Converter Documentation

Release 3.2

/ELSA/MU-09020/V3.2

Feb 09, 2021

CONTENTS

CHAPTER

ONE

PREAMBLE

This module provides functions for CFD data conversion (both file format and grid topology).

This module is part of Cassiopee, a free open-source pre- and post-processor for CFD simulations.

This module can manipulate two different data structures: the first one is called an **array** and is very close to a numpy array, the second one is called a **pyTree** and it implements the CGNS/python standard.

- An **array** is a simple definition of a mesh using classical numpy arrays.

An array can be a structured array defined by a python list **['x,y,z,...', an, ni, nj, nk]**, where *ni, nj, nk* are the dimension of the grid and *an* is a (*nfld*, *nixnjxnk*) numpy array containing data (coordinates and fields).

An array can also be an unstructured array defined by **['x,y,z,...', an, cn, 'ELTTYPE']**, where *cn* is a numpy array storing the elements-to-nodes connectivity and *an* is a numpy array of data. If *an* stores fields on nodes, 'ELTTYPE' can be **'NODE', 'BAR', 'TRI', 'QUAD', 'TETRA', 'PYRA', 'PENTA', 'HEXA'**. If *an* stores field on elements, 'ELTTYPE' can be **'NODE*', 'BAR*', 'TRI*', 'QUAD*', 'TETRA*', 'PYRA*', 'PENTA*', 'HEXA*'**. Finally, an unstructured array can be of type **'NGON'**, describing meshes made of polyhedral elements. For those arrays, the connectivity *cn* is made of a faces-to-nodes connectivity (FN) and a elements-to-faces connectivity (EF). *cn* is then a flat numpy array [nfaces, sizeofFN, ..FN.., nelts, sizeofEF, ..EF..], where nfaces is the number of faces in mesh, nelts the number of elements in mesh. For each face, FN is [number of nodes, ..nodes indices..]. For each element, EF is [number of faces, ..face indices..].

To use the array interface:

```
import Converter as C
```

- A **pyTree** is a CGNS/Python tree, that is a mapping of the CGNS standard in Python, using lists and numpy arrays.

Each node of the tree is a Python list defined by **['name', ar, [...], 'CGNSType_t']**, where

ar is the value stored by this node (*ar* is a numpy array), and [...] designates a list of nodes that are the children of the current node.

Important note: Numpy arrays stored in pyTrees stores only one variable for each node. Numpy arrays for a structured zone can be accessed by *ar* [i,j,k] and by *ar* [ind] for a unstructured zone.

To use the pyTree interface:

```
import Converter.PyTree as C
```

CHAPTER

TWO

NAME STANDARDISATION

Some functions of Converter, Post and other modules perform specific treatments for given variables. For instance, the computeVariables function in the Post module can compute the pressure automatically if density and velocity are defined with their CGNS names. Recognised names are CGNS names, but some alternative names are also recognised. Naming convention is described in the following table.

Description	CGNS	Alternative names
Coordinate in x direction	CoordinateX	x, X
Coordinate in y direction	CoordinateY	y, Y
Coordinate in z direction	CoordinateZ	z, Z
Density	Density	ro
Momentum in x direction	MomentumX	rou, rovx
Momentum in y direction	MomentumY	rov, rovy
Momentum in z direction	MomentumZ	row, rovz
Density times total energy	EnergyStagnationDensity	roE
Density times turbulence kinetic energy	TurbulentEnergyKineticDensity	rok
Density times dissipation rate of turbulence kinetic energy	TurbulentDissipationDensity	roeps
Static pressure	Pressure	p, P
Dynamic pressure	PressureDynamic	
Enthalpy	Enthalpy	
Entropy	Entropy	
Stagnation pressure	PressureStagnation	
Stagnation temperature	TemperatureStagnation	
x-component of the absolute velocity	VelocityX	vx, u
y-component of the absolute velocity	VelocityY	vy, v
z-component of the absolute velocity	VelocityZ	vz, w
Absolute velocity magnitude	VelocityMagnitude	
Absolute mach number	Mach	
Molecular viscosity	ViscosityMolecular	
Cell nature field (0:blanked, 1:discretised, 2:interpolated)		cellN, cellnf
Cell nature field (0:blanked, 1:discretised, -Id interp block)		cellNF, cellnf
Cell nature field (-1:orphan, 0:blanked, 1:discretised, 2: interpolated explicitely, 3: extrapolated, 4: interp implicit)		status

CHAPTER

THREE

LIST OF FUNCTIONS

– Array creation and manipulations

Converter.array(vars, n1, n2, sub[, api])	Create a structured or unstructured array.
Converter.getValue(array, ind)	Return the values of an array for a point of index ind or (i,j,k)...
Converter.setValue(array, ind, values)	Set the values in an array for a point of index ind or (i,j,k)...
Converter.addVars(array[, add])	Add variables to an array.
Converter.copy(array)	Copy an array.

– PyTree creation and manipulations

Converter.PyTree.newPyTree([args])	Create a new PyTree.
Converter.PyTree.getNobOfBase(base, t)	Return the nob of a base in t.
Converter.PyTree.getNobNozOfZone(a, t)	Return the (nob, noz) of a in t.
Converter.PyTree.breakConnectivity(t)	Break a multi-element zone into single element zones.
Converter.PyTree.mergeConnectivity(z1, z2[, ...])	Gather an additional zone connectivity in z1.
Converter.PyTree.deleteEmptyZones(t)	Delete zones with null number of points or elements.
Converter.PyTree.addBase2PyTree(t, baseName)	Add a base name to a pyTree.
Converter.PyTree.addState(t[, state, value, ...])	Add single state value or a full reference state.
Converter.PyTree.addChimera2Base(base, ...)	Add chimera setting as node in base.
Converter.PyTree.addBC2Zone(a, bndName, bndType)	Add a BC to a zone node.
Converter.PyTree.fillEmptyBCWith(t, bndName, ...)	Fill empty BCs with given type.

Continued on next page

Table 2 – continued from previous page

Converter.PyTree.rmBCOfType(t, bndType)	Remove BCs of given type.
Converter.PyTree.rmBCOfName(t, bndName)	Remove BCs of given name.
Converter.PyTree.rmBCDataVars(t, var)	Remove variables var from t in every BCDataSet.
Converter.PyTree.extractBCOfType(t, bndType)	Extract the grid coordinates of given BC type as zones.
Converter.PyTree.extractBCOfName(t, bndName)	Extract the grid coordinates of given BC name as zones.
Converter.PyTree.getEmptyBC(a[, dim, ...])	Return the range or facelist of unset boundary conditions.
Converter.PyTree.getBCs(t[, reorder, extrapFlow])	Return geometry, names and types of boundary conditions.
Converter.PyTree.recoverBCs(a, T[, tol])	Recover given BCs on a tree.
Converter.PyTree.extractBCFields(z[, varList])	Extract fields on BCs.
Converter.PyTree.getConnectedZones(a, topTree)	Return the list of zones connected to a through match or nearMatch.
Converter.PyTree.addFamily2Base(base, familyName)	Add a family node to a base node.
Converter.PyTree.tagWithFamily(z, familyName)	Tag a zone node or a BC node with a familyName.
Converter.PyTree.getFamilyZones(t, familyName)	Return all zones that have this familyName.
Converter.PyTree.getFamilyBCs(t, familyName)	Return all BC nodes that have this familyName.
Converter.PyTree.getFamilyZoneNames(t)	Return the family zone names in a tree or a base.
Converter.PyTree.getFamilyBCNamesOfType(t[, ...])	Return the family BC names of a given type.
Converter.PyTree.getFamilyBCNamesDict(t)	Return the dictionary of familyBCs.
Converter.PyTree.getValue(t, var, ind)	Return the values for a point of index ind or (i,j,k)...
Converter.PyTree.setValue(t, var, ind, val)	Set the values in an array for a point of index ind or (i,j,k)...
Converter.PyTree.setPartialFields(t, arrays, ...)	Set some field values for given indices.
Converter.PyTree.addVars(t, vars)	Add variables to a pyTree.

Continued on next page

Table 2 – continued from previous page

Converter.PyTree.fillMissingVariables(t)	Fill FlowSolution nodes with variables, such that all the zones have the same variables.
Converter.PyTree.cpVars(t1, var1, t2, var2)	Copy field variables.

– Array / PyTree common manipulations

Converter.getVarNames(a)	Get variable names.
Converter.isNamePresent(a, varname)	Test if varName is present in a.
Converter.getNPts(a)	Return the total number of points.
Converter.getNCells(a)	Return the total number of cells.
Converter.initVars(a, var[, v1, v2])	Initialize a variable by a value or a formula.
Converter.extractVars(array, vars)	Extract variables from a.
Converter.rmVars(a, var)	Remove variables.
Converter.convertArray2Tetra(array[, split])	Convert a array in an unstructured tetra array.
Converter.convertArray2Hexa(array)	Convert a array in an unstructured hexa array.
Converter.convertArray2NGon(array)	Convert a array in a NGON array.
Converter.convertArray2Node(array)	Convert an array in an unstructured node array.
Converter.convertBAR2Struct(array)	Convert a BAR array without ramifications, closed into an i-array.
Converter.convertTri2Quad(array[, alpha])	Convert a TRI array to a QUAD array.
Converter.convertHO2LO(a[, mode])	Convert a HO mesh to a low order mesh.
Converter.convertLO2HO(a[, mode, order])	Convert a LO mesh to a high order mesh.
Converter.conformizeNGon(array[, tol])	Conformize topologically a NGON array.
Converter.node2Center(array[, accurate])	Convert array defined on nodes to array defined on centers.
Converter.center2Node(array[, cellNType])	Convert array defined on centers to array defined on nodes.

– Array / PyTree analysis

Converter.diffArrays(arrays1, arrays2[, arrays3])	Diff arrays defining solutions.

Continued on next page

Table 4 – continued from previous page

Converter.getMinValue(array, varName)	Get the minimum value of variable defined by varName in array.
Converter.getMaxValue(array, varName)	Get the maximum value of variable defined by varName in array.
Converter.getMeanValue(array, varName)	Get the mean value of the variable defined by varName in an array.
Converter.getMeanRangeValue(array, varName, ...)	Get the mean value of the variable defined by varName for a sorted range in an array.
Converter.normL0(array, varName)	Get the L0 norm of the field defined by varName in the array.
Converter.normL2(array, varName)	Get the L2 norm of the field defined by varName in the array.
Converter.normalize(a, vars)	Get the normalisation of the fields defined by vars in the array.
Converter.magnitude(array, vars)	Get the magnitude of the fields defined by vars in the array.
Converter.randomizeVar(array, var, deltaMin, ...)	Randomize a field defined by var within a range [a-deltaMin, a+deltaMax].
Converter.isFinite(array[, var])	Return true if all fields have no NAN or INF values.

– Array / PyTree input/output

Converter.convertFile2Arrays(fileName[, ...])	Read file and return arrays containing file data.
Converter.convertArrays2File(arrays, fileName)	Write arrays to output file.
Converter.PyTree.convertFile2PyTree(fileName)	Read a file and return a pyTree containing file data.
Converter.PyTree.convertPyTree2File(t, fileName)	Write a pyTree to a file.

– Preconditioning

Converter.createHook(a[, function])	Create a hook for a given function.
Converter.createGlobalHook(a[, function, indir])	Create a hook for a set of zones and for a given function.
Converter.freeHook(hook)	Free hook.

– Geometrical/topological identification

Converter.identifyNodes(hook, a[, tol])	Identify nodes of a in KDT.
Converter.identifyFaces(hook, a[, tol])	Identify face centers of a in KDT.
Converter.identifyElements(hook, a[, tol])	Identify element centers of a in KDT.
Converter.identifySolutions(coordsRcv, ...)	Identify points in a hook to mesh points and set the solution if donor and receptor points are distant from tol.
Converter.nearestNodes(hook, a)	Find in KDT nearest points to nodes of a.
Converter.nearestFaces(hook, a)	Find in KDT nearest points to face centers of a.
Converter.nearestElements(hook, a)	Find in KDT nearest points to element centers of a.
Converter.createGlobalIndex(a[, start])	Create the global index field.
Converter.recoverGlobalIndex(a, b)	Recover fields of b in a following the global index field.

– Client/server to exchange arrays/pyTrees

Converter.createSockets([nprocs, port])	Create sockets for communication.
Converter.listen(s)	Listen for sends.
Converter.send(data[, host, rank, port])	Send data to socket.

– Converter arrays/3D arrays conversion

Converter.Array3D.convertArrays2Arrays3D(CArrays)	Convert a standard array to a 3D array.
Converter.Array3D.convertArrays3D2Arrays(CArrays)	Convert a 3D array to a standard array.

CHAPTER

FOUR

CONTENTS

4.1 Array creation and manipulations

Converter.**array**(*vars*, *ni*, *nj*, *nk*)

Create a structured array containing variables x,y,z on a nixnjxnk structured grid.

Parameters

- **vars** (string) – variables stored in array
- **ni,nj,nk** (int) – grid dimensions

Return type structured array

Converter.**array**(*vars*, *np*, *ne*, *eltType*)

Create a unstructured array containing variables x,y,z on a unstructured grid. The grid has np points, ne elements of type eltType. eltType can be 'NODE', 'BAR', 'TRI', 'QUAD', 'TETRA', 'PYRA', 'PENTA', 'HEXA', 'NGON'.

Parameters

- **vars** (string) – variables stored in array
- **np** (int) – number of points of grid
- **ne** (int) – number of elements of grid
- **eltType** (string) – type of elements

Return type unstructured array

Example of use:

- Array creation (array):

```
# - array (array) -
import Converter as C

# Structured
```

(continues on next page)

(continued from previous page)

```
b = C.array('x,y,z', 12, 9, 12); print(b)
#>> ['x,y,z', array(...), 12, 9, 12]

# Unstructured
a = C.array('x,y,z', 12, 9, 'QUAD'); print(a)
#>> ['x,y,z', array(...), array(..., dtype=int32), 'QUAD']
```

Converter.**getValue**(*array*, *ind*)

Return the list of values defined in array for point of index ind (for both structured and unstructured arrays). For structured arrays, you can specify (i,j,k) instead of ind. For unstructured arrays, the index ind corresponds to the location type of point defining array a: for instance, if array a describes a field at element vertices, ind is a vertex index (ind starts at 0 and (i,j,k) start at 1).

Parameters

- **array** (array) – input array
- **ind** (int or tuple of int) – index

Return type list of floats corresponding to field values

Example of use:

- Get values for a given grid index (array):

```
# - getValue (array) -
import Converter as C
import Generator as G

# Structured array
Ni = 40; Nj = 50; Nk = 20
a = G.cart((0,0,0), (1./(Ni-1), 0.5/(Nj-1),1./(Nk-1)), (Ni,Nj,Nk))
# Get variable values contained in a (x,y,z) in point (10,1,1)
print(C.getValue(a, (10,1,1)))
#>> [0.23076923076923075, 0.0, 0.0]
print(C.getValue(a, 9)) # It is the same point!
#>> [0.23076923076923075, 0.0, 0.0]

# Unstructured array
Ni = 40; Nj = 50; Nk = 20
a = G.cartTetra((0,0,0), (1./(Ni-1), 0.5/(Nj-1),1./(Nk-1)), (Ni,Nj,Nk))
print(C.getValue(a, 9))
#>> [0.23076923076923075, 0.0, 0.0]
```

Converter.**setValue**(*array*, *ind*, *values*)

Set the values of one point of index ind in array. values must be a list corresponding to the variables stored in array.

Parameters

- **array** (array) – input array
- **ind** (int or tuple of int) – index
- **values** (list of floats) – values of field to set in this point

Example of use:

- Set values at a given grid index (array):

```
# - setValue (array) -
import Converter as C
import Generator as G

a = G.cart((0,0,0), (1,1,1), (5,5,1))

# Set point (1,1,1) with value x=0.1, y =0.1, z=1.
C.setValue(a, (1,1,1), [0.1,0.1,1.]); print(a)

# Same thing with a global index
C.setValue(a, 0, [0.1,0.1,1.])
```

Converter.**addVars**(*array*, *add='Density'*)

Add variable(s) to an array. Variables argument can be a string name ('ro') or a list of string names (['ro', 'rou']). Variables are set to 0.

Parameters

- **array** ([array, list of arrays]) – input array
- **add** (string or list of strings) – variable to add

Return type array with additional variables

Converter.**addVars**([*a*, *b*, *c*])

Concatenate array fields with the same dimensions. Variables defined by a list of arrays are put in the same array.

Parameters **arrays** (list of arrays with same dimension) – input arrays

Return type array with all variables concanated

Example of addVars(array, 'Density'):

- Adding variables (array):

```
# - addVars (array) -
import Converter as C
import Generator as G

a = G.cart((0,0,0), (1,1,1), (10,10,11))

# Add a variable defined by a string
a = C.addVars(a, 'ro')
a = C.addVars(a, 'cellN')
C.convertArrays2File(a, 'out1.plt')

# Add variables defined by a list of varNames
a = C.addVars(a, ['rou','rov'])
C.convertArrays2File(a, 'out2.plt')
```

Example of addVars([a,b,c]):

- Adding many variables (array):

```
# - addVars (array) -
import Converter as C
import Generator as G

a = G.cart((0,0,0), (1,1,1), (10,10,11))
b = C.array('cell', a[2], a[3], a[4])
c = C.array('t,u', a[2], a[3], a[4])
f = C.addVars([a, b, c])
C.convertArrays2File(f, 'out.plt')
```

Converter.**copy**(*array*)

Copy an array (return a new duplicated array).

Parameters **array** ([array, list of arrays]) – input array

Return type identical to input

Example of use:

- Array copy (array):

```
# - copy (array) -
import Converter as C
import Generator as G

a = G.cart((0,0,0,), (1,1,1), (10,10,10))
b = C.copy(a)
C.convertArrays2File([b], "out.plt")
```

4.2 pyTree creation and manipulation

Converter.PyTree.**newPyTree**(*args*)

Create a new pyTree. You can specify base names, cell dimension in base, and attached zones eventually. See below example for all possibilities of input.

Parameters **args** ([list of baseNames, list of baseNames and dimension, list of zones]) – input

Return type a new pyTree

Example of use:

- Create pyTree (pyTree):

```
# - newPyTree (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal

# Create a tree with two bases
t = C.newPyTree(['Base1','Base2'])

# Create a tree with two bases with their dims
t = C.newPyTree(['Base1',2,'Base2',3])

# Create a tree with an attached existing Base node
base = Internal.newCGNSBase('Base', 3)
t = C.newPyTree([base])

# Create a tree with attached existing zones
z1 = Internal.newZone('Zone1')
z2 = Internal.newZone('Zone2')
t1 = C.newPyTree(['Base', z1, z2])
t2 = C.newPyTree(['Base', z1, 'Base2', z2])
t3 = C.newPyTree(['Base', [z1,z2]])
C.convertPyTree2File(t3, 'out.cgns')
```

Converter.PyTree.**getNobOfBase**(*base*, *t*)

Get the number of a given base in tree base list, such that t[2][nob] = base.

Parameters

- **base** (CGNS base node) – a base node
- **t** (pyTree) – tree containing base

Return type the no of base in t children list

Example of use:

- Get base number (pyTree):

```
# - getNobOfBase (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal

t = C.newPyTree(['Base', 'Base2'])
b = Internal.getNodeFromName(t, 'Base2')
nob = C.getNobOfBase(b, t); print(nob)
#>> 2
# This means that t[2][nob] = b
```

Converter.PyTree.**getNobNozOfZone**(*zone*, *t*)

Get the number (nob, noz) of a given zone a tree base and zone list , such that t[2][nob][2][noz] = zone.

Parameters

- **zone** (CGNS zone node) – a zone node
- **t** (pyTree) – top tree containing zone

Return type the no of base and zone in t children list

Example of use:

- Get zone number (pyTree):

```
# - getNobNozOfZone (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
b = G.cart( (0,0,0), (1,1,1), (10,10,10) )
t = C.newPyTree(['Base', 'Base2'])
t[2][1][2] += [a]; t[2][2][2] += [b]
(nob, noz) = C.getNobNozOfZone(a, t); print(nob, noz)
#>> 1 0
# This means that t[2][nob][2][noz] = a
```

Converter.PyTree.**breakConnectivity**(*a*)

Break a multi-element zone (unstructured) into single type element zones. If a is a zone node, return a list of single type element zones. If a is a base, a tree or a list of zones return a base, a tree or a list of zones containing single type element zones.

Parameters **a** ([pyTree, base, zone, list of zones]) – Input data

Return type list of single element zones or identical to input

Example of use:

- Break connectivity (pyTree):

```
# - breakConnectivity (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cartHexa((0,0,0), (1,1,1), (10,10,10))
b = G.cartTetra((9,0,0), (1,1,1), (10,10,5))
c = C.mergeConnectivity(a, b, boundary=0)
# c is a zone with two connectivities (one HEXA and one TETRA)
t = C.newPyTree(['Base',c])
t = C.breakConnectivity(t)
# t contains now two zones (one pure HEXA, one pure TETRA)
C.convertPyTree2File(t, 'out.cgns')

# You can directly break a zone
A = C.breakConnectivity(c)
# A contains 2 zones
C.convertPyTree2File(A, 'out2.cgns')
```

Converter.PyTree.**mergeConnectivity**(*a*, *b*, *boundary=0*)

Merge two zones (unstructured) into a single zone with a multiple connectivity. If boundary=1, b will be a BC connectivity in a (b must be a subzone of a), if boundary=0, b will be a element connectivity.

Parameters

- **a** (CGNS Zone node) – first zone
- **b** (CGNS Zone node) – second zone
- **boundary** (0 or 1) – 1 if b is boundary connectivity, 0 if b is an element connectivity

Return type single zone with multiple connectivity

Example of use:

- Merge connectivity (pyTree):

```
# - mergeConnectivity (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cartHexa((0,0,0), (1,1,1), (10,10,10))
b = G.cartHexa((0,0,0), (1,1,1), (10,10,1))
c = C.mergeConnectivity(a, b, boundary=1)
```

(continues on next page)

(continued from previous page)

```
# c contains now a volume HEXA connectivity and a QUAD boundary connectivity.
C.convertPyTree2File(c, 'out0.cgns')

a = G.cartHexa((0,0,0), (1,1,1), (10,10,10))
b = G.cartTetra((0,0,0), (1,1,1), (10,10,10))
c = C.mergeConnectivity(a, b, boundary=0)
# c is now a multiple-element zone containing a volume HEXA connectivity and
# a volume TETRA connectivity.

C.convertPyTree2File(c, 'out.cgns')
```

Converter.PyTree.**deleteEmptyZones**(*a*)

Delete structured zones with a null ni, nj or nk, delete unstructured zones with a null number of nodes or elements.

Exists also as in place version (_deleteEmptyZones) that modifies a and returns None.

Parameters **a** ([pyTree, base, list of zones]) – Input data

Return type Identical to input

Example of use:

- Delete empty zones (pyTree):

```
# - deleteEmptyZones (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Post.PyTree as P

a = G.cart((0,0,0), (1,1,1), (3,3,3))
b = P.selectCells(a, '{CoordinateX} > 12')
c = P.selectCells(a, '{CoordinateX} > 15')

t = C.newPyTree(['Base',c,a,b])
C._deleteEmptyZones(t)
C.convertPyTree2File(t, 'out.cgns')
```

Converter.PyTree.**addBase2PyTree**(*a*, *baseName*, *cellDim=3*)

Add a base named 'baseName' to a pyTree. Third argument specifies the cell dimension (cellDim=3 for volume meshes, cellDim=2 for surface meshes).

Exists also as in place version (_addBase2PyTree) that modifies a and returns None.

Parameters

- **a** (CGNS pyTree node) – pyTree

- **baseName** (string) – name of created base
- **cellDim** (int) – cell dimension of zones in base

Return type pyTree with new base added

Example of use:

- Add base to pyTree (pyTree):

```
# - addBase2PyTree (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal

t = C.newPyTree(['Base', 3]) # must contain volume zones
t = C.addBase2PyTree(t, 'Base2', 2) # must contain surface zones
Internal.printTree(t)
#>> ['CGNSTree',None,[3 sons],'CGNSTree_t']
#>>   |_['CGNSLibraryVersion',array([3.1],dtype='float64'),[0 son],
↪'CGNSLibraryVersion_t']
#>>   |_['Base',array(shape=(2,),dtype='int32',order='F'),[0 son],'CGNSBase_t']
#>>   |_['Base2',array(shape=(2,),dtype='int32',order='F'),[0 son],'CGNSBase_t']
```

Converter.PyTree.**addState**(*a*, *state*, *value*)

Add a FlowEquation or a ReferenceState data.

Exists also as in place version (_addState) that modifies a and returns None.

Parameters

- **a** ([pyTree, base, zone, list of zones]) – Input data
- **state** (string) – the state to add or modify
- **value** (int, float, string, numpy) – the value of state

Return type Identical to input

Converter.PyTree.**addState**(*a*, *adim='adim1'*, *MInf=None*, *alphaZ=0.*, *alphaY=0.*, *ReInf=1.e8*, *UInf=None*, *TInf=None*, *PInf=None*, *RoInf=None*, *LInf=None*, *Mus=None*, *MutSMuInf=0.2*, *TurbLevelInf=1.e-4*, *EquationDimension=None*, *GoverningEquations=None*)

Add a full reference state built from Adim. See Initiator documentation.

Exists also as in place version (_addState) that modifies a and returns None.

Parameters

- **a** ([pyTree, base, zone, list of zones]) – Input data
- **adim** (string) – type of adimensioning

- **alphaZ...** (MInf,) – data for adimensioning

Return type pyTree with new base added

Example of addState(a, state, value):

- Add single state (pyTree):

```
# - addState (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal

a = G.cylinder((0,0,0), 1., 1.5, 0., 360., 1., (80,30,2))
t = C.newPyTree(['Base',a])

# Specifie des valeurs
b = Internal.getNodeFromName1(t, 'Base')
C._addState(b, 'EquationDimension', 2)
C._addState(b, 'GoverningEquations', 'Euler')
C._addState(b, 'Mach', 0.6)
C._addState(b, 'Reynolds', 100000)
```

Example of addState(a, adim, ...):

- Add reference state (pyTree):

```
# - addState (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cylinder((0,0,0), 1., 1.5, 0., 360., 1., (80,30,2))
t = C.newPyTree(['Base',a])

# Specifie un etat de reference adimensionne par:
# Mach, alpha, Re, MutSMu, TurbLevel (adim1)
C._addState(t, adim='adim1', MInf=0.5, alphaZ=0., alphaY=0.,
            ReInf=1.e8, MutSMuInf=0.2, TurbLevelInf=1.e-4)

# Specifie un etat de reference adimensionne par:
# Mach, alpha, Re, MutSMu, TurbLevel (adim2)
C._addState(t, adim='adim2', MInf=0.5, alphaZ=0., alphaY=0.,
            ReInf=1.e8, MutSMuInf=0.2, TurbLevelInf=1.e-4)

# Specifie un etat de reference dimensionne par:
# U, T, P, L, MutSMu, TurbLevel (dim1)
C._addState(t, adim='dim1', UInf=35, TInf=294., PInf=101325, LInf=1.,
            alphaZ=0., alphaY=0., MutSMuInf=0.2, TurbLevelInf=1.e-4)
```

(continues on next page)

(continued from previous page)

```
# Specifie un etat de reference dimensionne par:
# U, T, Ro, L, MutSMu, TurbLevel (dim2)
C._addState(t, adim='dim2', UInf=35, TInf=294., RoInf=1.2, LInf=1.,
            alphaZ=0., alphaY=0., MutSMuInf=0.2, TurbLevelInf=1.e-4)

# Specifie un etat de reference dimensionne par:
# U, P, Ro, L, MutSMu, TurbLevel (dim2)
C._addState(t, adim='dim3', UInf=35, PInf=101325., RoInf=1.2, LInf=1.,
            alphaZ=0., alphaY=0., MutSMuInf=0.2, TurbLevelInf=1.e-4)

C.convertPyTree2File(t, 'out.cgns')
```

Converter.PyTree.**addChimera2Base**(*base*, *setting*, *value*)

Add a Chimera setting to a node in base. Settings are added in a .Solver#Chimera user defined node. Chen using chimera, a CGNS base defines one component. They are used to define priority in grid assembly, setting of xray blanking, tolerance in double wall, and the kind of relationship for assembling components. '+' means union, '-' means difference, '0' means inactive, 'N': means neutral.

Exists also as in place version (_addChimera2Base) that modifies base and returns None.

Parameters

- **base** (CGNS base node) – input base node
- **setting** (string in ['Priority', 'XRayTol', 'XRayDelta', 'DoubleWallTol', '+', '-', '0', 'N']) – type of chimera setting
- **value** (int, float, string) – value for setting

Return type reference copy of input

Example of use:

- Add Chimera to base (pyTree):

```
# - addChimera2Base (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cylinder((0,0,0), 1., 1.5, 0., 360., 1., (80,30,2))
t = C.newPyTree(['Base', a])
t[2][1] = C.addChimera2Base(t[2][1], 'XRayTol', 1.e-6)
t[2][1] = C.addChimera2Base(t[2][1], 'XRayDelta', 0.1)
t[2][1] = C.addChimera2Base(t[2][1], 'DoubleWallTol', 100.)
```

(continues on next page)

(continued from previous page)

```
t[2][1] = C.addChimera2Base(t[2][1], 'Priority', 1)
C.convertPyTree2File(t, 'out.cgns')
```

Converter.PyTree.**addBC2Zone**(*a, bndName, bndType, wrange=[], zoneDonor=[], rangeDonor=[], trirac=[1,2,3], rotationCenter=[], rotationAngle=[], translation=[], faceList=[], elementList=[], elementRange=[], data=None, subzone=None, faceListDonor=None, elementListDonor=None, elementRangeDonor=None, tol=1.e-12, unitAngle=None*)

Add a physical boundary condition (BC) or a grid connectivity (GC) to a structured/basic element/NGON zone of a PyTree. Parameter bndName is the name of the BC or GC. Exists also as in place version (_addBC2Zone) modifying zone and returning None.

Parameters

- **a** (CGNS zone node) – zone in which the BC/GC is defined
- **bndName** (string) – name of the BC/GC
- **bndType** (string as a CGNS BC type or ['BCMatch', 'BCNearMatch','BCOverlap','FamilySpecified:'+myFamilyBC]) – type of BC or GC defined either by a CGNS type or by a family of BCs of name myFamilyBC. Joins between stages must be defined by a familyBC prefixed by 'BCStage'
- **wrange** (a list of integers defining the window [imin,imax, jmin,jmax,kmin,kmax] or a string in ['imin','imax','jmin', 'jmax,'kmin','kmax']) – for structured grids only. Defines the range of the BC/GC
- **zoneDonor** (zone node for abutting GC and list of [zone nodes, zone names, family of zones prefixed by 'FamilySpecified:]) – donor zone(s)
- **rangeDonor** (a list of integers defining the window [imin, imax,jmin,jmax,kmin,kmax] or a string in ['imin','imax', 'jmin','jmax,'kmin','kmax'] or 'doubly_defined') – range of donor zone for abutting GC, 'doubly_defined' for a doubly defined overlap GC.
- **trirac** (list of three signed integers as a permutation of [1,2,3]) – for an abutting GC, defines the transformation from the window of zone to the donor window

- **rotationCenter** (3-tuple of floats) – for GC with periodicity by rotation, coordinates of the rotation center
- **rotationAngle** (3-tuple of floats) – for GC with periodicity by rotation, angles of rotation in the three directions
- **unitAngle** ('Radian','Degree',None) – defines the units of the rotationAngle (if None, the rotation angle is assumed in degrees)
- **translation** (3-tuple of floats) – for GC with periodicity by translation
- **faceList** (list of integers (>0)) – list of indices of faces for unstructured BE/NGON
- **elementList** (list of integers) – list of indices of elements defining the BC (unstructured basic elements only)
- **elementRange** (list of two integers: [rangeMin, rangeMax]) – range of elements referencing an existing boundary connectivity (unstructured basic elements only)
- **data** (numpy array of floats) – Dirichlet data set in a BCDataSet node with name 'State'
- **subzone** (pyTree zone) – zone corresponding to the window where the BC is defined (for unstructured zones only)
- **faceListDonor** (list of integers) – list of donor faces (unstructured zones only)
- **elementListDonor** (list of integers) – list of elements defining the donor window (unstructured basic elements only)
- **elementRangeDonor** (list of two integers [rangeMin, rangeMax]) – range of elements defining an existing boundary connectivity corresponding to the donor window (unstructured basic elements only)
- **tol** (float) – tolerance for abutting GC

Return type reference copy of input

Example of use:

- Add boundary condition to zone (pyTree):

```
# - addBC2Zone (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

# - Structured grids -
```

(continues on next page)

(continued from previous page)

```
a = G.cylinder((0,0,0), 1., 1.5, 0., 360., 1., (80,30,2))
b = G.cart((-0.1,0.9,0), (0.01,0.01,1.), (20,20,2))

# Physical BC (here BCWall)
a = C.addBC2Zone(a, 'wall1', 'BCWall', 'jmin')
# Matching BC
a = C.addBC2Zone(a, 'match1', 'BCMatch', 'imin', a, 'imax', [1,2,3])
# Matching BC with donor zone name
a = C.addBC2Zone(a, 'match1', 'BCMatch', 'imin', a[0], [80,80,1,30,1,2],
                 [1,2,3])
# Overlap BC (with automatic donor zones)
a = C.addBC2Zone(a, 'overlap1', 'BCOverlap', [1,80,30,30,1,2])
# Overlap BC (with given donor zones and doubly defined)
a = C.addBC2Zone(a, 'overlap2', 'BCOverlap', 'jmin', zoneDonor=[b],
                 rangeDonor='doubly_defined')
# BC defined by a family name
b = C.addBC2Zone(b, 'wall', 'FamilySpecified:myBCWall', 'imin')
# Periodic matching BC
b = C.addBC2Zone(b, 'match', 'BCMatch', 'jmin', b, 'jmax', [1,2,3],
                 translation=(0,2,0))

t = C.newPyTree(['Base',a,b])
C.convertPyTree2File(t, 'out.cgns')

# - Unstructured grids -
a = G.cartTetra((0,0,0), (1,1,1), (10,10,10))
bc = G.cartTetra((0,0,0), (1,1,1), (10,10,1))
a = C.addBC2Zone(a, 'wall1', 'BCWall', subzone=bc)
C.convertPyTree2File(a, 'out.cgns')
```

- Add boundary condition to NGON zone (pyTree):

```
# - addBC2Zone (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

# - NGons -
a = G.cartNGon((2,0,0), (0.1,0.1,1), (10,10,2))
C._addBC2Zone(a, 'wall', 'BCWall', faceList=[1,2])
C.convertPyTree2File(a, 'out.cgns')
```

Converter.PyTree.**fillEmptyBCWith**(*a*, *bndName*, *bndType*, *dim=3*)

Fill empty boundary conditions of grids with the given boundary condition. Parameter dim can be 2 or 3.

Exists also as in place version (_fillEmptyBCWith) that modifies a and returns None.

Parameters

- **a** ([pyTree, base, zone, list of zones]) – input data
- **bndName** (string) – generic name of bnd
- **bndType** (string) – type of bnd
- **dim** (2 or 3) – dimension of problem

Return type reference copy of input

Example of use:

- Fill empty BC (pyTree):

```
# - fillEmpyBCWith (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0),(1,1,1),(10,10,2))
a = C.addBC2Zone(a, 'overlap', 'BCOverlap', 'imin')
a = C.addBC2Zone(a, 'match1', 'BCMatch', 'jmin', a, 'jmax', [1,2,3])
a = C.fillEmptyBCWith(a, 'wall', 'BCWall', dim=2)
C.convertPyTree2File(a, 'out.cgns')
```

Converter.PyTree.**rmBCOfType**(*a*, *bndType*)

Remove all boundaries of a given type. bndType accepts wildcards. bndType can also be a family BC name. In this case, to remove a family named 'myFamily', you must set bndType to 'FamilySpecified:myFamily'.

Exists also as in place version (_rmBCOfType) that modifies a and returns None.

Parameters

- **a** ([pyTree, base, zone, list of zones]) – input data
- **bndType** (string) – type of bnd to remove (accepts wildcards)

Return type reference copy of input

Example of use:

- Remove BC of type (pyTree):

```
# - rmBCOfType (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
```

(continues on next page)

(continued from previous page)

```
a = G.cylinder((0,0,0), 1., 1.5, 0., 360., 1., (80,30,2))
b = G.cart((-0.1,0.9,0), (0.01,0.01,1.), (20,20,2))

a = C.addBC2Zone(a, 'wall1', 'BCWallInviscid', 'jmin')
a = C.addBC2Zone(a, 'match1', 'BCMatch', 'imin', a, 'imax', [1,2,3])
a = C.addBC2Zone(a, 'match2', 'BCMatch', 'imax', a, 'imin', [1,2,3])
a = C.addBC2Zone(a, 'overlap1', 'BCOverlap', 'jmax')
b = C.addBC2Zone(b, 'wall2', 'BCWall', 'imin')
b = C.addBC2Zone(b, 'loin', 'FamilySpecified:LOIN', 'imax')

t = C.newPyTree(['Base',a,b])
t[2][1] = C.addFamily2Base(t[2][1], 'LOIN', bndType='BCFarfield')

t = C.rmBCOfType(t, 'BCWall*') # rm Wall BCs
t = C.rmBCOfType(t, 'BCMatch') # rm match GC
t = C.rmBCOfType(t, 'BCFarfield') # rm FarField BCs
t = C.rmBCOfName(t, 'FamilySpecified:LOIN') # rm Family
C.convertPyTree2File(t, 'out.cgns')
```

Converter.PyTree.**rmBCOfName**(*a*, *bndName*)

Remove all boundaries of a given name. bndName accepts wildcards. bndName can also be a family BC name. In this case, to remove a family named 'myFamily', you must set bndName to 'FamilySpecified:myFamily'.

Exists also as in place version (_rmBCOfName) that modifies a and returns None.

Parameters

- **a** ([pyTree, base, zone, list of zones]) – input data
- **bndName** (string) – name of bnd to remove (accepts wildcards)

Return type reference copy of input

Example of use:

- Remove BC of name (pyTree):

```
# - rmBCOfName (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cylinder((0,0,0), 1., 1.5, 0., 360., 1., (80,30,2))
b = G.cart((-0.1,0.9,0), (0.01,0.01,1.), (20,20,2))

a = C.addBC2Zone(a, 'wall1', 'BCWall', 'jmin')
a = C.addBC2Zone(a, 'match1', 'BCMatch', 'imin', a, 'imax', [1,2,3])
```

(continues on next page)

(continued from previous page)

```
a = C.addBC2Zone(a, 'match2', 'BCMatch', 'imax', a, 'imin', [1,2,3])
a = C.addBC2Zone(a, 'overlap1', 'BCOverlap', 'jmax')
b = C.addBC2Zone(b, 'wall2', 'BCWall', 'imin')
b = C.addBC2Zone(b, 'loin', 'FamilySpecified:LOIN', 'imax')

t = C.newPyTree(['Base',a,b])
t[2][1] = C.addFamily2Base(t[2][1], 'LOIN', bndType='BCFarfield')

t = C.rmBCOfName(t, 'wall1') # rm BC named wall1
t = C.rmBCOfName(t, 'match*') # rm all BC starting with match
t = C.rmBCOfName(t, 'FamilySpecified:LOIN') # rm all BCs of family LOIN

C.convertPyTree2File(t, 'out.cgns')
```

Converter.PyTree.**rmBCDataVars**(*a*, *varName*)

Remove variables given by varName in all BCDataSet. a can be tree, zone or list of zones. varName can be single variable name or a list of variable name.

Exists also as in place version (_rmBCDataVars) that modifies a and returns None.

Parameters

- **a** ([pyTree, base, zone, list of zones]) – input data
- **varName** (string or list of strings) – name of variable (or list of name) to remove

Return type reference copy of input

Example of use:

- Remove BCDataSet variable (pyTree):

```
# - rmBCDataVars (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0),(1,1,1),(10,10,10))
a = C.addBC2Zone(a, 'wall1', 'BCWall', 'jmin')
a = C.addBC2Zone(a, 'wall2', 'BCWall', 'jmax')
C._initBCDataSet(a,'{var1}=1.')
C._initBCDataSet(a,'{var2}=2.')
C._initBCDataSet(a,'{var3}=3.')
a = C.rmBCDataVars(a,'var1')
a = C.rmBCDataVars(a,['var2','var3'])
```

Note: new in version 2.7.

Converter.PyTree.**extractBCOfType**(*a*, *bndType*, *topTree=None*, *reorder=True*)

Extract all boundaries of a given type. Returns a list of zones. Each zone corresponds to one boundary condition. If a BCDataSet exists in boundary condition, it is contained as zones flow solution. If flow solution exists and no BCDataSet exists, the flow solution is extracted. bndType accepts wildcards. bndType can also be a family BC name. Int this case, to extract the BCs of 'myFamily', you must set bntType to 'FamilySpecified:myFamily'.

Parameters

- **a** ([pyTree, base, zone, list of zones]) – input data
- **bndType** (string) – type of BC to extract (accepts wildcards)
- **topTree** (pyTree) – top tree if a is a zone and contains families of BCs.
- **reorder** (Boolean) – if True, extracted zones are reordered such that normals are oriented towards the interior of a.

Return type list of zones

Example of use:

- Extract BCs of type (pyTree):

```
# - extractBCOfType (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cylinder((0,0,0), 1., 1.5, 360., 0., 1., (100,30,10))
a = C.addBC2Zone(a, 'wall1', 'BCWall', 'jmin')
Z = C.extractBCOfType(a, 'BCWall')
C.convertPyTree2File(Z, 'out.cgns')
```

Converter.PyTree.**extractBCOfName**(*a*, *bndName*, *reorder=True*)

Extract all boundaries of a given name. Returns a list of zones. Each zone corresponds to one boundary condition. If a BCDataSet exists in boundary condition, it is contained as zones flow solution. If flow solution exists and no BCDataSet exists, the flow solution is extracted. bndName accepts wildcards. bndName can also be a family BC name. Int this case, to extract the BCs of 'myFamily', you must set bntName to 'FamilySpecified:myFamily'.

Parameters

- **a** ([pyTree, base, zone, list of zones]) – input data
- **bndName** (string) – name of BC to extract (accepts wildcards)
- **reorder** (Boolean) – if True, extracted zones are reordered such that normals are oriented towards the interior of a.

Return type list of zones

Example of use:

- Extract BC(s) of name (pyTree):

```
# - extractBCOfName (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cylinder((0,0,0), 1., 1.5, 360., 0., 1., (100,30,10))
a = C.addBC2Zone(a, 'wall1', 'BCWall', 'jmin')
a = C.addBC2Zone(a, 'walla', 'FamilySpecified:CARTER', 'imin')
t = C.newPyTree(['Base',3,'Skin',2]); t[2][1][2] += [a]
t[2][1] = C.addFamily2Base(t[2][1], 'CARTER', bndType='BCWall')

Z1 = C.extractBCOfName(a, 'wall*')
Z2 = C.extractBCOfName(a, 'FamilySpecified:CARTER')
C.convertPyTree2File(Z1+Z2, 'out.cgns')
```

Converter.PyTree.**getEmptyBC**(*a*, *dim=3*, *splitFactor=180.*)

For each zone, undefined boundary conditions is a list of ranges [imin,imax,jmin,jmax,kmin,kmax] of undefined boundaries for structured zones or is a list of face indices for unstructured zones. The complete return is a list of of undefined boundary condition for each zone.

Lists can be empty ([[],...,[]]) if all the boundary conditions of a zone have been defined. Parameter dim can be 2 or 3. For unstructured grids, undefined boundaries can be split if the angle between neighbouring elements exceeds splitFactor in degrees (default no split).

Parameters

- **a** ([pyTree, base, zone, list of zones]) – input data
- **dim** (2 or 3) – dimension of problem

Return type list of ranges or face indices

Example of use:

- Get empty BCs (pyTree):

```
# - getEmptyBC (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C

a1 = G.cart((0.,0.,0.), (0.1, 0.1, 0.1), (11, 21, 2)); a1[0] = 'cart1'
a1 = C.addBC2Zone(a1, 'wall1', 'BCWall', 'imin')
a2 = G.cart((1., 0.2, 0.), (0.1, 0.1, 0.1), (11, 21, 2)); a2[0] = 'cart2'
a2 = C.addBC2Zone(a2, 'wall1', 'BCWall', 'imax')
t = C.newPyTree(['Base',a1,a2])
# Returns undefined windows (as range list since structured)
wins = C.getEmptyBC(t,2); print(wins)
#>> [[ [[11, 11, 1, 21, 1, 2], ..., [1, 11, 21, 21, 1, 2]], [[1, 1, 1, 21, 1,␣
↪2], ..., [1, 11, 21, 21, 1, 2]] ]]
# Returns undefined windows (as face indices list)
t = C.convertArray2NGon(t)
faceList = C.getEmptyBC(t,2); print(faceList)
#>> [[ [array([ 11, 230, ...], dtype=int32)], [array([  1, 221, 222, 632, ...],␣
↪dtype=int32)] ]]
C.convertPyTree2File(t, 'out.cgns')
```

Converter.PyTree.**getBCs**(*t*, *reorder=True*)

Return the BCs with their complete geometries, names and types.

Parameters

- **t** ([pyTree, base, zone, list of zones]) – input data
- **reorder** (Boolean) – if True, extracted BCs are reordered such that normals are oriented towards the interior of a.

Return type tuple (BCs, BCNames, BCTypes) where BCs is a list of BC nodes, BCNames a list of BC names and BCTypes a list of BC types.

Example of use:

- Get bcs (pyTree):

```
# - getBCs (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0),(1,1,1),(10,10,2))
a = C.addBC2Zone(a, 'overlap', 'BCOverlap', 'imin')
a = C.addBC2Zone(a, 'match1', 'BCMatch', 'jmin', a, 'jmax', [1,2,3])
a = C.fillEmptyBCWith(a, 'wall', 'BCWall', dim=2)
(BCs,BCNames,BCTypes) = C.getBCs(a)
print (BCs,BCNames,BCTypes)
```

Converter.PyTree.**recoverBCs**(*t*, *(BCs, BCNames, BCTypes)*, *tol=1.e-11*)

Recover given BCs onto a NGon tree. BCs are given by a tuple of geometries, names and types has obtained by getBCs. Exists also as in place version (_recoverBCs) that modifies t and returns None.

Parameters

- **t** ([pyTree, base, zone, list of zones]) – input NGon data
- **BCNames, BCTypes)** ((BCs,) – tuple (BCs, BCNames, BCTypes) where BCs is a list of BC nodes, BCNames a list of BC names and BCTypes a list of BC types.

Return type reference copy of t

Example of use:

- Recover boundary conditions (pyTree):

```
# - recoverBCs (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0),(1,1,1),(10,10,2))
a = C.addBC2Zone(a, 'overlap', 'BCOverlap', 'imin')
a = C.addBC2Zone(a, 'match1', 'BCMatch', 'jmin', a, 'jmax', [1,2,3])
a = C.fillEmptyBCWith(a, 'wall', 'BCWall', dim=2)
(BCs,BCNames,BCTypes) = C.getBCs(a)
b = C.convertArray2NGon(a)
C._recoverBCs(b,(BCs,BCNames,BCTypes))
C.convertPyTree2File(b,'out.cgns')
```

Converter.PyTree.**extractBCFields**(*a*, *varList=None*)

Extract fields defined at BCs of a zone *z*. If no BCDataSet is defined then a 0th-order extrapolation from interior cells is done. If a BCDataSet is defined, it has priority on the extrapolation. List of variables can be specified by the user. If not, the variables that are extracted are those defined in the FlowSolution node located at cell centers. Currently, this function works for structured and NGON zones. It returns the list of variables that could have been extracted and the indices of the face centers of the corresponding BCs.

Parameters

- **a** (zone) – input data

- **varList** (list of strings defining variables or None) – list of variables to be extracted at BCs

Return type tuple (varList, fields, indicesBC) where varList is a list of extracted variables, fields the list of numpy arrays defining extracted fields at BCs, indicesBC the numpy array of indices of BC faces.

Example of use:

- extractBCFields (pyTree):

```
# - extractBCFields (pyTree) -
import Converter.Internal as Internal
import Converter.PyTree as C
import Generator.PyTree as G

ni = 10; nj = 10; nk = 10
nfaces = (nj-1)*(nk-1)
a = G.cart((0,0,0), (1,1,1), (ni,nj,nk))
C._addBC2Zone(a, 'wall', 'BCWall', 'imin')

C._initVars(a,'centers:VelocityX={centers:CoordinateX}')
C._initVars(a,'centers:Density=1.05')

b = Internal.getNodeFromName2(a, 'wall')
d = Internal.newBCDataSet(name='BCDataSet', value='UserDefined',
                          gridLocation='FaceCenter', parent=b)
d1 = Internal.newBCData('BCNeumann', parent=d)
d = Internal.newDataArray('Density', value=nfaces*[1.], parent=d1)
d = Internal.newDataArray('MomentumX', value=nfaces*[0.3], parent=d1)

# Get data array node list
#varList=['Density','MomentumX']
varList=None
res = C.extractBCFields(a,varList=varList)
print('variables = ', res[0])
print('fields = ',  res[1])
print('indices = ', res[2])
```

Converter.PyTree.**getConnectedZones**(*a*, *topTree*, *type='all'*)

Get zones connected to a given zone a by 'BCMatch' or 'BCNearMatch' or 'all' (defined in zone GridConnectivity).

Parameters

- **a** ([pyTree, base, zone, list of zones]) – input data

- **topTree** (pyTree) – the pyTree containing a

Return type list of zones (shared with a)

Example of use:

- Get connected zones (pyTree):

```
# - getConnectedZones (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C
import Connector.PyTree as X

a = G.cart((0,0,0), (1,1,1), (11,11,1))
b = G.cart((10,0,0), (1,1,1), (11,11,1))

t = C.newPyTree(['Base',a,b])
t = X.connectMatch(t, dim=2)

zones = C.getConnectedZones(t[2][1][2][1], topTree=t)
for z in zones: print(z[0])
#>> cart
```

Converter.PyTree.**addFamily2Base**(*a*, *familyName*, *bndType=None*)

Add a family node to a base node of a tree. The family can designates a set of zone (family of zones) or a set of boundary conditions (family of BCs). If the family designates a family BC, then bndType can be defined with a CGNS BC type. This family name can then be referenced in zones or in boundary conditions.

Exists also as in place version (_addFamily2Base) that modifies a and returns None.

Parameters **a** ([pyTree, base]) – input data

Return type reference copy of a

Example of use:

- Add family to base (pyTree):

```
# - addFamily2Base (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cylinder((0,0,0), 1., 1.5, 0, 360, 1, (50,20,20))
t = C.newPyTree(['Base', a])
# Add family name referencing a BCWall BC type
C._addFamily2Base(t[2][1], 'flap', 'BCWall')
# Add just a family name
```

(continues on next page)

(continued from previous page)

```
C._addFamily2Base(t[2][1], 'component1')
C.convertPyTree2File(t, 'out.cgns')
```

Converter.PyTree.**tagWithFamily**(*a*, *familyName*, *add=False*)

Tag zones or a BC nodes with a family name. If a is a pyTree, base, zone or list of zones, family is supposed to be a family of zones. If a is a BC node or a list of BC nodes, family is supposed to be a familyBC. If add=True and a family already exists, the family is added as a AdditionalFamilyName.

Exists also as in place version (_tagWithFamily) that modifies a and returns None.

Parameters

- **a** ([pyTree, base, zone, list of zones, BC node, list of BC nodes]) – input data
- **add** (True or False) – if True, family is added otherwise it replaces an eventual existing family

Return type reference copy of a

Example of use:

- Tag zones or BC with family (pyTree):

```
# - tagWithFamily (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cylinder((0,0,0), 1., 1.5, 0., 360., 1., (80,30,2))
b = G.cart((-0.1,0.9,0), (0.01,0.01,1.), (20,20,2))
C._tagWithFamily(a, 'CYLINDER')
C._tagWithFamily(b, 'CART')

t = C.newPyTree(['Base',a,b])
C._addFamily2Base(t[2][1], 'CYLINDER')
C._addFamily2Base(t[2][1], 'CART')

C.convertPyTree2File(t, 'out.cgns')
```

Converter.PyTree.**getFamilyZones**(*a*, *familyName*)

Get all zones of given family (family of zones).

Parameters **a** ([pyTree, base, zone, list of zones]) – input data

Return type list of zones (shared with a)

Example of use:

- Get family zones (pyTree):

```
# - getFamilyZones (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cylinder((0,0,0), 1., 1.5, 0., 360., 1., (80,30,2))
b = G.cylinder((3,0,0), 1., 1.5, 0., 360., 1., (80,30,2))
c = G.cart((-0.1,0.9,0), (0.01,0.01,1.), (20,20,2))

C._tagWithFamily(a, 'CYLINDER')
C._tagWithFamily(b, 'CYLINDER')
C._tagWithFamily(c, 'CART')

t = C.newPyTree(['Base',a,b,c])
C._addFamily2Base(t[2][1], 'CYLINDER')
C._addFamily2Base(t[2][1], 'CART')
zones = C.getFamilyZones(t, 'CYLINDER')
for z in zones: print(z[0])
#>> cylinder cylinder.0
```

Converter.PyTree.**getFamilyBCs**(*a*, *familyName*)

Get all BC nodes corresponding to a given familyName (family of BCs).

Parameters **a** ([pyTree, base, zone, list of zones]) – input data

Return type list of BC nodes (shared with a)

Example of use:

- Get family BC nodes (pyTree):

```
# - getFamilyBCs (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal

a = G.cart((0.,0.,0), (0.01,0.01,1.), (20,20,2))
b = G.cart((1.,0.,0), (0.01,0.01,1.), (20,20,2))

a = C.addBC2Zone(a, 'walla', 'FamilySpecified:CARTER', 'imin')
b = C.addBC2Zone(b, 'wallb', 'FamilySpecified:CARTER', 'jmin')

t = C.newPyTree(['Base',a,b])
C._addFamily2Base(t[2][1], 'CARTER', bndType='BCWall')
B1 = C.getFamilyBCs(t, 'CARTER'); Internal.printTree(B1)
```

(continues on next page)

(continued from previous page)

```
#>> ['walla',array('FamilySpecified',dtype='|S1'),[2 sons],'BC_t']
#>>   |_['PointRange',array(shape=(3, 2),dtype='int32',order='F'),[0 son],
↪'IndexRange_t']
#>>   |_['FamilyName',array('CARTER',dtype='|S1'),[0 son],'FamilyName_t']
#>> ['wallb',array('FamilySpecified',dtype='|S1'),[2 sons],'BC_t']
#>>   |_['PointRange',array(shape=(3, 2),dtype='int32',order='F'),[0 son],
↪'IndexRange_t']
#>>   |_['FamilyName',array('CARTER',dtype='|S1'),[0 son],'FamilyName_t']
```

Converter.PyTree.**getFamilyZoneNames**(*a*)

Return all family zone names defined in a.

Parameters **a** ([pyTree, base]) – input data

Return type list of familyZone names

Example of use:

- Get familyZone names (pyTree):

```
# - getFamilyZoneNames (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0.,0.,0), (0.01,0.01,1.), (20,20,2))
b = G.cart((1.,0.,0), (0.01,0.01,1.), (20,20,2))
C._tagWithFamily(a, 'CARTER')
C._tagWithFamily(b, 'CARTER')

t = C.newPyTree(['Base',a,b])
C._addFamily2Base(t[2][1], 'CARTER')

# Toutes les family zone names de l'arbre
names = C.getFamilyZoneNames(t); print(names)
#>> ['CARTER']
```

Converter.PyTree.**getFamilyBCNamesOfType**(*a*, *bndType=None*)

Return all family BC names of a given type. If type is None, return all family BC names.

Parameters **a** ([pyTree, base]) – input data

Return type list of familyBC names

Example of use:

- Get familyBC names of given type (pyTree):

```
# - getFamilyBCNamesOfType (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0.,0.,0), (0.01,0.01,1.), (20,20,2))
b = G.cart((1.,0.,0), (0.01,0.01,1.), (20,20,2))

a = C.addBC2Zone(a, 'walla', 'FamilySpecified:CARTER', 'imin')
b = C.addBC2Zone(b, 'wallb', 'FamilySpecified:CARTER', 'jmin')

t = C.newPyTree(['Base',a,b])

C._addFamily2Base(t[2][1], 'CARTER', bndType='BCWall')

# Toutes les familyBCs de type BCwall
names = C.getFamilyBCNamesOfType(t, 'BCWall'); print(names)
#>> ['CARTER']
# Toutes les familyBCs de l'arbre
names = C.getFamilyBCNamesOfType(t); print(names)
#>> ['CARTER']
```

Converter.PyTree.**getFamilyBCNamesDict**(*a*)

Return all family BC names contained in a as a dictionary 'familyName', 'BCType'. The dictionary is dict['familyName'] = 'BCType'.

Parameters **a** ([pyTree, base]) – input data

Return type dictionary of familyBC names with their type

Example of use:

- Get familyBC names as a dictionary (pyTree):

```
# - getFamilyBCNamesDict (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0.,0.,0), (0.01,0.01,1.), (20,20,2))
b = G.cart((1.,0.,0), (0.01,0.01,1.), (20,20,2))

a = C.addBC2Zone(a, 'walla', 'FamilySpecified:CARTER', 'imin')
b = C.addBC2Zone(b, 'wallb', 'FamilySpecified:CARTER', 'jmin')

t = C.newPyTree(['Base',a,b])

C._addFamily2Base(t[2][1], 'CARTER', bndType='BCWall')
```

(continues on next page)

(continued from previous page)

```
# Toutes les familyBCs
dict = C.getFamilyBCNamesDict(t); print(dict)
#>> {'CARTER': 'BCWall'}
```

Converter.PyTree.**getValue**(*a*, *var*, *ind*)

Return the field value(s) defined in a zone a for point of index ind (for both structured and unstructured zones). For structured zones, you can specify (i,j,k) instead of ind. For unstructured zones, the index ind corresponds to the location type of point defining zone a. For instance, if a describes a field at element vertices, ind is a vertex index. var is the name of the field variable or a list of field variables or a container name. Variable name can be preceded with 'centers:' or 'nodes:'. This routine is slow and must not be used to access all points of a zone. In this case, it is better to access the field numpy with Internal.getNodeFromName for example.

Parameters

- **a** (zone) – input zone
- **var** (string) – field name
- **ind** (int or tuple of ints) – index

Return type float or list of floats

Example of use:

- Get field value (pyTree):

```
# - getValue (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

# Structured array
Ni = 40; Nj = 50; Nk = 20
a = G.cart((0,0,0), (1./(Ni-1), 0.5/(Nj-1),1./(Nk-1)), (Ni,Nj,Nk))
# Get variable values contained in a in point (10,1,1)
print(C.getValue( a, 'CoordinateX', (10,1,1) ))
#>> 0.230769230769
print(C.getValue( a, 'CoordinateX', 9 )) # It's the same point
#>> 0.230769230769
print(C.getValue( a, 'nodes:CoordinateX', 9 )) # It's the same point
#>> 0.230769230769
print(C.getValue( a, 'GridCoordinates', 9 )) # return [x,y,z]
#>> [0.23076923076923075, 0.0, 0.0]
print(C.getValue( a, ['CoordinateX', 'CoordinateY'], 9 )) # return [x,y]
```

(continues on next page)

(continued from previous page)

```
#>> [0.23076923076923075, 0.0]
# Unstructured array
Ni = 40; Nj = 50; Nk = 20
a = G.cartTetra((0,0,0), (1./(Ni-1), 0.5/(Nj-1),1./(Nk-1)), (Ni,Nj,Nk))
print(C.getValue( a, 'CoordinateX', 9 ))
#>> 0.230769230769
```

Converter.PyTree.**setValue**(*a*, *var*, *ind*, *value*)

Set the values of one point of index ind in a zone a. var is the name of the field variable or a container name. Variable name can be preceded with 'centers:' or 'nodes:'. value can be a float or a list of floats corresponding to the values of the variables to be modified. This routine is slow and must not be used to access all points of a zone. In this case, it is better to use setPartialFields.

Parameters

- **a** (zone) – input zone
- **var** (string) – field name
- **ind** (int or tuple of ints) – index

Example of use:

- Set field value (pyTree):

```
# - setValue (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

# Structured array
Ni = 40; Nj = 50; Nk = 20
a = G.cart((0,0,0), (1./(Ni-1), 0.5/(Nj-1),1./(Nk-1)), (Ni,Nj,Nk))
C.setValue(a, 'CoordinateX', (10,1,1), 0.25)
C.setValue(a, 'GridCoordinates', (11,1,1), [0.3,0.2,0.1]); print(a)

# Unstructured array
Ni = 40; Nj = 50; Nk = 20
a = G.cartTetra((0,0,0), (1./(Ni-1), 0.5/(Nj-1),1./(Nk-1)), (Ni,Nj,Nk))
C.setValue(a, 'CoordinateX', 9, 0.1 ); print(a)
```

Converter.PyTree.**setPartialFields**(*a*, *F*, *I*, *loc='nodes'*, *startFrom=0*)

Set the values for a given list of indices. Field values are given as a list of arrays in F (one array for each zone), indices are given as a list of numpys in I (one numpy for each zone), loc can be 'nodes' or 'centers'.

Exists also as in place version (_setPartialFields) that modifies a and returns None.

Parameters

- **a** ([pyTree, base, zone, list of zones]) – input data
- **F** (list of arrays) – list of arrays of fields values
- **I** (list of numpys) – list of indices
- **loc** ('centers' or 'nodes') – location of fields in zone
- **startFrom** (integer) – starting indice of I (e.g. 0 or 1)

Return type reference copy of input

Example of use:

- Set values for a list of indices (pyTree):

```
# - setPartialFields (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C
import Converter
import numpy

a = G.cart((0,0,0), (1,1,1), (10,10,10))
a = C.initVars(a, 'F', 2.)
f1 = Converter.array('F', 5,1,1)
f1[1][:] = 1.
inds = numpy.array([0,1,2], dtype=numpy.int32)
b = C.setPartialFields(a, [f1], [inds], loc='nodes')
t = C.newPyTree(['Base',b])
C.convertPyTree2File(t, 'out.cgns')
```

Converter.PyTree.**addVars**(*a*, *vars*)

Add given variables. Variables are added to the flow container as described by Internal.__FlowSolutionNodes__ or Internal.__FlowSolutionCenters__. Prefix the variable names with 'centers:' or 'nodes:' to specify variable location. Exists also as in place version (_addVars) that modifies a and returns None.

Parameters

- **a** ([pyTree, base, list of zones]) – input data
- **vars** (list of strings) – list of variable names to add.

Return type reference copy of intput

Example of use:

- Add variables (pyTree):

```
# - addVars (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0), (1,1,1), (10,10,11))
a = C.addVars(a, 'rou')
a = C.addVars(a, 'centers:cellN')
a = C.addVars(a, ['Density', 'Hx', 'centers:Hy'])
t = C.newPyTree(['Base',a])
C.convertPyTree2File(t, 'out.cgns')
```

Converter.PyTree.**fillMissingVariables**(*a*)

Add missing variables and reorder variables for all zones, such that all zones have the same variables at the end.

Exists also as in place version (_fillMissingVariables) that modifies a and returns None.

Parameters **a** ([pyTree, base, list of zones]) – input data

Return type reference copy of intput

Example of use:

- Fill missing variables (pyTree):

```
# - fillMissingVariables (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0), (1,1,1), (10,10,11)); a[0] = 'cart1'
b = G.cart((1,0,0), (2,1,1), (10,10,11)); b[0] = 'cart2'
C._addVars(a, 'rou'); C._addVars(a, 'rov')
C._addVars(a, 'centers:cellN')
C._addVars(a, ['Density', 'Hx', 'centers:Hy'])

t = C.newPyTree(['Base',a,b])
t = C.fillMissingVariables(t)
C.convertPyTree2File(t, 'out.cgns')
```

Converter.PyTree.**cpVars**(*a1*, *var1*, *a2*, *var2*)

Copy a variable from zone a1, with name var1, to zone a2, with name var2. The var location must be coherent. a1 and a2 can be identical.

Exists also as in place version (_cpVars) that modifies a2 and returns None.

Parameters

- **a1** (zone node) – input zone 1
- **var1** (string) – variable name (can be preceded of 'centers:' or 'nodes:')
- **a2** (zone node) – receiver zone 2
- **var2** (string) – variable name (can be preceded of 'centers:' or 'nodes:')

Return type reference copy of a2

Example of use:

- Variables copy (pyTree):

```
# - cpVars (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
a = G.cart((0,0,0),(1,1,1),(10,10,10)); a[0] = 'cart1'
b = G.cart((0,0,0),(1,1,1),(10,10,10)); b[0] = 'cart2'
C._initVars(a, 'Density', 2.)
C._initVars(b, 'centers:H', 4.)
a = C.cpVars(a, 'Density', a, 'G') # copy in a
C._cpVars(a, 'Density', b, 'Density') # copy from a to b
a = C.cpVars(b, 'centers:H', a, 'centers:H') # copy from b to a
t = C.newPyTree(['Base',a,b])
C.convertPyTree2File(t, 'out.cgns')
```

4.3 Array / PyTree common manipulations

Converter.**getVarNames**(*a*, *excludeXYZ=False*, *loc='both'*)

Return the list of variable names contained in a. Localization of variables can be specified ('nodes', 'centers', 'both'). Coordinates can be excluded. Only containers defined in Internal.__GridCoordinates__, Internal.__FlowSolutionNodes__ and Internal.__FlowSolutionCenters__ are scanned.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **excludeXYZ** – if True, Coordinates are not scanned
- **loc** ('nodes', 'centers', 'both') – variable localisations

Return type list of field names (one for each zone of a)

Example of use:

- Get variable names (array):

```
# - getVarNames (array) -
import Converter as C
a = C.array('x,y,z,ro', 12, 9, 12)
print(C.getVarNames(a))
#>> ['x', 'y', 'z', 'ro']
```

- Get variable names (pyTree):

```
# - getVarNames (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
a = G.cart((0,0,0),(1,1,1),(10,10,10))
C._addVars(a, ['Density', 'centers:cellN'])
# one zone
print(C.getVarNames(a, loc='nodes'))
#>> [['CoordinateX', 'CoordinateY', 'CoordinateZ', 'Density']]
print(C.getVarNames(a, loc='centers'))
#>> [['CoordinateX', 'CoordinateY', 'CoordinateZ', 'centers:cellN']]
print(C.getVarNames(a, excludeXYZ=True, loc='both'))
#>> [['Density', 'centers:cellN']]
```

Converter.**isNamePresent**(*a*, *varName*)

Return -1 if a doesn't contain field varName, 0 if at least one zone in a contains varName, 1 if all zones in a contain varName.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **varName** (string) – variable name (can be preceded by 'nodes:' or 'centers:')

Return type -1, 0, 1

Example of use:

- Is variable present (array):

```
# - isNamePresent (array) -
import Converter as C
import Generator as G

a = G.cart((0,0,0), (1,1,1), (50,50,50))
C._initVars(a, 'F', 1.)
```

(continues on next page)

(continued from previous page)

```
b = G.cart((0,0,0), (1,1,1), (50,50,50))
C._initVars(b, 'centers:G', 2.)

print(C.getVarNames([a, b]))
#>> [['x', 'y', 'z', 'F'], ['x', 'y', 'z', 'G']]

print(C.isNamePresent(a, 'F'))
#>> 1
print(C.isNamePresent([a, b], 'F'))
#>> 0
print(C.isNamePresent([a, b], 'K'))
#>> -1
```

- Is variable present (pyTree):

```
# - isNamePresent (PyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0), (1,1,1), (50,50,50))
C._initVars(a, 'F', 1.)
C._initVars(a, 'centers:G', 0.)

b = G.cart((0,0,0), (1,1,1), (50,50,50))
C._initVars(b, 'F', 2.)
C._initVars(b, 'centers:H', 3.)

t = C.newPyTree(['Base',a,b])

print(C.getVarNames([a, b]))
#>> [['CoordinateX', 'CoordinateY', 'CoordinateZ', 'F', 'centers:G'], [
↪'CoordinateX', 'CoordinateY', 'CoordinateZ', 'F', 'centers:H']]

print(C.isNamePresent(a, 'F'))
#>> 1
print(C.isNamePresent(a, 'centers:F'))
#>> -1
print(C.isNamePresent([a, b], 'F'))
#>> 1
print(C.isNamePresent([a, b], 'centers:G'))
#>> 0
```

Converter.**getNPts**(*a*)

Return the total number of points in a.

Parameters **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data

Return type int

Example of use:

- Get number of points (array):

```
# - getNPts (array) -
import Converter as C
import Generator as G

a = G.cart((0,0,0), (1,1,1), (10,10,11))
npts = C.getNPts(a); print(npts)
#>> 1100
```

- Get number of points (pyTree):

```
# - getNPts (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0), (1,1,1), (10,10,11))
npts = C.getNPts(a); print(npts)
#>> 1100
```

Converter.**getNCells**(*a*)

Return the total number of cells in a.

Parameters **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data

Return type int

Example of use:

- Get number of cells (array):

```
# - getNCells (array) -
import Converter as C
import Generator as G

a = G.cart((0,0,0), (1,1,1), (10,10,11))
```

(continues on next page)

(continued from previous page)

```
ncells = C.getNCells(a); print(ncells)
#>> 810
```

- Get number of cells (pyTree):

```
# - getNCells (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0), (1,1,1), (10,10,11))
ncells = C.getNCells(a); print(ncells)
#>> 810
```

Converter.**initVars**(*a*, *varNameString*, *value*)

Initialize a variable given by a string to a constant value, or by using a string formula or by using an external function.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **varNameString** (string) – string describing variable or formula
- **value** (float or function and parameters) – value in case of constant init or function.

Return type identical to input

Example of use:

- Init a variable to a constant value (array):

```
# - initVars (array) -
import Converter as C
a = C.array('x,y,z', 10, 10, 10)
a = C.initVars(a, 'celln', 2.)
C.convertArrays2File(a, 'out.plt')
```

- Init a variable with a function (array):

```
# - initVars (array) -
import Converter as C
import Generator as G

```

(continues on next page)

(continued from previous page)

```
# Create a function
def F(x1, x2): return 3.*x1+2.*x2

a = G.cart((0,0,0), (1,1,1), (11,11,1))
a = C.initVars(a, 'F', F, ['x','y'])
C.convertArrays2File(a, "out.plt")
```

- Init a variable with a formula (array):

```
# - initVars (array) -
import Converter as C
import Generator as G
a = G.cart((0,0,0), (1,1,1), (10,10,10))
b = C.initVars(a, '{Density} = 3 * {x} + sin({y})')
C.convertArrays2File(b, 'out.plt')
```

Note: When initializing variables using string formulas, functions correspond to the numpy library.

- Init a variable to a constant value (pyTree):

```
# - initVars (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0), (1,1,1), (10,10,10))
a = C.initVars(a, 'F', 0.)
a = C.initVars(a, 'centers:G', 1.)
C.convertPyTree2File(a, 'out.cgns')
```

- Init a variable with a function (pyTree):

```
# - initVars (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0), (1,1,1), (10,10,10))

# Init from a function
def F(x1, x2): return 3.*x1+2.*x2
```

(continues on next page)

(continued from previous page)

```
a = C.initVars(a, 'Density', F, ['CoordinateX','CoordinateY'])
a = C.initVars(a, 'centers:F', F, ['centers:CoordinateX','centers:CoordinateY'])
C.convertPyTree2File(a, 'out.cgns')
```

- Init a variable with a formula (pyTree):

```
# - initVars (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
a = G.cart((0,0,0), (1,1,1), (10,10,10))
a = C.initVars(a, '{Density} = 3 * {CoordinateX} + sin({CoordinateY})')
a = C.initVars(a, '{centers:MomentumX} = 3 * {centers:CoordinateX} + sin(
↪{centers:CoordinateY})')
C.convertPyTree2File(a, 'out.cgns')
```

Converter.**extractVars**(*a*, *varNames*)

Extract variables defined in varNames from a (other variables are removed).

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **varNames** (string or list of strings.) – names of variable to extract (can starts with 'nodes:' or 'centers:")

Return type Identical to input

Example of use:

- Extract some variables from array (array):

```
# - extractVars (array) -
import Generator as G
import Converter as C

a = G.cart((0,0,0), (1,1,1), (10,10,10))
a = C.initVars(a, 'F', 2.)
# Var defined by a string
r = C.extractVars(a, 'F')
# Vars defined by a list
r = C.extractVars(a, ['x','y'])
C.convertArrays2File(r, 'out.plt')
```

- Extract some variables from zone (array):

```
# - extractVars (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C

a = G.cart((0,0,0), (1,1,1), (10,10,10))
a = C.addVars(a, ['F', 'G', 'centers:H'])
# Keep only F
a = C.extractVars(a, ['F'])
C.convertPyTree2File(a, 'out.cgns')
```

Converter.**rmVars**(*a*, *varNames*)

Remove variable(s) from a. varNames is a string name or a list of string names.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **varNames** (string or list of strings.) – names of variable to remove (can starts with 'nodes:' or 'centers:")

Return type Identical to input

Example of use:

- Remove some variables from array (array):

```
# - rmVars (array) -
import Converter as C
import Generator as G
a = G.cart((0,0,0),(1,1,1),(10,10,10))
b = C.addVars(a, 'Density')
b = C.addVars(a, 'Alpha')
b = C.rmVars(b, 'Density')
C.convertArrays2File(b, 'out.plt')
```

- Remove some variables from zone (array):

```
# - rmVars (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
a = G.cart((0,0,0),(1,1,1),(10,10,10))
b = C.addVars(a, ['MomentumX','centers:Density'])
b = C.rmVars(b, ['MomentumX','centers:Density'])
C.convertPyTree2File(b, 'out.cgns')
```

Converter.**convertArray2Tetra**(*a*, *split='simple'*)

Create tetra unstructured array from an any type of mesh. 2D elements are made triangular, else they are made tetrahedral. If split='simple', conversion does not create new points. If split='withBarycenters', barycenters of elements and faces are added.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **split** (string) – 'simple' or 'withBarycenters'

Return type Identical to input

Example of use:

- Convert structured mesh to tetra (array):

```
# - convertArray2Tetra (array) -
import Converter as C
import Generator as G

# 2D: triangles
a = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,10,1))
b = C.convertArray2Tetra(a)
C.convertArrays2File(b, 'out1.plt')

# 3D: tetrahedras
a = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,10,10))
b = C.convertArray2Tetra(a)
C.convertArrays2File(b, 'out2.plt')
```

- Convert hexa mesh to tetra (array):

```
# - convertArray2Tetra (array) -
import Converter as C
import Generator as G

# 2D: quads -> triangles
a = G.cartHexa((0.,0.,0.), (0.1,0.1,0.2), (10,10,1))
b = C.convertArray2Tetra(a)
C.convertArrays2File(b, 'out1.plt')

# 3D: hexa -> tetrahedra
a = G.cartHexa((0.,0.,0.), (0.1,0.1,0.2), (10,10,10))
b = C.convertArray2Tetra(a)
C.convertArrays2File(b, 'out2.plt')
```

- Convert structured mesh to tetra (pyTree):

```
# - convertArray2Tetra (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

# 2D : triangles
a = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,10,1))
b = C.convertArray2Tetra(a)

# 3D : tetrahedras
a = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,10,10))
b = C.convertArray2Tetra(a)
C.convertPyTree2File([a,b], 'out.cgns')
```

Converter.**convertArray2Hexa**(*a*)

Create hexa unstructured array from an any type of mesh. 2D elements are made quadrangular, else they are made hexahedral.

Parameters **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data

Return type Identical to input

Example of use:

- Convert structured mesh to hexa (array):

```
# - convertArray2Hexa (array) -
import Converter as C
import Generator as G

# 2D: quad
a = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,10,1))
a = C.convertArray2Hexa(a)

# 3D: hexa
b = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,10,10))
b = C.convertArray2Hexa(b)
C.convertArrays2File([a,b], 'out.plt')
```

- Convert structured mesh to hexa (pyTree):

```
# - convertArray2Hexa (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
```

(continues on next page)

(continued from previous page)

```
# 2D: quad
a = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,10,1))
b = C.convertArray2Hexa(a)

# 3D: hexa
a = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,10,10))
b = C.convertArray2Hexa(a)

C.convertPyTree2File([a,b], 'out.cgns')
```

Converter.**convertArray2NGon**(*a*, *recoverBC=1*)

Create NGON array from an any type of mesh.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **recoverBC** (integer (0 or 1)) – BCs can be recovered (=1) or not(=0) on the NGON a (not valid for arrays).

Return type Identical to input

Example of use:

- Convert structured mesh to NGON (array):

```
# - convertArray2NGon(array) -
import Converter as C
import Generator as G

a = G.cartTetra((0.,0.,0.), (0.1,0.1,0.2), (2,2,1))
b = C.convertArray2NGon(a)
C.convertArrays2File(b, 'out.plt')
```

- Convert structured mesh to NGON (pyTree):

```
# - convertArray2NGon(pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cartTetra((0.,0.,0.), (0.1,0.1,0.2), (2,2,1))
b = C.convertArray2NGon(a)
C.convertPyTree2File(b, 'out.cgns')
```

Converter.**convertArray2Node**(*a*)

Create NODE array from an any type of mesh. A node array only contains node and no connectivity.

Parameters **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data

Return type Identical to input

Example of use:

- Convert structured mesh to NODE (array):

```
# - convertArray2Node (array) -
import Converter as C
import Generator as G

a = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,10,1))
b = C.convertArray2Node(a)
C.convertArrays2File([b], 'out.plt')
```

- Convert structured mesh to NODE (pyTree):

```
# - convertArray2Node (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,10,1))
a = C.convertArray2Node(a)
C.convertPyTree2File(a, 'out.cgns')
```

Converter.**convertBAR2Struct**(*a*)

Create a structured 1D array from a BAR array. The BAR array must not contain branches. To split a branched BAR, you may consider T.splitTBranches.

Parameters **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data (BAR)

Return type Identical to input

Example of use:

- Convert BAR to i-array (array):

```
# - convertBAR2Struct (array) -
import Converter as C
import Generator as G
import Geom as D
a = D.circle((0.,0.,0.),1.)
a = C.convertArray2Hexa(a); a = G.close(a)
b = C.convertBAR2Struct(a)
C.convertArrays2File([a,b], 'out.plt')
```

- Convert BAR to i-array (pyTree):

```
# - convertBAR2Struct (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Geom.PyTree as D
a = D.circle((0.,0.,0.),1.)
a = C.convertArray2Hexa(a); a = G.close(a)
b = C.convertBAR2Struct(a)
C.convertPyTree2File(b,'out.cgns')
```

Converter.**convertTri2Quad**(*a*, *alpha=30.*)

Convert a TRI-array to a QUAD-array. Neighbouring cells with an angle lower than alpha can be merged. It returns the QUAD-array b and the rest of not merged cells in a TRI-array c.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data (TRI)
- **alpha** (float) – angle for merging

Return type Identical to input

Example of use:

- Convert TRI to QUAD-array (array):

```
# - convertTri2Quad (array) -
import Converter as C
import Generator as G

a = G.cartTetra((0.,0.,0.), (0.1,0.1,0.2), (10,10,1))
a, b = C.convertTri2Quad(a, 30.)
C.convertArrays2File([a,b], 'out.plt')
```

- Convert TRI to QUAD-array (pyTree):

```
# - convertTri2Quad (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cartTetra((0.,0.,0.), (0.1,0.1,0.2), (10,10,1))
a, b = C.convertTri2Quad(a, 30.)
C.convertPyTree2File([a,b], 'out.cgns')
```

Converter.**convertHO2LO**(*a*, *mode=0*)

Convert a high order element mesh into a low order (linear) mesh. If mode=1, only valid for BAR_3, TRI_6, QUAD_8, QUAD_9, TETRA_10, HEXA_20, HEXA_27, PENTA_18, PYRA_14.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **mode** (int) – if 0, coarse extraction, 1, tesselate all points

Return type Identical to input

Example of use:

- Convert High order mesh to Low order mesh (array):

```
# - convertHO2LO (array) -
import Converter as C
import Geom as D

a = D.triangle((0,0,0), (1,0,0), (1,1,0))
a = C.convertLO2HO(a, mode=0)
a = C.convertHO2LO(a, mode=0)
C.convertArrays2File(a, 'out.plt')
```

- Convert High order to Low order mesh (pyTree):

```
# - convertHO2LO (pyTree) -
import Converter.PyTree as C
import Geom.PyTree as D

a = D.triangle((0,0,0), (1,0,0), (1,1,0))
a = C.convertLO2HO(a, mode=0)
a = C.convertHO2LO(a, mode=0)
C.convertPyTree2File(a, 'out.cgns')
```

Converter.**convertLO2HO**(*a*, *mode=0*, *order=2*)

Convert a low order element mesh into a high order mesh. Points are added linearly on edges or faces. Order 2 can give: BAR_3, TRI_6, QUAD_8, QUAD_9, TETRA_10, HEXA_20, HEXA_27, PENTA_18, PYRA_14. Order 3 can give: BAR_4, TRI_9, ...

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **mode** (int) – specify the type of generated high order elements
- **order** (int) – specify the order of output elements

Return type Identical to input

Example of use:

- Convert Low order mesh to High order mesh (array):

```
# - convertLO2HO (array) -
import Converter as C
import Geom as D

a = D.triangle((0,0,0), (1,0,0), (1,1,0))
a = C.convertLO2HO(a, mode=0)
# save to add when ready
```

- Convert Low order to High order mesh (pyTree):

```
# - convertLO2HO (pyTree) -
import Converter.PyTree as C
import Geom.PyTree as D

a = D.triangle((0,0,0), (1,0,0), (1,1,0))
a = C.convertLO2HO(a, mode=0)
C.convertPyTree2File(a, 'out.cgns')
```

Converter.**conformizeNGon**(*a*, *tol=1.e-6*)

Conformize the cell faces of a NGon, such that a face of a cell corresponds to a unique face of another cell. Typically, a mesh with hanging nodes will be made conform.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data (NGON)
- **tol** (float) – tolerance for face matching

Return type Identical to input

Example of use:

- NGON mesh conformization (array):

```
# - conformizeNGon (array) -
import Generator as G
import Converter as C
import Transform as T
a = G.cartNGon((0,0,0),(0.1,0.1,1),(11,11,1))
b = G.cartNGon((1.,0,0),(0.1,0.2,1),(11,6,1))
a = G.cartNGon((0,0,0),(1,1,1),(3,3,1))
b = G.cartNGon((2.,0,0),(2,2,1),(2,2,1))
res = T.join(a,b)
res2 = C.conformizeNGon(res)
C.convertArrays2File(res2, 'out.plt')
```

- NGON mesh conformization (pyTree):

```
# - conformizeNGon (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C
import Transform.PyTree as T

a = G.cartNGon((0,0,0),(0.1,0.1,1),(11,11,1))
b = G.cartNGon((1.,0,0),(0.1,0.2,1),(11,6,1))
res = T.join(a,b)
res = C.conformizeNGon(res)
C.convertPyTree2File(res, 'out.cgns')
```

Converter.**node2Center**(*a*, *var=''*, *accurate=0*)

Change data location from nodes to centers. If no variable is specified, the mesh coordinates are also put to centers, resulting in a "all in nodes" mesh. If a variable is specified, only this variable is passed to centers and stored in __FlowSolutionCenters__ container.

Parameters

- **a** (`[array, list of arrays]` or `[pyTree, base, zone, list of zones]`) – input data
- **var** (`string or list of strings or container name`) – variables to modify
- **accurate** (`integer (0 or 1)`) – the center can be computed by sorting the vertices of the cell (accurate=1)

Return type Identical to input

Example of use:

- Nodes to centers conversion (array):

```
# - node2Center (array) -
import Converter as C
import Generator as G

a = G.cart((0,0,0), (1,1,1), (30,40,1))
a = C.initVars(a, '{ro}=2*{x}+{y}')
ac = C.node2Center(a)
C.convertArrays2File([a,ac], "out.plt")
```

- Nodes to centers conversion (pyTree):

```
# - node2Center (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0), (1,1,1), (30,40,1))
a = C.initVars(a, '{Density}=2*{CoordinateX}+{CoordinateY}')

# node2Center: passe une variable en centres (dans la meme zone)
a = C.node2Center(a, 'Density')
C.convertPyTree2File(a, 'out1.cgns')

# node2Center: cree une nouvelle zone contenant les centres
a = G.cart((0,0,0), (1,1,1), (30,40,1))
a = C.initVars(a, '{Density}=2*{CoordinateX}+{CoordinateY}')
b = C.node2Center(a); b[0] = a[0]+'_centers'
C.convertPyTree2File([a,b], 'out2.cgns')
```

Converter.**center2Node**(*a*, *var=''*, *cellNType=0*)

Change data location from centers to nodes. If no variable is specified, the mesh coordinates are also put to nodes, resulting in a "all in nodes" mesh. If a variable is specified, only this variable is passed to nodes and stored in __FlowSolutionNodes__ container. cellNType indicates the treatement for blanked points when cellN field is present. cellNType=0, means that, if a node receives at least one cellN=0 value from a center, its cellN is set to 0. cellNType=1 means that, only if all values of neighbouring centers are cellN=0, its cellN is set to 0.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data

- **var** (string or list of strings or container name) – variables to modify
- **cellNType** (int) – describes the type of treatment for cellN variables.

Return type Identical to input

Example of use:

- Centers to nodes conversion (array):

```
# - center2Node (array) -
import Converter as C
import Generator as G

a = G.cart((0,0,0), (1,1,1), (30,40,1))
a = C.initVars(a, 'ro', 1.)
an = C.center2Node(a)
C.convertArrays2File(an, "out.plt")
```

- Centers to nodes conversion (pyTree):

```
# - center2Node (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

# center2Node: create a new zone
a = G.cart((0,0,0), (1,1,1), (30,40,1))
C._initVars(a, 'centers:Density', 1.)
b = C.center2Node(a); b[0] = a[0]+'_nodes'
C.convertPyTree2File(b, 'out0.cgns')

# center2Node: modify a variable
a = G.cart((0,0,0), (1,1,1), (30,40,1))
C._initVars(a, 'centers:Density', 1.)
a = C.center2Node(a, 'centers:Density')

# center2Node: modify a container
a = C.center2Node(a, 'FlowSolution#Centers')

C.convertPyTree2File(a, 'out.cgns')
```

4.4 Array / PyTree analysis

Converter.**diffArrays**(*a*, *b*, *removeCoordinates=True*)

Given a solution in a and a solution in b, both defined on the same mesh, return the differences.

Parameters

- **a** ([list of arrays] or [pyTree, base, zone, list of zones]) – input data 1
- **b** ([list of arrays] or [pyTree, base, zone, list of zones]) – input data 2
- **removeCoordinates** (boolean) – if True, remove original coordinates (pyTree)

Return type Identical to input 1

Example of use:

- Difference between two solutions (array):

```
# - diffArrays (array) -
import Converter as C
import Generator as G

a = G.cart((0,0,0), (1,1,1), (11,11,11))
a = C.initVars(a, "F", 1.)
a = C.initVars(a, "Q", 1.2)

b = G.cart((0,0,0), (1,1,1), (11,11,11))
b = C.initVars(b, "Q", 2.)
b = C.initVars(b, "F", 3.)

ret = C.diffArrays([a], [b])
C.convertArrays2File(ret, 'out.plt')
```

- Difference between two solutions (pyTree):

```
# - diffArrays (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0), (1,1,1), (11,11,11))
a1 = C.initVars(a, "F", 1.); a1 = C.initVars(a1, "centers:Q", 1.2)
a2 = C.initVars(a, "F", 3.); a2 = C.initVars(a2, "centers:Q", 2.)
```

(continues on next page)

(continued from previous page)

```
ret = C.diffArrays(a1, a2)
C.convertPyTree2File(ret, 'out.cgns')
```

Converter.**getMinValue**(*a*, *var*)

Return the minimum value of field 'var' on input.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **var** (string) – variable name

Return type minimum value

Example of use:

- Get the minimum of F (array):

```
# - getMinValue (array) -
import Converter as C
import Generator as G

a = G.cart((0,0,0), (1.,1.,1.), (11,1,1))
minval = C.getMinValue(a, 'x'); print(minval)
#>> 0.0
```

- Get the minimum of F (pyTree):

```
# - getMinValue (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0), (1.,1.,1.), (11,1,1))
C._initVars(a, '{centers:F}={centers:CoordinateX}')
minval = C.getMinValue(a, 'CoordinateX'); print(minval)
#>> 0.0
minval = C.getMinValue(a, 'centers:F'); print(minval)
#>> 0.5
minval = C.getMinValue(a, ['CoordinateX', 'CoordinateY']); print(minval)
#>> [0.0, 0.0]
minval = C.getMinValue(a, 'GridCoordinates'); print(minval)
#>> [0.0, 0.0, 0.0]
```

Converter.**getMaxValue**(*a*, *var*)

Return the maximum value of field 'var' on input.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **var** (string) – variable name

Return type maximum value

Example of use:

- Get the maximum of F (array):

```
# - getMaxValue (array) -
import Converter as C
import Generator as G

a = G.cart((0,0,0), (1.,1.,1.), (11,1,1))
maxval = C.getMaxValue(a, 'x'); print(maxval)
#>> 10.0
```

- Get the maximum of F (pyTree):

```
# - getMaxValue (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0), (1.,1.,1.), (11,2,2))
C._initVars(a, '{centers:F}={centers:CoordinateX}')
maxval = C.getMaxValue(a, 'CoordinateX'); print(maxval)
#>> 10.0
maxval = C.getMaxValue(a, 'centers:F'); print(maxval)
#>> 9.5
maxval = C.getMaxValue(a, ['CoordinateX', 'CoordinateY']); print(maxval)
#>> [10.0, 1.0]
maxval = C.getMaxValue(a, 'GridCoordinates'); print(maxval)
#>> [10.0, 1.0, 1.0]
```

Converter.**getMeanValue**(*a*, *var*)

Return the mean value of field 'var' on input.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data

- **var** (string) – variable name

Return type mean value

Example of use:

- Get the mean of F (array):

```
# - getMeanValue (array) -
import Converter as C
import Generator as G

a = G.cart((0,0,0), (1.,1.,1.), (11,1,1))
meanval = C.getMeanValue(a, 'x'); print(meanval)
#>> 5.0
```

- Get the mean of F (pyTree):

```
# - getMeanValue (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0), (1.,1.,1.), (11,1,1))
meanval = C.getMeanValue(a, 'CoordinateX'); print(meanval)
#>> 5.0
```

Converter.**getMeanRangeValue**(*a*, *var*, *rmin*, *rmax*)

Return the mean value of variable 'var' for a given range of value. The field 'var' is sorted. Then the mean value of 'var' on the given range is returned. For instance, getMeanRangeValue(a, 'F', 0., 0.3) return the mean value of F for the 30% lowest values.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **var** (string) – variable name
- **rmin** (float) – min of range in [0,1]
- **rmax** – max of range in [0,1]

Returns mean value of var for the min-max %

Return type float

Example of use:

- Get the mean value of x in a sorted range of values (array):

```
# - getMeanRangeValue (array) -
import Converter as C
import Generator as G

a = G.cart((0,0,0), (1.,1.,1.), (11,1,1))
# return the mean of the 30% smallest values
meanval = C.getMeanRangeValue(a, 'x', 0., 0.3); print(meanval)
# >> 0.75
```

- Get the mean value of x in a sorted range of values (pyTree):

```
# - getMeanRangeValue (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0), (1.,1.,1.), (11,1,1))
# get the mean of the 30% smallest values
meanval = C.getMeanRangeValue(a, 'CoordinateX', 0., 0.3); print(meanval)
#>> 0.75
```

Converter.**normL0**(*a*, *var*)

Return the L0 norm of field 'var' on input.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **var** (string) – variable name

Return type L0 norm

Example of use:

- Get the L0 norm of F (array):

```
# - normL0 (array) -
import Converter as C
import Generator as G

a = G.cart((0,0,0), (1,1,1), (11,11,11))
a = C.initVars(a, 'F', 1.)
print('normL0 =', C.normL0(a, 'F'))
#>> normL0 = 1.0
```

- Get the L0 norm of F (pyTree):

```
# - normL0 (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0), (1,1,1), (11,11,11))
C._initVars(a, 'centers:F', 1.)
print('normL0 =', C.normL0(a, 'centers:F'))
#>> normL0 = 1.0
```

Converter.**normL2**(*a*, *var*)

Return the L2 norm of field 'var' on input.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **var** (string) – variable name

Return type L2 norm

Example of use:

- Get the L2 norm of F (array):

```
# - normL2 (array) -
import Converter as C
import Generator as G

a = G.cart((0,0,0), (1,1,1), (11,11,11))
a = C.initVars(a, "F", 1.)
print('normL2 =', C.normL2(a, "F"))
#>> normL2 = 1.0

# cellN variable IS taken into account
cellnf = C.array('celln', 11, 11, 11)
cellnf = C.initVars(cellnf, "celln", 1.)
cellnf[1][0][1] = 0.
cellnf[1][0][2] = 0.
a = C.addVars([a, cellnf])
print('normL2 =', C.normL2(a, "F"))
#>> normL2 = 1.0
```

- Get the L2 norm of F (pyTree):

```
# - normL2 (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0), (1,1,1), (11,11,11))
a = C.initVars(a, "F", 1.)
print('normL2 =', C.normL2(a, "F"))
#>> normL2 = 1.0
```

Converter.**normalize**(*a*[, *'sx', 'sy', sz'*])

Normalize a vector defined by its 3 vector components. The vector component values are modified such that the vector (a.sx,a.sy,a.sz) has a unit norm for each point.

Exists also as in place version (_normalize) that modifies a and returns None.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **sx,sy,sz** (list of strings) – names of field used as vector components

Return type Identical to input

Example of use:

- Normalize a vector (array):

```
# - normalize (array) -
import Converter as C
import Generator as G
import Geom as D

a = D.sphere((0,0,0), 1., 50)
n = G.getNormalMap(a)
n = C.center2Node(n)
n[1] = n[1]*10
n = C.normalize(n, ['sx','sy','sz'])
a = C.addVars([a, n])
C.convertArrays2File(a, 'out.plt')
```

- Normalize a vector (pyTree):

```
# - normalize (pyTree) -
import Converter.PyTree as C
```

(continues on next page)

(continued from previous page)

```
import Geom.PyTree as D
import Generator.PyTree as G

a = D.sphere((0,0,0), 1., 50)
a = G.getNormalMap(a)
a = C.normalize(a, ['centers:sx','centers:sy','centers:sz'])
C.convertPyTree2File(a, 'out.cgns')
```

Converter.**magnitude**(*a*[, *'sx', 'sy', sz'*])

Get the magnitude of a vector defined by its 3 vector components for each point. The name of created field is composed from the components names. For instance 'sx,sy,sz' will create a 'sMagnitude' field.

Exists also as in place version (_magnitude) that modifies a and returns None.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **sx,sy,sz** (list of strings) – names of field used as vector components

Return type Identical to input

Example of use:

- Compute a vector magnitude (array):

```
# - magnitude (array) -
import Converter as C
import Generator as G
import Geom as D

a = D.sphere((0,0,0), 1., 50)
n = G.getNormalMap(a)
n = C.magnitude(n, ['sx','sy','sz'])
a = C.addVars([a, n])
C.convertArrays2File(a, 'out.plt')
```

- Compute a vector magnitude (pyTree):

```
# - magnitude (pyTree) -
import Converter.PyTree as C
import Geom.PyTree as D
```

(continues on next page)

(continued from previous page)

```
import Generator.PyTree as G

a = D.sphere((0,0,0), 1., 50)
a = G.getNormalMap(a)
a = C.magnitude(a, ['centers:sx','centers:sy','centers:sz'])
C.convertPyTree2File(a, 'out.cgns')
```

Converter.**randomizeVar**(*a*, *varName*, *deltaMin*, *deltaMax*)

Randomize a fied varName. The modified field is bounded by [f-deltaMin,f+deltaMax] where f is the local field value.

Exists also as in place version (_randomizeVar) that modifies a and returns None.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **varName** (string) – field to randomize
- **deltaMin,deltaMax** (floats) – range for random

Return type Identical to input

Example of use:

- Randomize a field (array):

```
# - randomizeVar (array) -
import Converter as C
import Generator as G
a = G.cart((0,0,0),(1,1,1),(11,11,1))
a = C.initVars(a, '{F}={x}*{y}')
b = C.randomizeVar(a,'F',0.1,0.5)
C.convertArrays2File([a,b],"out.plt")
```

- Randomize a field (pyTree):

```
# - randomizeVar (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
a = G.cart((0,0,0),(1,1,1),(11,11,1))
C._randomizeVar(a, 'CoordinateZ', 0.1, 1.)
C.convertPyTree2File(a, "out.cgns")
```

Converter.**isFinite**(*a*, *var=None*)

Return True if a contains only finite values (no NAN, no INF).

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **var** (string) – variable name (optional)

Return type True or False

Example of use:

- Test if fields are finite (array):

```
# - isFinite (array) -
import Generator as G
import Converter as C

a = G.cart((0,0,0), (1,1,1), (10,10,10))
a = C.initVars(a, 'F', 1.)
print(C.isFinite(a))
#>> True
print(C.isFinite(a, var='x'))
#>> True
print(C.isFinite(a, var='F'))
#>> True
```

- Test if fields are finite (pyTree):

```
# - isFinite (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C

a = G.cart((0,0,0), (1,1,1), (10,10,10))
C._initVars(a, 'F', 1.)
print(C.isFinite(a))
#>> True
print(C.isFinite(a, var='CoordinateZ'))
#>> True
print(C.isFinite(a, var='F'))
#>> True
```

Note: new in version 3.2.

4.5 Array / PyTree input/output

Converter.**convertFile2Arrays**(*fileName*, *format=None*, *options*)

Read a file and return a list of arrays (array interface). For format needing multiple files (for ex: plot3d), multiple files can be specified in file name string as: "file.gbin,file.qbin". In file format where variables name are undefined, the following ones are adopted: x, y, z, ro, rou, rov, row, roE, cellN. If format is unspecified, the format is guessed from file extension or from file header, if possible. For a list of available format, see *FileFormats*. Several options are available to specify the discretization of vector elements (for vector formats such as xfig or svg). For a list of available options, see *ReadOptions*.

Parameters

- **fileName** (string) – name of file to read
- **format** (string) – file format (see *FileFormats*)
- **options** (keywords) – options for vector formats such as svg, xfig (see *ReadOptions*)

Returns list of arrays

Return type list of Converter arrays

Example of use:

- Binary tecplot file read:

```
# - convertFile2Arrays (arrays) -
import Generator as G
import Converter as C

# Create and save test meshes
cart = G.cart((0,0,0), (0.1, 0.2, 1.), (11, 11, 2))
C.convertArrays2File(cart, 'out.plt')

# Read it
A = C.convertFile2Arrays('out.plt'); print(A)
#>> [['x,y,z', array([[ 0. ,  0.1,  0.2,  ...]]), 11,11,2]]
```

Converter.**convertArrays2File**(*a*, *fileName*, *format=None*, *options*)

Write array or list of arrays to a file (array interface). If format is not given, it is guessed from fileName extension. For a list of available formats, see *FileFormats*. For a list of available options, see *WriteOptions*.

Parameters

- **a** ([array, list of arrays]) – input data

- **fileName** (string) – name of file to read
- **format** (string) – file format (see *FileFormats*)
- **options** (keywords) – writing options (see *WriteOptions*)

Example of use:

- Binary tecplot file read/write:

```
# - convertArrays2File (array) -
import Generator as G
import Converter as C

# Create a cartesian mesh and save it as binary tecplot file
a = G.cart((0,0,0), (0.1, 0.2, 1.), (11, 11, 2))
C.convertArrays2File(a, 'out.plt', 'bin_tp')
```

Converter.PyTree.**convertFile2PyTree**(*fileName*, *format=None*, *options*)

Read a file and return a CGNS pyTree (pyTree interface). If format is not given, it is guessed from file header or extension. For a list of available format, see *FileFormats*. For a list of available options, see *ReadOptions*.

Parameters

- **fileName** (string) – name of file to read
- **format** (string) – file format (see *FileFormats*)
- **options** (keywords) – reading options (see *ReadOptions*)

Returns a pyTree

Return type pyTree

Example of use:

- CGNS file read:

```
# - convertFile2PyTree (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0.,0.,0.),(0.1,0.1,0.1),(11,11,11))
t = C.newPyTree(['Base',a])
C.convertPyTree2File(t, 'in.cgns')
t1 = C.convertFile2PyTree('in.cgns'); print(t1)
```

Converter.PyTree.**convertPyTree2File**(*t*, *fileName*, *format=None*, *options*)

Write a pyTree to a file (pyTree interface). If format is not given, it is guessed from file name extension. For a list of available format, see *FileFormats*. For a list of available options, see *WriteOptions*.

Parameters

- **t** ([pyTree, base, zone, list of zones]) – input data
- **fileName** (string) – name of file to read
- **format** (string) – file format (see *FileFormats*)
- **options** (keywords) – writing options (see *WriteOptions*)

Example of use:

- CGNS file write:

```
# - convertPyTree2File (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0.,0.,0.),(0.1,0.1,0.1),(11,11,11))
t = C.newPyTree(['Base', a])
C.convertPyTree2File(t, 'out.cgns')
C.convertPyTree2File(a, 'out.plt')
```

Known formats for read/write functions (array and pyTree interface):

Format	Extension	Description
bin_tp	.plt	binary tecplot file
fmt_tp	.dat, .tp	formatted tecplot file
bin_v3d	.v3d	binary v3d file (ONERA)
fmt_v3d	.fv3d	formatted v3d file (ONERA)
bin_plot3d	.gbin	binary plot 3d file (NASA)
fmt_plot3d	.gfmt	formatted plot 3d file (NASA)
fmt_pov	.pov	formatted povray raytracer file
fmt_mesh	.mesh	formatted mesh file (INRIA)
fmt_gmsh	.msh	formatted GMSH mesh file (UCL)
bin_gmsh	.msh	binary GMSH mesh file (UCL)
fmt_su2	.su2	formatted SU2 file (STANFORD)
fmt_cedre	.d	formatted CEDRE file (ONERA)
bin_stl	.stl	binary STL file
fmt_stl	.fstl	formatted STL file
fmt_obj	.obj	formatted OBJ file (WAVEFRONT)
bin_gltf	.gltf	binary gltf file (KHRONOS, only read)
bin_3ds	.3ds	binary 3DS file (3D STUDIO)
bin_ply	.ply	binary PLY file (STANFORD)
bin_pickle	.ref	binary python pickle file
bin_wav	.wav	binary wav 8 bits sound file
fmt_xfig	.fig	formatted XFIG file
fmt_svg	.svg	formatted SVG file (INKSCAPE)
bin_png	.png	binary PNG file

Known formats for read/write functions specific to pyTree interface:

Format	Extension	Description
bin_adf	.cgns .adf	binary CGNS ADF file
bin_hdf	.hdf	binary CGNS HDF file

Options for reading:

Option	Description	Default value
nptsCurve	Number of discretization points for curved vector elements	20
nptsLine	Number of discretization points for lines	2
density	Number of discretization points per unit length	-1: not used. If > 0, overides npts
skipTypes	list of strings (CGNS types) that stop reading when met	None
links	list of list of 4 strings (see after)	None

Links option:

For hdf format only, when reading, links are always followed but a list of links can be returned. If you specify links=[] to convertFile2PyTree, a list of links is returned. Each link is a list ['directoryOfPointedFile', 'pointedFile', 'targetNodePath', 'currentNodePath']. The 'directoryOfPointedFile' is the directory where the pointed file must be found, 'pointedFile' is the pointed file name, 'targetNodePath' is the path of pointed node (in pointed file), 'currentNodePath' is the path of node in current file.

Example of use:

- HDF file read with links:

```
# - HDF read/write with links -

import Generator.PyTree as G
import Converter.PyTree as C
import Converter.Filter as Filter

a = G.cart((0,0,0),(1,1,1),(50,50,50))
t = C.newPyTree(['Base',a])
C.convertPyTree2File(t, 'coord.hdf')
C._initVars(t, 'Density=1.')

# Save file with links
links=[['.','coord.hdf','/Base/cart/GridCoordinates','/Base/cart/
↪GridCoordinates']]
C.convertPyTree2File(t, 'main.hdf', links=links)

# full read of main returning links
LC=[]
t = C.convertFile2PyTree('main.hdf', links=LC); print(LC)
#>> [['.', './coord.hdf', '/Base/cart/GridCoordinates', '/Base/cart/
↪GridCoordinates']]
```

(continues on next page)

(continued from previous page)

```
# Read links with skeleton
LC=[]
t = Filter.convertFile2SkeletonTree('main.hdf', links=LC); print(LC)
#>> [['.', './coord.hdf', '/Base/cart/GridCoordinates', '/Base/cart/
↪GridCoordinates']]
```

Options for writing:

Option	Description	Possible values	Default value
int	Size of integer	4,8	4
real	Size of real	4,8	8
endian	Data endianess	'little', 'big'	'big'
dataFormat	'printf' like format for formatted files (%[width][.precision]specifier)	'%f', '%.9e', '%16.9e',...	'%.9e'
zoneNames	list of zone names (first struct, the unstruct zones)	['Zone1','Zone2',...]	[]
links	list of list of 4 strings (see after)	[['.', 'cart.hdf', '/Base', '/Base']]	None

Links option:

For hdf format only, when writing, link node path can be specified. These nodes are then not written with data but are written as links to a pointed file. A link is a list ['directoryOfPointedFile', 'pointedFile', 'targetNodePath', 'currentNodePath']. The 'directoryOfPointedFile' is the directory where the pointed file must be found, 'pointedFile' is the pointed file name, 'targetNodePath' is the path of pointed node (in pointed file), 'currentNodePath' is the path of node in current file. This function doesn't write the pointed file. You must explicitely write it with another call to convertPyTree2File.

Example of use:

- HDF file write with links:

```
# - HDF write with links -
import Generator.PyTree as G
import Converter.PyTree as C
import Converter.Internal as Internal

a = G.cart((0,0,0),(1,1,1),(50,50,50))
```

(continues on next page)

(continued from previous page)

```
C._initVars(a, 'Density=1.')
t = C.newPyTree(['Base',a])

# Save file with links
links=[['.','coord.hdf','/Base/cart/GridCoordinates','/Base/cart/
↪GridCoordinates']]
C.convertPyTree2File(t, 'main.hdf', links=links)

# Write pointed file
Internal._rmNodeByPath(t, '/Base/cart/FlowSolution')
C.convertPyTree2File(t, 'coord.hdf')
```

4.6 Preconditionning (hook)

Preconditionning is used to create pre-computed opaque search structures on zones. These opaque search structures are called hook and are used in the Geometrical identification functions and other functions of Connector and Post.

Converter.**createHook**(*a*, *functionName*)

Create a hook for use with identification function 'functionName'. For "extractMesh" and "adt", input is intended to be a set of zones, otherwise a hook is intended to be created on a single zone.

Parameters

- **a** ([array] or [zone]) – input data
- **functionName** (string) – function the hook is made for (see *functionName*)

Returns hook

Return type opaque structure

Example of use:

- Create hook (array):

```
# - createHook (array) -
import Converter as C
import Generator as G

a = G.cart((0,0,0), (1,1,1), (10,10,10))
hook = C.createHook(a, function='nodes')
```

- Create hook (pyTree):

```
# - createHook (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0), (1,1,1), (10,10,10))
hook = C.createHook(a, function='nodes')
```

Function name	Type of storage	Usage
'extractMesh'	Bounding boxes of cells stored in an ADT. Valid for structured and TETRA zones.	Post.extractMesh, Post.extractPoint
'adt'	Bounding boxes of cells stored in an ADT. Valid for structured and TETRA zones.	Connector.setInterpData, Connector.setIBCData
'nodes'	Mesh nodes stored in a k-d tree	Converter.identifyNodes, Converter.nearestNodes
'faceCenters'	Mesh face centers stored in a k-d tree	Converter.identifyFaces, Converter.nearestFaces
'elementCenters'	Mesh element centers stored in a k-d tree	Converter.identifyElements, Converter.nearestElements

---

Converter.**createGlobalHook**(*a*, *functionName*, *indir=0*)

Create a global hook (one single search structure) for a set of zones and for use with identification function 'functionName' or identifySolutions. If indir=1, the function also returns an indirection specifying the zone number of each index.

Parameters

- **a** ([arrays] or [zones]) – input data
- **functionName** (string) – function the hook is made for (see *functionName2*)

Returns hook

Return type opaque structure

Example of use:

- Create global hook (array):

```
# - createGlobalHook (array) -
import Converter as C
import Generator as G

a = G.cart((0,0,0), (1,1,1), (10,10,10))
b = G.cart((9,0,0), (1,1,1), (10,10,10))
hook = C.createGlobalHook([a,b], function='nodes')
```

- Create global hook (pyTree):

```
# - createGlobalHook (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0), (1,1,1), (10,10,10))
b = G.cart((9,0,0), (1,1,1), (10,10,10))
hook, indir = C.createGlobalHook([a,b], function='nodes', indir=1)
```

Function name	Type of storage	Usage
'nodes'	Mesh nodes stored in a k-d tree	Converter.identifyNodes, Converter.nearestNodes
'faceCenters'	Mesh face centers stored in a k-d tree	Converter.identifyFaces, Converter.nearestFaces
'elementCenters'	Mesh element centers stored in a k-d tree	Converter.identifyElements, Converter.nearestElements

---

Converter.**freeHook**(*hook*)

Free a hook created with createHook.

Parameters **hook** (opaque search structure as created by createHook) – hook

Example of use:

- Free hook (array):

```
# - freeHook (array) -
import Converter as C
import Generator as G
```

(continues on next page)

(continued from previous page)

```
a = G.cart((0,0,0), (1,1,1), (10,10,10))
hook = C.createHook([a], function='extractMesh')
C.freeHook(hook)
```

- Free hook (pyTree):

```
# - freeHook (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0), (1,1,1), (10,10,10))
hook = C.createHook([a], function='extractMesh')
C.freeHook(hook)
```

4.7 Geometrical identification

Converter.**identifyNodes**(*hook*, *a*, *tol=1.e-11*)

Identify nodes of a with nodes stored in hook. Return the indices of hook corresponding to the nodes of a. If a point is not identified, its returned index is -1.

Parameters

- **hook** (created by createHook) – hook
- **a** ([array,list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **tol** (float) – matching tolerance

Returns indices of identified points

Return type numpy array or list of numpy arrays

Example of use:

- Identify nodes in a hook (array):

```
# - identifyNodes (array) -
import Converter as C
import Generator as G
import Post as P

a = G.cart((0,0,0), (1,1,1), (10,10,10))
# Enregistre les noeuds de a dans le hook
hook = C.createHook(a, function='nodes')
```

(continues on next page)

(continued from previous page)

```
# Indices des noeuds de a correspondant aux noeuds de f
f = P.exteriorFaces(a)
nodes = C.identifyNodes(hook, f); print(nodes)
#>> [   1    2    3    4    5    6    7  ...]
```

- Identify nodes in a hook (pyTree):

```
# - identifyNodes (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Post.PyTree as P

a = G.cart((0,0,0), (1,1,1), (10,10,10))
hook = C.createHook(a, function='nodes')

# Indices des noeuds de a correspondant aux noeuds de f
f = P.exteriorFaces(a)
nodes = C.identifyNodes(hook, f); print(nodes)
#>> [   1    2    3    4    5    6    7  ...]
```

- Identify nodes in multiple zones (pyTree):

```
# - identifyNodes (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Geom.PyTree as D

a = G.cart((0,0,0), (1,1,1), (10,10,10))
b = G.cart((12,0,0), (1,1,1), (10,10,10))
hook, indir = C.createGlobalHook([a,b], function='nodes', indir=1)
offset = [0, C.getNPts(a), C.getNPts(b)]

f = D.point((13,3,3))
nodes = C.identifyNodes(hook, f)
ind = nodes[0]
print('Le premier point de f a pour indice', ind-offset[indir[ind]], 'sur la␣
↪zone', indir[ind])
#>> Le premier point de f a pour indice 332 sur la zone 1
```

Converter.**identifyFaces**(*hook*, *a*, *tol=1.e-11*)

Identify face centers of a with points stored in hook. Return the indices of hook corresponding to the faces of a. If a face is not identified, its returned index is -1.

Parameters

- **hook** (created by createHook) – hook
- **a** ([array,list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **tol** (float) – matching tolerance

Returns indices of identified faces

Return type numpy array or list of numpy arrays

Example of use:

- Identify faces in a hook (array):

```
# - identifyFaces (array) -
import Converter as C
import Generator as G

a = G.cartNGon((0,0,0), (1,1,1), (10,10,10))
b = G.cartNGon((9,0,0), (1,1,1), (10,10,10))

# Enregistre les centres des faces de a dans le hook
hook = C.createHook(a, function='faceCenters')
# Indices des faces de a correspondant aux faces de b
faces = C.identifyFaces(hook, b); print(faces)
#>> [10 -1 -1 ..., -1 -1 -1]
```

- Identify faces in a hook (pyTree):

```
# - identifyFaces (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cartNGon((0,0,0), (1,1,1), (10,10,10))
b = G.cartNGon((9,0,0), (1,1,1), (10,10,10))

# Enregistre les centres des faces de a dans le hook
hook = C.createHook(a, function='faceCenters')
# Indices des faces de a correspondant aux faces de b
faces = C.identifyFaces(hook, b); print(faces)
#>> [10 -1 -1 ..., -1 -1 -1]
```

Converter.**identifyElements**(*hook*, *a*, *tol=1.e-11*)

Identify element centers of a with points stored in hook. Return the indices of hook

corresponding to the elements of a. If a elements is not identified, its returned index is -1.

Parameters

- **hook** (created by createHook) – hook
- **a** ([array,list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **tol** (float) – matching tolerance

Returns indices of identified elements

Return type numpy array or list of numpy arrays

Example of use:

- Identify elements in a hook (array):

```
# - identifyElements (array) -
import Converter as C
import Generator as G
import Post as P

a = G.cartNGon( (0,0,0), (1,1,1), (10,10,10) )
f = P.exteriorElts(a)

# Enregistre les centres des elements dans le hook
hook = C.createHook(a, function='elementCenters')
# Indices des elements de a correspondant aux centres des elts de f
elts = C.identifyElements(hook, f); print(elts)
#>> [  1   2   3   4   5 ... 726 727 728 729]
```

- Identify elements in a hook (pyTree):

```
# - identifyElements (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Post.PyTree as P

a = G.cartNGon((0,0,0), (1,1,1), (10,10,10))
f = P.exteriorElts(a)

# Enregistre les centres des faces dans le hook
hook = C.createHook(a, function='elementCenters')
# Indices des faces de a correspondant aux centres des elts de f
elts = C.identifyElements(hook, f); print(elts)
#>> [  1   2   3  ... 726 727 728 729]
```

Converter.**identifySolutions**(*tRcv*, *tDnr*, *hookN=None*, *hookC=None*, *vars=[]*, *tol=1.e6*)

Set the solution field in tRcv with the nearest point solution of tDnr. Hooks must be global hooks on tDnr.

Exists also as an in-place version (_identifySolutions) which modifies tRcv and returns None.

Parameters

- **tRcv** ([array,list of arrays] or [pyTree, base, zone, list of zones]) – receiver data
- **tDnr** ([array,list of arrays] or [pyTree, base, zone, list of zones]) – donor data
- **hookN** (created by createGlobalHook) – global hook if field on nodes
- **hookC** (created by createGlobalHook) – global hook if field on centers
- **vars** (list of strings) – variable names list
- **tol** (float) – tolerance for matching

Returns a reference copy of tRcv

Return type identical to input

Example of use:

- Identify solutions (array):

```
# - identifySolutions (array) -
import Converter as C
import Generator as G
import Geom as D
ni = 21; nj = 21; nk = 21
m = G.cart((0,0,0), (1./(ni-1),1./(nj-1),1./(nk-1)), (ni,nj,nk))
hook = C.createGlobalHook([m], function='nodes')
sol = C.initVars(m, 'ro={x}')
sol = C.extractVars(sol,['ro'])

# Create extraction mesh
a = D.sphere((0,0,0),0.1)
# Identify solutions of sol in a
a2 = C.identifySolutions(a, sol, hook)
C.freeHook(hook)
```

(continues on next page)

(continued from previous page)

```
a = C.addVars([a,a2])
m = C.addVars([m,sol])
C.convertArrays2File([m,a], 'out.plt')
```

- Identify solutions (pyTree):

```
# - identifySolutions (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Geom.PyTree as D

# A mesh with a field
ni = 21; nj = 21; nk = 21
m = G.cart((0,0,0), (1./(ni-1),1./(nj-1),1./(nk-1)), (ni,nj,nk))
m = C.initVars(m, '{Density}={CoordinateX}')

# Create extraction mesh
a = D.sphere((0,0,0),0.1)

# Identify solutions
hook = C.createGlobalHook([m], 'nodes')
a = C.identifySolutions(a, m, hookN=hook)
C.freeHook(hook)
C.convertPyTree2File(a, 'out.cgns')
```

Converter.**nearestNodes**(*hook*, *a*)

Find nearest points stored in hook to the nodes of a. Return the indices of hook nearest of a given node of a and the corresponing distance.

Parameters

- **hook** (created by createHook) – hook
- **a** ([array,list of arrays] or [pyTree, base, zone, list of zones]) – input data

Returns indices and distance of nearest points

Return type tuple of 2 numpys or list of tuple of 2 numpys

Example of use:

- Find nearest nodes in a hook (array):

```
# - nearestNodes (array) -
import Converter as C
```

(continues on next page)

(continued from previous page)

```
import Generator as G
import Transform as T
import Post as P

a = G.cart((0,0,0), (1,1,1), (10,10,10))
# Enregistre les noeuds de a dans le hook
hook = C.createHook(a, function='nodes')

# Indices des noeuds de a les plus proches des noeuds de f
# et distance correspondante
b = T.translate(a,(0.15,0.,0.))
f = P.exteriorFaces(b)
nodes,dist = C.nearestNodes(hook, f); print(nodes, dist)
#>> [   1    2    3   ...] [0.15  0.15  0.15  ...]
```

- Find nearest nodes in a hook (pyTree):

```
# - nearestNodes (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Transform.PyTree as T
import Post.PyTree as P

a = G.cart((0,0,0), (1,1,1), (10,10,10))
b = T.translate(a,(0.15,0.,0.))
f = P.exteriorFaces(b)

hook = C.createHook(a, function='nodes')
# Indices des noeuds de a les plus proches des noeuds de f
# et distance correspondante
nodes,dist = C.nearestNodes(hook, f)
print(nodes, dist)
```

Converter.**nearestFaces**(*hook*, *a*)

Find nearest points stored in hook to the face centers of a. Return the indices of hook nearest of a given face of a and the corresponing distance.

Parameters

- **hook** (created by createHook) – hook
- **a** ([array,list of arrays] or [pyTree, base, zone, list of zones]) – input data

Returns indices and distance of nearest points

Return type tuple of 2 numpys or list of tuple of 2 numpys

Example of use:

- Find nearest faces in a hook (array):

```
# - nearestFaces (array) -
import Converter as C
import Generator as G

a = G.cartNGon((0,0,0), (1,1,1), (10,10,10))
b = G.cartNGon((9.1,0,0), (1,1,1), (10,10,10))

# Enregistre les centres des faces de a dans le hook
hook = C.createHook(a, function='faceCenters')
# Indices des faces de a les plus proches des faces de b
# et distance correspondante
faces,dist = C.nearestFaces(hook, b)
print(faces,dist)
C.freeHook(hook)
```

- Find nearest faces in a hook (pyTree):

```
# - nearestFaces (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cartNGon((0,0,0), (1,1,1), (10,10,10))
b = G.cartNGon((9.1,0,0), (1,1,1), (10,10,10))

# Enregistre les centres des faces de a dans le hook
hook = C.createHook(a, function='faceCenters')
# Indices des faces de a les plus proches des faces de b
# et distance correspondante
faces,dist = C.nearestFaces(hook, b)
print(faces,dist)
```

Converter.**nearestElements**(*hook*, *a*)

Find nearest points stored in hook to the elements centers of a. Return the indices of hook nearest of a given element of a and the corresponing distance.

Parameters

- **hook** (created by createHook) – hook
- **a** ([array,list of arrays] or [pyTree, base, zone, list of zones]) – input data

Returns indices and distance of nearest points

Return type tuple of 2 numpys or list of tuple of 2 numpys

Example of use:

- Find nearest elements in a hook (array):

```
# - nearestElements (array) -
import Converter as C
import Generator as G
import Transform as T
import Post as P

a = G.cartNGon( (0,0,0), (1,1,1), (10,10,10) )
b = T.translate(a,(0.15,0.,0.))
f = P.exteriorElts(b)

# Enregistre les centres des faces dans le hook
hook = C.createHook(a, function='elementCenters')
# Indices des faces de a les plus proches des centres des elts de f
# et distance correspondante
elts,dist = C.nearestElements(hook, f)
print(elts,dist)
```

- Find nearest elements in a hook (pyTree):

```
# - nearestElements (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Transform.PyTree as T
import Post.PyTree as P

a = G.cartNGon((0,0,0), (1,1,1), (10,10,10))
b = T.translate(a,(0.15,0.,0.))
f = P.exteriorElts(b)

# Enregistre les centres des faces dans le hook
hook = C.createHook(a, function='elementCenters')
# Indices des faces de a les plus proches des centres des elts de f
# et distance correspondante
elts,dist = C.nearestElements(hook, f)
print(elts,dist)
```

Converter.**createGlobalIndex**(*a*)

Create a index field corresponding to the vertex number.

Parameters **a** ([array,list of arrays] or [pyTree, base, zone, list of zones]) – input data

Returns input data with a 'globalIndex' field

Return type Identical to input

Example of use:

- Create global index (array):

```
# - createGlobalIndex (array) -
import Converter as C
import Generator as G

a = G.cart((0,0,0), (1,1,1), (10,10,10))
C._createGlobalIndex(a)
C.convertArrays2File(a, 'out.plt')
```

- Create global index (pyTree):

```
# - createGlobalIndex (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0), (1,1,1), (10,10,10))
C._createGlobalIndex(a)
C.convertPyTree2File(a, 'out.cgns')
```

Converter.**recoverGlobalIndex**(*a*, *b*)

Push the field of b in a follwing the global index field.

Parameters

- **a** ([array,list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **b** ([array,list of arrays] or [pyTree, base, zone, list of zones]) – input data with 'globalIndex' field

Returns modified a with the field of b

Return type Identical to a

Example of use:

- Recover global index (array):

```
# - recoverGlobalIndex (array) -
import Converter as C
import Generator as G
import Transform as T

a = G.cart((0,0,0), (1,1,1), (10,10,10))
C._createGlobalIndex(a)

b = T.splitNParts(a, 2)
C._initVars(b[0], 'f=1')
C._initVars(b[1], 'f=2')

C._recoverGlobalIndex(a, b[0])
C._recoverGlobalIndex(a, b[1])
a = C.rmVars(a, 'globalIndex')

C.convertArrays2File(a, 'out.plt')
```

- Create global index (pyTree):

```
# - recoverGlobalIndex (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Transform.PyTree as T

a = G.cart((0,0,0), (1,1,1), (10,10,10))
C._createGlobalIndex(a)

b = T.splitNParts(a, 2)
C._initVars(b[0], 'f=1')
C._initVars(b[1], 'f=2')

C._recoverGlobalIndex(a, b[0])
C._recoverGlobalIndex(a, b[1])
C._rmVars(a, 'globalIndex')

C.convertPyTree2File(a, 'out.cgns')
```

4.8 Client/server to exchange arrays/pyTrees

Converter.**createSockets**(*nprocs=1*, *port=15555*)

Create sockets for receiving arrays/pyTrees. If you are sending from a MPI run with nprocs, set nprocs accordingly.

Parameters

- **nprocs** (int) – the number of process the sender job is running with.
- **port** (int) – the communication port

Returns socket

Return type socket

Converter.**listen**(*sockets*)

Listen for client sends.

Parameters **sockets** (sockets) – sockets (created with createSockets)

Returns arrays or pyTrees

Example of use:

- Listen for arrays from server (array):

```
# - listen (array) -
import Converter as C
import CPlot
sockets = C.createSockets()

while True:
    out = []
    for s in sockets:
        a = C.listen(s)
        if a is not None: out.append(a)
    if out != []: CPlot.display(out)
```

- Listen for pyTrees from server (pyTree):

```
# - listen (pyTree) -
import Converter.PyTree as C
import CPlot.PyTree as CPlot

sockets = C.createSockets()
```

(continues on next page)

(continued from previous page)

```
while True:
    out = []
    for s in sockets:
        a = C.listen(s)
        if a is not None: out.append(a)
    if out != []: CPlot.display(out)
```

Converter.**send**(*a*, *host='localhost'*, *rank=0*, *port=15555*)

Send data to the server.

Parameters

- **a** ([array,list of arrays] or [pyTree, base, zone, list of zones]) – input data
- **host** (string) – host we are sending to
- **rank** (int) – rank of sending process
- **port** (int) – communication port (must be the same as createSockets)

Example of use:

- Send arrays to server (array):

```
# - send (array) -
import Converter as C
import Generator as G
import Transform as T

a = G.cart((0,0,0), (1,1,1), (100,100,300))
C.send(a, 'localhost')

for i in range(30):
    a = T.rotate(a, (0,0,0), (0,0,1), 10.)
    C.send(a, 'localhost')
```

- Send pyTrees to server (pyTree):

```
# - send (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Transform.PyTree as T

a = G.cart((0,0,0), (1,1,1), (100,100,30))
```

(continues on next page)

(continued from previous page)

```
C.send(a, 'localhost')

for i in range(30):
    a = T.rotate(a, (0,0,0), (0,0,1), 10.)
    C.send(a, 'localhost')
```

4.9 Converter arrays / 3D arrays conversion

In some applications, arrays must be seen as 3D arrays, that is (ni,nj,nk) numpy arrays instead of (nfld, ni*nj*nk) arrays. A 3D array is defined as [['x','y',...],[ax, ay, ...]] where ax is a (ni,nj,nk) numpy array corresponding to variable x, and so on...

Converter.Array3D.**convertArrays2Arrays3D**(*a*)

Convert arrays to 3D arrays (ni,nj,nk).

Parameters **a** (`[list of arrays]`) – input data

Returns list of 3D arrays

Example of use:

- Create 3D arrays from Converter arrays (array):

```
# - convertArrays2Arrays3D -
import Generator as G
import Converter.Array3D

a = G.cart((0,0,0), (0.1, 0.2, 1.), (11, 4, 1))
b = Converter.Array3D.convertArrays2Arrays3D([a]); print(b)
#>> [[['x', 'y', 'z'], [array([[[ 0. ], ...]]]
```

Converter.Array3D.**convertArrays3D2Arrays**(*a*)

Convert 3D arrays to Converter arrays.

Parameters **a** (`[list of 3D arrays]`) – input data

Returns list of arrays

Example of use:

- Create Converter arrays from 3D arrays (array):

```
# - convertArrays3D2Arrays -
import Converter as C
import Generator as G
import Converter.Array3D

a = G.cart( (0,0,0), (0.1, 0.2, 1.), (11, 4, 2))
b = Converter.Array3D.convertArrays2Arrays3D([a])
c = Converter.Array3D.convertArrays3D2Arrays(b); print(c)
```

CHAPTER

FIVE

INDEX

- genindex
- modindex
- search