Converter 2.4

Stephanie Peron, Christophe Benoit, Gaelle Jeanfaivre, Pascal Raud,
Benoit Rodriguez, Simon Verley, Bruno Maugars, Thomas Renaud
- Onera -

1 Converter: CFD data conversion module

1.1 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**, the second one is called a **pyTree**.

- 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..].

In this documentation, we note a or b an array, and A or B a list of arrays.

Important note: For accessing numpy arrays in Python, the first index corresponds to the variable number, the second index to the cell index. For instance: ar[0,0] for an 'x,y,z' array is variable x of first cell, ar[1,0] is variable y of first cell, and so on... For a structured array, the cell of index i,j,k will be accessed with index ind=i+j*ni+k*ni*nj, by ar[0,ind].

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 can be a numpy array, a float64, an int32 or a string), and [...] designates a list of nodes that are the children of the current node.
  In this case, in the following, we note a or b a zone node, and A or B a list of zone nodes or a complete Python tree.
  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
```

1.2 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.
Names are 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	
Dynamic pressure	PressureDynamic	
Static temperature	Temperature	
Enthalpy	Enthalpy	
Entropy	Entropy	
Stagnation pressure	PressureStagnation	
Stagnation temperature	TemperatureStagnation	
x-component of the absolute velocity	VelocityX	
y-component of the absolute velocity	VelocityY	
z-component of the absolute velocity	VelocityZ	
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:-interpolation block Id)		cellNF, cellnf, cellNF, ichim
Cell Status (-1:orphan, 0:blanked, 1:discretised, 2:interpolated explicitly, 3:extrapolated, 4:interpolated implicitly)		status

1.3 Array creation and manipulations

C.array: create a structured array containing variables x,y,z on a nixnjxnk grid:

```
a = C.array('x,y,z', ni, nj, nk)
```

Create an unstructured array containing variables x,y,z on a grid containing np points and ne elements of type ELTTYPE (=NODE, BAR, TRI, QUAD, TETRA, PYRA, PENTA, HEXA, NGON):

```
a = C.array('x,y,z', np, ne, ELTTYPE)
```

(See: arrayK.py)

C.getValue: return the list of values defined in array a 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):

```
v = C.getValue(a, ind)
```

(See: getValue.py)

C.setValue: set the values of one point of index ind in array a. v must be a list corresponding to the variables stored in array a:

```
C.setValue(a, ind, v)
```

(See: setValue.py)

C.addVars: add variable(s) to an array. Variables argument can be a string name ('ro') or a list of string names (['ro', 'rou']):

```
b = C.addVars(a, 'ro') .or. B = C.addVars(A, 'ro')
```

(See: addVar.py)

C.addVars: concatenate array fields with the same dimensions. Variables defined by a list of arrays are put in the same array:

```
f = C.addVars([a, b, c])
```

(See: addVars.py)

C.copy: copy an array (return a new duplicated array):

```
b = C.copy(a) .or. B = C.copy(A)
```

(See: copya.py)

1.4 pyTree creation and manipulations

C.newPyTree: create a new pyTree. You can specify base names, cell dimension in base, and attach zone nodes eventually:

```
A = C.newPyTree(['Base1', 2, 'Base2', 3]) .or. A = C.newPyTree(['Base1', z1, z2])
```

(See: newPyTree.py)

C.getNobOfBase: get the number of a given base a in topTree base list, such that topTree[2][nob] = a:

```
nob = C.getNobOfBase(a, topTree)
```

(See: getNobOfBasePT.py)

C.getNobNozOfZone: get the number (nob, noz) of a given zone a in topTree base and zone list , such that topTree[2][nob][2][noz] = a:

```
(nob, noz) = C.getNobNozOfZone(a, topTree)
```

(See: getNobNozOfZonePT.py)

C.breakConnectivity: break a multi-element zone (unstructured) into single element zones:

```
B = C.breakConnectivity(a) .or. B = C.breakConnectivity(A)
```

(See: breakConnectivityPT.py)

C.mergeConnectivity: 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:

```
a = C.mergeConnectivity(a, b, boundary=0)
```

(See: mergeConnectivityPT.py)

C.mergeTrees: merge two trees in one. Only basis of B are added to A:

```
D = C.mergeTrees(A, B)
```

(See: mergeTrees.py)

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

```
B = C.deleteEmptyZones(A)
```

(See: deleteEmptyZones.py)

C.addBase2PyTree: add a base named 'baseName' to a pyTree. Third argument specifies the cell dimension (cellDim=3 for volume meshes, cellDim=2 for surface meshes):

```
B = C.addBase2PyTree(A, baseName, cellDim=3)
```

(See: addBase2PyTree.py)

C.addState: add a FlowEquation or a ReferenceState data to a base or a zone or a node:

```
B = C.addState(A, state, value)
```

(See: addStatePT.py)

Add a full ReferenceState data to a base or a zone or a node:

```
B = C.addState(A, MInf=0.5, alphaZ=0., alphaY=0., ReInf=1.e8, MutSMuInf=0.2, TurbRateInf=1.e-8)
```

(See: addState2PT.py)

C.addChimera2Base: add Chimera settings data to a base. Settings are added in a .Solver#Chimera

user defined node:

```
B = C.addChimera2Base(A, setting, value)
```

(See: addChimera2Base.py)

C.addBC2Zone: 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. Parameter bndType is the type of BC, according to the CGNS Standard or 'BCMatch' for a 1-to-1 abutting GC, 'BCNearMatch' for 'n-to-m' abutting GC (only for structured grids), 'BCOverlap' for an overset GC, or 'FamilySpecified:FAMILY' for a BC specified in a family BC named FAMILY. Data is optional data that will be stored in a BCDataSet.

For structured grids, parameter 'range' specifies the window where the BC is applied. It can be defined by a list of indices [imin, imax, jmin, jmax, kmin, kmax] or by 'imin', 'jmin', ... if the BC is defined on the whole window ('imin' stands for i=1, 'imax' for i=imax). A physical BC for a structured grid is defined by:

```
b = C.addBC2Zone(a, bndName, bndType, range, data=None)
```

For a 1-to-1 abutting GC, donor zone and range and transformation must be specified:

```
b = C.addBC2Zone(a, bndName, 'BCMatch', range, zoneDonor=donorZone, rangeDonor=donorRange, trirac=[-1,2,3])
```

For periodic 1-to-1 GC, you must specify rotationCenter, rotationAngle and translation:

```
b = C.addBC2Zone(a, bndName, 'BCMatch', range, zoneDonor=donorZone, rangeDonor=donorRange, trirac=[-1,2,3], rotationCenter=(0,0,0), rotationAngle(0,0,25.), translation=(0,0,0))
```

For an overlap GC, donor zones can be provided as a list of zones or a list of zone names (optional). If the window range defines an overset GC and a physical BC, then 'doubly_defined' must be set for the 'rangeDonor' parameter.

```
b = C.addBC2Zone(a, bndName, 'BCOverlap', range, zoneDonor=donorZones, rangeDonor='doubly_defined')
```

For basic element unstructured zones, the location of the BC/GC can be specified either by a list of faces defined by 'faceList', either by 'elementList' or 'elementRange' referencing an existing boundary connectivity, or by a subzone:

```
b = C.addBC2Zone(a, bndName, bndType, faceList=[], data=None).or. b = C.addBC2Zone(a, bndName, bndType, elementList=[], data=None) .or. b = C.addBC2Zone(a, bndName, bndType, elementRange=[], data=None) .or. b = C.addBC2Zone(a, bndName, bndType, subzone=z, data=None)
```

For NGON zones, only faceList or subzone can be used:

```
b = C.addBC2Zone(a, bndName, bndType, faceList=[], data=None).or. b = C.addBC2Zone(a, bndName, bndType, subzone=z, data=None)
```

(See: addBC2Zone.py) (See: addBC2ZoneU.py)

C.fillEmptyBCWith: fill empty boundary conditions of a structured tree/base/list of zones/zone with the given boundary condition. Parameter dim can be 2 or 3:

```
b = C.fillEmptyBCWith(a, bndName, bndType, dim=3) .or. B = C.fillEmptyBCWith(A, bndName, bndType, dim=3)
```

(See: fillEmptyBCWith.py)

C.rmBCOfType: remove all boundaries of a given type from a pyTree. bndType can also be a family BC name ('FamilySpecified:FAMILY'):

```
b = C.rmBCOfType(a, 'BCWall') .or. B = C.rmBCOfType(A, 'BCWall')
```

(See: rmBCOfType.py)

C.rmBCOfName: remove the boundary of given name from a pyTree. bndName accepts wildcard and can also be a family BC name ('FamilySpecified:FAMILY'):

```
b = C.rmBCOfName(a, 'wall*') .or. B = C.rmBCOfName(A, 'wall*')
```

(See: rmBCOfNamePT.py)

C.extractBCOfType: extract all boundaries of a given type from a pyTree, return a list of zones. bndType can also be a family BC name ('FamilySpecified:FAMILY):

```
Z = C.extractBCOfType(a, 'BCWall') .or. Z = C.extractBCOfType(A, 'BCWall')
```

(See: extractBCOfTypePT.py)

C.extractBCOfName: extract all boundaries of a given name from a pyTree, return a list of zones. bndName accepts wildcards or can be a family name ('FamilySpecified:FAMILY'):

```
Z = C.extractBCOfName(a, 'wall1') .or. Z = C.extractBCOfName(A, 'wall1')
```

(See: extractBCOfNamePT.py)

C.getEmptyBC: return undefined boundary conditions for a zone node, a list of zone nodes or a complete pyTree, as a list of the range [imin,imax,jmin,jmax,kmin,kmax] of undefined boundaries for structured zones and as a list of face indices for unstructured zones. Lists are empty ([[],...,[]]) if all the boundary conditions 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):

```
wins = C.getEmptyBC(a, dim=3, splitFactor=180.) .or. wins = C.getEmptyBC(A, dim=3, splitFactor=180.)
```

(See: getEmptyBC.py)

C.getConnectedZones: get zones connected to a given zone a by 'BCMatch' or 'BCNearMatch' or 'all' (defined in zone GridConnectivity):

```
Z = C.getConnectedZones(a, topTree, type='all') .or. Z = C.getConnectedZones(A, topTree, type='all')
```

(See: getConnectedZonesPT.py)

C.addFamily2Base: add a family node to a base node of a tree. If this node defines a BC type,

bndType must be defined:

```
B = C.addFamily2Base(A, familyName, bndType=None)
```

(See: addFamily2Base.py)

C.tagWithFamily: Tag a zone node or a BC node a with a family name:

```
b = C.tagWithFamily(a, familyName)
```

(See: tagWithFamilyPT.py)

C.getFamilyZones: get all zones of a family name:

```
b = C.getFamilyZones(a, familyName)
```

(See: getFamilyZonesPT.py)

C.getFamilyBCs: get all BC nodes of a given familyName:

```
b = C.getFamilyBCs(a, familyName)
```

(See: getFamilyBCsPT.py)

C.getFamilyZoneNames: get all family zone names in A (A must be a base node or a tree node):

```
names = C.getFamilyZoneNames(A)
```

(See: getFamilyZoneNamesPT.py)

C.getFamilyBCNamesOfType: get all family BC names of this type (A must be a base node or a tree node):

```
names = C.getFamilyBCNamesOfType(A, bndType=None)
```

(See: getFamilyBCNamesOfTypePT.py)

C.rmNodes: remove nodes named 'name' from a for each zone:

```
b = C.rmNodes(a, name) .or. B = C.rmNodes(A, name)
```

(See: rmNodes.py)

C.getValue: return the list of values 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 variable:

```
v = C.getValue(a, var, ind)
```

ind starts at 0 and (i,j,k) start at 1.

(See: getValuePT.py)

C.setValue: set the values of one point of index ind in a zone a. var is the name of the variables to be modified, value can be a float or a list of floats corresponding to the values of the variables to be modified:

```
C.setValue(a, var, ind, value)
```

(See: setValuePT.py)

C.setPartialFields: 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':

```
b = C.setPartialFields(a, F, I, loc='nodes') .or. B = C.setPartialFields(A, F, I, loc='nodes')
```

(See: setPartialFieldsPT.py)

C.setPartialFields1: set the values for a given list of indices. Field values are given as a list of numpys in F (a list of numpys for each zone and for each field), indices are given as a list of numpys in I (one numpy for each zone), loc can be 'nodes' or 'centers':

```
b = C.setPartialFields1(a, F, I, loc='nodes') .or. B = C.setPartialFields1(A, F, I, loc='nodes')
```

(See: setPartialFields1PT.py)

C.addVars: add a variable(s) to a zone. var is a string name or a list of string names (e.g. 'Density',...), variable localisation ('nodes' or 'centers') can be specified in var:

```
b = C.addVars(a, var) .or. B = C.addVars(A, var)
```

(See: addVarsPT.py)

C.fillMissingVariables: fill FlowSolution_t nodes with variables defined in a zone, such that all the zones have the same variables:

```
b = C.fillMissingVariables(a)
```

(See: fillMissingVariablesPT.py)

C.cpVars: copy a variable from zone a1, with name var1, to zone a2, with name var2. The var location must be coherent:

```
b = C.cpVars(a1, var1, a2, var2)
```

(See: cpVars.py)

C.printTree: pretty print a pyTree to screen or in a file:

```
C.printTree(A, file=None)
```

(See: printTree2PT.py)

1.5 Array / pyTree common manipulations

C.getVarNames: return the list of variable names contained in a. Localization of variables can be specified ('nodes', 'centers', 'both'):

```
V = C.getVarNames(a, excludeXYZ=False, loc='both') .or. V = C.getVarNames(A)
```

(See: getVarNames.py) (See: getVarNamesPT.py)

C.isNamePresent: return -1 if A doesn't contain varName, 0 if at least one zone in A contains varName, 1 if all zones in A contain varName:

```
ret = C.isNamePresent(a, varName) .or. ret = C.isNamePresent(A, varName)
```

(See: isNamePresent.py) (See: isNamePresentPT.py)

C.getNPts: return the number of points in a:

```
npts = C.getNpts(a) .or. npts = C.getNpts(A)
```

(See: getNPts.py)

C.getNCells: return the number of cells in a:

```
ncells = C.getNCells(a) .or. ncells = C.getNCells(A)
```

(See: getNCells.py)

C.initVars: init variable given by a string to a constant value val:

```
b = C.initVars(a, 'cellN', val) .or. B = C.initVars(A, 'cellN', val)
```

(See: initVars.py) (See: initVarsPT.py)

Init a variable (named 'x') with a function f. Function f has two arguments here, named 'x','y'. Variables location can not be mixed:

```
b = C.initVars(a, 'x', f, ['x','y']) .or. B = C.initVars(A, 'x', f, ['x','y'])
```

(See: initVar.py)

Init a variable by a formula string. Variables must be separated by . Variables location can not be mixed:

```
b = C.initVars(a, 'Density = 3 * x') .or. B = C.initVars(A, 'Density = 3 * x')
```

(See: initVarsByEq.py) (See: initVarsByEqPT.py)

C.extractVars: extract variables named 'F','G' from a (other variables are removed):

```
b = C.extractVars(a, ['F','G']) .or. B = C.extractVars(A, ['F','G'])
```

(See: extractVars.py) (See: extractVarsPT.py)

C.rmVars: remove a variable(s). var is a string name or a list of string names:

```
b = C.rmVars(a, var) .or. B = C.rmVars(A, var)
```

(See: rmVars.py) (See: rmVarsPT.py)

C.convertArray2Tetra: create tetra unstructured array from an array. 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:

```
b = C.convertArray2Tetra(a, split='simple') .or. B = C.convertArray2Tetra(A, split='simple')
```

(See: convertStruct2Tetra.py) (See: convertArray2TetraPT.py) (See: convertHexa2Tetra.py) (See: convertPrism

C.convertArray2Hexa: Create hexa unstructured array from an array. 2D elements are made quadrangular, else they are made hexahedral:

```
b = C.convertArray2Hexa(a) .or. B = C.convertArray2Hexa(A)
```

(See: convertStruct2Hexa.py) (See: convertArray2HexaPT.py)

C.convertArray2NGon: create an NGON array from an array:

```
b = C.convertArray2NGon(a) .or. B = C.convertArray2NGon(A)
```

(See: convertArray2NGon.py) (See: convertArray2NGonPT.py)

C.convertArray2Node: create a node array from an array (discarding any connectivity):

```
b = C.convertArray2Node(a) .or. B = C.convertArray2Node(A)
```
(See: convertArray2Node.py) (See: convertArray2NodePT.py)

C.convertBAR2Struct: create a structured 1D array from a BAR array. The BAR array must not contain branches:

```
b = C.convertBAR2Struct(a) .or. B = C.convertBAR2Struct(A)
```
(See: convertBAR2Struct.py) (See: convertBAR2StructPT.py)

C.convertTri2Quad: 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:

```
b, c = C.convertTri2Quad(a, alpha=30.) .or. B, C = C.convertTri2Quad(A, alpha=30.)
```
(See: convertTri2Quad.py) (See: convertTri2QuadPT.py)

C.conformizeNGon: conformize the cell faces of a NGon, such that a face of a cell corresponds to a unique face of another cell:

```
b = C.conformizeNGon(a, tol=1.e-6) .or. B = C.conformizeNGon(A, tol=1.e-6)
```
(See: conformizeNGon.py) (See: conformizeNGonPT.py)

C.node2Center: convert data (grid coordinates and solution), defined at nodes in a, to centers:

```
b = C.node2Center(a) .or. B = C.node2Center(A)
```
When using the pyTree interface, a varname can be additionaly specified. Then, only the variable 'varname' is computed at centers and set in returned zone FlowSolution#Centers location:

```
b = C.node2Center(a, 'Density') .or. B = C.node2Center(A, 'Density')
```
(See: node2Center.py) (See: node2CenterPT.py)

C.center2Node: Change data, defined at centers in a, to nodes. 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:

```
b = C.center2Node(a, cellNType=0) .or. B = C.center2Node(A, cellNType=0)
```
When using the pyTree interface, a varname can be additionaly specified. Then, only the variable 'varname' is computed at nodes and set in zone FlowSolution location:

```
b = C.center2Node(a, 'centers:Density') .or. B = C.center2Node(A, 'centers:Density')
```
(See: center2Node.py) (See: center2NodePT.py)

C.node2ExtCenters: Convert grid coordinates, defined at nodes in a, to extended centers:

```
b = C.node2ExtCenter(a) .or. B = C.node2ExtCenter(A)
```
(See: node2ExtCenter.py) (See: node2ExtCenterPT.py)

1.6 Array / pyTree analysis

C.diffArrays: given a solution A and a solution B both defined on the same mesh, return the differences:

```
C = C.diffArrays(A, B)
```

(See: diffArrays.py) (See: diffArraysPT.py)

C.getArgMin: return the field value where variable 'ro' is minimum:

```
min = C.getArgMin(a, 'ro') .or. min = C.getArgMin(A, 'ro')
```

(See: getArgMin.py) (See: getArgMinPT.py)

C.getArgMax: Return the field value where variable 'ro' is maximum:

```
max = C.getArgMax(a, 'ro') .or. max = C.getArgMax(A, 'ro')
```

(See: getArgMax.py) (See: getArgMaxPT.py)

C.getMinValue: return the min value of variable 'ro':

```
min = C.getMinValue(a, 'ro') .or. min = C.getMinValue(A, 'ro')
```

(See: getMinValue.py) (See: getMinValuePT.py)

C.getMaxValue: return the max value of variable 'ro':

```
max = C.getMaxValue(a, 'ro') .or. max = C.getMaxValue(A, 'ro')
```

(See: getMaxValue.py) (See: getMaxValuePT.py)

C.getMeanValue: return the mean value of variable 'ro':

```
mean = C.getMeanValue(a, 'ro') .or. mean = C.getMeanValue(A, 'ro')
```

(See: getMeanValue.py) (See: getMeanValuePT.py)

C.getMeanRangeValue: return the mean value of variable 'ro' for the 20

```
mean = C.getMeanRangeValue(a, 'ro', 0.8, 1.) .or. mean = C.getMeanRangeValue(A, 'ro', 0.8, 1.)
```

(See: getMeanRangeValue.py) (See: getMeanRangeValuePT.py)

C.normL0, C.normL2: L0 and L2 norms of a field defined in an array can be extracted. If cellnature field is defined in the array, then blanked points are not taken into account into the computation of the norm:

```
L0norm = C.normL0(a, 'ro') .or. L0norm = C.normL0(A, 'ro')
```

(See: normL0.py) (See: normL0PT.py)

```
L2norm = C.normL2(a, 'ro') .or. L2norm = C.normL2(A, 'ro')
```

(See: normL2.py) (See: normL2PT.py)

C.normalize: Normalize a vector defined by its 3 vector coordinates. The vector component values are modified:

```
b = C.normalize(a, ['sx', 'sy', 'sz']) .or. B = C.normalize(A, ['sx', 'sy', 'sz'])
```

(See: normalize.py) (See: normalizePT.py)

C.magnitude: get the magnitude of a vector for each point of array:

```
b = C.magnitude(a, ['sx', 'sy', 'sz']) .or. B = C.magnitude(A, ['sx', 'sy', 'sz'])
```

(See: magnitude.py) (See: magnitudePT.py)

C.randomizeVar: randomize a variable of name varname. The modified field is bounded by [f-deltamin,f+deltamax] where f is the local field.

```
b = C.randomizeVar(a, varname, deltamin, deltamax) .or. B = C.randomizeVar(A, varname, deltamin,deltamax)
```

(See: randomizeVar.py) (See: randomizeVarPT.py)

1.7 Array / pyTree conversion

C.convertPyTree2ZoneNames: return the list of zone paths (strings) contained in a Python tree:

```
P = C.convertPyTree2ZoneNames(T)
```

(See: convertPyTree2ZoneNames.py)

C.convertPyTree2Array: convert a Python tree node to an array. One have to provide the path of the corresponding node and the Python tree:

```
a = C.convertPyTree2Array("Zone-001/GridCoordinates/CoordinateX", T)
```

(See: convertPyTree2Array.py)

1.8 File / arrays or File / pyTree conversion

C.convertFile2Arrays: read a file and return a list of arrays:

```
A = C.convertFile2Arrays(fileName, format=None, options)
```

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 one is adopted: **x, y, z, ro, rou, rov, row, roE, cellN**.
If format is unspecified, the format is guessed from file extension.
Several options are available to specify the discretization of vector elements (as defined in xfig or svg):

Option name	Meaning	Default value
nptsCurve	Number of discretization points for curved vector elements	20
nptsLine	Number of discretization points for line vector elements	2
density	Number of discretization points per unit length in the output curve	No default value (nptsLine, nptsCurve are used by default)

C.convertArrays2File: write a list of arrays to a file:

```
C.convertArrays2File(A, fileName, format=None, options)
```

The following additionnal options are available:

Option name	Meaning	Possible values	Default value
int	size of integer	4, 8	4
real	size of real	4, 8	8
endian	endianess	'little', 'big'	'big'
colormap (pov)	colormap style	1, 2,...	0
dataFormat	'printf' compatible data format (	'	'
zoneNames	list of zone names (struct zones, then unstruct zones)	['Zone1','Zone2','ZoneTa[]pon']	

C.convertFile2PyTree: read a file to a pyTree:

```
A = C.convertFile2PyTree('in.cgns', 'bin_cgns')
```

(See: convertFile2PyTree.py)

C.convertPyTree2File: write a Python tree to a file:

```
C.convertPyTree2File(A, 'out.cgns', 'bin_cgns')
```

(See: convertPyTree2File.py)

Recognised formats are:

- bin_tp: binary tecplot file (.plt).
  (See: conv.py)
- fmt_tp: formatted tecplot file (.dat, .tp).
  (See: conv2.py)
- bin_v3d: binary v3d file (.v3d).
- fmt_v3d: formatted v3d file (.fv3d).
  (See: conv3.py)

- bin_plot3d: binary plot3d file (.gbin).

- fmt_plot3d: formatted plot3d file (.gfmt).
  (See: convPlot3d.py)
- fmt_pov: formatted povray raytracer file (.pov).
  (See: convPov.py)
- bin_df3: binary density file for povray raytracer (.df3).
  (See: convDf3.py)
- fmt_mesh: formatted mesh (INRIA-Gamma) file (.mesh).
  (See: convMesh.py)
- fmt_gmsh: formatted GMSH file (.msh).
  (See: convGmsh.py)
- bin_gmsh: binary GMSH file (.msh).

- fmt_su2: formatted SU2 (Stanford) file (.su2).
  (See: convSu2.py)
- fmt_cedre: formatted CEDRE (Onera) file (.d).
  (See: convCedre.py)
- bin_stl: binary STL file (.stl).
  (See: convSTL.py)
- fmt_stl: formatted STL file (.fstl).
  (See: convFSTL.py)
- fmt_obj: formatted wavefornt Obj file (.obj).
  (See: convObj.py)
- bin_3ds: binary 3D studio file (.3ds).
  (See: conv3DS.py)
- bin_ply: stanford binary PLY file (.ply).
  (See: convPly.py)
- bin_pickle: binary Python pickle format (.ref).
  (See: convPickle.py)

- bin_wav: binary wav 8-bits sound file (.wav).
  (See: convWav.py)
- fmt_xfig: formatted xfig file (.fig).
  (See: convXfig.py)
- fmt_svg: formatted svg file (.svg).
  (See: convSvg.py)
- bin_png: binary png file (.png).
  (See: convPng.py)
- bin_cgns or bin_adf: binary ADF CGNS file for pyTrees only (.cgns, .adf).
- bin_hdf: binary HDF CGNS file for pyTrees only (.hdf, .hdf5).

1.9 Conversion to elsA profile

Following functions are specific to the elsA CGNS profile.

C.elsAProfile.convert2elsAxdt: convert a pyTree a to a pyTree b usable with elsA. If pyTree contains periodic matching joins, elsA parameter 'axis_ang_2' is set to 1.:

```
import Converter.elsAProfile
b = Converter.elsAProfile.convert2elsAxdt(a)
```

(See: convert2elsAxdtPT.py)

C.elsAProfile.addReferenceState: add a reference state node in each base. You must specify conservative as a list of reference conservative values. You can optionaly specify temp, a reference temperature, turbmod, the turbulence model in 'komega', 'kkl', 'smith', 'spalart', 'keps', 'rsm', and reference values for k and eps. Name of node and comments can also be specified:

```
B = Converter.elsAProfile.addReferenceState(A, conservative=[1.,...], temp=None, turbmod='spalart', name='ReferenceState', comments=None, k=None, eps=None)
```

(See: addReferenceStatePT.py)

C.elsAProfile.addGlobalConvergenceHistory: add a convergence node in each base. The type of norm used in residual computation can be specified (0: L0, 1: L2):

```
B = Converter.elsAProfile.addGlobalConvergenceHistory(A, normValue=0)
```

(See: addGlobalConvergenceHistoryPT.py)

C.elsAProfile.addFlowSolution: add a FlowSolution node for each zone. name is a suffix that can be appened to the 'FlowSolution' name. loc must be 'Vertex' or 'CellCenter' or 'cellfict'. governingEquation is the name of the 'GoverningEquation' node. output can be optionaly a dictionary

specifying the '.Solver#Output' node data. If addBCExtract is true, the boundary windows are also extracted. protocol is an optional string in 'iteration', 'end', 'after', specifying when extraction is performed:

```
B = Converter.elsAProfile.addFlowSolution(A, name='', loc='CellCenter', variables=None, governingEquations=None, writingMode=None, writingFrame='relative', period=None, output=None, addBCExtract=False, protocol="end")
```

```
B = Converter.elsAProfile.addFlowSolutionEoR(A, name='', loc='CellCenter', variables=None, governingEquations=None, writingMode=None, writingFrame='relative', period=None, output=None, addBCExtract=False, protocol="end")
```

(See: addFlowSolutionPT.py)

C.elsAProfile.addOutputForces: add an output forces node to a BC or a family BC node. name is an optional string to complete output node name, var is a list of variables to extract, writingmode specifies the elsA writing mode. period specifies the number of iterations between two extractions. pinf is the value of infinite stagnation pressure. fluxcoeff is an optional value used to correct the extracted value. torquecoeff is an optional value used to correct the extracted value. xyz torque is the origin for torque. frame must be in ['relative','absolute']:

```
B = Converter.elsAProfile.addOutputForces(A, name="", var=None, loc=4, writingmode=1, period=None, pinf=None, fluxcoef=None, torquecoef=None, xyztorque=None, frame=None, governingEquations="NSTurbulent", xtorque=None, ytorque=None, ztorque=None)
```

(See: addOutputForcesPT.py)

C.elsAProfile.addOutputFriction: add an output friction node to a BC or a family BC node. name is an optional string to complete output node name, var is a list of variables to extract. loc specifies the location of extraction. writingmode specifies the elsA writing mode. period specifies the number of iterations between two extractions. fluxcoeff is an optional value used to correct the extracted value:

```
B = Converter.elsAProfile.addOutputFriction(A, name="", var=None, loc=4, writingmode=1, period=None, fluxcoef=None, torquecoef=None, writingframe=None)
```

(See: addOutputFrictionPT.py)

1.10 Preconditionning (hook)

C.createHook: create a hook for each zone in a. The hook is used to store pre-computed data on a, and can be reused by a function if the zones defined in a are not modified. Parameter 'function' can be:

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 functions
'adt'	Bounding boxes of cells stored in an ADT. Valid for 3D structured and TETRA and 2D structured zones (nk=1 and z constant).	Connector.setInterpData, Connector.setIBCData functions
'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'	Element centers stored in a k-d tree	Converter.identifyElements, Converter.nearestElements

Function usage:

```
hook = C.createHook(a, function)
```

(See: createHook.py) (See: createHookPT.py)

C.createGlobalHook: create a global hook on a preconditioning tree. If extended=0, returns the hook in a list. If extended=1, returns the hook and an indirection numpy array of block number for each point stored in the preconditioning tree defined in the hook.
Parameter 'function' can be:

function name	Type of storage	Usage
'nodes'	Mesh nodes stored in a global k-d tree	Converter.identifySolutions
'faceCenters'	Mesh face centers stored in a global k-d tree	Converter.identifySolutions
'elementCenters'	Element centers stored in a global k-d tree	Converter.identifySolutions

```
hook = C.createGlobalHook(A, function='None')
```

(See: createGlobalHook.py) (See: createGlobalHookPT.py)

C.freeHook: free a hook created by createHook:

```
C.freeHook(hook)
```

(See: freeHook.py) (See: createHookPT.py)

1.11 Geometrical identification

C.identifyNodes: identify nodes of a with points stored in hook. Return the indices of hook corre-

sponding to the nodes of a:

```
C.identifyNodes(hook, a, tol=1.e-12)
```

(See: identifyNodes.py) (See: identifyNodesPT.py)

C.identifyFaces: identify face centers of a with points stored in hook. Return the indices of hook corresponding to the faces of a:

```
C.identifyFaces(hook, a, tol=1.e-12)
```

(See: identifyFaces.py) (See: identifyFacesPT.py)

C.identifyElements: identify element centers of a with points stored in hook. Return the indices of hook corresponding to the elements of a:

```
C.identifyElements(hook, a, tol=1.e-12)
```

(See: identifyElements.py) (See: identifyElementsPT.py)

C.identifySolutions: Copy the field defined on given donor zones to receptor zones where points are at a distance inferior to tol.

This function doesn't perform interpolation but directly copy the field of the nearest donor points, if it is nearer than tol. Otherwise the field is set to 0.

The array interface returns the arrays corresponding to fields defined in solDnr. The arrays coordsRcv are the receptor coordinate arrays. Donor coordinates are stored in a kdtree hook:

```
b = C.identifySolutions(coordsRcv, solDnr, hookDnr, vars=[], tol=1.e-12)
```

The pyTree interface requires a donor and a receptor pyTree. hookN can store the coordinates of donor nodes, and hookC can store the coordinates of donor cell centers. If hookN=None or hookC=None, then solution is not identified for nodes or centers respectively. The variables that must be identified can be specified by the list of string vars. If vars=[], then all the variables of tDnr are set to tRcv:

```
t = C.identifySolutions(tRcv, tDnr, hookN=None, hookC=None, vars=[], tol=1.e-12)
```

(See: identifySolutions.py) (See: identifySolutionsPT.py)

C.nearestNodes: find nearest points stored in hook to nodes of a. Return the indices of hook nearest to the nodes of a and the corresponding distance:

```
C.nearestNodes(hook, a)
```

(See: nearestNodes.py) (See: nearestNodesPT.py)

C.nearestFaces: find nearest points stored in hook to face centers of a. Return the indices of hook nearest to the faces of a and the corresponding distance:

```
C.nearestFaces(hook, a)
```

(See: nearestFaces.py) (See: nearestFacesPT.py)

C.nearestElements: find nearest points stored in hook to element centers of a. Return the indices of hook nearest to the elements of a and the corresponding distance:

```
C.nearestElements(hook, a)
```

(See: nearestElements.py) (See: nearestElementsPT.py)

1.12 Distributed trees

Converter can also manage distributed trees, that is trees where zones are distributed over different processors. Three new concepts are introduced in addition of standard pyTrees: the skeleton tree, the loaded skeleton tree and the partial tree.
A skeleton tree (S) is a full pyTree where float numpy arrays with more than 100 elements are replaced by None.
A loaded skeleton tree (LS) is a skeleton tree with zones attributed to the current processor fully loaded.
A partial tree (P) is a pyTree with only zones attributed to the current processor fully loaded. It can be viewed as a ”cleaned” loaded skeleton tree.

Access to distributed tree functions is provided by:

```
import Converter.Mpi as Cmpi
```

Cmpi.convertFile2SkeletonTree: return a skeleton tree from a file (only ADF or HDF). If float data array size is lower than maxFloatSize then the array is loaded. If maxDepth is specified, load is limited to maxDepth levels:

```
A = Cmpi.convertFile2SkeletonTree('in.cgns', maxFloatSize=5, maxDepth=-1)
```

(See: convertFile2SkeletonTreePT.py)

Cmpi.readZones: load zones of a skeleton tree referenced by proc number in a .Solver#Param/proc node (only ADF or HDF):

```
B = Cmpi.readZones(A, 'in.cgns', proc=0)
```

(See: readZonesPT.py)

Cmpi.convertPyTree2File: write a loaded skeleton tree or a partial tree to a file:

```
Cmpi.convertPyTree2File(A, 'in.cgns', proc=0)
```

Cmpi.convert2PartialTree: convert a loaded skeleton tree to a partial tree If rank=-1, get rid of skeleton zones. If rank¿=0, get rid of zones with a proc different of rank:

```
B = Cmpi.convert2PartialTree(A, rank=-1)
```

(See: convert2PartialTreePT.py)

Cmpi.createBBoxTree: from a partial tree or a loaded skeleton tree, create a full tree containing bbox of zones (identical on all processors). Argument method can be 'AABB' (axis aligned bbox) or 'OBB' (oriented bbox):

```
B = Cmpi.createBBoxTree(A, method='AABB')
```

(See: createBBoxTreePT.py)

Cmpi.getProc: return the proc of a zone (can be a skeleton zone). If zone has not been affected, return -1:

```
proc = Cmpi.getProc(z)
```

(See: getProcPT.py)

Cmpi.setProc: set proc to a (can be skeleton):

```
b = Cmpi.setProc(a) .or. B = Cmpi.setProc(A)
```

(See: setProcPT.py)

Cmpi.getProcDict: return the dictionary proc['zoneName'] from a skeleton, loaded skeleton or partial tree:

```
procDict = Cmpi.getProcDict(A)
```

(See: getProcDictPT.py)

Cmpi.computeGraph: compute a graph. The graph is a dictionary such that graph[proc1][proc2] contains the names of zones of proc1 that are "connected" to at least one zone on proc2.

If type='bbox', a zone is connected to another if their bbox intersects. A must be a bbox tree.
If type='bbox2', a zone is connected to another if their bbox intersects and are not in the same base. A must be a bbox tree.
If type='bbox3', a zone is connected to another of another tree if their bbox intersects. A and t2 must be a bbox tree.

If type='match' (S/LS/P), a zone is connected to another if they have a match between them. A can be a skeleton, loaded skeleton or a partial tree.

If type='ID' (S/LS/P), a zone is connected to another if they have interpolation data between them. A can be a skeleton, a loaded skeleton or a partial tree.

If type='IBCD' (S/LS/P), a zone is connected to another if they have IBC data between them. A can be a skeleton, a loaded skeleton or a partial tree.

If type='ALLD' (S/LS/P), a zone is connected to another if they have Interpolation or IBC data between them. A can be a skeleton, a loaded skeleton or a partial tree.

If type='proc', a zone is attributed to another proc than the one it is loaded on. A can be a skeleton, a loaded skeleton tree or a partial tree.

```
graph = Cmpi.computeGraph(A, type='bbox', t2=None)
```

(See: computeGraphPT.py)

Cmpi.addXZones: add zones loaded on a different processor that are connected to local zones through the graph:

```
B = Cmpi.addXZones(A, graph)
```

(See: addXZonesPT.py)

Cmpi.rmXZones: remove zones created by addXZones:

```
B = Cmpi.rmXZones(A)
```

(See: rmXZonesPT.py)

Cmpi.center2Node: center2Node for distributed trees:

```
B = Cmpi.center2Node(A, 'centers:Density')
```

(See: center2NodeDPT.py)

1.13 Client/server to exchange arrays/pyTrees

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

```
sockets = C.createSockets(nprocs=1, port=15555)
```

C.listen: listen from created sockets. A is whatever has been sent:

```
A = C.listen(sockets)
```

(See: listen.py) (See: listenPT.py)

C.send: send information to server:

```
C.send(a, 'localhost', rank=0, port=15555)
```

(See: send.py) (See: sendPT.py)

1.14 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...

C.Array3D.convertArrays2Arrays3D: convert arrays to 3D arrays (ni,nj,nk):

```
B = C.Array3D.convertArrays2Arrays3D(A)
```

(See: convertArray2Array3D.py)

C.Array3D.convertArrays3D2Arrays: convert 3D arrays to arrays:

```
B = C.Array3D.convertArrays3D2Arrays(A)
```

(See: convertArray3D2Array.py)

1.15 More general examples of use

- See: Examples/Converter/restart.py
- See: Examples/Converter/conv8.py
- See: Examples/Converter/shell.py

1.16 Example files

Example file: arrayK.py

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

# Structured
b = C.array('x,y,z', 12, 9, 12); print b

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

Example file: getValue.py

```
# - 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) )
print C.getValue( a, 9 ) # It is the same point!

# 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 )
```

Example file: setValue.py

```
# - 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.])
```

Example file: addVar.py

```
# - 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 file: addVars.py

```
# - 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')
```

Example file: copya.py

```
# - 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")
```

Example file: newPyTree.py

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

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

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

# Create a tree with 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')
```

Example file: getNobOfBasePT.py

```
# - 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, topTree=t); print nob
# This means that t[2][nob] = b
```

Example file: getNobNozOfZonePT.py

```
# - 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, topTree=t); print nob, noz
# This means that t[2][nob][2][noz] = a
```

Example file: breakConnectivityPT.py

```
# - 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')
```

Example file: mergeConnectivityPT.py

```
# - 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)
# 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')
```

Example file: mergeTrees.py

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

t1 = C.newPyTree(['Base1', 2, 'Base2', 3])
```

```
t2 = C.newPyTree(['Other1', 'Other2', 'Base2'])
t = C.mergeTrees(t1, t2)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: deleteEmptyZones.py

```
# - 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])
t = C.deleteEmptyZones(t)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: addBase2PyTree.py

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

t = C.newPyTree(['Base', 3]) # must contain volume zones
t = C.addBase2PyTree(t, 'Base2', 2) # must contain surface zones
print t
```

Example file: addStatePT.py

```
# - 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)

# Specifie un etat de reference adimensionne par:
# Mach, alpha, Re, MutSMu, TurbRate (adim1)
C._addState(t, adim='adim1', MInf=0.5, alphaZ=0., alphaY=0.,
            ReInf=1.e8, MutSMuInf=0.2, TurbRateInf=1.e-8)

# Specifie un etat de reference adimensionne par:
# Mach, alpha, Re, MutSMu, TurbRate (adim2)
C._addState(t, adim='adim2', MInf=0.5, alphaZ=0., alphaY=0.,
            ReInf=1.e8, MutSMuInf=0.2, TurbRateInf=1.e-8)

# Specifie un etat de reference dimensionne par:
# U, T, P, L, MutSMu, TurbRate (dim1)
C._addState(t, adim='dim1', UInf=35, TInf=294., PInf=101325, LInf=1.,
            alphaZ=0., alphaY=0., MutSMuInf=0.2, TurbRateInf=1.e-8)

# Specifie un etat de reference dimensionne par:
```

```
# U, T, Ro, L, MutSMu, TurbRate (dim2)
C._addState(t, adim='dim2', UInf=35, TInf=294., RoInf=1.2, LInf=1.,
            alphaZ=0., alphaY=0., MutSMuInf=0.2, TurbRateInf=1.e-8)

# Specifie un etat de reference dimensionne par:
# U, P, Ro, L, MutSMu, TurbRate (dim2)
C._addState(t, adim='dim3', UInf=35, PInf=101325., RoInf=1.2, LInf=1.,
            alphaZ=0., alphaY=0., MutSMuInf=0.2, TurbRateInf=1.e-8)
```

Example file: addState2PT.py

```
# - 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])
t = C.addState(t, adim='adim1', MInf=0.5)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: addChimera2Base.py

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

Example file: addBC2Zone.py

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

# - Structured grids -
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')
```

Example file: addBC2ZoneU.py

```
# - 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))
a = C.addBC2Zone(a, 'wall', 'BCWall', faceList=[1,2])

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

Example file: fillEmptyBCWith.py

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

Example file: rmBCOfType.py

```
# - rmBCOfType (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])
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']); t[2][1][2] += [a,b]
t[2][1] = C.addFamily2Base(t[2][1], 'LOIN', bndType='BCFarfield')

t = C.rmBCOfType(t, 'BCWall')
t = C.rmBCOfType(t, 'BCMatch')
t = C.rmBCOfType(t, 'BCFarfield')
t = C.rmBCOfName(t, 'FamilySpecified:LOIN')
C.convertPyTree2File(t, 'out.cgns')
```

Example file: rmBCOfNamePT.py

```
# - 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])
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')
t = C.rmBCOfName(t, 'match*')
t = C.rmBCOfName(t, 'FamilySpecified:LOIN')

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

Example file: extractBCOfTypePT.py

```
# - 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')
t = C.newPyTree(['Base',3,'Skin',2]); t[2][1][2] += [a]; t[2][2][2] += Z
C.convertPyTree2File(t, 'out.cgns')
```

Example file: extractBCOfNamePT.py

```
# - 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')

t[2][2][2] += Z1+Z2
C.convertPyTree2File(t, 'out.cgns')
```

Example file: getEmptyBC.py

```
# - 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)
wins = C.getEmptyBC(t,2); print wins
# Returns undefined windows (as face list)
t= C.convertArray2NGon(t)
faceList = C.getEmptyBC(t,2); print faceList

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

Example file: getConnectedZonesPT.py

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

Example file: addFamily2Base.py

```
# - 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
t[2][1] = C.addFamily2Base(t[2][1], 'flap', 'BCWall')
# Add just a family name
t[2][1] = C.addFamily2Base(t[2][1], 'component1')
C.convertPyTree2File(t, 'out.cgns')
```

Example file: tagWithFamilyPT.py

```
# - 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))
a = C.tagWithFamily(a, 'CYLINDER')
b = C.tagWithFamily(b, 'CART')

t = C.newPyTree(['Base',a,b])
t[2][1] = C.addFamily2Base(t[2][1], 'CYLINDER')
t[2][1] = C.addFamily2Base(t[2][1], 'CART')

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

Example file: getFamilyZonesPT.py

```
# - 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))

a = C.tagWithFamily(a, 'CYLINDER')
b = C.tagWithFamily(b, 'CYLINDER')
c = C.tagWithFamily(c, 'CART')

t = C.newPyTree(['Base',a,b,c])
t[2][1] = C.addFamily2Base(t[2][1], 'CYLINDER')
t[2][1] = C.addFamily2Base(t[2][1], 'CART')

zones = C.getFamilyZones(t, 'CYLINDER')
t2 = C.newPyTree(['Base'])
t2[2][1][2] += zones
t2[2][1] = C.addFamily2Base(t2[2][1], 'CYLINDER')

C.convertPyTree2File(t2, 'out.cgns')
```

Example file: getFamilyBCsPT.py

```
# - getFamilyBCs (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])
t[2][1] = C.addFamily2Base(t[2][1], 'CARTER', bndType='BCWall')

B1 = C.getFamilyBCs(t, 'CARTER'); print B1
```

Example file: getFamilyZoneNamesPT.py

```
# - 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))

a = C.tagWithFamily(a, 'CARTER')
b = C.tagWithFamily(b, 'CARTER')

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

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

Example file: getFamilyBCNamesOfTypePT.py

```
# - 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])

t[2][1] = C.addFamily2Base(t[2][1], 'CARTER', bndType='BCWall')

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

Example file: rmNodes.py

```
# - rmNodes (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.addVars(a, ['Density', 'centers:cellN', 'rou', 'rov', 'Hx', 'Hy'])
b = C.rmNodes(a, 'FlowSolution#Centers')
t = C.newPyTree(['Base']); t[2][1][2].append(b)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: getValuePT.py

```
# - 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) )
print C.getValue( a, 'CoordinateX', 9 ) # It's the same point
print C.getValue( a, 'nodes:CoordinateX', 9 ) # It's the same point
print C.getValue( a, 'GridCoordinates', 9 ) # return [x,y,z]
print C.getValue( a, ['CoordinateX', 'CoordinateY'], 9 ) # return [x,y]

# 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 )
```

Example file: setValuePT.py

```
# - 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
```

Example file: setPartialFieldsPT.py

```
# - 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')
```

Example file: setPartialFields1PT.py

```
# - setPartialFields1 (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C
import numpy

a = G.cart((0,0,0), (1,1,1), (10,10,10))
a = C.initVars(a, 'F=2.')
a = C.initVars(a, 'G=2.')
f1 = numpy.array([0.,0.,0.,0.,0.,0.,0.,0.,0.,0.], dtype=numpy.float64)
inds = numpy.array([0,1,2,3,4,5,6,7,8,9], dtype=numpy.int32)
t = C.newPyTree(['Base']); t[2][1][2] += [a]
t = C.setPartialFields1(t, [ [[],f1]] , [inds], loc='nodes')
C.convertPyTree2File(t, 'out.cgns')
```

Example file: addVarsPT.py

```
# - 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')
```

Example file: fillMissingVariablesPT.py

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

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'
```

```
a = C.addVars(a, 'rou'); a = C.addVars(a, 'rov')
a = C.addVars(a, 'centers:cellN')
a = C.addVars(a, ['Density', 'Hx', 'centers:Hy'])

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

Example file: cpVars.py

```
# - 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'
a = C.initVars(a, 'Density',2.)
a = C.cpVars(a, 'Density', a, 'Density2')
c = C.cpVars(a, 'Density', b, 'Density')
t = C.newPyTree(['Base',a,c])
C.convertPyTree2File(t, 'out.cgns')
```

Example file: printTree2PT.py

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

a = G.cart((0,0,0), (1,1,1), (10,10,10))
t = C.newPyTree(['Base',a])
C.printTree(t)
```

Example file: getVarNames.py

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

Example file: getVarNamesPT.py

```
# - getVarNames (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.addVars(a, ['Density', 'centers:cellN'])
print C.getVarNames(a, loc='nodes')
print C.getVarNames(a, loc='centers')
print C.getVarNames(a, excludeXYZ=True, loc='both')
```

Example file: isNamePresent.py

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

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

b = G.cart( (0,0,0), (1,1,1), (50,50,50) )
b = C.initVars(b, 'G', 2.)

print C.getVarNames(a)
```

```
print C.getVarNames([a, b])

print C.isNamePresent(a, 'F')
print C.isNamePresent([a, b], 'F')
print C.isNamePresent([a, b], 'K')
```

Example file: isNamePresentPT.py

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

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

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

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

print C.getVarNames(a)
print C.getVarNames([a, b])
print C.getVarNames(t[2][1])
print C.getVarNames(t)

print C.isNamePresent(a, 'F')
print C.isNamePresent(a, 'centers:F')
print C.isNamePresent(a, 'centers:G')
print C.isNamePresent([a, b], 'F')
print C.isNamePresent([a, b], 'centers:G')

print C.isNamePresent(t[2][1], 'F')
print C.isNamePresent(t[2][1], 'centers:G')
print C.isNamePresent(t, 'F')
print C.isNamePresent(t, 'centers:G')
```

Example file: getNPts.py

```
# - 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
```

Example file: getNCells.py

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

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

Example file: initVars.py

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

Example file: initVarsPT.py

```
# - 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')
```

Example file: initVar.py

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

# 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")
```

Example file: initVarsByEq.py

```
# - 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')
```

Example file: initVarsByEqPT.py

```
# - 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')
```

Example file: extractVars.py

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

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
a = C.addVars(a, 'F')
# 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")
```

Example file: extractVarsPT.py

```
# - 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'])
t = C.newPyTree(['Base',a])
C.convertPyTree2File(t, "out.cgns")
```

Example file: rmVars.py

```
# - 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')
```

Example file: rmVarsPT.py

```
# - 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, 'Density')
b = C.rmVars(b, 'Density')
C.convertPyTree2File(b, 'out.cgns')
```

Example file: convertStruct2Tetra.py

```
# - 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], 'new1.plt', 'bin_tp')

# 3D : tetrahedras
a = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,10,10))
b = C.convertArray2Tetra(a)
C.convertArrays2File([b], 'new2.plt', 'bin_tp')
```

Example file: convertArray2TetraPT.py

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

t = C.newPyTree(['Base1',2,'Base2',3])

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

# 3D : tetrahedras
a = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,10,10))
b = C.convertArray2Tetra(a); t[2][2][2].append(b)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: convertHexa2Tetra.py

```
# - 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], 'new1.plt', 'bin_tp')

# 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], 'new2.plt', 'bin_tp')
```

Example file: convertPrism2Tetra.py

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

a = G.cartPenta((0.,0.,0.), (1,1,1), (10,10,3))
a = C.convertArray2Tetra(a)
C.convertArrays2File([a], 'out.plt', 'bin_tp')
```

Example file: convertStruct2Hexa.py

```
# - 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')
```

Example file: convertArray2HexaPT.py

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

t = C.newPyTree(['Base',3,'Base2',2])

# 2D: quad
a = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,10,1))
b = C.convertArray2Hexa(a); t[2][2][2].append(b)

# 3D: hexa
a = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,10,10))
b = C.convertArray2Hexa(a); t[2][1][2].append(b)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: convertArray2NGon.py

```
# - 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)
```

Example file: convertArray2NGonPT.py

```
# - 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')
```

Example file: convertArray2Node.py

```
# - 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')
```

Example file: convertArray2NodePT.py

```
# - 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)
t = C.newPyTree(['Base1',2,a])
C.convertPyTree2File(t, 'out.cgns')
```

Example file: convertBAR2Struct.py

```
# - 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')
```

Example file: convertBAR2StructPT.py

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

Example file: convertTri2Quad.py

```
# - 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')
```

Example file: convertTri2QuadPT.py

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

Example file: conformizeNGon.py

```
# - 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')
```

Example file: conformizeNGonPT.py

```
# - 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)
t = C.newPyTree(['Base',res])
C.convertPyTree2File(t, 'out.cgns')
```

Example file: node2Center.py

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

def F(x,y): return 2*x+y

ni = 30; nj = 40; nk = 1
a = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,nk))
a = C.initVars(a, 'ro', F, ['x','y'])
ac = C.node2Center(a)
C.convertArrays2File([a,ac], "out.plt")
```

Example file: node2CenterPT.py

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

def F(x,y): return 2*x+y

ni = 30; nj = 40; nk = 3
a = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,nk))
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), (10./(ni-1),10./(nj-1),1), (ni,nj,nk))
a = C.initVars(a, 'Density', F, ['CoordinateX','CoordinateY'])
b = C.node2Center(a); b[0] = a[0]+'_centers'
C.convertPyTree2File([a,b], 'out2.cgns')
```

Example file: center2Node.py

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

ni = 30; nj = 40; nk = 10
a = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,nk))
a = C.initVars(a, 'ro', 1.)
an = C.center2Node(a)
C.convertArrays2File([an], "out.plt")
```

Example file: center2NodePT.py

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

def F(x,y): return 2*x+y

# center2Node: create a new zone
ni = 30; nj = 40; nk = 2
a = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,nk))
a = C.initVars(a, 'centers:Density', 1.)
b = C.center2Node(a); b[0] = a[0]+'_nodes'
t = C.newPyTree(['Base1',3,b])
C.convertPyTree2File(t, 'out0.cgns')

# center2Node: modify a variable
ni = 30; nj = 40; nk = 3
a = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,nk))
a = C.initVars(a, 'centers:Density', 1.)
a = C.center2Node(a, 'centers:Density')
t = C.newPyTree(['Base',3,a])
C.convertPyTree2File(t, 'out.cgns')
```

Example file: node2ExtCenter.py

```
# - node2ExtCenter (array) -
import Converter as C
import Generator as G

ni = 30; nj = 40; nk = 1
a = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,nk))
ac = C.node2ExtCenter(a)
C.convertArrays2File([a,ac], "out.plt")
```

Example file: node2ExtCenterPT.py

```
# - node2ExtCenter (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

ni = 30; nj = 40; nk = 1
a = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,nk))
ac = C.node2ExtCenter(a)
t = C.newPyTree(['Base',2]); t[2][1][2].append(ac)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: diffArrays.py

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

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

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

ret = C.diffArrays([a], [b]); print ret
```

Example file: diffArraysPT.py

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

ni = 11; nj = 11; nk = 11
a = G.cart( (0,0,0), (1,1,1), (ni,nj,nk) )
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.)
ret = C.diffArrays(a1, a2)
```

Example file: getArgMin.py

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

a = G.cart((0,0,0), (1.,1.,1.), (10,10,10))
a = C.initVars(a, 'F={x}+{y}+{z}')
argmin = C.getArgMin(a, 'F'); print argmin
```

Example file: getArgMinPT.py

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

a = G.cart((0,0,0), (1.,1.,1.), (10,10,10))
argmin = C.getArgMin(a, 'CoordinateX'); print argmin
```

Example file: getArgMax.py

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

a = G.cart( (0,0,0), (1.,1.,1.), (10,10,10) )
argmax = C.getArgMax(a, 'x'); print argmax
```

Example file: getArgMaxPT.py

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

# test sur une zone
a = G.cart( (0,0,0), (1.,1.,1.), (10,10,10) )
argmax = C.getArgMax(a, 'CoordinateX'); print argmax
```

Example file: getMinValue.py

```
# - 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
```

Example file: getMinValuePT.py

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

a = G.cart((0,0,0), (1.,1.,1.), (11,1,1))
minval = C.getMinValue(a, 'CoordinateX'); print minval
minval = C.getMinValue(a, ['CoordinateX', 'CoordinateY']); print minval
minval = C.getMinValue(a, 'GridCoordinates'); print minval
```

Example file: getMaxValue.py

```
# - 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
```

Example file: getMaxValuePT.py

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

a = G.cart((0,0,0), (1.,1.,1.), (11,2,2))
maxval = C.getMaxValue(a, 'CoordinateX'); print maxval
maxval = C.getMaxValue(a, ['CoordinateX', 'CoordinateY']); print maxval
maxval = C.getMaxValue(a, 'GridCoordinates'); print maxval
```

Example file: getMeanValue.py

```
# - 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
```

Example file: getMeanValuePT.py

```
# - 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
```

Example file: getMeanRangeValue.py

```
# - getMeanRangeValue (array) -
import Converter as C
import Generator 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, 'x', 0., 0.3); print meanval
```

Example file: getMeanRangeValuePT.py

```
# - 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
```

Example file: normL0.py

```
# - 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")
```

Example file: normL0PT.py

```
# - normL0 (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 'normL0 = ', C.normL0(a, "F")
```

Example file: normL2.py

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

ni = 11; nj = 11; nk = 11
a = G.cart( (0,0,0), (1,1,1), (ni,nj,nk) )
a = C.initVars(a, "F", 1.)
print 'normL2 = ', C.normL2(a, "F")

# cellN variable IS taken into account
cellnf = C.array('celln', ni, nj, nk)
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")
```

Example file: normL2PT.py

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

ni = 11; nj = 11; nk = 11
a = G.cart((0,0,0), (1,1,1), (ni,nj,nk))

# Add variable F
a = C.addVars(a, "F")
a = C.initVars(a, "F", 1.)
print 'normL2 = ', C.normL2(a, "F")
```

Example file: normalize.py

```
# - 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')
```

Example file: normalizePT.py

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

Example file: magnitude.py

```
# - 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.center2Node(n)
n[1] = n[1]*10
n = C.magnitude(n, ['sx','sy','sz'])
a = C.addVars([a, n])
C.convertArrays2File([a], 'out.plt')
```

Example file: magnitudePT.py

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

Example file: randomizeVar.py

```
# - 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")
```

Example file: randomizeVarPT.py

```
# - randomizeVar (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
a = G.cart((0,0,0),(1,1,1),(11,11,1))
a = C.initVars(a, 'F=10.')
b = C.randomizeVar(a, 'F', 0.1, 1.)
C.convertPyTree2File(b, "out.cgns")
```

Example file: convertPyTree2ZoneNames.py

```
# - convertPyTree2ZoneNames (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))
b = G.cartHexa((0.,0.,0.),(0.1,0.1,0.1),(11,11,11)); b[0] = 'cartHexa'
t = C.newPyTree(['Base']); t[2][1][2] += [a,b]
zones = C.convertPyTree2ZoneNames(t); print zones
```

Example file: convertPyTree2Array.py

```
# - convertPyTree2Array (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter
```

```
a = G.cart((0.,0.,0.),(0.1,0.1,0.1),(11,11,11))
a = C.initVars(a,'F',1.); a = C.initVars(a,'centers:G',2.)
a = C.addBC2Zone(a, 'wall1', 'BCWall', 'imin')
b = G.cartHexa((0.,0.,0.),(0.1,0.1,0.1),(11,11,11)); b[0] = 'cartHexa'
t = C.newPyTree(['Base']); t[2][1][2] = t[2][1][2] + [a,b]
t[2][1] = C.addState(t[2][1], 'Mach', 0.6)

zones = C.convertPyTree2ZoneNames(t)

# 1 - Get one field
arrays = []
for i in zones:
    a = C.convertPyTree2Array(i+"/GridCoordinates/CoordinateX", t)
    b = C.convertPyTree2Array(i+"/GridCoordinates/CoordinateY", t)
    c = C.convertPyTree2Array(i+"/GridCoordinates/CoordinateZ", t)
    x = Converter.addVars([a,b,c])
    arrays.append(x)
Converter.convertArrays2File(arrays, "out.plt")

# 2 - Get a global field
arrays = []
for i in zones:
    a = C.convertPyTree2Array(i+"/GridCoordinates", t)
    arrays.append(a)
Converter.convertArrays2File(arrays, "out2.plt")

# 3 - Get the flow solution
a = C.convertPyTree2Array(zones[0]+'/FlowSolution', t)
print a
```

Example file: convertFile2PyTree.py

```
# - 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
```

Example file: convertPyTree2File.py

```
# - 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']); t[2][1][2] += [a]
C.convertPyTree2File(t, 'out.cgns')
C.convertPyTree2File(t, 'out.plt')
```

Example file: conv.py

```
# - convertFile2Arrays (binary tecplot) -
# - convertArrays2File (binary tecplot) -
# Read a file to a list of arrays
# Select some blocks in arrays
# Write the new arrays
import Converter as C
```

```
# Read file 'in.plt' to arrays
arrays = C.convertFile2Arrays("in.plt", "bin_tp")
n = len(arrays)
print 'number of blocks in file: ', n

# arrays is a list of n arrays.

# Write dimensions of first block : arrays[0]
print 'dimensions: ', arrays[0][2], arrays[0][3], arrays[0][4]

# Write variable list of first block : arrays[0]
print 'variables: ', arrays[0][0]

# Write coord. of first point of first block.
# ! : Note that index is not natural for Fortran programmers
print 'x,y,z of first element: ',\
      arrays[0][1][0,0], arrays[0][1][1,0],  arrays[0][1][2,0]

# Select blocks 1 to 3 (first block is 0) and put them in a list
# named 'res'
res = arrays[1:4]

# Add block 5 at the end of list
res.append(arrays[5])

# Insert block 6 at beginning of list
res.insert(0, arrays[6])

# Save arrays to file 'out.plt' (bin_tp format)
C.convertArrays2File(res, 'out.plt')
```

Example file: conv2.py

```
# - convertFile2Arrays (formatted tecplot) -
# - convertArrays2File (binary tecplot) -
import Converter as C

# Read a file into arrays
arrays = C.convertFile2Arrays("in360.tp", "fmt_tp")

# Write arrays to file
C.convertArrays2File(arrays, "out.plt", "bin_tp")
```

Example file: conv3.py

```
# - convertFile2Arrays (binary v3d) -
# - convertArrays2File (formatted and binary v3d) -
import Converter as C

# Read a file into arrays
arrays = C.convertFile2Arrays("infmt.v3d", "fmt_v3d")

# Write a binary file with endian conversion
C.convertArrays2File(arrays, "out.v3d", "bin_v3d", endian='big')

# Write a binary file with "elsA v3d format"
C.convertArrays2File(arrays, "out_elsa.v3d", "fmt_v3d", dataFormat='%14.7e')
```

Example file: convPlot3d.py

```
# - convertFile2Arrays (binary plot3d) -
# - convertArrays2File (binary plot3d) -
import Converter as C

# Read a file into arrays
arrays = C.convertFile2Arrays("in.dat", "bin_plot3d")
arrays = C.addVars(arrays, ["ro", "rou", "rov", "row", "roE"])

# Write arrays to file
C.convertArrays2File(arrays, "out.dat", "bin_plot3d", ['int',8])
C.convertArrays2File(arrays, "out.plt", "bin_tp")
```

Example file: convPov.py

```
# - convertArrays2File (fmt_pov) -
import Converter as C

a = C.convertFile2Arrays("dauphin_skin.plt", "bin_tp")

# fmt_pov only supports triangle meshes
b = C.convertArray2Tetra(a)

# This converts coordinates and Density field to fmt_pov
# using colormap 1 (c1). c0 means no color pigment.
C.convertArrays2File(b, "out.pov", "fmt_pov", ['colormap',1])

# Reread this file
a = C.convertFile2Arrays("out.pov", "fmt_pov")
C.convertArrays2File(a, "out.plt", "bin_tp")

# Execute povray
import os; os.system("povray -W800 -H600 +a0.3 +SP16 render.pov +P")
```

Example file: convDf3.py

```
# - convertArrays2File (bin_df3) -
import Converter as C
import Generator as G

a = G.cart( (0,0,0), (1,1,1), (11,11,11) )
a = C.addVar(a, 'ro')

# Convert density field to povray density file
C.convertArrays2File([a], 'out.df3', 'bin_df3')
```

Example file: convMesh.py

```
# - convertArrays2File (INRIA mesh format) -
import Converter as C

# Read mesh file
a = C.convertFile2Arrays("falcon.mesh", "fmt_mesh")
C.convertArrays2File(a, "out.plt", "bin_tp")

# Rewrite mesh file
C.convertArrays2File(a, "out.mesh", "fmt_mesh")
```

Example file: convGmsh.py

```
# - convertArrays2File (GMSH file format) -
import Converter as C
import Generator as G

a = G.cartHexa( (0,0,0), (1,1,1), (3,3,3) )
b = G.cartTetra( (5,0,0), (1,1,1), (3,3,3) )

# write
C.convertArrays2File([a,b], 'out.msh', 'fmt_gmsh')

# Reread
a = C.convertFile2Arrays("out.msh", "fmt_gmsh")
```

Example file: convSu2.py

```
# - convertArrays2File (SU2 format) -
import Converter as C

# Read mesh file
a = C.convertFile2Arrays("mesh_naca.su2", "fmt_su2")
C.convertArrays2File(a, "out.plt", "bin_tp")

# Rewrite mesh file
C.convertArrays2File(a, "out.su2", "fmt_su2")
```

Example file: convCedre.py

```
# - convertArrays2File (Cedre fmt format) -
import Converter as C
import Generator as G

# Create grid
a = G.cartNGon((0,0,0), (1,1,1), (10,10,10))
C.convertArrays2File([a], "out.d", "fmt_cedre")

# Reread file
a = C.convertFile2Arrays("out.d", "fmt_cedre")
```

Example file: convSTL.py

```
# - convertFile2Arrays (binary STL) -
import Converter as C

a = C.convertFile2Arrays('Data/piece-a-percer.stl', 'bin_stl')
C.convertArrays2File(a, 'out.plt', 'bin_tp')
```

Example file: convFSTL.py

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

a = G.cartTetra((0,0,0), (1,1,1), (10,10,1))
a = C.convertArrays2File([a], 'out.stl', 'fmt_stl')
a = C.convertFile2Arrays('out.stl', 'fmt_stl')
```

Example file: convObj.py

```
# - convertFile2Arrays (formatted Obj) -
import Converter as C

a = C.convertFile2Arrays('Data/cube.obj', 'fmt_obj')
C.convertArrays2File(a, 'out.plt', 'bin_tp')
```

Example file: conv3DS.py

```
# - convertFile2Arrays (binary 3DS) -
import Converter as C

a = C.convertFile2Arrays('Data/box.3ds', 'bin_3ds')
C.convertArrays2File(a, 'out.plt', 'bin_tp')
```

Example file: convPly.py

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

a = G.cartHexa( (0,0,0), (1,1,1), (10,10,1) )
C.convertArrays2File([a], 'out.ply')
a = C.convertFile2Arrays('out.ply')
```

Example file: convPickle.py

```
# - convertFile2Arrays (binary python pickle) -
import Converter as C
import Generator as G

a = G.cart((0,0,0), (1,1,1), (100,100,100))
C.convertArrays2File([a], 'out.pickle', 'bin_pickle')
b = C.convertFile2Arrays('out.pickle', 'bin_pickle')
```

Example file: convWav.py

```
# - convertArrays2File (wav) -
import Converter as C
import math

# Sampling time step
Deltat = 5.e-5

# Number of samples
N = 500000

# Length of sound
L = N*Deltat
print 'Total sound duration : ', L

# Sound frequency (Hertz)
# Human ear is between 20 Hz and 20 kHz
f1 = 1000; f2 = 1100; f3 = 1200

# Signal
def F(time):
    if (time < L/3.):
        return math.cos(2*math.pi*f1*time)
    elif (time < 2.*L/3.):
        return math.cos(2*math.pi*f2*time)
    else:
        return math.cos(2*math.pi*f3*time)

a = C.array('Time, Pressure', N, 1, 1)

# Time
for i in xrange(N):
    a[1][0,i] = Deltat*i
```

```
# Pressure
a = C.initVars(a, 'Pressure', F, ['Time'])

# Convert in wav uses Time and Pressure field
C.convertArrays2File([a], 'out.wav', 'bin_wav')
```

Example file: convXfig.py

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

a = C.convertFile2Arrays('test.svg', 'fmt_svg',['density',100])
C.convertArrays2File(a, 'out.plt', 'bin_tp')
C.convertArrays2File(a, 'out.fig', 'fmt_xfig')

a = G.cart( (0,0,0), (1,1,1), (3, 3, 3) )
b = G.cart( (4,0,-2), (1,1,1), (3, 3, 1) )
c = G.cart( (0,-3,0),(1,1,1), (3,3,3) )
d = G.cart( (4,-3,-2),(1,1,1), (3,3,1) )
import Transform as T
a = T.rotate(a, (0,0,0), (1,1,0), 20.)
c = T.rotate(c, (0,-3,0), (1,1,0), 20.)
C.convertArrays2File([a,b,c,d], 'out1.fig', 'fmt_xfig')

b = C.convertArray2Tetra(b)
d = C.convertArray2Hexa(d)
C.convertArrays2File([a,b,c,d], 'out2.fig', 'fmt_xfig')

a = C.convertArray2Tetra(a)
c = C.convertArray2Hexa(c)
C.convertArrays2File([a,b,c,d], 'out3.fig', 'fmt_xfig')
```

Example file: convSvg.py

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

a = C.convertFile2Arrays('test.svg', 'fmt_svg', nptsCurve=130)
C.convertArrays2File(a, 'out.plt', 'bin_tp')
C.convertArrays2File(a, 'out.svg', 'fmt_svg')

a = G.cart( (0,0,0), (1.,1.,1.), (10, 10, 1) )
C.convertArrays2File([a], 'out0.svg', 'fmt_svg')

a = G.cart( (0,0,0), (100,100,100), (3, 3, 3) )
b = G.cart( (400,0,-2), (100,100,1), (3, 3, 1) )
c = G.cart( (0,300,0),(100,100,100), (3,3,3) )
d = G.cart( (400,300,-2),(100,100,1), (3,3,1) )
import Transform as T
a = T.rotate(a, (0,0,0), (1,1,0), 20.)
c = T.rotate(c, (0,300,0), (1,1,0), 20.)
C.convertArrays2File([a,b,c,d], 'out1.svg', 'fmt_svg')

b = C.convertArray2Tetra(b)
d = C.convertArray2Hexa(d)
C.convertArrays2File([a,b,c,d], 'out2.svg', 'fmt_svg')

a = C.convertArray2Tetra(a)
c = C.convertArray2Hexa(c)
C.convertArrays2File([a,b,c,d], 'out3.svg', 'fmt_svg')
```

Example file: convPng.py

```
# - convertFile2Arrays (binary png) -
import Converter as C

a = C.convertFile2Arrays('Data/test.png', 'bin_png')
C.convertArrays2File(a, 'out.plt', 'bin_tp')
```

Example file: convert2elsAxdtPT.py

```
# - convert2elsAxdt (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.elsAProfile as elsAProfile

a = G.cart((0,0,0), (1,1,1), (10,10,10))
t = C.newPyTree(['Base',a])
tp = elsAProfile.convert2elsAxdt(t)
C.convertPyTree2File(tp, 'out.cgns')
```

Example file: addReferenceStatePT.py

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

a = G.cart((0,0,0), (1,1,1), (10,10,10))
t = C.newPyTree(['Base',a])
tp = elsAProfile.addReferenceState(t, conservative=[1.,0,0,0,1.7])
C.convertPyTree2File(tp, 'out.cgns')
```

Example file: addGlobalConvergenceHistoryPT.py

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

a = G.cart((0,0,0), (1,1,1), (10,10,10))
t = C.newPyTree(['Base',a])
tp = elsAProfile.addGlobalConvergenceHistory(t)
C.convertPyTree2File(tp, 'out.cgns')
```

Example file: addFlowSolutionPT.py

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

a = G.cart((0,0,0), (1,1,1), (10,10,10))
t = C.newPyTree(['Base', a])
tp = elsAProfile.addFlowSolution(t)
tp = elsAProfile.addFlowSolutionEoR(tp)
C.convertPyTree2File(tp, 'out.cgns')
```

Example file: addOutputForcesPT.py

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

a = G.cart((0,0,0), (1,1,1), (10,10,10))
a = C.addBC2Zone(a, 'wall', 'BCWall', 'imin')
bcs = Internal.getNodesFromName(a, 'wall')
for b in bcs: elsAProfile._addOutputForces(b)

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

Example file: addOutputFrictionPT.py

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

a = G.cart((0,0,0), (1,1,1), (10,10,10))
a = C.addBC2Zone(a, 'wall', 'BCWall', 'imin')
bcs = Internal.getNodesFromName(a, 'wall')
for b in bcs: elsAProfile._addOutputFriction(b)

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

Example file: createHook.py

```
# - 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='extractMesh')
```

Example file: createHookPT.py

```
# - 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='extractMesh')
```

Example file: createGlobalHook.py

```
# - 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')
```

Example file: createGlobalHookPT.py

```
# - 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 = C.createGlobalHook([a,b], function='nodes')
```

Example file: freeHook.py

```
# - freeHook (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='extractMesh')
C.freeHook(hook)
```

Example file: createHookPT.py

```
# - 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='extractMesh')
```

Example file: identifyNodes.py

```
# - 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) )
f = P.exteriorFaces(a)

# Enregistre les noeuds de a dans le hook
hook = C.createHook(a, function='nodes')
# Indices des noeuds de a correspondant aux noeuds de f
nodes = C.identifyNodes(hook, f)
print nodes
```

Example file: identifyNodesPT.py

```
# - 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))
f = P.exteriorFaces(a)

hook = C.createHook(a, function='nodes')
# Indices des noeuds de a correspondant aux noeuds de f
nodes = C.identifyNodes(hook, f)
print nodes
```

Example file: identifyFaces.py

```
# - 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
```

Example file: identifyFacesPT.py

```
# - 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
```

Example file: identifyElements.py

```
# - 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
```

Example file: identifyElementsPT.py

```
# - 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
```

Example file: identifySolutions.py

```
# - 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
```

```
a2 = C.identifySolutions(a,sol,hook,tol=1000.)
C.freeHook(hook)
a = C.addVars([a,a2])
m = C.addVars([m,sol])
C.convertArrays2File([m,a], 'out.plt')
```

Example file: identifySolutionsPT.py

```
# - 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, tol=1000.)
C.freeHook(hook)
C.convertPyTree2File(a, 'out.cgns')
```

Example file: nearestNodes.py

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

# 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
nodes,dist = C.nearestNodes(hook, f)
print nodes,dist
```

Example file: nearestNodesPT.py

```
# - 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
```

Example file: nearestFaces.py

```
# - nearestFaces (array) -
import Converter as C
import Transform as T
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
```

Example file: nearestFacesPT.py

```
# - 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
```

Example file: nearestElements.py

```
# - 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
```

Example file: nearestElementsPT.py

```
# - 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
```

Example file: convertFile2SkeletonTreePT.py

```
# - convertFile2SkeletonTree (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Mpi as Cmpi
import Converter

if Cmpi.rank == 0:
    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')
Cmpi.barrier()

t1 = Cmpi.convertFile2SkeletonTree('in.cgns'); print t1
```

Example file: readZonesPT.py

```
# - readZones (pyTree) -
import Converter.PyTree as C
import Converter.Mpi as Cmpi
import Distributor2.PyTree as Distributor2
import Generator.PyTree as G

# Cree le fichier test
if Cmpi.rank == 0:
    a = G.cart((0,0,0), (1,1,1), (10,10,10))
    b = G.cart((12,0,0), (1,1,1), (10,10,10))
    t = C.newPyTree(['Base',a,b])
    C.convertPyTree2File(t, 'test.cgns')
Cmpi.barrier()

# Relit des zones par procs
t = Cmpi.convertFile2SkeletonTree('test.cgns')
(t, dic) = Distributor2.distribute(t, NProc=Cmpi.size, algorithm='fast')
t = Cmpi.readZones(t, 'test.cgns', proc=Cmpi.rank)
if Cmpi.rank == 0: C.convertPyTree2File(t, 'out.cgns')
```

Example file: convert2PartialTreePT.py

```
# - convert2PartialTree (pyTree) -
import Converter.PyTree as C
import Converter.Mpi as Cmpi
import Distributor2.PyTree as Distributor2
import Generator.PyTree as G

# Cree le fichier test
if Cmpi.rank == 0:
    a = G.cart((0,0,0), (1,1,1), (10,10,10))
    b = G.cart((12,0,0), (1,1,1), (10,10,10))
    t = C.newPyTree(['Base',a,b])
    C.convertPyTree2File(t, 'test.cgns')
```

```
Cmpi.barrier()

# Relit des zones par procs
t = Cmpi.convertFile2SkeletonTree('test.cgns')
(t, dic) = Distributor2.distribute(t, NProc=Cmpi.size, algorithm='fast')
t = Cmpi.readZones(t, 'test.cgns', proc=Cmpi.rank)
# Arbre partiel (sans zones squelettes)
t = Cmpi.convert2PartialTree(t)
if Cmpi.rank == 0: C.convertPyTree2File(t, 'out.cgns')
```

Example file: createBBoxTreePT.py

```
# - createBBoxTree (pyTree) -
import Converter.PyTree as C
import Converter.Mpi as Cmpi
import Distributor2.PyTree as Distributor2
import Generator.PyTree as G

# Cree le fichier test
if Cmpi.rank == 0:
    a = G.cart((0,0,0), (1,1,1), (10,10,10))
    b = G.cart((12,0,0), (1,1,1), (10,10,10))
    t = C.newPyTree(['Base',a,b])
    C.convertPyTree2File(t, 'in.cgns')
Cmpi.barrier()

# Relit des zones par procs
t = Cmpi.convertFile2SkeletonTree('in.cgns')
(t, dic) = Distributor2.distribute(t, NProc=Cmpi.size, algorithm='fast')
t = Cmpi.readZones(t, 'in.cgns', proc=Cmpi.rank)
# Cree le bbox tree
tb = Cmpi.createBBoxTree(t)
if Cmpi.rank == 0: C.convertPyTree2File(tb, 'out.cgns')
```

Example file: getProcPT.py

```
# - getProc (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal
import Converter.Mpi as Cmpi
import Distributor2.PyTree as Distributor2
import Generator.PyTree as G
import KCore.test as test

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
b = G.cart( (12,0,0), (1,1,1), (10,10,10) )
t = C.newPyTree(['Base',a,b])
(t, dic) = Distributor2.distribute(t, NProc=2, algorithm='fast')

zones = Internal.getNodesFromType(t, 'Zone_t')
for z in zones:
    print z[0]+' -> '+str(Cmpi.getProc(z))
```

Example file: setProcPT.py

```
# - setProc (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal
import Converter.Mpi as Cmpi
import Distributor2.PyTree as Distributor2
import Generator.PyTree as G
```

```
import KCore.test as test

a = G.cart((0,0,0), (1,1,1), (10,10,10))
b = G.cart((12,0,0), (1,1,1), (10,10,10))
t = C.newPyTree(['Base',a,b])
t = Cmpi.setProc(t, 1)

zones = Internal.getNodesFromType(t, 'Zone_t')
for z in zones:
    print z[0]+' -> '+str(Cmpi.getProc(z))
```

Example file: getProcDictPT.py

```
# - getProcDict (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal
import Converter.Mpi as Cmpi
import Distributor2.PyTree as Distributor2
import Generator.PyTree as G
import KCore.test as test

# Cree le fichier test
if Cmpi.rank == 0:
    a = G.cart((0,0,0), (1,1,1), (10,10,10))
    b = G.cart((12,0,0), (1,1,1), (10,10,10))
    t = C.newPyTree(['Base',a,b])
    C.convertPyTree2File(t, 'in.cgns')
Cmpi.barrier()

# Relit des zones par procs
t = Cmpi.convertFile2SkeletonTree('in.cgns')
(t, dic) = Distributor2.distribute(t, NProc=2, algorithm='fast')

procDict = Cmpi.getProcDict(t)
if Cmpi.rank == 0: print procDict
```

Example file: computeGraphPT.py

```
# - computeGraph (pyTree) -
import Converter.PyTree as C
import Converter.Mpi as Cmpi
import Distributor2.PyTree as Distributor2
import Generator.PyTree as G

# Cree le fichier test
if Cmpi.rank == 0:
    a = G.cart((0,0,0), (1,1,1), (10,10,10))
    b = G.cart((9,0,0), (1,1,1), (10,10,10))
    t = C.newPyTree(['Base',a,b])
    C.convertPyTree2File(t, 'test.cgns')
Cmpi.barrier()

# Relit des zones par procs
t = Cmpi.convertFile2SkeletonTree('test.cgns')
(t, dic) = Distributor2.distribute(t, NProc=Cmpi.size, algorithm='fast')
t = Cmpi.readZones(t, 'test.cgns', proc=Cmpi.rank)
# Cree le bbox tree
tb = Cmpi.createBBoxTree(t)
# Cree le graph
graph = Cmpi.computeGraph(tb)
if Cmpi.rank == 0: print graph
```

Example file: addXZonesPT.py

```
# - addXZones (pyTree) -
import Converter.PyTree as C
import Converter.Mpi as Cmpi
import Distributor2.PyTree as Distributor2
import Generator.PyTree as G

# Cree le fichier test
if Cmpi.rank == 0:
    a = G.cart((0,0,0), (1,1,1), (10,10,10))
    b = G.cart((9,0,0), (1,1,1), (10,10,10))
    t = C.newPyTree(['Base',a,b])
    C.convertPyTree2File(t, 'test.cgns')
Cmpi.barrier()

# Relit des zones par procs
t = Cmpi.convertFile2SkeletonTree('test.cgns')
(t, dic) = Distributor2.distribute(t, NProc=Cmpi.size, algorithm='fast')
t = Cmpi.readZones(t, 'test.cgns', proc=Cmpi.rank)
# Cree le bbox tree
tb = Cmpi.createBBoxTree(t)
# Cree le graph
graph = Cmpi.computeGraph(tb)
# Add X Zones
t = Cmpi.addXZones(t, graph)
if Cmpi.rank == 0: C.convertPyTree2File(t, 'out.cgns')
```

Example file: rmXZonesPT.py

```
# - rmXZones (pyTree) -
import Converter.PyTree as C
import Converter.Mpi as Cmpi
import Distributor2.PyTree as Distributor2
import Generator.PyTree as G

# Cree le fichier test
if (Cmpi.rank == 0):
    a = G.cart((0,0,0), (1,1,1), (10,10,10))
    b = G.cart((9,0,0), (1,1,1), (10,10,10))
    t = C.newPyTree(['Base',a,b])
    C.convertPyTree2File(t, 'test.cgns')
Cmpi.barrier()

# Relit des zones par procs
t = Cmpi.convertFile2SkeletonTree('test.cgns')
(t, dic) = Distributor2.distribute(t, NProc=Cmpi.size, algorithm='fast')
t = Cmpi.readZones(t, 'test.cgns', proc=Cmpi.rank)
# Cree le bbox tree
tb = Cmpi.createBBoxTree(t)
# Cree le graph
graph = Cmpi.computeGraph(tb)
# Add X Zones
t = Cmpi.addXZones(t, graph)
t = Cmpi.rmXZones(t)
if Cmpi.rank == 0: C.convertPyTree2File(t, 'out.cgns')
```

Example file: center2NodeDPT.py

```
# - center2Node distribue -
import Converter.PyTree as C
```

```
import Distributor2.PyTree as Distributor2
import Converter.Mpi as Cmpi
import Transform.PyTree as T
import Connector.PyTree as X
import Converter.Internal as Internal
import Generator.PyTree as G

# Case
N = 11
t = C.newPyTree(['Base'])
pos = 0
for i in xrange(N):
    a = G.cart( (pos,0,0), (1,1,1), (10+i, 10, 10) )
    pos += 10 + i - 1
    t[2][1][2].append(a)
t = C.initVars(t, '{centers:Density} = {CoordinateX} + {CoordinateY}')
t = X.connectMatch(t)
if Cmpi.rank == 0: C.convertPyTree2File(t, 'in.cgns')
Cmpi.barrier()

# lecture du squelette
sk = Cmpi.convertFile2SkeletonTree('in.cgns')

# equilibrage
(sk, dic) = Distributor2.distribute(sk, NProc=Cmpi.size, algorithm='gradient0',
                                    useCom='match')

# load des zones locales dans le squelette
a = Cmpi.readZones(sk, 'in.cgns', proc=Cmpi.rank)

# center2Node
a = Cmpi.center2Node(a, 'centers:Density')
# a est maintenant un arbre partiel
a = C.rmVars(a, 'centers:Density')

# Reconstruit l'arbre complet a l'ecriture
Cmpi.convertPyTree2File(a, 'out.cgns')
```

Example file: listen.py

```
# - 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)
```

Example file: listenPT.py

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

Example file: send.py

```
# - 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 xrange(30):
    a = T.rotate(a, (0,0,0), (0,0,1), 10.)
    C.send(a, 'localhost')
```

Example file: sendPT.py

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

a = G.cart((0,0,0), (1,1,1), (100,100,30))
C.send(a, 'localhost')

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

Example file: convertArray2Array3D.py

```
# - 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
```

Example file: convertArray3D2Array.py

```
# - 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
```

Example file: restart.py

```
# Read a file 'output.plt' in centers
# Write each block to a separate file with bin_v3d format
import Converter as C

arrays = C.convertFile2Arrays("output.plt", "bin_tp")
```

```
i = 1
for ar in arrays:
    if i < 10:
        C.convertArrays2File([ar], "rep0"+repr(i)+".v3d", "bin_v3d")
    else:
        C.convertArrays2File([ar], "rep"+repr(i)+".v3d", "bin_v3d")
    i=i+1
```

Example file: conv8.py

```
# - convertFile2Arrays -
# Read structured blocks from a file
# Read unstructured blocks from a file
# Write a global file with all blocks
import Converter as C

arrays = C.convertFile2Arrays('dauphin.plt')
unsArrays = C.convertFile2Arrays('Data/unstr.plt')
arrays = arrays + unsArrays
C.convertArrays2File(arrays, 'out.plt')
```

Example file: shell.py

```
#!/usr/bin/env python

# You know how to write a script for converting files format:
# import Converter as C
# a = C.convertFile2Arrays("in.tp", "fmt_tp")
# C.convertArrays2File(a, "out.plt", "bin_tp")
#
# Now you can make it a shell command:

# First define a function equivalent to previous script:
def conv(fmttpFile, bintpFile):
  import Converter as C
  a = C.convertFile2Arrays(fmttpFile, "fmt_tp")
  C.convertArrays2File(a, bintpFile, "bin_tp")

# Then write a main function calling the previous function:
if (__name__ == "__main__"):
  import sys
  if (len(sys.argv) < 3):
    print "Two arguments are needed!"
  else:
    conv(sys.argv[1], sys.argv[2])
```