Internal 2.5

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

1 Internal: CGNS/python tree management

1.1 Preamble

This module provides simple and efficient functions for creating/traversing/manipulating CGNS/Python data tree (pyTree).

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

To use the module:

```
import Converter.Internal as Internal
```

All functions of Cassiopee modules (Converter.PyTree, Transform.PyTree, ...) work on 3 types of containers: a container for grid coordinates named __GridCoordinates__, a container for node solution named __FlowSolutionNodes__ and a container for center solution named __FlowSolutionCenters__. By default:

```
Internal.__GridCoordinates__ = 'GridCoordinates'
Internal.__FlowSolutionNodes__ = 'FlowSolution'
Internal.__FlowSolutionCenters__ = 'FlowSolution#Centers'
```

To make the functions operate on another named container for example 'FlowSolution#Init', start your script with:

```
Internal.__FlowSolutionNodes__ = 'FlowSolution#Init'
```

Internal.autoSetContainers: adapt automatically the containers name (__FlowSolutionNode__, __FlowSolutionCenters__) to match the containers of a zone or a pyTree:

```
Internal.autoSetContainers(t)
```

(See: autoSetContainersPT.py)

1.2 Node tests

Internal.isTopTree: return True if node is a top tree:

```
res = Internal.isTopTree(node )
```

(See: isTopTreePT.py)

Internal.isStdNode: return 0 if node is a list of standard nodes, -1 if node is a standard node, -2 otherwise:

```
res = Internal.isStdNode(node)
```

(See: isStdNodePT.py)

Internal.typeOfNode: return 1 if node is a zone node, 2 if node is a list of zones, 3 if node is a tree, 4 if node is a base, 5 if node is a list of bases, -1 otherwise:

```
res = Internal.typeOfNode(node)
```

(See: typeOfNodePT.py)

Internal.isType: compare given type and node type. Wildcards are accepted:

```
res = Internal.isType(node, 'Zone_t')
```

(See: isTypePT.py)

Internal.isName: compare given name and node name. Wildcards are accepted:

```
res = Internal.isName(node, 'Zone*')
```

(See: isNamePT.py)

Internal.isValue: compare given value and node value. If value is a string, accepts wildcards. Accepts also numpy arrays as value:

```
res = Internal.isValue(node, 1.)
```

(See: isValuePT.py)

1.3 Set/create generic nodes

Internal.setValue: set the given value in node (node is modified):

```
Internal.setValue(node, 1.)
```

(See: setValueIPT.py)

Internal.setName: set the given name to node (node is modified):

```
Internal.setName(node, 'myNode')
```

(See: setNamePT.py)

Internal.setType: set the given type to node (node is modified):

```
Internal.setType(node, 'Zone_t')
```

(See: setTypePT.py)

Internal.createNode: create a node with a given name and type and optional value and children:

```
node = Internal.createNode('myNode', 'DataArray_t', value=1., children=[])
```

(See: createNodePT.py)

Internal.addChild: add a child node to a given node. Position in children list can be specified (node is modified):

```
child = Internal.addChild(node, child, pos=-1)
```

(See: addChildPT.py)

Internal.createChild: create a child node and attach it to a given node. Child's node name, type, value and children can be specified. Position in children list can also be specified (node is modified) and newly created child node is returned:

```
child = Internal.createChild(node, 'myNode', 'DataArray_t', value=None, children=[], pos=-1)
```

(See: createChildPT.py)

Internal.createUniqueChild: same as createChild except that, if a node with the same name already exists in the children list, it is replaced by the new values (node is modified) and newly created or modified child node is returned:

```
child = Internal.createUniqueChild(node, 'myNode', 'DataArray_t', value=None, pos=-1)
```

(See: createUniqueChildPT.py)

1.4 Create CGNS specific nodes

Internal.newCGNSTree: create a tree node with the CGNS version node:

```
node = Internal.newCGNSTree()
```

(See: newCGNSTreePT.py)

Internal.newCGNSBase: create a base node. cellDim is the dimension of zone cells of this base, physDim is the dimension of physical space. For example, triangle zones in 3D space will correspond to cellDim=2 and physDim=3. If parent is not None, attach it to parent node:

```
node = Internal.newCGNBase(name='Base', cellDim=3, physDim=3, parent=None)
```

(See: newCGNSBasePT.py)

Internal.newZone: create a zone node. zsize is the dimension of zone, ztype is the type of zone ('Structured' or 'Unstructured'). If family is not None, attach a zone family node. If parent is not None, attach it to parent node:

```
node = Internal.newZone(name='Zone', zsize=None, ztype='Structured', family=None, parent=None)
```

(See: newZonePT.py)

Internal.newGridCoordinates: create a GridCoordinates node. If parent is not None, attach it to parent node:

```
node = Internal.newGridCoordinates(name=__GridCoordinates__, parent=None)
```

(See: newGridCoordinatesPT.py)

Internal.newDataArray: create a DataArray node. value can be a string, an int, a float, a numpy of ints, a numpy of floats. If parent is not None, attach it to parent node:

```
node = Internal.newDataArray(name='Data', value=None, parent=None)
```

(See: newDataArrayPT.py)

Internal.newDataClass: create a DataClass node. If parent is not None, attach it to parent node:

```
node = Internal.newDataClass(value='UserDefined', parent=None)
```

(See: newDataClassPT.py)

Internal.newDimensionalUnits: create a DimensionalUnits node. Arguments describe the units of the problem. If parent is not None, attach it to parent node:

```
node = Internal.newDimensionalUnits(massUnit='Kilogram', lengthUnit='Meter', timeUnit='Second', temperatureUnit='Kelvin', angleUnit='Radian', parent=None)
```

(See: newDimensionalUnitsPT.py)

Internal.newDimensionalExponents: create a DimensionalExponents node. Arguments describe the unit exponent of the problem. If parent is not None, attach it to parent node:

```
node = Internal.newDimensionalExponents(massExponent='Kilogram', lengthExponent='Meter', timeExponent='Second', temperatureExponent='Kelvin', angleExponent='Radian', parent=None)
```

(See: newDimensionalExponentsPT.py)

Internal.newDataConversion: create a DataConversion node. Arguments describe the conversion factors. If parent is not None, attach it to parent node:

```
node = Internal.newDataConversion(conversionScale=1., conversionOffset=0., parent=None)
```

(See: newDataConversionPT.py)

Internal.newDescriptor: create a Descriptor node. If parent is not None, attach it to parent node:

```
node = Internal.newDescriptor(name='Descriptor', value='', parent=None)
```

(See: newDescriptorPT.py)

Internal.newGridLocation: create a GridLocation node. If parent is not None, attach it to parent node:

```
node = Internal.newGridLocation(value='CellCenter', parent=None)
```

(See: newGridLocationPT.py)

Internal.newIndexArray: create a indexArray node. If parent is not None, attach it to parent node:

```
node = Internal.newIndexArray(name='Index', value=None, parent=None)
```

(See: newIndexArrayPT.py)

Internal.newPointList: create a PointList node. If parent is not None, attach it to parent node:

```
node = Internal.newPointList(name='PointList', value=None, parent=None)
```

(See: newPointListPT.py)

Internal.newPointRange: create a PointRange node. If parent is not None, attach it to parent node:

```
node = Internal.newPointRange(name='PointRange', value=None, parent=None)
```

(See: newPointRangePT.py)

Internal.newRind: create a Rind node. If parent is not None, attach it to parent node:

```
node = Internal.newRind(value=None, parent=None)
```

(See: newRindPT.py)

Internal.newSimulationType: create a SimulationType node. If parent is not None, attach it to parent node:

```
node = Internal.newSimulationType(value='TimeAccurate', parent=None)
```

(See: newSimulationTypePT.py)

Internal.newOrdinal: create a Ordinal node. If parent is not None, attach it to parent node:

```
node = Internal.newOrdinal(value=0, parent=None)
```

(See: newOrdinalPT.py)

Internal.newDiscreteData: create a DiscreteData node. If parent is not None, attach it to parent node:

```
node = Internal.newDiscreteData(name='DiscreteData', parent=None)
```

(See: newDiscreteDataPT.py)

Internal.newIntegralData: create a IntegralData node. If parent is not None, attach it to parent node:

```
node = Internal.newIntegralData(name='IntegralData', parent=None)
```

(See: newIntegralDataPT.py)

Internal.newElements: create a Elements node. etype is the element type ('BAR', 'TRI', 'QUAD', ...), econnectivity is the connectivity numpy array and eboundary ... If parent is not None, attach it to parent node:

```
node = Internal.newElements(name='Elements', etype='UserDefined', econnectivity=None,
eboundary=0, parent=None)
```

(See: newElementsPT.py)

Internal.newParentElements: create a ParentElements node. value is a numpy array. If parent is not None, attach it to parent node:

```
node = Internal.newParentElements(value=None, parent=None)
```

(See: newParentElementsPT.py)

Internal.newParentElementsPosition: create a ParentElementsPosition node. value is a numpy array. If parent is not None, attach it to parent node:

```
node = Internal.newParentElementsPosition(value=None, parent=None)
```

(See: newParentElementsPositionPT.py)

Internal.newZoneBC: create a ZoneBC node. If parent is not None, attach it to parent node:

```
node = Internal.newZoneBC(parent=None)
```

(See: newZoneBCPT.py)

Internal.newBC: create a BC node. It can be defined by a pointRange for structured zones or a pointList of faces for unstructured zones. btype specifies the BC type. A BC familyName can also be defined. If parent is not None, attach it to parent node:

```
node = Internal.newBC(name='BC', pointRange=None, pointList=None, btype='Null', fam-
ily=None, parent=None)
```

(See: newBCPT.py)

Internal.newBCDataSet: create a BCDataSet node. value must be a BCType. GridLocation ('FaceCenter', 'Vertex') can be specified. If parent is not None, attach it to parent node:

```
node = Internal.newBCDataSet(name='BCDataSet', value='Null', gridLocation=None, par-
ent=None)
```

(See: newBCPT.py)

Internal.newBCData: create a BCData node. If parent is not None, attach it to parent node:

```
node = Internal.newBCData(name='BCData', parent=None)
```

(See: newBCDataPT.py)

Internal.newBCProperty: create a BCProperty node. If parent is not None, attach it to parent node:

```
node = Internal.newBCProperty(wallFunction='Null', area='Null', parent=None)
```

(See: newBCPropertyPT.py)

Internal.newAxiSymmetry: create a AxiSymmetry node. If parent is not None, attach it to parent node:

```
node = Internal.newAxiSymmetry(referencePoint=[0.,0.,0.], axisVector=[0.,0.,0.], parent=None)
```

(See: newAxiSymmetryPT.py)

Internal.newRotatingCoordinates: create a RotatingCoordinates node. If parent is not None, attach it to parent node:

```
node = Internal.newRotatingCoordinates(rotationCenter=[0.,0.,0.], rotationRateVector=[0.,0.,0.],
parent=None)
```

(See: newRotatingCoordinatesPT.py)

Internal.newFlowSolution: create a newFlowSolution node. If parent is not None, attach it to parent node:

```
node = Internal.newFlowSolution(name=__FlowSolutionNodes__, gridLocation='Vertex', parent=None)
```

(See: newFlowSolutionPT.py)

Internal.newZoneGridConnectivity: create a newZoneGridConnectivity node. If parent is not None, attach it to parent node:

```
node = Internal.newZoneGridConnectivity(name='ZoneGridConnectivity', parent=None)
```

(See: newZoneGridConnectivityPT.py)

Internal.newGridConnectivity1to1: create a newGridConnectivity1to1 node. If parent is not None, attach it to parent node:

```
node = Internal.newGridConnectivity1to1(name='Match', donorName=None, pointRange=None, pointList=None, pointRangeDonor=None, pointListDonor=None, transform=None, parent=None)
```

(See: newGridConnectivityPT.py)

Internal.newGridConnectivity: create a newGridConnectivity node. If parent is not None, attach it to parent node:

```
node = Internal.newGridConnectivity(name='Overlap', donorName=None, ctype='Overset', parent=None)
```

(See: newGridConnectivityPT.py)

Internal.newGridConnectivityType: create a newGridConnectivityType node. If parent is not None, attach it to parent node:

```
node = Internal.newGridConnectivityType(ctype='Overset',parent=None)
```

(See: newGridConnectivityTypePT.py)

Internal.newGridConnectivityProperty: create a newGridConnectivityProperty node. If parent is not None, attach it to parent node:

```
node = Internal.newGridConnectivityProperty(parent=None)
```

(See: newGridConnectivityPropertyPT.py)

Internal.newPeriodic: create a Periodic node. If parent is not None, attach it to parent node:

```
node = Internal.newPeriodic(rotationCenter=[0.,0.,0.], rotationAngle=[0.,0.,0.], translation=[0.,0.,0.], parent=None)
```

(See: newPeriodicPT.py)

Internal.newOversetHoles: create a OversetHoles node. If parent is not None, attach it to parent node:

```
node = Internal.newOversetHoles(name='OversetHoles', pointRange=None, pointList=None, parent=None)
```

(See: newOversetHolesPT.py)

Internal.newFlowEquationSet: create a FlowEquationSet node. If parent is not None, attach it to parent node:

```
node = Internal.newFlowEquationSet(parent=None)
```

(See: newFlowEquationSetPT.py)

Internal.newGoverningEquations: create a GoverningEquations node. value is the equation type in 'FullPotential', 'Euler', 'NSLaminar','NSTurbulent', 'NSLaminarIncompressible', 'NSTurbulentIncompressible'. If parent is not None, attach it to parent node:

```
node = Internal.newGoverningEquations(value='Euler', parent=None)
```

(See: newGoverningEquationsPT.py)

Internal.newGasModel: create a GasModel node. value is the model type in 'Ideal', 'VanderWaals', 'CaloricallyPerfect', 'ThermallyPerfect', 'ConstantDensity', 'RedlichKwong'. If parent is not None, attach it to parent node:

```
node = Internal.newGasModel(value='Ideal', parent=None)
```

(See: newGasModelPT.py)

Internal.newThermalConductivityModel: create a ThermalConductivityModel node. value is the model type in 'ConstantPrandtl', 'PowerLaw', 'SutherlandLaw'. If parent is not None, attach it to parent node:

```
node = Internal.newThermalConductivityModel(value='Null', parent=None)
```

(See: newThermalConductivityModelPT.py)

Internal.newViscosityModel: create a ViscosityModel node. value is the model type in 'Constant', 'PowerLaw', 'SutherlandLaw'. If parent is not None, attach it to parent node:

```
node = Internal.newViscosityModel(value='Null', parent=None)
```

(See: newViscosityModelPT.py)

Internal.newTurbulenceClosure: create a TurbulenceClosure node. value is the closure type in 'EddyViscosity', 'ReynoldStress', 'ReynoldsStressAlgebraic'. If parent is not None, attach it to parent node:

```
node = Internal.newTurbulenceClosure(value='Null', parent=None)
```

(See: newTurbulenceClosurePT.py)

Internal.newTurbulenceModel: create a TurbulenceModel node. value is the model type in 'Algebraic_BaldwinLomax', 'Algebraic_CebeciSmith', 'HalfEquation_JohnsonKing', 'OneEquation_BaldwinBarth', 'OneEquation_SpalartAllmaras', 'TwoEquation_JonesLaunder', 'TwoEquation_MenterSST
'TwoEquation_Wilcox'. If parent is not None, attach it to parent node:

```
node = Internal.newTurbulenceModel(value='Null', parent=None)
```

(See: newTurbulenceModelPT.py)

Internal.newThermalRelaxationModel: create a ThermalRelaxtionModel node. value is the model type in 'Frozen', 'ThermalEquilib', 'ThermalNonequilb'. If parent is not None, attach it to parent node:

```
node = Internal.newThermalRelaxationModel(value='Null', parent=None)
```

(See: newThermalRelaxationModelPT.py)

Internal.newChemicalKineticsModel: create a ChemicalKineticsModel node. value is the model type in 'Frozen', 'ChemicalEquilibCurveFit', 'ChemicalEquilibMinimization', 'ChemicalNonequilib'. If parent is not None, attach it to parent node:

```
node = Internal.newChemicalKineticsModel(value='Null', parent=None)
```

(See: newChemicalKineticsModelPT.py)

Internal.newEMElectricFieldModel: create a EMElectricFieldModel node. value is the model type in 'Constant', 'Frozen', 'Interpolated', 'Voltage'. If parent is not None, attach it to parent node:

```
node = Internal.newEMElectricFieldModel(value='Null', parent=None)
```

(See: newEMElectricFieldModelPT.py)

Internal.newEMMagneticFieldModel: create a EMMagneticFieldModel node. value is the model type in 'Constant', 'Frozen', 'Interpolated'. If parent is not None, attach it to parent node:

```
node = Internal.newEMMagneticFieldModel(value='Null', parent=None)
```

(See: newEMMagneticFieldModelPT.py)

Internal.newEMConductivityModel: create a EMConductivityModel node. value is the model type in 'Constant', 'Frozen', 'Equilibrium_LinRessler', 'Chemistry_LinRessler'. If parent is not None, attach it to parent node:

```
node = Internal.newEMConductivityModel(value='Null', parent=None)
```

(See: newEMConductivityModelPT.py)

Internal.newBaseIterativeData: create a BaseIterativeData node. If parent is not None, attach it to parent node:

```
node = Internal.newBaseIterativeData(name='BaseIterativeData', parent=None)
```

(See: newBaseIterativeDataPT.py)

Internal.newZoneIterativeData: create a ZoneIterativeData node. If parent is not None, attach it to parent node:

```
node = Internal.newZoneIterativeData(name='ZoneIterativeData', parent=None)
```

(See: newZoneIterativeDataPT.py)

Internal.newRigidGridMotion: create a RigidGridMotion node. mtype is the motion type ('ConstantRate', 'VariableRate'). If parent is not None, attach it to parent node:

```
node = Internal.newRigidGridMotion(name='Motion', origin=[0.,0.,0.], mtype='Null', parent=None)
```

(See: newRigidGridMotionPT.py)

Internal.newRigidGridMotionType: create a RigidGridMotionType node. value is the motion type ('ConstantRate', 'VariableRate'). If parent is not None, attach it to parent node:

```
node = Internal.newRigidGridMotionType(value='ConstantRate', parent=None)
```
(See: newRigidGridMotionTypePT.py)

Internal.newReferenceState: create a ReferenceState node. If parent is not None, attach it to parent node:

```
node = Internal.newReferenceState(name='ReferenceState', parent=None)
```
(See: newReferenceStatePT.py)

Internal.newConvergenceHistory: create a ConvergenceHistory node. value is an iteration number. If parent is not None, attach it to parent node:

```
node = Internal.newConvergenceHistory(name='GlobalConvergenceHistory', value=0, parent=None)
```
(See: newConvergenceHistoryPT.py)

Internal.newFamily: create a zone Family node. If parent is not None, attach it to parent node:

```
node = Internal.newFamily(name='Family', parent=None)
```
(See: newFamilyPT.py)

Internal.newFamilyBC: create a FamilyBC node. value is a BC type string. If parent is not None, attach it to parent node:

```
node = Internal.newFamilyBC(value='UserDefined', parent=None)
```
(See: newFamilyBCPT.py)

Internal.newGeometryReference: create a GeometryReference node. value is a type of CAD ('NASA-IGES', 'SDRC', 'Unigraphics', 'ProEngineer', 'ICEM-CFD'). If parent is not None, attach it to parent node:

```
node = Internal.newGeometryReference(value='Null', file='MyCAD.iges', parent=None)
```
(See: newGeometryReferencePT.py)

Internal.newArbitraryGridMotion: create a ArbitraryGridMotion node. value is the type of motion ('NonDeformingGrid', 'DeformingGrid'). If parent is not None, attach it to parent node:

```
node = Internal.newArbitraryGridMotion(name='Motion', value='Null', parent=None)
```
(See: newArbitraryGridMotionPT.py)

Internal.newUserDefinedData: create a UserDefinedData node to store user specific data. If parent is not None, attach it to parent node:

```
node = Internal.newUserDefinedData(name='UserDefined', value=None, parent=None)
```
(See: newUserDefinedDataPT.py)

Internal.newGravity: create a Gravity node. value is the gravity vector. If parent is not None, attach it to parent node:

```
node = Internal.newGravity(value=[0.,0.,9.81], parent=None)
```
(See: newGravityPT.py)

1.5 Access nodes

Internal.getNodeFromPath: return a node from its path string. Note that node path is relative to input node A. If not found, it returns None:

```
node = Internal.getNodeFromPath(A, 'Base/Zone')
```

(See: getNodeFromPathPT.py)

Internal.getPathsFromType: return a list of paths of nodes of A from their type. A can be a standard node or a list of standard nodes:

```
paths = Internal.getPathsFromType(A, 'Zone_t')
```

(See: getPathsFromTypePT.py)

Internal.getNodesFromType: return a list of nodes of A from their type. A can be a standard node or a list of standard nodes:

```
nodes = Internal.getNodesFromType(A, 'Zone_t')
```

This function can be limited to 1, 2 or 3 levels from the starting node A by using Internal.getNodesFromType1,2,3
(See: getNodesFromTypePT.py)

Internal.getNodeFromType: return the first node of A of given type. A must be a standard node. Wildcards are NOT accepted. This is a fast routine:

```
node = Internal.getNodeFromType(A, 'Zone_t')
```

This function can be limited to 1, 2 or 3 levels from the starting node A by using Internal.getNodeFromType1,2,3.
(See: getNodeFromTypePT.py)

Internal.getByType: return a standard node containing the matching type nodes of A. A can be a standard node or a list of standard nodes:

```
nodes = Internal.getByType(A, 'Zone_t', recursive=-1)
```

(See: getByTypePT.py)

Internal.getNodesFromName: return a list of nodes of a pyTree from their name (wildcards allowed):

```
nodes = Internal.getNodesFromName(A, 'cart')
```

This function can be limited to 1, 2 or 3 levels from the starting node A by using Internal.getNodesFromName1,2,
(See: getNodesFromNamePT.py)

Internal.getNodeFromName: return the first node of A of given name. A must be a standard node. Wildcards are NOT accepted. If not found, it returns None. This is a fast routine:

```
node = Internal.getNodeFromName(A, 'cart')
```

This function can be limited to 1, 2 or 3 levels from the starting node A by using Internal.getNodeFromName1,2,3
(See: getNodeFromNamePT.py)

Internal.getByName: return a standard node containing the matching name nodes of A. A can be a standard node or a list of standard nodes. Wildcards are accepted:

```
nodes = Internal.getByName(A, 'cart', recursive=-1)
```

(See: getByNamePT.py)

Internal.getNodesFromValue: return a list of nodes of a pyTree from their value. Value can be a string, a numpy, a float or an int:

```
nodes = Internal.getNodesFromValue(A, 'Structured')
```

(See: getNodesFromValuePT.py)

Internal.getZones: return the list of zone nodes:

```
nodes = Internal.getZones(A)
```

(See: getZonesPT.py)

Internal.getBases: return the list of base nodes:

```
nodes = Internal.getBases(A)
```

(See: getBasesPT.py)

Internal.getParentOfNode: return the parent node of given node. start must be a higher node in the tree. It returns p (the parent node) and c (the position number of node in the parent's children list). If start is a node of a pyTree, then p[2][c] = node. If starts is a list of standard nodes, then p == start and p[c] = node. If node is not found, then p is None:

```
(p, c) = Internal.getParentOfNode(start, node)
```

(See: getParentOfNodePT.py)

Internal.getPath: return the path of a node in A:

```
path = Internal.getPath(A, node)
```

(See: getPathPT.py)

Internal.getName: return the name of a node:

```
name = Internal.getName(node)
```

(See: getNamePT.py)

Internal.getType: return the CGNS type of a node:

```
type = Internal.getType(node)
```

(See: getTypePT.py)

Internal.getChildren: return the children of a node. Children is a node list:

```
children = Internal.getChildren(node)
```

(See: getChildrenPT.py)

Internal.getValue: return the value of a node, if the node contains a string, a float or an integer. If the node contains an array, return the numpy array:

```
val = Internal.getValue(node)
```

(See: getValueIPT.py)

Internal.getZoneDim: return the dimension of a zone node. Return ['Structured', ni, nj, nk, celldim] for structured grids, return ['Unstructured', np, ne, eltsName, celldim] for unstructured grids. np is the number of points, ne the number of elements, celldim is 0, 1, 2, 3 depending on element dimension, eltsName is one of NODE, BAR, TRI, QUAD, TETRA, PYRA, PENTA, NGON, MULTIPLE:

```
dims = Internal.getZoneDim(node)
```

(See: getZoneDimPT.py)

Internal.getElementRange: return the range of an element connectivity. node must be a zone node. In case of multiple connectivity, the name of the connectivity, its number or its type can be specified:

```
range = Internal.getElementRange(node, name=None, number=None, type=None)
```

(See: getElementRangePT.py)

1.6 Check nodes

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

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

(See: printTreePT.py)

Internal.checkPyTree: check pyTree A following level (0: valid version node, 1: node conformity, 2: unique base name, 3: unique zone name, 4: unique BC name, 5: valid BC range, 6: valid opposite BC range for match and nearmatch, 7: referenced familyZone and familyBCs must be defined in bases, 8: valid CGNS types, 9: valid connectivity, 10: valid CGNS flowfield name). Return a list of pairs of invalid nodes and error message:

```
errors = Internal.checkPyTree(A, level=-20)
```

(See: checkPyTreePT.py)

Internal.correctPyTree: correct pyTree A following level (0: valid version node, 1:node conformity, 2: unique base name, 3: unique zone name, 4: unique BC names, 5: valid BC range, 6: valid opposite BC range for match and nearMatch, 7: referenced familyZone and familyBCs must be defined in bases, 8: valid CGNS types, 9: valid connectivity, 10: valid CGNS flowfield name). Non unique names are made unique. Missing families are added. Invalid nodes are removed. Return a new tree:

```
B = Internal.correctPyTree(A, level=-20)
```

(See: correctPyTreePT.py)

Internal.checkMultigrid: check if pyTree is compatible with multigrid of given level (only for structured zones). Zone dimensions, but also BCs and grid connectivities are checked. Check if coarse grids contain at least nbMinCoarseB points per direction. Check if coarse boundary and connectivity point ranges contain at least nbMinCoarseW points. Return a list of pairs of invalid nodes and error messages:

```
errors = Internal.checkMultigrid(A, level=1, nbMinCoarseB=5, nbMinCoarseW=3)
```

(See: checkMultigridPT.py)

Internal.checkSize: check if the number of points of some zones in pyTree exceed sizeMax points:

```
errors = Internal.checkSize(A, sizeMax=100000000)
```

(See: checkSizePT.py)

1.7 Copy nodes

Internal.copyRef: copy a tree sharing node values (in particular data numpys are shared):

```
B = Internal.copyRef(A)
```

(See: copyRefPT.py)

Internal.copyTree: fully copy a tree. Node values (in particular data numpys) are copied:

```
B = Internal.copyTree(A)
```

(See: copyTreePT.py)

Internal.copyValue: copy the value of nodes specified by byName or byType string. Wildcards are accepted:

```
B = Internal.copyValue(A, byName=None, byType=None)
```

(See: copyValuePT.py)

Internal.copyNode: copy only this node (no recursion). Node value (in particular data numpys) is copied:

```
B = Internal.copyNode(A)
```

(See: copyNodePT.py)

1.8 Add/Remove nodes

Internal.append: append a node a to A by its path. Note that the path is relative to A:

```
B = Internal.append(A, a, 'Base')
```

(See: appendPT.py)

Internal.rmNode: remove a given node b in a (node a is modified):

```
Internal.rmNode(a, b)
```

(See: rmNodePT.py)

Internal.rmNodeByPath: remove a node by its path:

```
a = Internal.rmNodeByPath(a, 'Base/cart') .or. B = Internal.rmNodeByPath(A, 'zone0')
```

(See: rmNodeByPathPT.py)

Internal.rmNodesByName: remove nodes by their name. Wildcards are accepted:

```
b = Internal.rmNodesByName(a, 'zone0') .or. B = Internal.rmNodesByName(A, 'zone0')
```
(See: rmNodesByNamePT.py)

Internal.rmNodesByType: remove nodes by their type:

```
b = Internal.rmNodesByType(a, 'Zone_t') .or. B = Internal.rmNodesByType(A, 'Zone_t')
```
(See: rmNodesByTypePT.py)

Internal.rmNodesByNameAndType: remove nodes verifying name and type at the same time. Wildcards accepted for name:

```
b = Internal.rmNodesByNameAndType(a, 'cart', 'Zone_t') .or. B = Internal.rmNodesByNameAndType(A, 'cart', 'Zone_t')
```
(See: rmNodesByNameAndTypePT.py)

Internal.rmNodesByValue: remove nodes by their value:

```
b = Internal.rmNodesByValue(a, value) .or. B = Internal.rmNodesByType(A, value)
```
(See: rmNodesByValuePT.py)

1.9 Modify nodes

Internal.sortByName: sort nodes by their name (alphabetical order). If recursive=True, sort also chidren nodes:

```
B = Internal.sortByName(A, recursive=True)
```
(See: sortByNamePT.py)

Internal.appendBaseName2ZoneName: add base name to zone name (resulting name is baseName_zoneName for each zone). If updateRef=True, reference to zone names in BC,... are replaced. Separator between base and zone name can be set with separator, a trailing string can also be added:

```
B = Internal.appendBaseName2ZoneName(A, updateRef=True, separator='_', trailing='')
```
(See: appendBaseName2ZoneNamePT.py)

Internal.renameNode: rename node 'name' with 'newName'. Occurances of name elsewhere in the tree is replaced with 'newName':

```
B = Internal.renameNode(A, name, newName)
```
(See: renameNodePT.py)

Internal.merge: merge a list of trees defined in [A1,A2,...]. Return a merged tree B. If a node appears more than once in different trees of the list, the first found node is kept:

```
B = Internal.merge([A1, A2, ...])
```
(See: mergePT.py)

Internal.groupBCByBCType: for each base, group all BCs of same BCType in a family named FamilyName:

```
b = Internal.groupBCByBCType(a, type='BCWall', name='FamWall')
```
(See: groupBCByBCTypePT.py)

Internal.adaptNFace2PE: add the ParentElements node to a NGON/NFace zone:
```
b = Internal.adaptNFace2PE(a, remove=True)
```
(See: adaptNFace2PEPT.py)

Internal.adaptPE2NFace: add the NFace node to a NGON zone with ParentElements:
```
b = Internal.adaptPE2NFace(a, remove=True)
```
(See: adaptPE2NFacePT.py)

Internal.adaptBCFace2BCC: add boundary connectivities for boundaries defined by BCFace:
```
b = Internal.adaptBCFace2BCC(a, remove=True)
```
(See: adaptBCFace2BCCPT.py)

Internal.adaptZoneBCEltRange2EltList: For unstructured zones, when the boundary conditions in ZoneBC are defined with IndexRange. Change this to IndexArray. Usefull for fetch users:
```
b = Internal.adaptZoneBCEltRange2EltList(a)
```
(See: adaptZoneBCEltRange2EltListPT.py)

1.10 Example files

Example file: autoSetContainersPT.py

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

a = G.cart((0,0,0), (1,1,1), (10,10,10))
Internal.__FlowSolutionCenters__ = 'FlowSolution#EndOfRun'
C._initVars(a, 'F={CoordinateX}')
C._initVars(a, 'centers:G={CoordinateY}')
Internal.__FlowSolutionCenters__ = 'FlowSolution#Centers'
t = C.newPyTree(['Base',a])
Internal.autoSetContainers(t)
print Internal.__FlowSolutionNodes__
print Internal.__FlowSolutionCenters__
```

Example file: isTopTreePT.py

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

t = C.newPyTree(['Base1', 'Base2'])
print Internal.isTopTree(t)
#>> True
print Internal.isTopTree(t[2][1])
#>> False
```

Example file: isStdNodePT.py

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

# This is a standard node
a = ['toto', numpy.zeros(12), [], 'DataArray_t']
print Internal.isStdNode(a)
#>> -1

# This is not a standard node
b = ['toto', 'tata']
print Internal.isStdNode(b)
#>> -2

# This is a list of standard nodes
c = ['titi', numpy.zeros(13), [], 'DataArray_t']
print Internal.isStdNode([a,c])
#>> 0
```

Example file: typeOfNodePT.py

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

t = C.newPyTree(['Base1', 'Base2'])
print Internal.typeOfNode(t)
#>> 3
```

Example file: isTypePT.py

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

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

# This is true
print Internal.isType(t, 'CGNSTree_t')
#>> True

# This is false
print Internal.isType(t, 'CGNSBase_t')
#>> False

# Check with wildcard: answer true if the type matches the string
print Internal.isType(t, 'CGNS*')
#>> True
```

Example file: isNamePT.py

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

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

# This is false
print Internal.isName(t[2][1], 'Base3')
#>> False
```

```
# This is true if node name matches the string
print Internal.isName(t[2][1], 'Base*')
#>> True
```

Example file: isValuePT.py

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

# Check a scalar value
node = Internal.createNode('node1', 'DataArray_t', value=1.)
print Internal.isValue(node, 1.)
#>> True

# Check a numpy array values
node = Internal.createNode('node1', 'DataArray_t', value=numpy.zeros(10))
print Internal.isValue(node, numpy.zeros(10))
#>> True

# Check a string value
node = Internal.createNode('node1', 'DataArray_t', value='toto')
print Internal.isValue(node, 'toto')
#>> True
```

Example file: setValueIPT.py

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

node = Internal.createNode('node1', 'DataArray_t', value=12.)

# Set a scalar value in node
Internal.setValue(node, 1.); print node
#>> ['node1', array([ 1.]), [], 'DataArray_t']

# Set a numpy array in node
Internal.setValue(node, numpy.zeros(10)); print node
#>> ['node1', array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]), [], 'DataArray_t']

# Set an array as a list
Internal.setValue(node, [1.,12.,13.]); print node
#>> ['node1', array([  1.,  12.,  13.]), [], 'DataArray_t']
```

Example file: setNamePT.py

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

node = Internal.createNode('node1', 'DataArray_t', value=1.)
Internal.setName(node, 'myNode'); print node
#>> ['myNode', array([ 1.]), [], 'DataArray_t']
```

Example file: setTypePT.py

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

node = Internal.createNode('node1', 'DataArray_t', value=1.)
Internal.setType(node, 'Zone_t'); print node
#>> ['node1', array([ 1.]), [], 'Zone_t']
```

Example file: createNodePT.py

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

# Create a node named myNode, of type DataArray_t, with one value 1.
node = Internal.createNode('myNode', 'DataArray_t', value=1., children=[]); print node
#>> ['myNode', array([ 1.]), [], 'DataArray_t']

# Create a node named myNode, of type DataArray_t, with an array valued in a list
node = Internal.createNode('myNode', 'DataArray_t', value=[12.,14.,15.], children=[]); print node
#>> ['myNode', array([ 12.,  14.,  15.]), [], 'DataArray_t']

# Create a node named myNode, of type DataArray_t, with a given numpy
a = numpy.zeros( (10) )
a[1] = 1.; a[8] = 2.
node = Internal.createNode('myNode', 'DataArray_t', value=a, children=[]); print node
#>> ['myNode', array([ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  2.,  0.]), [], 'DataArray_t']

# Create a node named GoverningEquation, of type 'GoverningEquation_t' and value 'Euler'
node = Internal.createNode('GoverningEquation', 'GoverningEquation_t', value='Euler'); print node
#>> ['GoverningEquation', array(['E', 'u', 'l', 'e', 'r'], dtype='|S1'), [], 'GoverningEquation_t']
```

Example file: addChildPT.py

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

node = Internal.createNode('myNode', 'DataArray_t', value=1.)
child1 = Internal.createNode('child1', 'DataArray_t', value=2.)
child2 = Internal.createNode('child2', 'DataArray_t', value=3.)

# add children nodes to node
Internal.addChild(node, child1, pos=-1) # at the end
Internal.addChild(node, child2, pos=0) # first
print node
#>> ['myNode', array([ 1.]), [['child2', array([ 3.]), [], 'DataArray_t'], ['child1', array([ 2.]), [], 'DataAr
```

Example file: createChildPT.py

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

node = Internal.createNode('myNode', 'DataArray_t', value=1., children=[])
Internal.createChild(node, 'childName', 'DataArray_t', value=2., children=[])
print node
#>> ['myNode', array([ 1.]), [['childName', array([ 2.]), [], 'DataArray_t']], 'DataArray_t']
```

Example file: createUniqueChildPT.py

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

node = Internal.createNode('myNode', 'DataArray_t', value=1.)
Internal.createUniqueChild(node, 'childName', 'DataArray_t', value=2.)
# Since childName node already exists. Only the value will be set.
Internal.createUniqueChild(node, 'childName', 'DataArray_t', value=3.); print node
#>> ['myNode', array([ 1.]), [['childName', array([ 3.]), [], 'DataArray_t']], 'DataArray_t']
```

Example file: newCGNSTreePT.py

```
# - newCGNSTree (pyTree) -
import Converter.Internal as Internal
t = Internal.newCGNSTree(); print t
#>> ['CGNSTree', None, [['CGNSLibraryVersion', array([ 3.1]), [], 'CGNSLibraryVersion_t']], 'CGNSTree_t']
```

Example file: newCGNSBasePT.py

```
# - newCGNSBase (pyTree) -
import Converter.Internal as Internal

# Create a base node
b = Internal.newCGNSBase('Base'); print b
#>> ['Base', array([3, 3], dtype=int32), [], 'CGNSBase_t']

# Create a base node and attach it to tree
t = Internal.newCGNSTree()
Internal.newCGNSBase('Base', 3, 3, parent=t); Internal.printTree(t)
#>> ['CGNSTree',None,[2 sons],'CGNSTree_t']
#>>    |_['CGNSLibraryVersion',array([3.1],dtype='float64'),[0 son],'CGNSLibraryVersion_t']
#>>    |_['Base',array(shape=(2,),dtype='int32',order='F'),[0 son],'CGNSBase_t']
```

Example file: newZonePT.py

```
# - newZone (pyTree) -
import Converter.Internal as Internal

# Create a zone node
z = Internal.newZone('Zone', zsize=[[10],[2],[0]], ztype='Structured'); Internal.printTree(z)
#>> ['Zone',array(shape=(3, 1),dtype='int32',order='F'),[1 son],'Zone_t']
#>>    |_['ZoneType',array('Structured',dtype='|S1'),[0 son],'ZoneType_t']

# Create a zone node and attach it to tree
t = Internal.newCGNSTree()
b = Internal.newCGNSBase('Base', 3, 3, parent=t)
z = Internal.newZone('Zone', [[10],[2],[0]], 'Structured', parent=b)
```

Example file: newGridCoordinatesPT.py

```
# - newGridCoordinates (pyTree) -
import Converter.Internal as Internal

# Create a GridCoordinates node
n = Internal.newGridCoordinates(); print n
#>> ['GridCoordinates', None, [], 'GridCoordinates_t']

# Create a zone node
z = Internal.newZone('Zone', zsize=[[10],[2],[0]], ztype='Structured')
n = Internal.newGridCoordinates(parent=z); Internal.printTree(z)
#>> ['Zone',array(shape=(3, 1),dtype='int32',order='F'),[2 sons],'Zone_t']
#>>    |_['ZoneType',array('Structured',dtype='|S1'),[0 son],'ZoneType_t']
#>>    |_['GridCoordinates',None,[0 son],'GridCoordinates_t']
```

Example file: newDataArrayPT.py

```
# - newDataArray (pyTree) -
import Converter.Internal as Internal
import numpy

# Create a DataArray node
n = Internal.newDataArray('CoordinateX', numpy.zeros(10)); print n
#>> ['CoordinateX', array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]), [], 'DataArray_t']

# Attach it to a parent node
g = Internal.newGridCoordinates()
Internal.newDataArray('CoordinateX', value=numpy.arange(0,10), parent=g)
Internal.newDataArray('CoordinateY', value=numpy.zeros(10), parent=g)
Internal.newDataArray('CoordinateZ', value=numpy.zeros(10), parent=g); Internal.printTree(g)
#>> ['GridCoordinates',None,[3 sons],'GridCoordinates_t']
#>>   |_['CoordinateX',array(shape=(10,),dtype='int64',order='F'),[0 son],'DataArray_t']
#>>   |_['CoordinateY',array(shape=(10,),dtype='float64',order='F'),[0 son],'DataArray_t']
#>>   |_['CoordinateZ',array(shape=(10,),dtype='float64',order='F'),[0 son],'DataArray_t']
```

Example file: newDataClassPT.py

```
# - newDataClass (pyTree) -
import Converter.Internal as Internal

# Create a DataClass node
n = Internal.newDataClass('Dimensional'); Internal.printTree(n)
#>> ['DataClass',array('Dimensional',dtype='|S1'),[0 son],'DataClass_t']

# Attach it to a parent node
d = Internal.newDiscreteData('DiscreteData')
Internal.newDataClass('Dimensional', parent=d)
```

Example file: newDimensionalUnitsPT.py

```
# - newDimensionalUnits (pyTree) -
import Converter.Internal as Internal

# Create a DimensionalUnits node
n = Internal.newDimensionalUnits(massUnit='Kilogram', lengthUnit='Meter', timeUnit='Second', temperatureUnit='K
Internal.printTree(n)
#>> ['DimensionalUnits',array(shape=(5,),dtype='object',order='F'),[1 son],'DimensionalUnits_t']
#>>   |_['AdditionalUnits',array('NullNullNullNullNull',dtype='|S1'),[0 son],'AdditionalUnits_t']

# Attach it to a parent node
d = Internal.newGridCoordinates()
Internal.newDataClass('Dimensional', parent=d)
Internal.newDimensionalUnits('Kilogram', 'Meter', 'Second', 'Kelvin', 'Radian', parent=d)
```

Example file: newDimensionalExponentsPT.py

```
# - newDimensionalExponents (pyTree) -
import Converter.Internal as Internal

# Create a DimensionalExponents node
# Example for velocity exponents (m/s)
n = Internal.newDimensionalExponents(massExponent=0., lengthExponent=1., timeExponent=-1., temperatureExponent=
#>> ['DimensionalExponents',array(shape=(5,),dtype='float64',order='F'),[0 son],'DimensionalExponents_t']

# Attach it to a parent node
d = Internal.newGridCoordinates()
Internal.newDataClass('Dimensional', parent=d)
Internal.newDimensionalExponents(massExponent=0., lengthExponent=1., timeExponent=0., temperatureExponent=0., a
```

Example file: newDataConversionPT.py

```
# - newDataConversion (pyTree) -
import Converter.Internal as Internal

# Data(raw) = Data(nondimensional)*ConversionScale + ConversionOffset

# Create a DataConversion node
n = Internal.newDataConversion(conversionScale=1., conversionOffset=0.); Internal.printTree(n)
#>> ['DataConversion',array(shape=(2,),dtype='float64',order='F'),[0 son],'DataConversion_t']

# Attach it to a parent node
d = Internal.newDiscreteData('DiscreteData')
Internal.newDataClass('NondimensionalParameter', parent=d)
Internal.newDataConversion(conversionScale=1., conversionOffset=0., parent=d)
```

Example file: newDescriptorPT.py

```
# - newDescriptor (pyTree) -
import Converter.Internal as Internal

# Create a Descriptor node
n = Internal.newDescriptor(name='Descriptor', value='Mesh exported from Cassiopee 2.2'); print n
#>> ['Descriptor', array(..), [], 'Descriptor_t']

# Attach it to a parent node
b = Internal.newCGNSBase('Base')
Internal.newDescriptor('Descriptor', 'Aircraft with nacelle Mach=0.7 Re=4 million alpha=-2', parent=b)
```

Example file: newGridLocationPT.py

```
# - newGridLocation (pyTree) -
import Converter.Internal as Internal

# Create a GridLocation node
n = Internal.newGridLocation(value='CellCenter'); Internal.printTree(n)
#>> ['GridLocation',array('CellCenter',dtype='|S1'),[0 son],'GridLocation_t']

# Attach it to a parent node
d = Internal.newBC('wall', [1,80,30,30,1,2], 'BCWall')
Internal.newGridLocation('Vertex', parent=d)
```

Example file: newIndexArrayPT.py

```
# - newIndexArray (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newIndexArray(name='Index', value=[101,51,22,1024,78]); Internal.printTree(n)
#>> ['Index',array(shape=(5,),dtype='int32',order='F'),[0 son],'IndexArray_t']

# Attach it to a parent node
d = Internal.newFlowSolution('FlowSolution', 'Vertex')
Internal.newIndexArray('Index', [101,51,22,1024,78], parent=d)
```

Example file: newPointListPT.py

```
# - newPointList (pyTree) -
import Converter.Internal as Internal

# Create a Descriptor node
n = Internal.newPointList(name='PointList', value=[101,51,22,1036,2]); Internal.printTree(n)
```

```
#>> ['PointList',array(shape=(5,),dtype='int32',order='F'),[0 son],'IndexArray_t']

# Attach it to a parent node
d = Internal.newBC('wall', [1,80,30,30,1,2], 'BCWall')
Internal.newPointList('PointList', [101,51,22,1036,2], parent=d)
```

Example file: newPointRangePT.py

```
# - newPointRange (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newPointRange(name='PointRange', value=[22,56]); Internal.printTree(n)
#>> ['PointRange',array(shape=(2,),dtype='int32',order='F'),[0 son],'IndexRange_t']

# Attach it to a parent node
d = Internal.newBC('wall', [1,80,30,30,1,2], 'BCWall')
Internal.newPointRange('PointRange', [1,71,29,29,1,2], parent=d)
```

Example file: newRindPT.py

```
# - newRind (pyTree) -
import Converter.Internal as Internal

# Create a Rind node (Rind contains the number of ghost cells for a structured block)
n = Internal.newRind([0,0,0,0,1,1]); Internal.printTree(n)
#>> ['Rind',array(shape=(6,),dtype='int32',order='F'),[0 son],'Rind_t']

# Attach it to a parent node
d = Internal.newGridCoordinates()
Internal.newRind([0,0,0,0,1,1], parent=d)
```

Example file: newSimulationTypePT.py

```
# - newSimulationType (pyTree) -
import Converter.Internal as Internal

# Create a simulation type node
b = Internal.newSimulationType(value='NonTimeAccurate'); Internal.printTree(b)
#>> ['SimulationType',array('NonTimeAccurate',dtype='|S1'),[0 son],'SimulationType_t']

# Attach it to parent
b = Internal.newCGNSBase('Base', 3, 3)
Internal.newSimulationType('TimeAccurate', parent=b)
```

Example file: newOrdinalPT.py

```
# - newOrdinal (pyTree) -
import Converter.Internal as Internal

# Create a ordinal node
b = Internal.newOrdinal(value=1); Internal.printTree(b)
#>> ['Ordinal',array([1],dtype='int32'),[0 son],'Ordinal_t']

# Attach it to zone
z = Internal.newZone('Zone', [[10],[2],[0]], 'Structured')
Internal.newOrdinal(value=1, parent=z)
```

Example file: newDiscreteDataPT.py

```
# - newDiscreteData (pyTree) -
import Converter.Internal as Internal

# Create a discrete data node
b = Internal.newDiscreteData(name='DiscreteData'); Internal.printTree(b)
#>> ['DiscreteData',None,[0 son],'DiscreteData_t']

# Attach it to zone
z = Internal.newZone('Zone', [[10],[2],[0]], 'Structured')
Internal.newDiscreteData('DiscreteData', parent=z)
```

Example file: newIntegralDataPT.py

```
# - newIntegralData (pyTree) -
import Converter.Internal as Internal

# Create an integral data node
n = Internal.newIntegralData(name='IntegralData'); Internal.printTree(n)
#>> ['IntegralData',None,[0 son],'IntegralData_t']

# Attach it to base
b = Internal.newCGNSBase('Base', 3, 3)
Internal.newIntegralData('IntegralData', parent=b)
```

Example file: newElementsPT.py

```
# - newElements (pyTree) -
import Converter.Internal as Internal

# Create an elements node
b = Internal.newElements(name='Elements', etype='UserDefined', econnectivity=None, eboundary=0); Internal.print
#>> ['Elements',array(shape=(2,),dtype='int32',order='F'),[2 sons],'Element_t']
#>>    |_['ElementConnectivity',None,[0 son],'DataArray_t']
#>>    |_['ElementRange',None,[0 son],'IndexRange_t']

# Attach it to zone
z = Internal.newZone('Zone', [[10],[2],[0]], 'Unstructured')
Internal.newElements(name='Elements', etype='UserDefined', econnectivity=None, eboundary=0, parent=z)
```

Example file: newParentElementsPT.py

```
# - newParentElements (pyTree) -
import Converter.Internal as Internal

# Create a parent elements node
n = Internal.newParentElements(value=None); Internal.printTree(n)
#>> ['ParentElements',None,[0 son],'DataArray_t']

# Attach it to elements
d = Internal.newElements('Elements', 'UserDefined', None, 0)
Internal.newParentElements(None, parent=d)
```

Example file: newParentElementsPositionPT.py

```
# - newParentElementsPosition (pyTree) -
import Converter.Internal as Internal

# Create a parent elements position node
n = Internal.newParentElementsPosition(value=None); Internal.printTree(n)
#>> ['ParentElementsPosition',None,[0 son],'DataArray_t']

# Attach it to elements
d = Internal.newElements(name='Elements', etype='UserDefined', econnectivity=None, eboundary=0)
Internal.newParentElementsPosition(None, parent=d)
```

Example file: newZoneBCPT.py

```
# - newZoneBC (pyTree) -
import Converter.Internal as Internal

# Create a zoneBC node
n = Internal.newZoneBC(); Internal.printTree(n)
#>> ['ZoneBC',None,[0 son],'ZoneBC_t']

# Attach it to a parent node
z = Internal.newZone('Zone', zsize=[[10],[2],[0]], ztype='Structured')
Internal.newZoneBC(parent=z)
```

Example file: newBCPT.py

```
# - newBC (pyTree) -
import Converter.Internal as Internal

# Create a BC node
n = Internal.newBC(name='BC', pointList=[22,1036,101,43], btype='BCFarfield')
Internal.printTree(n)
#>> ['BC',array('BCFarfield',dtype='|S1'),[1 son],'BC_t']
#>>    |_['PointList',array(shape=(4,),dtype='int32',order='F'),[0 son],'IndexArray_t']

# Attach it to a parent node
d = Internal.newZoneBC()
Internal.newBC('BC', [1,45,1,21,1,1], 'BCWall', parent=d)
```

Example file: newBCPT.py

```
# - newBC (pyTree) -
import Converter.Internal as Internal

# Create a BC node
n = Internal.newBC(name='BC', pointList=[22,1036,101,43], btype='BCFarfield')
Internal.printTree(n)
#>> ['BC',array('BCFarfield',dtype='|S1'),[1 son],'BC_t']
#>>    |_['PointList',array(shape=(4,),dtype='int32',order='F'),[0 son],'IndexArray_t']

# Attach it to a parent node
d = Internal.newZoneBC()
Internal.newBC('BC', [1,45,1,21,1,1], 'BCWall', parent=d)
```

Example file: newBCDataPT.py

```
# - newBCData (pyTree) -
import Converter.Internal as Internal

# Create a BC data node
n = Internal.newBCData(name='BCData'); Internal.printTree(n)
#>> ['BCData',None,[0 son],'BCData_t']

# Attach it to a parent node
d = Internal.newBCDataSet(name='BCDataSet', value='UserDefined')
Internal.newBCData('BCNeumann', parent=d); Internal.printTree(d)
#>> ['BCDataSet',array('UserDefined',dtype='|S1'),[1 son],'BCDataSet_t']
#>>    |_['BCNeumann',None,[0 son],'BCData_t']
```

Example file: newBCPropertyPT.py

```
# - newBCProperty (pyTree) -
import Converter.Internal as Internal

# Create a BC property node
n = Internal.newBCProperty(wallFunction='Null', area='Null'); Internal.printTree(n)
#>> ['BCProperty',None,[2 sons],'BCProperty_t']
#>>    |_['WallFunctionType',array('Null',dtype='|S1'),[0 son],'WallFunctionType_t']
#>>    |_['Area',array('Null',dtype='|S1'),[0 son],'Area_t']

# Attach it to a parent node
d = Internal.newBC(name='BC', pointList=[22,1036,101,43], btype='BCWall')
Internal.newBCProperty(wallFunction='Null', area='Null', parent=d)
```

Example file: newAxiSymmetryPT.py

```
# - newAxiSymmetry (pyTree) -
import Converter.Internal as Internal

# Create an axisymmetry node
n = Internal.newAxiSymmetry(referencePoint=[0.,0.,0.], axisVector=[0.,0.,0.]); Internal.printTree(n)
#>> ['AxiSymmetry',None,[2 sons],'AxiSymmetry_t']
#>>    |_['AxiSymmetryReferencePoint',array(shape=(3,),dtype='float64',order='F'),[0 son],'DataArray_t']
#>>    |_['AxiSymmetryAxisVector',array(shape=(3,),dtype='float64',order='F'),[0 son],'DataArray_t']

# Attach it to base
b = Internal.newCGNSBase('Base', 3, 3)
Internal.newAxiSymmetry([0.,0.,0.], [0.,0.,0.], parent=b)
```

Example file: newRotatingCoordinatesPT.py

```
# - newRotatingCoordinates (pyTree) -
import Converter.Internal as Internal

# Create an rotating coordinates node
n = Internal.newRotatingCoordinates(rotationCenter=[0.,0.,0.], rotationRateVector=[0.,0.,0.]); Internal.printTr
#>> ['RotatingCoordinates',None,[2 sons],'RotatingCoordinates_t']
#>>    |_['RotationCenter',array(shape=(3,),dtype='float64',order='F'),[0 son],'DataArray_t']
#>>    |_['RotationRateVector',array(shape=(3,),dtype='float64',order='F'),[0 son],'DataArray_t']

# Attach it to base
b = Internal.newCGNSBase('Base', 3, 3)
Internal.newRotatingCoordinates([0.,0.,0.], [0.,0.,0.], parent=b)
```

Example file: newFlowSolutionPT.py

```
# - newFlowSolution (pyTree) -
import Converter.Internal as Internal

# Create a flow solution node
n = Internal.newFlowSolution(name='FlowSolution', gridLocation='Vertex'); Internal.printTree(n)
#>> ['FlowSolution',None,[1 son],'FlowSolution_t']
#>>    |_['GridLocation',array('Vertex',dtype='|S1'),[0 son],'GridLocation_t']

# Attach it to a parent node
z = Internal.newZone('Zone', zsize=[[10],[2],[0]], ztype='Structured')
Internal.newFlowSolution(name='FlowSolution', gridLocation='Vertex', parent=z)
```

Example file: newZoneGridConnectivityPT.py

```
# - newZoneGridConnectivity (pyTree) -
import Converter.Internal as Internal
```

```
# Create a node
n = Internal.newZoneGridConnectivity(name='ZoneGridConnectivity'); Internal.printTree(n)
#>> ['ZoneGridConnectivity',None,[0 son],'ZoneGridConnectivity_t']

# Attach it to a parent node
z = Internal.newZone('Zone', zsize=[[10],[2],[0]], ztype='Structured')
Internal.newZoneGridConnectivity('ZoneGridConnectivity', parent=z)
```

Example file: newGridConnectivityPT.py

```
# - newGridConnectivity (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newGridConnectivity(name='Match', donorName='blk1', ctype='Abutting1to1'); Internal.printTree(n)
#>> ['Match',array('blk1',dtype='|S1'),[1 son],'GridConnectivity_t']
#>>    |_['GridConnectivityType',array('Abutting1to1',dtype='|S1'),[0 son],'GridConnectivityType_t']

# Attach it to a parent node
d = Internal.newZoneGridConnectivity(name='ZoneGridConnectivity')
Internal.newGridConnectivity(name='Match', donorName='blk1', ctype='Abutting1to1', parent=d)
```

Example file: newGridConnectivityPT.py

```
# - newGridConnectivity (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newGridConnectivity(name='Match', donorName='blk1', ctype='Abutting1to1'); Internal.printTree(n)
#>> ['Match',array('blk1',dtype='|S1'),[1 son],'GridConnectivity_t']
#>>    |_['GridConnectivityType',array('Abutting1to1',dtype='|S1'),[0 son],'GridConnectivityType_t']

# Attach it to a parent node
d = Internal.newZoneGridConnectivity(name='ZoneGridConnectivity')
Internal.newGridConnectivity(name='Match', donorName='blk1', ctype='Abutting1to1', parent=d)
```

Example file: newGridConnectivityTypePT.py

```
# - newGridConnectivityType (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newGridConnectivityType(ctype='Abutting1to1'); Internal.printTree(n)
#>> ['GridConnectivityType',array('Abutting1to1',dtype='|S1'),[0 son],'GridConnectivityType_t']

# Attach it to a parent node
d = Internal.newGridConnectivity(name='Match', donorName='blk1', ctype=None)
Internal.newGridConnectivityType(ctype='Abutting1to1', parent=d)
```

Example file: newGridConnectivityPropertyPT.py

```
# - newGridConnectivityProperty (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newGridConnectivityProperty(); Internal.printTree(n)
#>> ['GridConnectivityProperty',None,[0 son],'GridConnectivityProperty_t']

# Attach it to a parent node
d = Internal.newGridConnectivity(name='Match', donorName='blk1', ctype='Abutting1to1')
Internal.newGridConnectivityProperty(parent=d)
```

Example file: newPeriodicPT.py

```
# - newPeriodic (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newPeriodic(rotationCenter=[0.,0.,0.], rotationAngle=[0.,0.,0.], translation=[0.,0.,0.]); Internal
#>> ['Periodic',None,[3 sons],'Periodic_t']
#>>    |_['RotationCenter',array(shape=(3,),dtype='float64',order='F'),[0 son],'DataArray_t']
#>>    |_['RotationAngle',array(shape=(3,),dtype='float64',order='F'),[0 son],'DataArray_t']
#>>    |_['Translation',array(shape=(3,),dtype='float64',order='F'),[0 son],'DataArray_t']

# Attach it to a parent node
d = Internal.newGridConnectivityProperty()
Internal.newPeriodic(rotationCenter=[0.,0.,0.], rotationAngle=[0.,0.,0.], translation=[0.,0.,0.], parent=d)
```

Example file: newOversetHolesPT.py

```
# - newOversetHoles (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newOversetHoles(name='OversetHoles', pointRange=None, pointList=None); Internal.printTree(n)
#>> ['OversetHoles',None,[0 son],'OversetHoles_t']

# Attach it to a parent node
d = Internal.newZoneGridConnectivity(name='ZoneGridConnectivity')
Internal.newOversetHoles(name='OversetHoles', pointList=[22,1036,101,43], parent=d)
```

Example file: newFlowEquationSetPT.py

```
# - newFlowEquationSet (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newFlowEquationSet(); Internal.printTree(n)
#>> ['FlowEquationSet',None,[0 son],'FlowEquationSet_t']

# Create a node and attach it to parent
t = Internal.newCGNSTree()
b = Internal.newCGNSBase('Base', 3, 3, parent=t)
n = Internal.newFlowEquationSet(parent=b)
```

Example file: newGoverningEquationsPT.py

```
# - newGoverningEquation (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newGoverningEquations(value='Euler'); Internal.printTree(n)
#>> ['GoverningEquations',array('Euler',dtype='|S1'),[0 son],'GoverningEquations_t']

# Create a node and attach it to parent
t = Internal.newFlowEquationSet()
n = Internal.newGoverningEquations(value='NSTurbulent', parent=t)
```

Example file: newGasModelPT.py

```
# - newGasModel (pyTree) -
import Converter.Internal as Internal

# Create a node
```

```
z = Internal.newGasModel(value='Ideal'); Internal.printTree(z)
#>> ['GasModel',array('Ideal',dtype='|S1'),[0 son],'GasModel_t']

# Create a node and attach it to parent
t = Internal.newFlowEquationSet()
n = Internal.newGasModel(value='Ideal', parent=t)
```

Example file: newThermalConductivityModelPT.py

```
# - newThermalConductivityModel (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newThermalConductivityModel(value='Null'); Internal.printTree(n)
#>> ['ThermalConductivityModel',array('Null',dtype='|S1'),[0 son],'ThermalConductivityModel_t']

# Create a node and attach it to parent
t = Internal.newFlowEquationSet()
n = Internal.newThermalConductivityModel(value='SutherlandLaw', parent=t); Internal.printTree(t)
```

Example file: newViscosityModelPT.py

```
# - newViscosityModel (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newViscosityModel(value='Null'); Internal.printTree(n)
#>> ['ViscosityModel',array('Null',dtype='|S1'),[0 son],'ViscosityModel_t']

# Create a node and attach it to parent
t = Internal.newFlowEquationSet()
n = Internal.newViscosityModel(value='SutherlandLaw', parent=t)
```

Example file: newTurbulenceClosurePT.py

```
# - newTurbulenceClosure (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newTurbulenceClosure(value='Null'); Internal.printTree(n)
#>> ['TurbulenceClosure',array('Null',dtype='|S1'),[0 son],'TurbulenceClosure_t']

# Create a node and attach it to parent
t = Internal.newFlowEquationSet()
n = Internal.newTurbulenceClosure(value='ReynoldsStress', parent=t)
```

Example file: newTurbulenceModelPT.py

```
# - newTurbulenceModel (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newTurbulenceModel(value='TwoEquation_MenterSST'); Internal.printTree(n)
#>> ['TurbulenceModel',array('TwoEquation_MenterSST',dtype='|S1'),[0 son],'TurbulenceModel_t']

# Create a node and attach it to parent
t = Internal.newFlowEquationSet()
n = Internal.newTurbulenceModel(value='OneEquation_SpalartAllmaras', parent=t)
```

Example file: newThermalRelaxationModelPT.py

```
# - newThermalRelaxationModel (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newThermalRelaxationModel(value='Null'); Internal.printTree(n)
#>> ['ThermalRelaxationModel',array('Null',dtype='|S1'),[0 son],'ThermalRelaxationModel_t']

# Create a node and attach it to parent
t = Internal.newFlowEquationSet()
n = Internal.newThermalRelaxationModel(value='ThermalNonequilib', parent=t)
```

Example file: newChemicalKineticsModelPT.py

```
# - newChemicalKineticsModel (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newChemicalKineticsModel(value='Null'); Internal.printTree(n)
#>> ['ChemicalKineticsModel',array('Null',dtype='|S1'),[0 son],'ChemicalKineticsModel_t']

# Create a node and attach it to parent
t = Internal.newFlowEquationSet()
n = Internal.newChemicalKineticsModel(value='ChemicalNonequilib', parent=t)
```

Example file: newEMElectricFieldModelPT.py

```
# - newEMElectricFieldModel (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newEMElectricFieldModel(value='Null'); Internal.printTree(n)
#>> ['EMElectricFieldModel',array('Null',dtype='|S1'),[0 son],'EMElectricFieldModel_t']

# Create a node and attach it to parent
t = Internal.newFlowEquationSet()
n = Internal.newEMElectricFieldModel(value='Voltage', parent=t)
```

Example file: newEMMagneticFieldModelPT.py

```
# - newEMMagneticFieldModel (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newEMMagneticFieldModel(value='Null'); Internal.printTree(n)
#>> ['EMMagneticFieldModel',array('Null',dtype='|S1'),[0 son],'EMMagneticFieldModel_t']

# Create a node and attach it to parent
t = Internal.newFlowEquationSet()
z = Internal.newEMMagneticFieldModel(value='Interpolated', parent=t)
```

Example file: newEMConductivityModelPT.py

```
# - newEMConductivityModel (pyTree) -
import Converter.Internal as Internal

# Create a zone node
n = Internal.newEMConductivityModel(value='Null'); Internal.printTree(n)
#>> ['EMConductivityModel',array('Null',dtype='|S1'),[0 son],'EMConductivityModel_t']

# Create a node and attach it to parent
t = Internal.newFlowEquationSet()
z = Internal.newEMConductivityModel(value='Chemistry_LinRessler', parent=t)
```

Example file: newBaseIterativeDataPT.py

```
# - newBaseIterativeData (pyTree) -
import Converter.Internal as Internal

# Create a zone node
n = Internal.newBaseIterativeData(name='BaseIterativeData', nsteps=100, itype='IterationValues'); Internal.prin
#>> ['BaseIterativeData',array([100],dtype='int32'),[1 son],'BaseIterativeData_t']
#>>    |_['IterationValues',array(shape=(100,),dtype='int32',order='F'),[0 son],'DataArray_t']

# Create a node and attach it to parent
t = Internal.newCGNSTree()
b = Internal.newCGNSBase('Base', 3, 3, parent=t)
n = Internal.newBaseIterativeData(name='BaseIterativeData', nsteps=100, itype='IterationValues', parent=b)
```

Example file: newZoneIterativeDataPT.py

```
# - newZoneIterativeData (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newZoneIterativeData(name='ZoneIterativeData'); Internal.printTree(n)
#>> ['ZoneIterativeData',None,[0 son],'ZoneIterativeData_t']

# Attach it to a parent node
z = Internal.newZone('Zone', zsize=[[10],[2],[0]], ztype='Structured')
Internal.newZoneIterativeData(name='ZoneIterativeData', parent=z)
```

Example file: newRigidGridMotionPT.py

```
# - newRigidGridMotion (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newRigidGridMotion(name='Motion', origin=[0.,0.,0.], mtype='ConstantRate'); Internal.printTree(n)
#>> ['Motion',None,[2 sons],'RigidGridMotion_t']
#>>    |_['OriginLocation',array(shape=(3,),dtype='float64',order='F'),[0 son],'DataArray_t']
#>>    |_['RigidGridMotionType',array('ConstantRate',dtype='|S1'),[0 son],'RigidGridMotionType_t']

# Attach it to a parent node
z = Internal.newZone('Zone', zsize=[[10],[2],[0]], ztype='Structured')
Internal.newRigidGridMotion(name='Motion', origin=[0.,0.,0.], mtype='ConstantRate', parent=z)
```

Example file: newRigidGridMotionTypePT.py

```
# - newRigidGridMotionType (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newRigidGridMotionType(value='ConstantRate'); Internal.printTree(n)
#>> ['RigidGridMotionType',array('ConstantRate',dtype='|S1'),[0 son],'RigidGridMotionType_t']

# Attach it to a parent node
z = Internal.newRigidGridMotion(name='Motion', origin=[0.,0.,0.], mtype='Null')
Internal.newRigidGridMotionType(value='ConstantRate', parent=z)
```

Example file: newReferenceStatePT.py

```
# - newReferenceState (pyTree) -
import Converter.Internal as Internal

# Create a ReferenceState node
```

```
n = Internal.newReferenceState(name='ReferenceState'); Internal.printTree(n)
#>> ['ReferenceState',None,[0 son],'ReferenceState_t']

# Create a ReferenceState node and attach it to base
t = Internal.newCGNSTree()
b = Internal.newCGNSBase('Base', 3, 3, parent=t)
n = Internal.newReferenceState(name='ReferenceState', parent=b); Internal.printTree(n)
```

Example file: newConvergenceHistoryPT.py

```
# - newConvergenceHistory (pyTree) -
import Converter.Internal as Internal

# Create a ConvergenceHistory node
n = Internal.newConvergenceHistory(name='ZoneConvergenceHistory', value=100); Internal.printTree(n)
#>> ['ZoneConvergenceHistory',array([100],dtype='int32'),[0 son],'ConvergenceHistory_t']

# Create a ConvergenceHistory node and attach it to base
t = Internal.newCGNSTree()
b = Internal.newCGNSBase('Base', 3, 3, parent=t)
n = Internal.newConvergenceHistory(name='GlobalConvergenceHistory', value=100, parent=b)
```

Example file: newFamilyPT.py

```
# - newFamily (pyTree) -
import Converter.Internal as Internal

# Create a Family node
n = Internal.newFamily(name='FamWall'); Internal.printTree(n)
#>> ['FamWall',None,[0 son],'Family_t']

# Create a Family node and attach it to base
t = Internal.newCGNSTree()
b = Internal.newCGNSBase('Base', 3, 3, parent=t)
n = Internal.newFamily(name='FamWall', parent=b)
```

Example file: newFamilyBCPT.py

```
# - newFamilyBC (pyTree) -
import Converter.Internal as Internal

# Create a FamilyBC node
n = Internal.newFamilyBC(value='BCWall'); Internal.printTree(n)
#>> ['FamilyBC',array('BCWall',dtype='|S1'),[0 son],'FamilyBC_t']

# Create a FamilyBC node and attach it to Family
b = Internal.newFamily(name='FamInjection')
n = Internal.newFamilyBC(value='UserDefined', parent=b)
```

Example file: newGeometryReferencePT.py

```
# - newGeometryReference (pyTree) -
import Converter.Internal as Internal

# Create a GeometryReference node
n = Internal.newGeometryReference(value='ICEM-CFD', file='MyCAD.tin'); Internal.printTree(n)
#>> ['GeometryReference',None,[2 sons],'GeometryReference_t']
#>>    |_['GeometryFormat',array('ICEM-CFD',dtype='|S1'),[0 son],'GeometryFormat_t']
#>>    |_['GeometryFile',array('MyCAD.tin',dtype='|S1'),[0 son],'GeometryFile_t']

# Create a GeometryReference node and attach it to a family
b = Internal.newFamily(name='FamWall')
n = Internal.newGeometryReference(value='NASA-IGES', file='MyCAD.iges', parent=b)
```

Example file: newArbitraryGridMotionPT.py

```
# - newArbitraryGridMotion (pyTree) -
import Converter.Internal as Internal

# Create a node
n = Internal.newArbitraryGridMotion(name='Motion', value='DeformingGrid'); Internal.printTree(n)
#>> ['Motion',None,[1 son],'ArbitraryGridMotion_t']
#>>    |_['ArbitraryGridMotion',array('DeformingGrid',dtype='|S1'),[0 son],'ArbitraryGridMotionType_t']

# Attach it to a parent node
z = Internal.newZone('Zone', zsize=[[10],[2],[0]], ztype='Structured')
Internal.newArbitraryGridMotion(name='Motion', value='NonDeformingGrid', parent=z)
```

Example file: newUserDefinedDataPT.py

```
# - newUserDefinedData (pyTree) -
import Converter.Internal as Internal

# Create a UserDefinedData node
n = Internal.newUserDefinedData(name='UserDefined', value=None); Internal.printTree(n)
#>> ['UserDefined',None,[0 son],'UserDefinedData_t']

# Create a UserDefinedData node and attach it to a Family node (just an example)
f = Internal.newFamily(name='FamInjection')
n = Internal.newFamilyBC(value='BCInflow', parent=f)
n = Internal.newUserDefinedData(name='.Solver#BC', value=None, parent=n)
```

Example file: newGravityPT.py

```
# - newGravity (pyTree) -
import Converter.Internal as Internal

# Create a gravity node
n = Internal.newGravity(value=[0.,0.,9.81]); Internal.printTree(n)
#>> ['Gravity',None,[1 son],'Gravity_t']
#>>    |_['GravityVector',array(shape=(3,),dtype='float64',order='F'),[0 son],'DataArray_t']

# Create a Gravity node and attach it to base
t = Internal.newCGNSTree()
b = Internal.newCGNSBase('Base', 3, 3, parent=t)
n = Internal.newGravity(value=[0.,0.,9.81], parent=b)
```

Example file: getNodeFromPathPT.py

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

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

# Return GridCoordinates node
coords = Internal.getNodeFromPath(t, 'Base/cart/GridCoordinates'); print coords
#>> ['GridCoordinates', None, [..], 'DataArray_t']

# Return GridCoordinates node (path is relative to input node)
coords = Internal.getNodeFromPath(a, 'GridCoordinates'); print coords
#>> ['GridCoordinates', None, [..], 'DataArray_t']
```

Example file: getPathsFromTypePT.py

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

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

# Return nodes of type 'Zone_t'
zonePaths = Internal.getPathsFromType(t, 'Zone_t'); print zonePaths
#>> ['CGNSTree/Base/cart']
```

Example file: getNodesFromTypePT.py

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

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

# Return nodes of type 'Zone_t'
zones = Internal.getNodesFromType(t, 'Zone_t'); print zones
#>> [['cart', array(..), [..], 'Zone_t']]

# Limit search to 2 levels (faster)
zones = Internal.getNodesFromType2(t, 'Zone_t'); print zones
#>> [['cart', array(..), [..], 'Zone_t']]
```

Example file: getNodeFromTypePT.py

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

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

# Return the first node of type 'Zone_t'
node = Internal.getNodeFromType(t, 'Zone_t'); print node

# Limit search to second level (faster)
node = Internal.getNodeFromType2(t, 'Zone_t'); print node
```

Example file: getByTypePT.py

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

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

# Return a standard node containing nodes of type 'Zone_t' as children
zones = Internal.getByType(t, 'Zone_t'); print zones
```

Example file: getNodesFromNamePT.py

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

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

# Return nodes named 'cart'
nodes = Internal.getNodesFromName(t, 'cart'); print nodes
#>> [['cart', array([..]), [..], 'Zone_t']]

# Return the 3 coordinate nodes
nodes = Internal.getNodesFromName(t, 'Coordinate*'); print nodes
#>> [['CoordinateX', array([..]), [], 'DataArray_t'], ['CoordinateY', array([]), 'DataArray_t'], ['CoordinateX'
```

Example file: getNodeFromNamePT.py

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

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

# Return the node named 'cart'
node = Internal.getNodeFromName(t, 'cart'); print node
#>> ['cart', array([..]), [..], 'Zone_t']
```

Example file: getByNamePT.py

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

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

# Return a standard node containing nodes of name 'cart' as children
node = Internal.getByName(t, 'cart'); print node
#>> ['cart', None, [['cart', array(...), [..], 'Zone_t']], None]
```

Example file: getNodesFromValuePT.py

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

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

# Return nodes with given value
nodes = Internal.getNodesFromValue(t, 2.4); print nodes
#>> []
nodes = Internal.getNodesFromValue(t, 'Structured'); print nodes
#>> [['ZoneType', array(..), [], 'ZoneType_t']]
```

Example file: getZonesPT.py

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

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

# Return nodes of type 'Zone_t'
zones = Internal.getZones(t); print zones
#>> [['cart', array(..), [..], 'Zone_t']]
```

Example file: getBasesPT.py

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

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

# Return nodes of type 'Zone_t'
bases = Internal.getBases(t); print bases
#>> [['Base', array(..), [..], 'CGNSBase_t']]
```

Example file: getParentOfNodePT.py

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

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

(p, c) = Internal.getParentOfNode(t, a); print p
#>> ['Base', array(..), [..], 'CGNSBase_t']
```

Example file: getPathPT.py

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

a = G.cart((0,0,0), (1,1,1), (10,10,10))
t = C.newPyTree(['Base', a])
print Internal.getPath(t, a)
#>> CGNSTree/Base/cart
```

Example file: getNamePT.py

```
# - getName (pyTree) -
import Converter.Internal as Internal

node = Internal.createNode('myNode', 'DataArray_t', value=1.)
print Internal.getName(node)
#>> myNode
```

Example file: getTypePT.py

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

node = Internal.createNode('myNode', 'DataArray_t', value=1.)
print Internal.getType(node)
#>> DataArray_t
```

Example file: getChildrenPT.py

```
# - getChildren (pyTree) -
import Converter.Internal as Internal

node = Internal.createNode('myNode', 'DataArray_t', value=1.)
print Internal.getChildren(node)
#>> []
```

Example file: getValueIPT.py

```
# - getValue of a node (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal

# Structured array
a = G.cart((0,0,0), (1., 0.5,1.), (40,50,20))

# Get value stored in a zone node
print Internal.getValue(a)
#>> [[40 39  0] [50 49  0] [20 19  0]]

# Get type of a zone (from ZoneType node)
node = Internal.getNodeFromName(a, 'ZoneType')

# Print node[1], which is a numpy array
print node[1]
#>> array(['S', 't', 'r', 'u', 'c', 't', 'u', 'r', 'e', 'd']

# getValue, return a string in this case
print Internal.getValue(node)
#>> Structured
```

Example file: getZoneDimPT.py

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

a = G.cart((0,0,0), (1,1,1), (10,10,10))
dim = Internal.getZoneDim(a); print dim
#>> ['Structured', 10, 10, 10, 3]

a = G.cartTetra((0,0,0), (1,1,1), (10,10,10))
dim = Internal.getZoneDim(a); print dim
#>> ['Unstructured', 1000, 3645, 'TETRA', 3]
```

Example file: getElementRangePT.py

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

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

# Get the range of first connectivity
print Internal.getElementRange(a, number=0)
# Get the range of the first connectivity of TETRA type
print Internal.getElementRange(a, type='TETRA')
# Get the range of the connectivity named 'GridElements'
print Internal.getElementRange(a, name='GridElements')
```

Example file: printTreePT.py

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

a = G.cart((0,0,0), (1,1,1), (10,10,10))
Internal.printTree(a) # can print a zone (or any node or any list of nodes)
#>> ['cart',array(shape=(3, 3),dtype='int32',order='F'),[2 sons],'Zone_t']
#>>    |_['ZoneType',array('Structured',dtype='|S1'),[0 son],'ZoneType_t']
#>>    |_['GridCoordinates',None,[3 sons],'GridCoordinates_t']
#>>        |_['CoordinateX',array(shape=(10, 10, 10),dtype='float64',order='F'),[0 son],'DataArray_t']
#>>        |_['CoordinateY',array(shape=(10, 10, 10),dtype='float64',order='F'),[0 son],'DataArray_t']
#>>        |_['CoordinateZ',array(shape=(10, 10, 10),dtype='float64',order='F'),[0 son],'DataArray_t']

t = C.newPyTree(['Base',a])
Internal.printTree(t) # can print a tree
#>> ['CGNSTree',None,[2 sons],'CGNSTree_t']
#>>    |_['CGNSLibraryVersion',array([3.1],dtype='float64'),[0 son],'CGNSLibraryVersion_t']
#>>    ..

Internal.printTree(t, file='toto.txt') # in a file
f = open('toto.txt', 'a')
Internal.printTree(t, stdOut=f) # in a file object
Internal.printTree(t, editor='emacs') # with an editor
```

Example file: checkPyTreePT.py

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

a = G.cart((0,0,0), (1,1,1), (10,10,10))
b = G.cart((9,0,0), (1,1,1), (10,10,10))
c = G.cartTetra((9,9,0), (1,1,1), (10,10,10))
a = C.addBC2Zone(a, 'wall1', 'BCWall', 'imin')
t = C.newPyTree(['Base',a,b,c])
t = X.connectMatch(t)

errors = []
# check unique base names
errors += Internal.checkPyTree(t, level=2)
# check unique zone names
errors += Internal.checkPyTree(t, level=3)
# check unique BC names
errors += Internal.checkPyTree(t, level=4)
# check BC ranges
```

```
errors += Internal.checkPyTree(t, level=5)
# check opposite ranges
errors += Internal.checkPyTree(t, level=6)
# check family definition
errors += Internal.checkPyTree(t, level=7)
# check CGNSTypes
errors += Internal.checkPyTree(t, level=8)
# check element nodes
errors += Internal.checkPyTree(t, level=9); print errors
#>> []

# Introduce errors (on purpose!)
n = Internal.getNodeFromType(t, 'Zone_t')
Internal.setType(n, 'Zon_t')
errors = Internal.checkPyTree(t, level=8); print errors
#>> [['cart', array(..), [..], 'Zon_t'], 'Unknown CGNS type Zon_t for node cart.\n']
```

Example file: correctPyTreePT.py

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

a = G.cart((0,0,0), (1,1,1), (10,10,10))
b = G.cart((9,0,0), (1,1,1), (10,10,10))
a = C.addBC2Zone(a, 'wall1', 'BCWall', 'imin')
t = C.newPyTree(['Base', a, b])
t = X.connectMatch(t)

t = Internal.correctPyTree(t)
```

Example file: checkMultigridPT.py

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

a = G.cart((0,0,0), (1,1,1), (11,11,11))
b = G.cart((10,0,0), (1,1,1), (11,11,11))
a = C.addBC2Zone(a, 'wall1', 'BCWall', 'imin')
t = C.newPyTree(['Base',a,b])
t = X.connectMatch(t)
C.convertPyTree2File(t, 'out.cgns')

# check multigrid compatibility
errors = Internal.checkMultigrid(t, level=1)
print errors
```

Example file: checkSizePT.py

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

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

```
b = G.cart((10,0,0), (1,1,1), (11,11,11))
a = C.addBC2Zone(a, 'wall1', 'BCWall', 'imin')
t = C.newPyTree(['Base',a,b])
t = X.connectMatch(t)
C.convertPyTree2File(t, 'out.cgns')

# check nodes conformity
errors = Internal.checkSize(t, sizeMax=500)
print errors
```

Example file: copyRefPT.py

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

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

# t2 and t are sharing values (also data numpys)
t2 = Internal.copyRef(t)
```

Example file: copyTreePT.py

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

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

# t2 is a full copy of t (values/numpys are also copied)
t2 = Internal.copyTree(t)
```

Example file: copyValuePT.py

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

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

# t2 has numpy copy only for nodes of type 'DataArray_t'
t2 = Internal.copyValue(t, byType='DataArray_t')

# t3 has numpy copy only for nodes of name 'Coordinate*'
t3 = Internal.copyValue(t, byName='Coordinate*')
```

Example file: copyNodePT.py

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

a = Internal.newDataArray(value=[1,2,3])
# Copy only the numpy of this node
b = Internal.copyNode(a)
```

```
# Modify numpy of b
b[1][0]=5
# a is not modified
print a
#>> ['Data', array([1, 2, 3], dtype=int32), [], 'DataArray_t']
print b
#>> ['Data', array([5, 2, 3], dtype=int32), [], 'DataArray_t']
```

Example file: appendPT.py

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

t = C.newPyTree(['Base', 'Base2'])
a = G.cart((0,0,0), (1,1,1), (10,10,10))

# Append a to 'Base' node of t
t = Internal.append(t, a, 'Base')

#>> ['CGNSTree',None,[3 sons],'CGNSTree_t']
#>>    |_['CGNSLibraryVersion',array([3.1],dtype='float64'),[0 son],'CGNSLibraryVersion_t']
#>>    |_['Base',array(shape=(2,),dtype='int32',order='F'),[1 son],'CGNSBase_t']
#>>    |   |_['cart',array(shape=(3, 3),dtype='int32',order='F'),[2 sons],'Zone_t']
#>>    |        ...
#>>    |_['Base2',array(shape=(2,),dtype='int32',order='F'),[0 son],'CGNSBase_t']
C.convertPyTree2File(t, 'out.cgns')
```

Example file: rmNodePT.py

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

t = C.newPyTree(['Base', 'Base2'])
a = G.cart((0,0,0), (1,1,1), (10,10,10))
t[2][1][2] += [a]

# rm the node a from t
Internal._rmNode(t, a)

#>> ['CGNSTree',None,[3 sons],'CGNSTree_t']
#>>   |_['CGNSLibraryVersion',array([3.1],dtype='float64'),[0 son],'CGNSLibraryVersion_t']
#>>   |_['Base',array(shape=(2,),dtype='int32',order='F'),[0 son],'CGNSBase_t']
#>>   |_['Base2',array(shape=(2,),dtype='int32',order='F'),[0 son],'CGNSBase_t']
```

Example file: rmNodeByPathPT.py

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

t = C.newPyTree(['Base', 'Base2'])
for i in xrange(10):
    a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
    a[0] = 'Cart'+str(i)
    t[2][1][2].append(a)

t = Internal.rmNodeByPath(t, 'Base/Cart2')
t = Internal.rmNodeByPath(t, 'Base/Cart4')
C.convertPyTree2File(t, 'out.cgns')
```

Example file: rmNodesByNamePT.py

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

t = C.newPyTree(['Base', 'Base2'])
for i in xrange(10):
    a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
    a[0] = 'Cart'+str(i)
    t[2][1][2].append(a)

t = Internal.rmNodesByName(t, 'Cart0')
t = Internal.rmNodesByName(t, 'Cart*')
C.convertPyTree2File(t, 'out.cgns')
```

Example file: rmNodesByTypePT.py

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

t = C.newPyTree(['Base', 'Base2'])
for i in xrange(10):
    a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
    a[0] = 'Cart'+str(i)
    t[2][1][2].append(a)

t = Internal.rmNodesByType(t, 'Zone_t')
C.convertPyTree2File(t, 'out.cgns')
```

Example file: rmNodesByNameAndTypePT.py

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

t = C.newPyTree(['Base', 'Base2'])
for i in xrange(10):
    a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
    a[0] = 'Cart'+str(i)
    t[2][1][2].append(a)

t = Internal.rmNodesByNameAndType(t, 'Cart0', 'Zone_t')
C.convertPyTree2File(t, 'out.cgns')
```

Example file: rmNodesByValuePT.py

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

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
a = C.fillEmptyBCWith(a, 'far', 'BCFarfield')

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

Example file: sortByNamePT.py

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

a = G.cart((0,0,0), (1,1,1), (10,10,10)); a[0] = 'b'
b = G.cart((12,0,0), (1,1,1), (10,10,10)); b[0] = 'a'
t = C.newPyTree(['Base',a,b])
t = Internal.sortByName(t)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: appendBaseName2ZoneNamePT.py

```
# - appendBaseName2ZoneName (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal

a = G.cart((0,0,0), (1,1,1), (10,10,10)); a[0] = 'a'
b = G.cart((11,0,0), (1,1,1), (10,10,10)); b[0] = 'b'
t = C.newPyTree(['Base',a,b])

t = Internal.appendBaseName2ZoneName(t)

#>> ['CGNSTree',None,[2 sons],'CGNSTree_t']
#>>    |_['CGNSLibraryVersion',array([3.1],dtype='float64'),[0 son],'CGNSLibraryVersion_t']
#>>    |_['Base',array(shape=(2,),dtype='int32',order='F'),[2 sons],'CGNSBase_t']
#>>        |_['Base_a',array(shape=(3, 3),dtype='int32',order='F'),[2 sons],'Zone_t']
#>>        |   |_['ZoneType',array('Structured',dtype='|S1'),[0 son],'ZoneType_t']
#>>        |   ...
#>>        |_['Base_b',array(shape=(3, 3),dtype='int32',order='F'),[2 sons],'Zone_t']
#>>            |_['ZoneType',array('Structured',dtype='|S1'),[0 son],'ZoneType_t']
#>>            ...
C.convertPyTree2File(t, 'out.cgns')
```

Example file: renameNodePT.py

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

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])
t = X.connectMatch(t)

# Change the zone named 'cart' and its reference in BCMatch (GridConnectivity)
t = Internal.renameNode(t, 'cart', 'myCart')
C.convertPyTree2File(t, 'out.cgns')
```

Example file: mergePT.py

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

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

```
t2 = C.newPyTree(['Other1', 'Other2', 'Base2'])
a = G.cart((0,0,0), (1,1,1), (10,10,10)); t1[2][1][2] += [a]
t = Internal.merge([t1, t2])

#>> ['CGNSTree',None,[5 sons],'CGNSTree_t']
#>>   |_['CGNSLibraryVersion',array([3.1],dtype='float64'),[0 son],'CGNSLibraryVersion_t']
#>>   |_['Base1',array(shape=(2,),dtype='int32',order='F'),[1 son],'CGNSBase_t']
#>>   |   |_['cart',array(shape=(3, 3),dtype='int32',order='F'),[2 sons],'Zone_t']
#>>   |       ...
#>>   |_['Base2',array(shape=(2,),dtype='int32',order='F'),[0 son],'CGNSBase_t']
#>>   |_['Other1',array(shape=(2,),dtype='int32',order='F'),[0 son],'CGNSBase_t']
#>>   |_['Other2',array(shape=(2,),dtype='int32',order='F'),[0 son],'CGNSBase_t']
C.convertPyTree2File(t, 'out.cgns')
```

Example file: groupBCByBCTypePT.py

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

a = G.cart((0.,0.,0.),(0.1,0.1,0.1),(10,10,10))
a = C.fillEmptyBCWith(a, 'wall', 'BCWall')

t = C.newPyTree(['Base',a])
Internal._groupBCByBCType(t, 'BCWall', 'FamWall')

#>> ['CGNSTree',None,[2 sons],'CGNSTree_t']
#>>   |_['CGNSLibraryVersion',array([3.1],dtype='float64'),[0 son],'CGNSLibraryVersion_t']
#>>   |_['Base',array(shape=(2,),dtype='int32',order='F'),[2 sons],'CGNSBase_t']
#>>       |_['cart',array(shape=(3, 3),dtype='int32',order='F'),[3 sons],'Zone_t']
#>>       |   ...
#>>       |   |_['ZoneBC',None,[6 sons],'ZoneBC_t']
#>>       |       |_['wall1',array('FamilySpecified',dtype='|S1'),[2 sons],'BC_t']
#>>       |       |   |_['PointRange',array(shape=(3, 2),dtype='int32',order='F'),[0 son],'IndexRange_t']
#>>       |       |   |_['FamilyName',array('FamWall',dtype='|S1'),[0 son],'FamilyName_t']
#>>       |       |_['wall2',array('FamilySpecified',dtype='|S1'),[2 sons],'BC_t']
#>>       |       |   |_['PointRange',array(shape=(3, 2),dtype='int32',order='F'),[0 son],'IndexRange_t']
#>>       |   ...
#>>       |_['FamWall',None,[1 son],'Family_t']
#>>           |_['FamilyBC',array('BCWall',dtype='|S1'),[0 son],'FamilyBC_t']
C.convertPyTree2File(t, 'out.cgns')
```

Example file: adaptNFace2PEPT.py

```
# - adaptNFace2PE (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal
import Generator.PyTree as G

a = G.cartNGon((0,0,0), (1,1,1), (10,10,10))
a = Internal.adaptNFace2PE(a, remove=False)
C.convertPyTree2File(a, 'out.cgns')
```

Example file: adaptPE2NFacePT.py

```
# - adaptPE2NFace (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal
import Generator.PyTree as G
```

```
a = G.cartNGon((0,0,0), (1,1,1), (10,10,10))
a = Internal.adaptNFace2PE(a, remove=True)

a = Internal.adaptPE2NFace(a, remove=True)
C.convertPyTree2File(a, 'out.cgns')
```

Example file: adaptBCFace2BCCPT.py

```
# - adaptBCFace2BCC (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal
import Generator.PyTree as G

a = G.cartHexa((0,0,0), (1,1,1), (10,10,10))
a = C.addBC2Zone(a, 'wall', 'BCWall', faceList=[1,7])
a = Internal.adaptBCFace2BCC(a)
C.convertPyTree2File(a, 'out.cgns')
```

Example file: adaptZoneBCEltRange2EltListPT.py

```
# - adaptZoneBCEltRange2EltList (pyTree) -
import Converter.Internal as Internal
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))
a = C.addBC2Zone(a, 'wall', 'BCWall', subzone=b)

a = Internal.adaptZoneBCEltRange2EltList(a)
C.convertPyTree2File(a, 'out.cgns')
```