDB_s            = "SoundIntensityDB"
SoundIntensity_s              = "SoundIntensity"
VorticityX_s                  = "VorticityX"
VorticityY_s                  = "VorticityY"
VorticityZ_s                  = "VorticityZ"
VorticityMagnitude_s          = "VorticityMagnitude"
SkinFrictionX_s               = "SkinFrictionX"
SkinFrictionY_s               = "SkinFrictionY"
SkinFrictionZ_s               = "SkinFrictionZ"
SkinFrictionMagnitude_s       = "SkinFrictionMagnitude"
VelocityAngleX_s              = "VelocityAngleX"
VelocityAngleY_s              = "VelocityAngleY"
VelocityAngleZ_s              = "VelocityAngleZ"
VelocityUnitVectorX_s         = "VelocityUnitVectorX"
VelocityUnitVectorY_s         = "VelocityUnitVectorY"
VelocityUnitVectorZ_s         = "VelocityUnitVectorZ"
MassFlow_s                    = "MassFlow"
ViscosityKinematic_s          = "ViscosityKinematic"
ViscosityMolecular_s          = "ViscosityMolecular"
ViscosityEddyDynamic_s        = "ViscosityEddyDynamic"
ViscosityEddy_s               = "ViscosityEddy"
ThermalConductivity_s         = "ThermalConductivity"
PowerLawExponent_s            = "PowerLawExponent"
SutherlandLawConstant_s       = "SutherlandLawConstant"
TemperatureReference_s        = "TemperatureReference"
ViscosityMolecularReference_s = "ViscosityMolecularReference"
ThermalConductivityReference_s = "ThermalConductivityReference"
IdealGasConstant_s            = "IdealGasConstant"
SpecificHeatPressure_s        = "SpecificHeatPressure"
SpecificHeatVolume_s          = "SpecificHeatVolume"
ReynoldsStressXX_s            = "ReynoldsStressXX"
ReynoldsStressXY_s            = "ReynoldsStressXY"
ReynoldsStressXZ_s            = "ReynoldsStressXZ"
ReynoldsStressYY_s            = "ReynoldsStressYY"
ReynoldsStressYZ_s            = "ReynoldsStressYZ"
ReynoldsStressZZ_s            = "ReynoldsStressZZ"
LengthReference_s             = "LengthReference"
MolecularWeight_s             = "MolecularWeight"
MolecularWeight_ps            = "MolecularWeight%s"
HeatOfFormation_s             = "HeatOfFormation"
HeatOfFormation_ps            = "HeatOfFormation%s"
FuelAirRatio_s                = "FuelAirRatio"
ReferenceTemperatureHOF_s     = "ReferenceTemperatureHOF"
```

```
MassFraction_s                = "MassFraction"
MassFraction_ps               = "MassFraction%s"
LaminarViscosity_s            = "LaminarViscosity"
LaminarViscosity_ps           = "LaminarViscosity%s"
ThermalConductivity_ps        = "ThermalConductivity%s"
EnthalpyEnergyRatio_s         = "EnthalpyEnergyRatio"
CompressibilityFactor_s       = "CompressibilityFactor"
VibrationalElectronEnergy_s   = "VibrationalElectronEnergy"
VibrationalElectronTemperature_s = "VibrationalElectronTemperature"
SpeciesDensity_s              = "SpeciesDensity"
SpeciesDensity_ps             = "SpeciesDensity%s"
MoleFraction_s                = "MoleFraction"
MoleFraction_ps               = "MoleFraction%s"
ElectricFieldX_s              = "ElectricFieldX"
ElectricFieldY_s              = "ElectricFieldY"
ElectricFieldZ_s              = "ElectricFieldZ"
MagneticFieldX_s              = "MagneticFieldX"
MagneticFieldY_s              = "MagneticFieldY"
MagneticFieldZ_s              = "MagneticFieldZ"
CurrentDensityX_s             = "CurrentDensityX"
CurrentDensityY_s             = "CurrentDensityY"
CurrentDensityZ_s             = "CurrentDensityZ"
LorentzForceX_s               = "LorentzForceX"
LorentzForceY_s               = "LorentzForceY"
LorentzForceZ_s               = "LorentzForceZ"
ElectricConductivity_s        = "ElectricConductivity"
JouleHeating_s                = "JouleHeating"
TurbulentDistance_s           = "TurbulentDistance"
TurbulentEnergyKinetic_s      = "TurbulentEnergyKinetic"
TurbulentDissipation_s        = "TurbulentDissipation"
TurbulentDissipationRate_s    = "TurbulentDissipationRate"
TurbulentBBReynolds_s         = "TurbulentBBReynolds"
TurbulentSANuTilde_s          = "TurbulentSANuTilde"
Mach_s                        = "Mach"
Mach_Velocity_s               = "Mach_Velocity"
Mach_VelocitySound_s          = "Mach_VelocitySound"
Reynolds_s                    = "Reynolds"
Reynolds_Velocity_s           = "Reynolds_Velocity"
Reynolds_Length_s             = "Reynolds_Length"
Reynolds_ViscosityKinematic_s = "Reynolds_ViscosityKinematic"
Prandtl_s                     = "Prandtl"
Prandtl_ThermalConductivity_s = "Prandtl_ThermalConductivity"
Prandtl_ViscosityMolecular_s  = "Prandtl_ViscosityMolecular"
Prandtl_SpecificHeatPressure_s = "Prandtl_SpecificHeatPressure"
PrandtlTurbulent_s            = "PrandtlTurbulent"
SpecificHeatRatio_s           = "SpecificHeatRatio"
SpecificHeatRatio_Pressure_s  = "SpecificHeatRatio_Pressure"
SpecificHeatRatio_Volume_s    = "SpecificHeatRatio_Volume"
CoefPressure_s                = "CoefPressure"
CoefSkinFrictionX_s           = "CoefSkinFrictionX"
CoefSkinFrictionY_s           = "CoefSkinFrictionY"
CoefSkinFrictionZ_s           = "CoefSkinFrictionZ"
Coef_PressureDynamic_s        = "Coef_PressureDynamic"
Coef_PressureReference_s      = "Coef_PressureReference"
Vorticity_s                   = "Vorticity"
Acoustic_s                    = "Acoustic"
RiemannInvariantPlus_s        = "RiemannInvariantPlus"
RiemannInvariantMinus_s       = "RiemannInvariantMinus"
CharacteristicEntropy_s       = "CharacteristicEntropy"
CharacteristicVorticity1_s    = "CharacteristicVorticity1"
CharacteristicVorticity2_s    = "CharacteristicVorticity2"
CharacteristicAcousticPlus_s  = "CharacteristicAcousticPlus"
CharacteristicAcousticMinus_s = "CharacteristicAcousticMinus"
```

```
ForceX_s                      = "ForceX"
ForceY_s                      = "ForceY"
ForceZ_s                      = "ForceZ"
ForceR_s                      = "ForceR"
ForceTheta_s                  = "ForceTheta"
ForcePhi_s                    = "ForcePhi"
Lift_s                        = "Lift"
Drag_s                        = "Drag"
MomentX_s                     = "MomentX"
MomentY_s                     = "MomentY"
MomentZ_s                     = "MomentZ"
MomentR_s                     = "MomentR"
MomentTheta_s                 = "MomentTheta"
MomentPhi_s                   = "MomentPhi"
MomentXi_s                    = "MomentXi"
MomentEta_s                   = "MomentEta"
MomentZeta_s                  = "MomentZeta"
Moment_CenterX_s              = "Moment_CenterX"
Moment_CenterY_s              = "Moment_CenterY"
Moment_CenterZ_s              = "Moment_CenterZ"
CoefLift_s                    = "CoefLift"
CoefDrag_s                    = "CoefDrag"
CoefMomentX_s                 = "CoefMomentX"
CoefMomentY_s                 = "CoefMomentY"
CoefMomentZ_s                 = "CoefMomentZ"
CoefMomentR_s                 = "CoefMomentR"
CoefMomentTheta_s             = "CoefMomentTheta"
CoefMomentPhi_s               = "CoefMomentPhi"
CoefMomentXi_s                = "CoefMomentXi"
CoefMomentEta_s               = "CoefMomentEta"
CoefMomentZeta_s              = "CoefMomentZeta"
Coef_PressureDynamic_s        = "Coef_PressureDynamic"
Coef_Area_s                   = "Coef_Area"
Coef_Length_s                 = "Coef_Length"
TimeValues_s                  = "TimeValues"
IterationValues_s             = "IterationValues"
NumberOfZones_s               = "NumberOfZones"
NumberOfFamilies_s            = "NumberOfFamilies"
DataConversion_s              ="DataConversion"

ZonePointers_s                = "ZonePointers"
FamilyPointers_s              = "FamilyPointers"
RigidGridMotionPointers_s     = "RigidGridMotionPointers"
ArbitraryGridMotionPointers_s = "ArbitraryGridMotionPointers"
GridCoordinatesPointers_s     = "GridCoordinatesPointers"
FlowSolutionsPointers_s       = "FlowSolutionsPointers"
PointerNames_l = [ZonePointers_s,FamilyPointers_s,RigidGridMotionPointers_s,
                  ArbitraryGridMotionPointers_s,GridCoordinatesPointers_s,
                  FlowSolutionsPointers_s]

OriginLocation_s              = "OriginLocation"
RigidRotationAngle_s          = "RigidRotationAngle"
Translation_s                 = "Translation"
RotationAngle_s               = "RotationAngle"
RigidVelocity_s               = "RigidVelocity"
RigidRotationRate_s           = "RigidRotationRate"
GridVelocityX_s               = "GridVelocityX"
GridVelocityY_s               = "GridVelocityY"
GridVelocityZ_s               = "GridVelocityZ"
GridVelocityR_s               = "GridVelocityR"
GridVelocityTheta_s           = "GridVelocityTheta"
GridVelocityPhi_s             = "GridVelocityPhi"
GridVelocityXi_s              = "GridVelocityXi"
```

```
GridVelocityEta_s             = "GridVelocityEta"
GridVelocityZeta_s            = "GridVelocityZeta"

ArbitraryGridMotion_ts        = "ArbitraryGridMotion_t"
ArbitraryGridMotion_s         = "ArbitraryGridMotion"
ArbitraryGridMotionType_l     = [Null_s,NonDeformingGrid_s,
                                 DeformingGrid_s,UserDefined_s]

ArbitraryGridMotionType_s     ="ArbitraryGridMotionType"
ArbitraryGridMotionType_ts    ="ArbitraryGridMotionType_t"

Area_ts                       = "Area_t"
Area_s                        = "Area"
AreaType_ts                   = "AreaType_t"
AreaType_s                    = "AreaType"
SurfaceArea_s                 = "SurfaceArea"
RegionName_s                  = "RegionName"
AverageInterface_ts           = "AverageInterface_t"
Axisymmetry_ts                = "Axisymmetry_t"
Axisymmetry_s                 = "Axisymmetry"
AxisymmetryReferencePoint_s   = "AxisymmetryReferencePoint"
AxisymmetryAxisVector_s       = "AxisymmetryAxisVector"
AxisymmetryAngle_s            = "AxisymmetryAngle"
BCDataSet_ts                  = "BCDataSet_t"
BCData_ts                     = "BCData_t"
BCData_s                      = "BCData"

BCProperty_ts                 = "BCProperty_t"
BCProperty_s                  = "BCProperty"
BC_ts                         = "BC_t"

BaseIterativeData_ts          = "BaseIterativeData_t"
BaseIterativeData_s           = "BaseIterativeData"

CGNSBase_ts                   = "CGNSBase_t"
CGNSLibraryVersion_ts         = "CGNSLibraryVersion_t"

# ----------------------------------------------
ConvergenceHistory_ts         = "ConvergenceHistory_t"
ZoneConvergenceHistory_s      = "ZoneConvergenceHistory"
GlobalConvergenceHistory_s    = "GlobalConvergenceHistory"

ConvergenceHistory_l          = [ZoneConvergenceHistory_s,
                                 GlobalConvergenceHistory_s]

NormDefinitions_s             ="NormDefinitions"

DataArray_ts                  = "DataArray_t"
DataConversion_ts             = "DataConversion_t"
Descriptor_ts                 = "Descriptor_t"

# ----------------------------------------------
DimensionalExponents_ts       = "DimensionalExponents_t"
DimensionalExponents_s        = "DimensionalExponents"
DimensionalUnits_ts           = "DimensionalUnits_t"
AdditionalUnits_ts            = "AdditionalUnits_t"
AdditionalExponents_ts        = "AdditionalExponents_t"

DiscreteData_ts               = "DiscreteData_t"
DiscreteData_s                = "DiscreteData"

FamilyBC_s                    = "FamilyBC"
FamilyBC_ts                   = "FamilyBC_t"
```

```
FamilyName_ts                = "FamilyName_t"
FamilyName_s                 = "FamilyName"
AdditionalFamilyName_ts      = "AdditionalFamilyName_t"
AdditionalFamilyName_s       = "AdditionalFamilyName"
Family_ts                    = "Family_t"
Family_s                     = "Family"
FlowEquationSet_ts           = "FlowEquationSet_t"
FlowEquationSet_s            = "FlowEquationSet"
FlowSolution_ts              = "FlowSolution_t"
GasModel_ts                  = "GasModel_t"
GasModel_s                   = "GasModel"

GeometryEntity_ts            = "GeometryEntity_t"
GeometryFile_ts              = "GeometryFile_t"
GeometryFile_s               = "GeometryFile"

GeometryFormat_s             = "GeometryFormat"
GeometryFormat_ts            = "GeometryFormat_t"

# not supported '-'
NASAIGES_s                   ="NASA-IGES"
ICEMCFD_s                    ="ICEM-CFD"

SDRC_s                       ="SDRC"
Unigraphics_s                ="Unigraphics"
ProEngineer_s                ="ProEngineer"
GeometryFormat_l             =[Null_s,NASAIGES_s,SDRC_s,Unigraphics_s,
                               ProEngineer_s,ICEMCFD_s,UserDefined_s]
GeometryReference_ts         = "GeometryReference_t"
GeometryReference_s          = "GeometryReference"

Gravity_ts                   = "Gravity_t"
Gravity_s                    = "Gravity"
GravityVector_s              = "GravityVector"

GridConnectivity1to1_ts      = "GridConnectivity1to1_t"
GridConnectivityProperty_ts  = "GridConnectivityProperty_t"
GridConnectivityProperty_s   = "GridConnectivityProperty"

GridCoordinates_ts           = "GridCoordinates_t"
IndexArray_ts                = "IndexArray_t"
IndexRange_ts                = "IndexRange_t"
IntegralData_ts              = "IntegralData_t"
InwardNormalList_ts          = "InwardNormalList_t"
InwardNormalList_s           = "InwardNormalList"
InwardNormalIndex_s          = "InwardNormalIndex"
Ordinal_ts                   = "Ordinal_t"
Ordinal_s                    = "Ordinal"
Transform_s                  = "Transform"
OversetHoles_ts              = "OversetHoles_t"
OversetHoles_s               = "OversetHoles"
Periodic_ts                  = "Periodic_t"
Periodic_s                   = "Periodic"

ReferenceState_ts            = "ReferenceState_t"
ReferenceState_s             = "ReferenceState"
ReferenceStateDescription_s  = "ReferenceStateDescription"

RigidGridMotion_ts           = "RigidGridMotion_t"
RigidGridMotion_s            = "RigidGridMotion"

Rind_s                       = "Rind"
```

```
Rind_ts                       = "Rind_t"

RotatingCoordinates_s         = "RotatingCoordinates"
RotatingCoordinates_ts        = "RotatingCoordinates_t"
RotationRateVector_s          = "RotationRateVector"
RotationCenter_s              = "RotationCenter"

GoverningEquations_s          = "GoverningEquations"
GoverningEquations_ts         = "GoverningEquations_t"
GoverningEquationsType_l      = [Euler_s,NSLaminar_s,NSTurbulent_s]
GoverningEquationsType_s      = "GoverningEquationsType"
GoverningEquationsType_ts     = "GoverningEquationsType_t"

# ----------------------------------------------
BCType_s                    = "BCType"
BCType_ts                   = "BCType_t"
BCTypeSimple_s              = "BCTypeSimple"
BCTypeSimple_ts             = "BCTypeSimple_t"

BCAxisymmetricWedge_s       = "BCAxisymmetricWedge"
BCDegenerateLine_s          = "BCDegenerateLine"
BCDegeneratePoint_s         = "BCDegeneratePoint"
BCDirichlet_s               = "BCDirichlet"
BCExtrapolate_s             = "BCExtrapolate"
BCFarfield_s                = "BCFarfield"
BCGeneral_s                 = "BCGeneral"
BCInflow_s                  = "BCInflow"
BCInflowSubsonic_s          = "BCInflowSubsonic"
BCInflowSupersonic_s        = "BCInflowSupersonic"
BCNeumann_s                 = "BCNeumann"
BCOutflow_s                 = "BCOutflow"
BCOutflowSubsonic_s         = "BCOutflowSubsonic"
BCOutflowSupersonic_s       = "BCOutflowSupersonic"
BCSymmetryPlane_s           = "BCSymmetryPlane"
BCSymmetryPolar_s           = "BCSymmetryPolar"
BCTunnelInflow_s            = "BCTunnelInflow"
BCTunnelOutflow_s           = "BCTunnelOutflow"
BCWall_s                    = "BCWall"
BCWallInviscid_s            = "BCWallInviscid"
BCWallViscous_s             = "BCWallViscous"
BCWallViscousHeatFlux_s     = "BCWallViscousHeatFlux"
BCWallViscousIsothermal_s   = "BCWallViscousIsothermal"

BCType_l = [Null_s,UserDefined_s,
            BCAxisymmetricWedge_s,BCDegenerateLine_s,BCDegeneratePoint_s,
            BCDirichlet_s,BCExtrapolate_s,BCFarfield_s,
            BCGeneral_s,BCInflow_s,BCInflowSubsonic_s,BCInflowSupersonic_s,
            BCNeumann_s,BCOutflow_s,BCOutflowSubsonic_s,BCOutflowSupersonic_s,
            BCSymmetryPlane_s,BCSymmetryPolar_s,
            BCTunnelInflow_s,BCTunnelOutflow_s,
            BCWall_s,BCWallInviscid_s,BCWallViscous_s,
            BCWallViscousHeatFlux_s,BCWallViscousIsothermal_s,
            FamilySpecified_s]
BCType    = stringAsKeyDict(BCType_l)
BCType_   = enumAsKeyDict(BCType_l)
(Null,UserDefined,
 BCAxisymmetricWedge,BCDegenerateLine,BCDegeneratePoint,
 BCDirichlet,BCExtrapolate,BCFarfield,
 BCGeneral,BCInflow,BCInflowSubsonic,BCInflowSupersonic,
 BCNeumann,BCOutflow,BCOutflowSubsonic,BCOutflowSupersonic,
 BCSymmetryPlane,BCSymmetryPolar,
 BCTunnelInflow,BCTunnelOutflow,
 BCWall,BCWallInviscid,BCWallViscous,
```

```
 BCWallViscousHeatFlux,BCWallViscousIsothermal,
 FamilySpecified)=BCType_.keys()

FamilyBC_l = BCType_l
FamilyBC   = BCType
FamilyBC_  = BCType_

# CAUTION, index of values in the lists below cannot be used as enumerate,
# the lists are subset of the global list and some index are missing.
BCTypeSimple_l   =[Null_s,BCGeneral_s,BCDirichlet_s,BCNeumann_s,
                   BCExtrapolate_s,BCWallInviscid_s,BCWallViscousHeatFlux_s,
                   BCWallViscousIsothermal_s,BCWallViscous_s,BCWall_s,
                   BCInflowSubsonic_s,BCInflowSupersonic_s,BCOutflowSubsonic_s,
                   BCOutflowSupersonic_s,BCTunnelInflow_s,BCTunnelOutflow_s,
                   BCDegenerateLine_s,BCDegeneratePoint_s,BCSymmetryPlane_s,
                   BCSymmetryPolar_s,BCAxisymmetricWedge_s,FamilySpecified_s,
                   UserDefined_s]
BCTypeCompound_l = [BCInflow_s,BCOutflow_s,BCFarfield_s,
                    Null_s,UserDefined_s]

# ------------------------------------------------
ThermalConductivityModel_ts          = "ThermalConductivityModel_t"
ThermalConductivityModel_s           = "ThermalConductivityModel"
ThermalConductivityModelType_l       = [Null_s,ConstantPrandtl_s,PowerLaw_s,
                                        SutherlandLaw_s,UserDefined_s]
ThermalConductivityModelType_s       = "ThermalConductivityModelType"
ThermalConductivityModelType_ts      = "ThermalConductivityModelType_t"
ThermalConductivityModelIdentifier_l = [(Prandtl_s),(PowerLawExponent_s),
                                        (SutherlandLawConstant_s),
                                        (TemperatureReference_s),
                                        (ThermalConductivityReference_s)]

TurbulenceClosure_ts          = "TurbulenceClosure_t"
TurbulenceClosure_s           = "TurbulenceClosure"
TurbulenceClosureType_l       = [Null_s,EddyViscosity_s,ReynoldsStress_s,
                                 ReynoldsStressAlgebraic_s,UserDefined_s]
TurbulenceClosureType_s       = "TurbulenceClosureType"
TurbulenceClosureType_ts      = "TurbulenceClosureType_t"
TurbulenceClosureIdentifier_l = [PrandtlTurbulent_s]

TurbulenceModel_ts     = "TurbulenceModel_t"
TurbulenceModel_s      = "TurbulenceModel"
TurbulenceModelType_l  = [Null_s,Algebraic_BaldwinLomax_s,
                          Algebraic_CebeciSmith_s,
                          HalfEquation_JohnsonKing_s,
                          OneEquation_BaldwinBarth_s,
                          OneEquation_SpalartAllmaras_s,
                          TwoEquation_JonesLaunder_s,
                          TwoEquation_MenterSST_s,TwoEquation_Wilcox_s]
TurbulenceModelType_s  = "TurbulenceModelType"
TurbulenceModelType_ts = "TurbulenceModelType_t"

DiffusionModel_s    = 'DiffusionModel'
EquationDimension_s = 'EquationDimension'

ViscosityModel_ts          = "ViscosityModel_t"
ViscosityModel_s           = "ViscosityModel"
ViscosityModelType_l       = [Constant_s,PowerLaw_s,SutherlandLaw_s,
                              Null_s,UserDefined_s]
ViscosityModelType_s       = "ViscosityModelType"
ViscosityModelType_ts      = "ViscosityModelType_t"
ViscosityModelIdentifier_l = [(PowerLawExponent_s),(SutherlandLawConstant_s),
                              (TemperatureReference_s),
```

```
                              (ViscosityMolecularReference_s)]

GasModelType_l       = [Null_s,Ideal_s,VanderWaals_s,CaloricallyPerfect_s,
                        ThermallyPerfect_s,ConstantDensity_s,RedlichKwong_s,
                        UserDefined_s]
GasModelType_s       = "GasModelType"
GasModelType_ts      = "GasModelType_t"
GasModelIdentifier_l = [IdealGasConstant_s,SpecificHeatRatio_s,
                        SpecificHeatVolume_s,SpecificHeatPressure_s]

ThermalRelaxationModel_ts     = "ThermalRelaxationModel_t"
ThermalRelaxationModel_s      = "ThermalRelaxationModel"
ThermalRelaxationModelType_l  = [Null_s,Frozen_s,ThermalEquilib_s,
                                 ThermalNonequilib_s,UserDefined_s]
ThermalRelaxationModelType_s  = "ThermalRelaxationModelType"
ThermalRelaxationModelType_ts = "ThermalRelaxationModelType_t"

ChemicalKineticsModel_ts          = "ChemicalKineticsModel_t"
ChemicalKineticsModel_s           = "ChemicalKineticsModel"
ChemicalKineticsModelType_l       = [Null_s,Frozen_s,ChemicalEquilibCurveFit_s,
                                     ChemicalEquilibMinimization_s,
                                     ChemicalNonequilib_s,
                                     UserDefined_s]
ChemicalKineticsModelType_s       = "ChemicalKineticsModelType"
ChemicalKineticsModelType_ts      = "ChemicalKineticsModelType_t"
ChemicalKineticsModelIdentifier_l = [FuelAirRatio_s,ReferenceTemperatureHOF_s]

EMElectricFieldModel_s      = "EMElectricFieldModel"
EMElectricFieldModel_ts     = "EMElectricFieldModel_t"
EMElectricFieldModelType_l  = [Null_s,Constant_s,Frozen_s,
                               Interpolated_s,Voltage_s,UserDefined_s]
EMElectricFieldModelType_s  = "EMElectricFieldModelType"
EMElectricFieldModelType_ts = "EMElectricFieldModelType_t"

EMMagneticFieldModel_s      = "EMMagneticFieldModel"
EMMagneticFieldModel_ts     = "EMMagneticFieldModel_t"
EMMagneticFieldModelType_l  = [Null_s,Constant_s,Frozen_s,
                               Interpolated_s,UserDefined_s]
EMMagneticFieldModelType_s  = "EMMagneticFieldModelType"
EMMagneticFieldModelType_ts = "EMMagneticFieldModelType_t"

EMConductivityModel_s           = "EMConductivityModel"
EMConductivityModel_ts          = "EMConductivityModel_t"
EMConductivityModelType_l       = [Null_s,Constant_s,Frozen_s,
                                   Equilibrium_LinRessler_s,
                                   Chemistry_LinRessler_s,UserDefined_s]
EMConductivityModelType_s       = "EMConductivityModelType"
EMConductivityModelType_ts      = "EMConductivityModelType_t"
EMConductivityModelIdentifier_l = [ElectricFieldX_s,ElectricFieldY_s,
                                   ElectricFieldZ_s,MagneticFieldX_s,
                                   MagneticFieldY_s,MagneticFieldZ_s,
                                   CurrentDensityX_s,CurrentDensityY_s,
                                   CurrentDensityZ_s,ElectricConductivity_s,
                                   LorentzForceX_s,LorentzForceY_s,
                                   LorentzForceZ_s,JouleHeating_s]

AverageInterfaceType_s  = "AverageInterfaceType"
AverageInterfaceType_ts = "AverageInterfaceType_t"
AverageInterfaceType_l  = [Null_s,AverageAll_s,AverageCircumferential_s,
                           AverageRadial_s,AverageI_s,AverageJ_s,AverageK_s,
                           UserDefined_s]
AverageInterface_s      = "AverageInterface"
AverageInterface_ts     = "AverageInterface_t"
```

```
NODE_s      = "NODE"
BAR_2_s     = "BAR_2"
BAR_3_s     = "BAR_3"
TRI_3_s     = "TRI_3"
TRI_6_s     = "TRI_6"
QUAD_4_s    = "QUAD_4"
QUAD_8_s    = "QUAD_8"
QUAD_9_s    = "QUAD_9"
TETRA_4_s   = "TETRA_4"
TETRA_10_s  = "TETRA_10"
PYRA_5_s    = "PYRA_5"
PYRA_13_s   = "PYRA_13"
PYRA_14_s   = "PYRA_14"
PENTA_6_s   = "PENTA_6"
PENTA_15_s  = "PENTA_15"
PENTA_18_s  = "PENTA_18"
HEXA_8_s    = "HEXA_8"
HEXA_20_s   = "HEXA_20"
HEXA_27_s   = "HEXA_27"
MIXED_s     = "MIXED"
NGON_n_s    = "NGON_n"
NFACE_n_s   = "NFACE_n"

Null_npe        = 0
UserDefined_npe = 0

NODE_npe      = 1
BAR_2_npe     = 2
BAR_3_npe     = 3
TRI_3_npe     = 3
TRI_6_npe     = 6
QUAD_4_npe    = 4
QUAD_8_npe    = 8
QUAD_9_npe    = 9
TETRA_4_npe   = 4
TETRA_10_npe  = 10
PYRA_5_npe    = 5
PYRA_13_npe   = 13
PYRA_14_npe   = 14
PENTA_6_npe   = 6
PENTA_15_npe  = 15
PENTA_18_npe  = 18
HEXA_8_npe    = 8
HEXA_20_npe   = 20
HEXA_27_npe   = 27
MIXED_npe     = 0
NGON_n_npe    = 0
NFACE_n_npe   = 0

Elements_ts     = "Elements_t"
ElementType_ts  = "ElementType_t"
ElementType_s   = "ElementType"
Elements_s      = "Elements"
ElementType_l   = [Null_s, UserDefined_s, NODE_s, BAR_2_s, BAR_3_s,
                   TRI_3_s, TRI_6_s, QUAD_4_s, QUAD_8_s, QUAD_9_s,
                   TETRA_4_s, TETRA_10_s, PYRA_5_s, PYRA_14_s,
                   PENTA_6_s, PENTA_15_s, PENTA_18_s,
                   HEXA_8_s, HEXA_20_s, HEXA_27_s, MIXED_s, PYRA_13_s,
                   NGON_n_s, NFACE_n_s]
ElementTypeNPE_l = [Null_npe, UserDefined_npe, NODE_npe, BAR_2_npe, BAR_3_npe,
                    TRI_3_npe, TRI_6_npe, QUAD_4_npe, QUAD_8_npe, QUAD_9_npe,
                    TETRA_4_npe, TETRA_10_npe, PYRA_5_npe, PYRA_14_npe,
```

```
                  PENTA_6_npe, PENTA_15_npe, PENTA_18_npe,
                  HEXA_8_npe, HEXA_20_npe, HEXA_27_npe, MIXED_npe,
                  PYRA_13_npe, NGON_n_npe, NFACE_n_npe]
ElementType     = stringAsKeyDict(ElementType_l)
ElementType_    = enumAsKeyDict(ElementType_l)
ElementTypeNPE = dict(zip(ElementType_l,ElementTypeNPE_l))
(Null, UserDefined, NODE, BAR_2, BAR_3,
 TRI_3, TRI_6, QUAD_4, QUAD_8, QUAD_9,
 TETRA_4, TETRA_10, PYRA_5, PYRA_14,
 PENTA_6, PENTA_15, PENTA_18,
 HEXA_8, HEXA_20, HEXA_27, MIXED, PYRA_13,
 NGON_n, NFACE_n)=ElementType_.keys()
#

WallFunction_ts               = "WallFunction_t"
WallFunction_s                = "WallFunction"
WallFunctionType_ts           = "WallFunctionType_t"
WallFunctionType_s            = "WallFunctionType"
ZoneBC_ts                     = "ZoneBC_t"
ZoneBC_s                      = "ZoneBC"
ZoneIterativeData_ts          = "ZoneIterativeData_t"
ZoneIterativeData_s           = "ZoneIterativeData"

UserDefinedData_ts            = "UserDefinedData_t"

# ---
cgnsnames=[globals()[k] for k in dir() if (k[-2:]=='_s')]
cgnstypes=[globals()[k] for k in dir() if (k[-3:]=='_ts')]
cgnsenums={}
for k in dir():
  if (k[-2:]=='_l'): cgnsenums[k[:-1]+'t']=locals()[k]
#
cgnsnames.sort()
cgnstypes.sort()
#
# --- last line
```

CHAPTER

FIVE

CGNS TYPES

5.1 “int”

- Name:
  - EquationDimension
- Data-type: I4
- Cardinality: Zero/One
- Children
- Parents

5.2 “int[1+...+IndexDimension]”

- Name:
  - DiffusionModel
- Data-type: I4
- Cardinality: Zero/One
- Children
- Parents

5.3 “int[IndexDimension]”

- Name:
  - InwardNormalIndex
- Data-type: I4
- Cardinality: Zero/One
- Children
- Parents

5.4 AdditionalExponents_t

- Name:
  - AdditionalExponents
- Data-type: R4 R8
- Cardinality: Zero/One
- Children
- Parents

5.5 AdditionalUnits_t

- Name:
  - AdditionalUnits
- Data-type: C1
- Enumerate:
- Cardinality: Zero/One
- Children
- Parents
  - *DimensionalUnits_t*

5.6 ArbitraryGridMotionType_t

- Name:
  - ArbitraryGridMotionType
- Data-type: C1
- Cardinality: One/One
- Children
- Parents

5.7 ArbitraryGridMotion_t

- Name:
  - {UserDefined}
- Data-type: C1
- Enumerate:
- Cardinality: Zero/N
- Children

  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *Descriptor_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
  - *GridLocation_t* (GridLocation)
  - *Rind_t* (Rind)
  - *DataArray_t* ({UserDefined})
- Parents
  - *Zone_t*

5.8 AreaType_t

- Name:
  - AreaType
- Data-type: C1
- Cardinality: One/One
- Children
- Parents
  - *Area_t*

5.9 Area_t

- Name:
  - Area
- Data-type: MT
- Cardinality: Zero/One
- Children
  - *Descriptor_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
  - *AreaType_t* (AreaType)
  - *DataArray_t* (SurfaceArea)
  - *DataArray_t* (RegionName)
- Parents
  - *BCProperty_t*

5.10 AverageInterfaceType_t

- Name:
  - AverageInterfaceType
- Data-type: C1
- Cardinality: One/One
- Children
- Parents
  - *AverageInterface_t*

5.11 AverageInterface_t

- Name:
  - AverageInterface
- Data-type: MT
- Cardinality: Zero/One
- Children
  - *Descriptor_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
  - *AverageInterfaceType_t* (AverageInterfaceType)
- Parents
  - *GridConnectivityProperty_t*

5.12 Axisymmetry_t

- Name:
  - Axisymmetry
- Data-type: MT
- Cardinality: Zero/One
- Children
  - *DataArray_t* (AxisymmetryReferencePoint)
  - *DataArray_t* (AxisymmetryAxisVector)
  - *DataArray_t* (AxisymmetryAngle)
  - *DataArray_t* (CoordinateNames)
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *Descriptor_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})

- Parents
  - *CGNSBase_t*

5.13 BCDataSet_t

- Name:
  - {UserDefined}
- Data-type: C1
- Enumerate:
- Cardinality: Zero/N
- Children
  - *BCData_t* (NeumannData)
  - *BCData_t* (DirichletData)
  - *GridLocation_t* (GridLocation)
  - *IndexRange_t* (PointRange)
  - *IndexArray_t* (PointList)
  - *Descriptor_t* ({UserDefined})
  - *ReferenceState_t* (ReferenceState)
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *BC_t*
  - *FamilyBC_t*

5.14 BCData_t

- Name:
  - DirichletData
  - NeumannData
- Data-type: MT
- Cardinality: Zero/One
- Children
  - *DataArray_t* ({UserDefined})
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *Descriptor_t* ({UserDefined})

  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *BCDataSet_t*

5.15 BCProperty_t

- Name:
  - BCProperty
- Data-type: MT
- Cardinality: Zero/One
- Children
  - *Descriptor_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
  - *WallFunction_t* (WallFunction)
  - *Area_t* (Area)
- Parents
  - *BC_t*

5.16 BC_t

- Name:
  - {UserDefined}
- Data-type: C1
- Enumerate:
- Cardinality: Zero/N
- Children
  - *ReferenceState_t* (ReferenceState)
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *Descriptor_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
  - *Ordinal_t* (Ordinal)
  - *FamilyName_t* (FamilyName)
  - *IndexArray_t* (InwardNormalList)
  - *BCDataSet_t* ({UserDefined})
  - *InwardNormalIndex_t* (InwardNormalIndex)
  - *IndexArray_t* (ElementList)

  - *IndexArray_t* (PointList)
  - *IndexRange_t* (ElementRange)
  - *IndexRange_t* (PointRange)
  - *GridLocation_t* (GridLocation)
  - *BCProperty_t* (BCProperty)
- Parents
  - *ZoneBC_t*

5.17 BaseIterativeData_t

- Name:
  - {UserDefined}
- Data-type: I4
- Cardinality: Zero/One
- Children
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *Descriptor_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
  - *DataArray_t* ({UserDefined})
- Parents
  - *CGNSBase_t*

5.18 CGNSBase_t

- Name:
  - {UserDefined}
- Data-type: I4
- Cardinality: Zero/N
- Children
  - *Zone_t* ({UserDefined})
  - *SimulationType_t* (SimulationType)
  - *BaseIterativeData_t* ({UserDefined})
  - *IntegralData_t* ({UserDefined})
  - *ConvergenceHistory_t* (GlobalConvergenceHistory)
  - *Family_t* ({UserDefined})
  - *FlowEquationSet_t* (FlowEquationSet)

  - *ReferenceState_t* (ReferenceState)
  - *Axisymmetry_t* (Axisymmetry)
  - *RotatingCoordinates_t* (RotatingCoordinates)
  - *Gravity_t* (Gravity)
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *Descriptor_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *CGNSTree_t*

5.19 CGNSLibraryVersion_t

- Name:
  - CGNSLibraryVersion
- Data-type: R4
- Cardinality: One/One
- Children
- Parents
  - *CGNSTree_t*

5.20 CGNSTree_t

- Name:
  - CGNSTree
  - {UserDefined}
- Data-type: MT
- Cardinality: One/One
- Children
  - *CGNSLibraryVersion_t* (CGNSLibraryVersion)
  - *CGNSBase_t* ({UserDefined})
- Parents

5.21 ChemicalKineticsModel_t

- Name:
  - ChemicalKineticsModel
- Data-type: C1
- Enumerate:
- Cardinality: Zero/One
- Children
  - *Descriptor_t* ({UserDefined})
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *DataArray_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *FlowEquationSet_t*

5.22 ConvergenceHistory_t

- Name:
  - GlobalConvergenceHistory
  - ZoneConvergenceHistory
- Data-type: I4
- Cardinality: Zero/One
- Children
  - *Descriptor_t* ({UserDefined})
  - *Descriptor_t* (NormDefinitions)
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *DataArray_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *CGNSBase_t*
  - *Zone_t*

5.23 DataArray_t

- Name:
  - {UserDefined}
- Data-type: C1 MT I4 I8 R4 R8
- Cardinality: Zero/N
- Children
  - *DimensionalExponents_t* (DimensionalExponents)
  - *DataConversion_t* (DataConversion)
  - *DataClass_t* (DataClass)
  - *Descriptor_t* ({UserDefined})
  - *DimensionalUnits_t* (DimensionalUnits)
- Parents
  - *ArbitraryGridMotion_t*
  - *Area_t*
  - *Axisymmetry_t*
  - *BCData_t*
  - *BaseIterativeData_t*
  - *ChemicalKineticsModel_t*
  - *ConvergenceHistory_t*
  - *DiscreteData_t*
  - *EMConductivityModel_t*
  - *EMElectricFieldModel_t*
  - *EMMagneticFieldModel_t*
  - *Elements_t*
  - *FlowSolution_t*
  - *GasModel_t*
  - *Gravity_t*
  - *GridConnectivity_t*
  - *GridCoordinates_t*
  - *IntegralData_t*
  - *Periodic_t*
  - *ReferenceState_t*
  - *RigidGridMotion_t*
  - *RotatingCoordinates_t*
  - *ThermalConductivityModel_t*
  - *ThermalRelaxationModel_t*
  - *TurbulenceClosure_t*
  - *TurbulenceModel_t*

  - *UserDefinedData_t*
  - *ViscosityModel_t*
  - *ZoneIterativeData_t*

5.24 DataClass_t

- Name:
  - DataClass
- Data-type: C1
- Enumerate:
- Cardinality: Zero/One
- Children
- Parents
  - *ArbitraryGridMotion_t*
  - *Axisymmetry_t*
  - *BCDataSet_t*
  - *BCData_t*
  - *BC_t*
  - *BaseIterativeData_t*
  - *CGNSBase_t*
  - *ChemicalKineticsModel_t*
  - *ConvergenceHistory_t*
  - *DataArray_t*
  - *DiscreteData_t*
  - *EMConductivityModel_t*
  - *EMElectricFieldModel_t*
  - *EMMagneticFieldModel_t*
  - *FlowEquationSet_t*
  - *FlowSolution_t*
  - *GasModel_t*
  - *Gravity_t*
  - *GridCoordinates_t*
  - *IntegralData_t*
  - *Periodic_t*
  - *ReferenceState_t*
  - *RigidGridMotion_t*
  - *RotatingCoordinates_t*
  - *ThermalConductivityModel_t*

  - *ThermalRelaxationModel_t*
  - *TurbulenceClosure_t*
  - *TurbulenceModel_t*
  - *UserDefinedData_t*
  - *ViscosityModel_t*
  - *ZoneBC_t*
  - *ZoneIterativeData_t*
  - *Zone_t*

5.25 DataConversion_t

- Name:
  - DataConversion
- Data-type: R4 R8
- Cardinality: Zero/One
- Children
- Parents
  - *DataArray_t*

5.26 Descriptor_t

- Name:
  - {UserDefined}
- Data-type: C1
- Cardinality: Zero/N
- Children
- Parents
  - *ArbitraryGridMotion_t*
  - *Area_t*
  - *AverageInterface_t*
  - *Axisymmetry_t*
  - *BCDataSet_t*
  - *BCData_t*
  - *BCProperty_t*
  - *BC_t*
  - *BaseIterativeData_t*
  - *CGNSBase_t*

- *ChemicalKineticsModel_t*
- *ConvergenceHistory_t*
- *DataArray_t*
- *DiscreteData_t*
- *EMConductivityModel_t*
- *EMElectricFieldModel_t*
- *EMMagneticFieldModel_t*
- *Elements_t*
- *Family_t*
- *FlowEquationSet_t*
- *FlowSolution_t*
- *GasModel_t*
- *GeometryReference_t*
- *GoverningEquations_t*
- *Gravity_t*
- *GridConnectivity1to1_t*
- *GridConnectivityProperty_t*
- *GridConnectivity_t*
- *GridCoordinates_t*
- *IntegralData_t*
- *OversetHoles_t*
- *Periodic_t*
- *ReferenceState_t*
- *RigidGridMotion_t*
- *RotatingCoordinates_t*
- *ThermalConductivityModel_t*
- *ThermalRelaxationModel_t*
- *TurbulenceClosure_t*
- *TurbulenceModel_t*
- *UserDefinedData_t*
- *ViscosityModel_t*
- *WallFunction_t*
- *ZoneBC_t*
- *ZoneGridConnectivity_t*
- *ZoneIterativeData_t*
- *Zone_t*

5.27 DiffusionModel_t

- Name:
  - DiffusionModel
- Data-type: I4
- Cardinality: Zero/One
- Children
- Parents
  - *GoverningEquations_t*
  - *TurbulenceModel_t*

5.28 DimensionalExponents_t

- Name:
  - DimensionalExponents
- Data-type: R4 R8
- Cardinality: Zero/One
- Children
- Parents
  - *DataArray_t*

5.29 DimensionalUnits_t

- Name:
  - DimensionalUnits
- Data-type: C1
- Enumerate:
- Cardinality: Zero/One
- Children
  - *AdditionalUnits_t* (AdditionalUnits)
- Parents
  - *ArbitraryGridMotion_t*
  - *Axisymmetry_t*
  - *BCDataSet_t*
  - *BCData_t*
  - *BC_t*
  - *BaseIterativeData_t*
  - *CGNSBase_t*

  - *ChemicalKineticsModel_t*
  - *ConvergenceHistory_t*
  - *DataArray_t*
  - *DiscreteData_t*
  - *EMConductivityModel_t*
  - *EMElectricFieldModel_t*
  - *EMMagneticFieldModel_t*
  - *FlowEquationSet_t*
  - *FlowSolution_t*
  - *GasModel_t*
  - *Gravity_t*
  - *GridCoordinates_t*
  - *IntegralData_t*
  - *Periodic_t*
  - *ReferenceState_t*
  - *RigidGridMotion_t*
  - *RotatingCoordinates_t*
  - *ThermalConductivityModel_t*
  - *ThermalRelaxationModel_t*
  - *TurbulenceClosure_t*
  - *TurbulenceModel_t*
  - *UserDefinedData_t*
  - *ViscosityModel_t*
  - *ZoneBC_t*
  - *ZoneIterativeData_t*
  - *Zone_t*

5.30 DiscreteData_t

- Name:
  - {UserDefined}
- Data-type: MT
- Cardinality: Zero/N
- Children
  - *GridLocation_t* (GridLocation)
  - *DataArray_t* ({UserDefined})
  - *Rind_t* (Rind)
  - *DataClass_t* (DataClass)

  - *DimensionalUnits_t* (DimensionalUnits)
  - *Descriptor_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *Zone_t*

5.31 EMConductivityModel_t

- Name:
  - EMConductivityModel
- Data-type: C1
- Enumerate:
- Cardinality: Zero/One
- Children
  - *Descriptor_t* ({UserDefined})
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *DataArray_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *FlowEquationSet_t*

5.32 EMElectricFieldModel_t

- Name:
  - EMElectricFieldModel
- Data-type: C1
- Enumerate:
- Cardinality: Zero/One
- Children
  - *Descriptor_t* ({UserDefined})
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *DataArray_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *FlowEquationSet_t*

5.33 EMMagneticFieldModel_t

- Name:
  - EMMagneticFieldModel
- Data-type: C1
- Enumerate:
- Cardinality: Zero/One
- Children
  - *Descriptor_t* ({UserDefined})
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *DataArray_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *FlowEquationSet_t*

5.34 Elements_t

- Name:
  - {UserDefined}
- Data-type: I4
- Cardinality: Zero/N
- Children
  - *IndexRange_t* (ElementRange)
  - *DataArray_t* (ElementConnectivity)
  - *DataArray_t* (ParentData)
  - *Rind_t* (Rind)
  - *Descriptor_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *Zone_t*

5.35 EquationDimension_t

- Name:
  - EquationDimension
- Data-type: I4
- Cardinality: Zero/One

- Children
- Parents
  - *FlowEquationSet_t*

5.36 FamilyBC_t

- Name:
  - FamilyBC
- Data-type: C1
- Enumerate:
- Cardinality: Zero/One
- Children
  - *BCDataSet_t* ({UserDefined})
- Parents
  - *Family_t*

5.37 FamilyName_t

- Name:
  - FamilyName
- Data-type: C1
- Cardinality: Zero/One
- Children
- Parents
  - *BC_t*
  - *UserDefinedData_t*
  - *Zone_t*

5.38 Family_t

- Name:
  - {UserDefined}
- Data-type: MT
- Cardinality: Zero/N
- Children
  - *Descriptor_t* ({UserDefined})

  - *Ordinal_t* (Ordinal)
  - *FamilyBC_t* ({UserDefined})
  - *GeometryReference_t* ({UserDefined})
  - *RotatingCoordinates_t* (RotatingCoordinates)
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *CGNSBase_t*

5.39 FlowEquationSet_t

- Name:
  - FlowEquationSet
- Data-type: MT
- Cardinality: Zero/One
- Children
  - *GoverningEquations_t* (GoverningEquations)
  - *EquationDimension_t* (EquationDimension)
  - *GasModel_t* (GasModel)
  - *ViscosityModel_t* (ViscosityModel)
  - *ThermalRelaxationModel_t* (ThermalRelaxationModel)
  - *ThermalConductivityModel_t* (ThermalConductivityModel)
  - *TurbulenceModel_t* (TurbulenceModel)
  - *TurbulenceClosure_t* (TurbulenceClosure)
  - *ChemicalKineticsModel_t* (ChemicalKineticsModel)
  - *EMMagneticFieldModel_t* (EMMagneticFieldModel)
  - *EMElectricFieldModel_t* (EMElectricFieldModel)
  - *EMConductivityModel_t* (EMConductivityModel)
  - *Descriptor_t* ({UserDefined})
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *CGNSBase_t*
  - *Zone_t*

5.40 FlowSolution_t

- Name:
  - {UserDefined}
- Data-type: MT
- Cardinality: Zero/N
- Children
  - *GridLocation_t* (GridLocation)
  - *DataArray_t* ({UserDefined})
  - *Rind_t* (Rind)
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *Descriptor_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *Zone_t*

5.41 GasModel_t

- Name:
  - GasModel
- Data-type: C1
- Enumerate:
- Cardinality: Zero/One
- Children
  - *Descriptor_t* ({UserDefined})
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *DataArray_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *FlowEquationSet_t*

5.42 GeometryEntity_t

- Name:
  - {UserDefined}
- Data-type: MT
- Cardinality: Zero/N
- Children
- Parents
  - *GeometryReference_t*

5.43 GeometryFile_t

- Name:
  - GeometryFile
- Data-type: C1
- Cardinality: One/One
- Children
- Parents
  - *GeometryReference_t*

5.44 GeometryFormat_t

- Name:
  - GeometryFormat
- Data-type: C1
- Cardinality: One/One
- Children
- Parents
  - *GeometryReference_t*

5.45 GeometryReference_t

- Name:
  - {UserDefined}
- Data-type: MT
- Cardinality: Zero/N
- Children

  - *Descriptor_t* ({UserDefined})
  - *GeometryFile_t* (GeometryFile)
  - *GeometryFormat_t* (GeometryFormat)
  - *GeometryEntity_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *Family_t*

5.46 GoverningEquations_t

- Name:
  - GoverningEquations
- Data-type: C1
- Enumerate:
- Cardinality: Zero/One
- Children
  - *Descriptor_t* ({UserDefined})
  - *DiffusionModel_t* (DiffusionModel)
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *FlowEquationSet_t*

5.47 Gravity_t

- Name:
  - {UserDefined}
- Data-type: MT
- Cardinality: Zero/One
- Children
  - *DataArray_t* (GravityVector)
  - *Descriptor_t* ({UserDefined})
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *CGNSBase_t*

5.48 GridConnectivity1to1_t

- Name:
  - {UserDefined}
- Data-type: C1
- Cardinality: Zero/N
- Children
  - *Transform_t"* (Transform)
  - *IndexRange_t* (PointRange)
  - *IndexRange_t* (PointRangeDonor)
  - *Ordinal_t* (Ordinal)
  - *GridConnectivityProperty_t* (GridConnectivityProperty)
  - *Descriptor_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *ZoneGridConnectivity_t*

5.49 GridConnectivityProperty_t

- Name:
  - GridConnectivityProperty
- Data-type: MT
- Cardinality: Zero/One
- Children
  - *Descriptor_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
  - *Periodic_t* (Periodic)
  - *AverageInterface_t* (AverageInterface)
- Parents
  - *GridConnectivity1to1_t*
  - *GridConnectivity_t*

5.50 GridConnectivityType_t

- Name:
  - GridConnectivityType
- Data-type: C1
- Cardinality: One/One

- Children
- Parents
  - *GridConnectivity_t*

5.51 GridConnectivity_t

- Name:
  - {UserDefined}
- Data-type: C1
- Cardinality: Zero/N
- Children
  - *GridLocation_t* (GridLocation)
  - *Ordinal_t* (Ordinal)
  - *Descriptor_t* ({UserDefined})
  - *IndexRange_t* (PointRange)
  - *IndexArray_t* (PointList)
  - *IndexArray_t* (PointListDonor)
  - *IndexArray_t* (CellListDonor)
  - *GridConnectivityProperty_t* (GridConnectivityProperty)
  - *GridConnectivityType_t* (GridConnectivityType)
  - *DataArray_t* (InterpolantsDonor)
- Parents
  - *ZoneGridConnectivity_t*

5.52 GridCoordinates_t

- Name:
  - GridCoordinates
  - {UserDefined}
- Data-type: MT
- Cardinality: Zero/N
- Children
  - *DataArray_t* ({UserDefined})
  - *Rind_t* (Rind)
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *Descriptor_t* ({UserDefined})

  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *Zone_t*

5.53 GridLocation_t

- Name:
  - GridLocation
- Data-type: C1
- Cardinality: Zero/One
- Children
- Parents
  - *ArbitraryGridMotion_t*
  - *BCDataSet_t*
  - *BC_t*
  - *DiscreteData_t*
  - *FlowSolution_t*
  - *GridConnectivity_t*
  - *OversetHoles_t*
  - *UserDefinedData_t*

5.54 IndexArray_t

- Name:
  - PointList
  - PointListDonor
  - CellListDonor
  - InwardNormalList
  - {UserDefined}
- Data-type: I4 R4 R8
- Cardinality: Zero/One
- Children
- Parents
  - *BCDataSet_t*
  - *BC_t*
  - *GridConnectivity_t*
  - *OversetHoles_t*

  - *UserDefinedData_t*

5.55 IndexRange_t

- Name:
  - PointRange
  - PointRangeDonor
  - ElementRange
  - {UserDefined}
- Data-type: I4
- Cardinality: Zero/One
- Children
- Parents
  - *BCDataSet_t*
  - *BC_t*
  - *Elements_t*
  - *GridConnectivity1to1_t*
  - *GridConnectivity_t*
  - *OversetHoles_t*
  - *UserDefinedData_t*

5.56 IntegralData_t

- Name:
  - {UserDefined}
- Data-type: MT
- Cardinality: Zero/N
- Children
  - *Descriptor_t* ({UserDefined})
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *DataArray_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *CGNSBase_t*
  - *Zone_t*

5.57 InwardNormalIndex_t

- Name:
  - InwardNormalIndex
- Data-type: I4
- Cardinality: Zero/One
- Children
- Parents
  - *BC_t*

5.58 Ordinal_t

- Name:
  - Ordinal
- Data-type: I4
- Cardinality: Zero/One
- Children
- Parents
  - *BC_t*
  - *Family_t*
  - *GridConnectivity1to1_t*
  - *GridConnectivity_t*
  - *UserDefinedData_t*
  - *Zone_t*

5.59 OversetHoles_t

- Name:
  - {UserDefined}
- Data-type: MT
- Cardinality: Zero/N
- Children
  - *Descriptor_t* ({UserDefined})
  - *IndexArray_t* (PointList)
  - *GridLocation_t* (GridLocation)
  - *IndexRange_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents

  - *ZoneGridConnectivity_t*

5.60 Periodic_t

- Name:
  - Periodic
- Data-type: MT
- Cardinality: Zero/One
- Children
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *Descriptor_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
  - *DataArray_t* (RotationCenter)
  - *DataArray_t* (RotationAngle)
  - *DataArray_t* (Translation)
- Parents
  - *GridConnectivityProperty_t*

5.61 ReferenceState_t

- Name:
  - ReferenceState
- Data-type: MT
- Cardinality: Zero/One
- Children
  - *Descriptor_t* ({UserDefined})
  - *Descriptor_t* (ReferenceStateDescription)
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *DataArray_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *BCDataSet_t*
  - *BC_t*
  - *CGNSBase_t*
  - *ZoneBC_t*

  - *Zone_t*

5.62 RigidGridMotionType_t

- Name:
  - RigidGridMotionType
- Data-type: C1
- Cardinality: One/One
- Children
- Parents

5.63 RigidGridMotion_t

- Name:
  - {UserDefined}
- Data-type: C1
- Enumerate:
- Cardinality: Zero/N
- Children
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *Descriptor_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
  - *DataArray_t* ({UserDefined})
- Parents
  - *Zone_t*

5.64 Rind_t

- Name:
  - Rind
- Data-type: I4
- Cardinality: Zero/One
- Children
- Parents
  - *ArbitraryGridMotion_t*
  - *DiscreteData_t*

  - *Elements_t*
  - *FlowSolution_t*
  - *GridCoordinates_t*

5.65 RotatingCoordinates_t

- Name:
  - RotatingCoordinates
- Data-type: MT
- Cardinality: Zero/One
- Children
  - *DataArray_t* (RotationCenter)
  - *DataArray_t* (RotationRateVector)
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *Descriptor_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *CGNSBase_t*
  - *Family_t*
  - *Zone_t*

5.66 SimulationType_t

- Name:
  - SimulationType
- Data-type: C1
- Enumerate:
- Cardinality: One/One
- Children
- Parents
  - *CGNSBase_t*

5.67 ThermalConductivityModel_t

- Name:
  - ThermalConductivityModel
- Data-type: C1
- Enumerate:
- Cardinality: Zero/One
- Children
  - *Descriptor_t* ({UserDefined})
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *DataArray_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *FlowEquationSet_t*

5.68 ThermalRelaxationModel_t

- Name:
  - ThermalRelaxationModel
- Data-type: C1
- Enumerate:
- Cardinality: Zero/One
- Children
  - *Descriptor_t* ({UserDefined})
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *DataArray_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *FlowEquationSet_t*

5.69 Transform_t”

- Name:
  - Transform
- Data-type: I4
- Cardinality: Zero/One

- Children
- Parents
  - *GridConnectivity1to1_t*

5.70 TurbulenceClosure_t

- Name:
  - TurbulenceClosure
- Data-type: C1
- Enumerate:
- Cardinality: Zero/One
- Children
  - *Descriptor_t* ({UserDefined})
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *DataArray_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *FlowEquationSet_t*

5.71 TurbulenceModel_t

- Name:
  - {UserDefined}
- Data-type: C1
- Enumerate:
- Cardinality: Zero/One
- Children
  - *Descriptor_t* ({UserDefined})
  - *DataArray_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *DiffusionModel_t* (DiffusionModel)
- Parents
  - *FlowEquationSet_t*

5.72 UserDefinedData_t

- Name:
  - {UserDefined}
- Data-type: MT
- Cardinality: Zero/N
- Children
  - *Descriptor_t* ({UserDefined})
  - *GridLocation_t* (GridLocation)
  - *IndexRange_t* (PointRange)
  - *IndexArray_t* (PointList)
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *DataArray_t* ({UserDefined})
  - *FamilyName_t* (FamilyName)
  - *UserDefinedData_t* ({UserDefined})
  - *Ordinal_t* (Ordinal)
- Parents
  - *ArbitraryGridMotion_t*
  - *Area_t*
  - *AverageInterface_t*
  - *Axisymmetry_t*
  - *BCDataSet_t*
  - *BCData_t*
  - *BCProperty_t*
  - *BC_t*
  - *BaseIterativeData_t*
  - *CGNSBase_t*
  - *ChemicalKineticsModel_t*
  - *ConvergenceHistory_t*
  - *DiscreteData_t*
  - *EMConductivityModel_t*
  - *EMElectricFieldModel_t*
  - *EMMagneticFieldModel_t*
  - *Elements_t*
  - *Family_t*
  - *FlowEquationSet_t*
  - *FlowSolution_t*
  - *GasModel_t*

  - *GeometryReference_t*
  - *GoverningEquations_t*
  - *Gravity_t*
  - *GridConnectivity1to1_t*
  - *GridConnectivityProperty_t*
  - *GridCoordinates_t*
  - *IntegralData_t*
  - *OversetHoles_t*
  - *Periodic_t*
  - *ReferenceState_t*
  - *RigidGridMotion_t*
  - *RotatingCoordinates_t*
  - *ThermalConductivityModel_t*
  - *ThermalRelaxationModel_t*
  - *TurbulenceClosure_t*
  - *TurbulenceModel_t*
  - *ViscosityModel_t*
  - *WallFunction_t*
  - *ZoneBC_t*
  - *ZoneGridConnectivity_t*
  - *ZoneIterativeData_t*
  - *Zone_t*

5.73 ViscosityModel_t

- Name:
  - ViscosityModel
- Data-type: C1
- Enumerate:
- Cardinality: Zero/One
- Children
  - *Descriptor_t* ({UserDefined})
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *DataArray_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *FlowEquationSet_t*

5.74 WallFunctionType_t

- Name:
  - WallFunctionType
- Data-type: C1
- Cardinality: One/One
- Children
- Parents
  - *WallFunction_t*

5.75 WallFunction_t

- Name:
  - WallFunction
- Data-type: MT
- Cardinality: Zero/One
- Children
  - *Descriptor_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
  - *WallFunctionType_t* (WallFunctionType)
- Parents
  - *BCProperty_t*

5.76 ZoneBC_t

- Name:
  - ZoneBC
- Data-type: MT
- Cardinality: Zero/One
- Children
  - *BC_t* ({UserDefined})
  - *ReferenceState_t* (ReferenceState)
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *Descriptor_t* ({UserDefined})

  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *Zone_t*

5.77 ZoneGridConnectivity_t

- Name:
  - ZoneGridConnectivity
- Data-type: MT
- Cardinality: Zero/One
- Children
  - *GridConnectivity1to1_t* ({UserDefined})
  - *GridConnectivity_t* ({UserDefined})
  - *OversetHoles_t* ({UserDefined})
  - *Descriptor_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *Zone_t*

5.78 ZoneIterativeData_t

- Name:
  - {UserDefined}
- Data-type: MT
- Cardinality: Zero/One
- Children
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)
  - *Descriptor_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
  - *DataArray_t* ({UserDefined})
- Parents
  - *Zone_t*

5.79 ZoneType_t

- Name:
  - ZoneType
- Data-type: C1
- Enumerate:
- Cardinality: One/One
- Children
- Parents
  - *Zone_t*

5.80 Zone_t

- Name:
  - {UserDefined}
- Data-type: I4
- Cardinality: Zero/N
- Children
  - *GridCoordinates_t* (GridCoordinates)
  - *GridCoordinates_t* ({UserDefined})
  - *DiscreteData_t* ({UserDefined})
  - *Elements_t* ({UserDefined})
  - *ZoneBC_t* (ZoneBC)
  - *FlowSolution_t* ({UserDefined})
  - *ZoneType_t* (ZoneType)
  - *Ordinal_t* (Ordinal)
  - *ZoneGridConnectivity_t* (ZoneGridConnectivity)
  - *ZoneIterativeData_t* ({UserDefined})
  - *RigidGridMotion_t* ({UserDefined})
  - *ReferenceState_t* (ReferenceState)
  - *IntegralData_t* ({UserDefined})
  - *ArbitraryGridMotion_t* ({UserDefined})
  - *FamilyName_t* (FamilyName)
  - *FlowEquationSet_t* (FlowEquationSet)
  - *ConvergenceHistory_t* (ZoneConvergenceHistory)
  - *RotatingCoordinates_t* (RotatingCoordinates)
  - *DataClass_t* (DataClass)
  - *DimensionalUnits_t* (DimensionalUnits)

  - *Descriptor_t* ({UserDefined})
  - *UserDefinedData_t* ({UserDefined})
- Parents
  - *CGNSBase_t*

CHAPTER

SIX

ERROR CODES AND FUNCTIONS

The errors are managed using exceptions. The base class is *cgnsException*, the derived classes are in the list below, for each class you can have several error codes. For example you can catch *cgnsNameError* and have a more detailled error diagnostic with the error code:

```
try:
  CGU.checkName('.')
except CGE.cgnsNameError:
  # skip exception
  # a cgnsNameError is a cgnsException
  pass

try:
  CGU.checkName('zapzap/s')
except CGE.cgnsException,why:
  # get message and print it
  # actually 'why' is the exception object but print calls its __str__
  print why

try:
  CGU.checkName('')
except CGE.cgnsNameError,exc:
  # a cgnsException has a 'code' attribute (the integer error code)
  # a 'value' attribute with a tuple of arguments set at raise time
  # a cgnsNameError is a cgnsException
  if (exc.code==21): print 'Cannot find node ',exc.value
```

6.1 cgnsNameError

code	Message
21	No node with name [%s]
22	Node name should have type string
23	Empty string is not allowed for a node name
24	Node name should not contain a '/'
25	Node name length should not be greater than 32 chars
102	Duplicated child name [%s] in [%s]

6.2 cgnsNodeError

code	Message
1	Node is empty !
2	Node should be a list of <name, value, children, type>
3	Node name should be a string
4	Node [%s] children list should be a list
5	Node [%s] bad value: should be a numpy object

6.3 cgnsTypeError

code	Message
103	Node type of [%s] not [%s]
104	Node type of [%s] not in %s

6.4 cgnsValueError

code	Message
000	

CHAPTER

SEVEN

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.

7.1 PAT Index

- *genindex*

PYTHON MODULE INDEX