ONERA

THE FRENCH AEROSPACE LAB

Converter.elsAProfile Documentation

Release 3.1

/ELSA/MU-09020/V3.1

May 28, 2020

CONTENTS

CHAPTER

ONE

PREAMBLE

This module provides functions to adapt a standard CGNS/python tree for use with ONERA aerodynamic solver *elsA*.

To use the module:

```
import Converter.elsAProfile as elsAProfile
```

CHAPTER

TWO

LIST OF FUNCTIONS

– Conversion to elsA CGNS

Converter.elsAProfile.adaptPeriodicMatch(t)	Convert Periodic Match Grid Connectivity (GC) data to be compliant with elsA solver.
Converter.elsAProfile.adaptNearMatch(t)	Convert Nearmatch Grid Connectivity (GC) to be compliant with elsA solver.
Converter.elsAProfile.rmGCOverlap(t)	Remove the Overlap boundary conditions described as Grid Connectivities (GC).
Converter.elsAProfile.overlapGC2BC(t)	Convert the Overlap boundary conditions from GC (Grid Connectivity) to BC (Boundary Condition) for elsA solver.
Converter.elsAProfile.fillNeighbourList(t[, ...])	Fill neighbour list by families of zones according to intersection.
Converter.elsAProfile.prefixDnrInSubRegions(t)	Prefix zone subregion ID_* donor names with base name.

– Addition of elsA specific nodes

Converter.elsAProfile.addPeriodicDataInSolverParam(a)	Add periodic data for grid connectivity in zone in a .Solver#Param node.
Converter.elsAProfile.addOutput(a, Dict[, ...])	Add a node '.Solver#Output'+name to extract required value from a node.
Converter.elsAProfile.addOutputForces(node)	Add an output node for forces.
Converter.elsAProfile.addOutputFriction(node)	Add an output node for frictions.
Converter.elsAProfile.addGlobalConvergenceHistory(t)	Create a node for global convergence history storage for each base.
Converter.elsAProfile.addReferenceState(t[, ...])	Add a reference state for each base.

Continued on next page

Table 2 – continued from previous page

Converter.elsAProfile.addFlowSolution(t[, ...])	Add a node to extract the flow solution.
Converter.elsAProfile.addFlowSolutionEoR(t)	Add a node to extract the flow solution at the end of the run for each zone of the pyTree.
Converter.elsAProfile.addTurbulentDistanceIndex(t)	Add the TurbulentDistance index node for elsA solver.
Converter.elsAProfile.createElsaHybrid(t[, ...])	Create elsAHybrid node necessary for NGON zones.

– Miscellaneous

Converter.elsAProfile.getCGNSkeys(key[, verbose])	Return the CGNS name (if it exists) corresponding to the elsA key.
Converter.elsAProfile.buildMaskFiles(t[, ...])	Write the mask files for elsA solver.
Converter.elsAProfile.convert2elsAxdt(t[, ...])	Perform all necessary transformations to obtain a computable tree for elsA.

CHAPTER

THREE

CONTENTS

3.1 Conversion

Converter.elsAProfile.**adaptPeriodicMatch**(*t*, *clean=False*)

Convert periodic grid connectivities to periodic information for elsA. A '.Solver#Property' node is added in the grid connectivity node with children nodes of name: 'type', 'jtype', 'jtopo', 'ptype'... A '.Solver#Param' node is added in the zone, providing periodicity information. Children node names are: 'axis_ang_1', 'axis_ang_2', 'axis_pnt_x', 'axis_pnt_y', 'axis_pnt_z', 'axis_vct_x', 'axis_vct_y', 'axis_vct_z', 'periodic_dir'. If 'Periodic_t' nodes are not removed, if RotationAngle exists, it is set in radians if it was defined in degrees (regarding the AngleUnits subchild node). Exists also as an in-place function (_adaptPeriodicMatch) that modifies t and returns None.

Parameters

- **t** (pyTree, list of bases, base, list of zones, zone) – input data
- **clean** (Boolean) – if True, removes useless nodes (Periodic_t nodes) for elsA

Returns modified reference copy of t

Return type Identical to input

Example of use:

- Adapt periodic match condition for elsA solver (pyTree):

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

(continues on next page)

(continued from previous page)

```
a = G.cylinder((0.,0.,0.), 0.1, 1., 0., 90., 5., (11,11,11))
t = C.newPyTree(['Base',a])
t = X.connectMatchPeriodic(t, rotationCenter=[0.,0.,0.],
                           translation=[0.,0.,5.])
t = X.connectMatchPeriodic(t, rotationCenter=[0.,0.,0.],
                           rotationAngle=[0.,0.,90.])
Internal.printTree(Internal.getNodeFromName(t, 'match1_0'))
#>> ['match1_0',array('cylinder',dtype='|S1'),[4 sons],'GridConnectivity1to1_t']
#>>    |_['PointRange',array(shape=(3, 2),dtype='int32',order='F'),[0 son],
↪'IndexRange_t']
#>>    |_['PointRangeDonor',array(shape=(3, 2),dtype='int32',order='F'),[0 son],
↪'IndexRange_t']
#>>    |_['Transform',array(shape=(3,),dtype='int32',order='F'),[0 son],'
↪"int[IndexDimension]"']
#>>    |_['GridConnectivityProperty',None,[1 son],'GridConnectivityProperty_t']
#>>        |_['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']
tp = elsAProfile.adaptPeriodicMatch(t)
Internal.printTree(Internal.getNodeFromName(tp, 'match1_0'))
#>> ['match1_0',array('cylinder',dtype='|S1'),[5 sons],'GridConnectivity1to1_t']
#>>    |_['PointRange',array(shape=(3, 2),dtype='int32',order='F'),[0 son],
↪'IndexRange_t']
#>>    |_['PointRangeDonor',array(shape=(3, 2),dtype='int32',order='F'),[0 son],
↪'IndexRange_t']
#>>    |_['Transform',array(shape=(3,),dtype='int32',order='F'),[0 son],'
↪"int[IndexDimension]"']
#>>    |_['GridConnectivityProperty',None,[1 son],'GridConnectivityProperty_t']
#>>    |   |_['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']
#>>    |_['.Solver#Property',None,[7 sons],'UserDefinedData_t']
#>>        |_['type',array('join',dtype='|S1'),[0 son],'DataArray_t']
#>>        |_['jtopo',array('periodic',dtype='|S1'),[0 son],'DataArray_t']
#>>        |_['jtype',array('match',dtype='|S1'),[0 son],'DataArray_t']
#>>        |_['ptype',array('tra',dtype='|S1'),[0 son],'DataArray_t']
```

(continues on next page)

(continued from previous page)

```
#>>         |_['xtran',array([0.0],dtype='float64'),[0 son],'DataArray_t']
#>>         |_['ytran',array([0.0],dtype='float64'),[0 son],'DataArray_t']
#>>         |_['ztran',array([5.0],dtype='float64'),[0 son],'DataArray_t']
C.convertPyTree2File(tp, 'out.cgns')
```

Converter.elsAProfile.**adaptNearMatch**(*t*)

Convert the nearmatch grid connectivities compliant with Cassiopee for use with elsA. In particular, a ".Solver#Property" node is added with all the information required for elsA. Exists also as an in-place function (_adaptNearMatch) that modifies t and returns None.

Parameters **t** (pyTree, list of bases, base, list of zones, zone) – input data

Returns modified reference copy of t

Return type Identical to input

Example of use:

- Adapt nearmatch condition for elsA solver (pyTree):

```
# - connectNearMatch (pyTree) -
import Generator.PyTree as G
import Connector.PyTree as X
import Converter.PyTree as C
import Transform.PyTree as T
import Converter.Internal as Internal
import Converter.elsAProfile as elsAProfile

a1 = G.cart((0.,0.,0.), (0.1, 0.1, 0.1), (11, 21, 3)); a1[0] = 'cart1'
a2 = G.cart((1., 0.2, 0.), (0.1, 0.1, 0.1), (11, 21, 3)); a2[0] = 'cart2'
a2 = T.oneovern(a2,(1,2,1))
t = C.newPyTree(['Base',a1,a2])
t = X.connectNearMatch(t)
Internal.printTree(Internal.getNodeFromName(t, 'nmatch1_0'))
#>> ['nmatch1_0',array('cart2',dtype='|S1'),[4 sons],'GridConnectivity_t']
#>>    |_['PointRange',array(shape=(3, 2),dtype='int32',order='F'),[0 son],
↪'IndexRange_t']
#>>    |_['GridConnectivityType',array('Abutting',dtype='|S1'),[0 son],
↪'GridConnectivityType_t']
#>>    |_['PointListDonor',array(shape=(3, 1),dtype='int32',order='C'),[0 son],
↪'IndexArray_t']
#>>    |_['UserDefinedData',None,[3 sons],'UserDefinedData_t']
#>>        |_['PointRangeDonor',array(shape=(3, 2),dtype='int32',order='F'),[0␣
↪son],'DataArray_t']
```

(continues on next page)

(continued from previous page)

```
#>>        |_['Transform',array(shape=(3,),dtype='int32',order='F'),[0 son],
↪'DataArray_t']
#>>        |_['NMRatio',array(shape=(3,),dtype='float64',order='F'),[0 son],
↪'DataArray_t']
tp = elsAProfile.adaptNearMatch(t)
Internal.printTree(Internal.getNodeFromName(tp, 'nmatch1_0'))
#>> ['nmatch1_0',array('cart2',dtype='|S1'),[4 sons],'GridConnectivity1to1_t']
#>>    |_['PointRange',array(shape=(3, 2),dtype='int32',order='F'),[0 son],
↪'IndexRange_t']
#>>    |_['PointRangeDonor',array(shape=(3, 2),dtype='int32',order='F'),[0 son],
↪'IndexRange_t']
#>>    |_['Transform',array(shape=(3,),dtype='int32',order='F'),[0 son],'
↪"int[IndexDimension]"']
#>>    |_['.Solver#Property',None,[6 sons],'UserDefinedData_t']
#>>        |_['jtype',array('nearmatch',dtype='|S1'),[0 son],'DataArray_t']
#>>        |_['type',array('join',dtype='|S1'),[0 son],'DataArray_t']
#>>        |_['matchside',array('fine',dtype='|S1'),[0 son],'DataArray_t']
#>>        |_['i_ratio',array([1],dtype='int32'),[0 son],'DataArray_t']
#>>        |_['j_ratio',array([2],dtype='int32'),[0 son],'DataArray_t']
#>>        |_['k_ratio',array([1],dtype='int32'),[0 son],'DataArray_t']
C.convertPyTree2File(tp, 'out.cgns')
```

Converter.elsAProfile.**rmGCOverlap**(*t*)

Remove the overlap grid connectivities described as a GridConnectivity type node of value 'Overset' in the t. Exists also as an in-place function (_rmGCOverlap) that modifies t and returns None.

Parameters **t** (pyTree, list of bases, base, list of zones, zone) – input data

Returns modified reference copy of t

Return type Identical to input

Example of use:

- Remove the Overlap condition described as Grid Connectivity (pyTree):

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

a = G.cylinder((0,0,0), 1., 1.5, 360., 0., 1., (30,30,10))
a = C.addBC2Zone(a, 'overlap1', 'BCOverlap', 'jmin')
t = C.newPyTree(['Base', a])
```

(continues on next page)

(continued from previous page)

```
tp = elsAProfile.rmGCOverlap(t)
C.convertPyTree2File(tp, 'out.cgns')
```

Converter.elsAProfile.**overlapGC2BC**(*t*)

Convert the 'Overset' grid connectivity nodes compliant with Cassiopee to an 'Overlap' BC compliant with elsA. The created BC node is attached to a family of BC of name 'Fam_Ovlp'+baseName, where baseName is the name of the parent base. Prefix '**Fam_Ovlp_**' of the family BC of 'Overlap' BCs can be redefined using Converter.elsAProfile.__FAMOVERLAPBC__ container (e.g. Converter.elsAProfile.__FAMOVERLAPBC__="OVERLAP"). In case of a doubly defined overlap grid connectivity, the default name of the family of bcs is prefixed by '**Fam_OvlpDD_**'. In that case, there is one family of classical overlap BCs per receptor base and one family of doubly defined overlap BCs too (if classical and doubly defined overlap bcs exist in that base). A '.Solver#Overlap' node is created in each family of BCs and contains the node 'NeighbourList' (where donor zones can be specified). The value is empty here. In case of a doubly defined family, a 'doubly_defined' node is added (with value 'active').

This function exists also as an in-place function (_overlapBC2GC) that modifies t and returns None.

Parameters **t** (pyTree) – input tree

Returns modified reference copy of t

Example of use:

- Convert overlap GCs to overlap BCs with families of BCs attached to each receptor base (pyTree):

```
# - overlapGC2BC (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.elsAProfile as elsAProfile

# - 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)
C._addBC2Zone(a, 'wall1', 'BCWall', 'jmin')
# Overlap BC (with automatic donor zones)
C._addBC2Zone(a, 'overlap1', 'BCOverlap', [1,80,30,30,1,2])
# Overlap BC (with given donor zones and doubly defined)
C._addBC2Zone(a, 'overlap2', 'BCOverlap', 'jmin', zoneDonor=[b],
```

(continues on next page)

(continued from previous page)

```
                rangeDonor='doubly_defined')
t = C.newPyTree(['BaseA',a,'BaseB',b])
elsAProfile._overlapGC2BC(t)
C.convertPyTree2File(t,"out.cgns")
```

Converter.elsAProfile.**fillNeighbourList**(*t*, *sameBase=0*)

Fill the NeighbourList node with families of donor zones. Donor zones are obtained by intersection of the receptor base. For doubly defined overlap BCs, the NeighbourList node is made of the family of specified donor zones. If sameBase=1, allows for donor zones to be in the same base.

This function exists also as an in-place function (_fillNeighbourList) that modifies t and returns None.

Parameters

- **t** (pyTree) – input tree
- **sameBase** (integer (0 or 1)) – if sameBase=1, donor zones are allowed on the same base as the receptor zone.

Returns modified reference copy of t

Example of use:

- Fill NeighbourList nodes (pyTree):

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

# - 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)
C._addBC2Zone(a, 'wall1', 'BCWall', 'jmin')
# Overlap BC (with automatic donor zones)
C._addBC2Zone(a, 'overlap', 'BCOverlap', 'imin')
C._addBC2Zone(a, 'overlap1', 'BCOverlap', [1,80,30,30,1,2])
# Overlap BC (with given donor zones and doubly defined)
C._addBC2Zone(a, 'overlap2', 'BCOverlap', 'jmin', zoneDonor=[b],
              rangeDonor='doubly_defined')
C._addBC2Zone(b,"OverlapDD","BCOverlap",'imin',zoneDonor=[a],rangeDonor='doubly_
↪defined')
t = C.newPyTree(['BaseA','BaseB'])
t[2][1][2]=[a]; t[2][2][2]=[b]
```

(continues on next page)

(continued from previous page)

```
elsAProfile._overlapGC2BC(t)
elsAProfile._rmGCOverlap(t)
elsAProfile._fillNeighbourList(t)
C.convertPyTree2File(t, "out.cgns")
```

Converter.elsAProfile.**prefixDnrInSubRegions**(*t*)

When Chimera connectivity (computed by Connector.PyTree.setInterpolations) is stored in the pyTree as subregion nodes of name 'ID*', corresponding donor zone names are defined by the value of the node 'ID_*'. To be read by elsA, the donor zone names are prefixed by their parent base name.

This function exists also as an in-place function (_prefixDnrInSubRegions) that modifies t and returns None.

Parameters **t** (pyTree) – input tree

Returns modified reference copy of t

Example of use:

- Prefix donor names by their base (pyTree):

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

a = G.cylinder((0,0,0),1.,3.,360,0,1,(200,30,2)); a[0] = 'cylindre1'
C._addBC2Zone(a, 'wall1', 'BCWall', 'jmin')
C._addBC2Zone(a, 'ov1', 'BCOverlap', 'jmax')
b = G.cylinder((4,0,0),1.,3.,360,0,1,(200,30,2)); b[0] = 'cylindre2'
C._addBC2Zone(b, 'wall1', 'BCWall', 'jmin')
C._addBC2Zone(b, 'ov1', 'BCOverlap', 'jmax')
c = G.cart((-5.,-7.5,0), (15./200,15./200,1), (200,200,2))
t = C.newPyTree(['Corps1', 'Corps2', 'Bgrd'])
t[2][1][2].append(a); t[2][2][2].append(b);t[2][3][2].append(c)
t = X.connectMatch(t, dim=2)
C._fillEmptyBCWith(t,'nref','BCFarfield', dim=2)
t = X.applyBCOverlaps(t, depth=1)
t = X.setInterpolations(t, loc='cell')
elsAProfile._prefixDnrInSubRegions(t)
C.convertPyTree2File(t, "out.cgns")
```

3.2 Addition of elsA nodes

Converter.elsAProfile.**addPeriodicDataInSolverParam**(*t, rotationCenter=[0.,0.,0.], rotationAngle=[0.,0.,0.], NAzimutalSectors=0, isChimera=False*)

Add information about periodicity by rotation in '.Solver#Param' node of zones of t. Exists also as an in-place function (_addPeriodicDataInSolverParam) that modifies t ans returns None.

Parameters

- **t** (pyTree, list of bases, base, list of zones, zone) – input data
- **rotationCenter** (list of 3 floats) – coordinates of rotation center
- **rotationAngle** (list of 3 floats) – rotation angle along each axis
- **NAzimutalSectors** (integer) – number of azimuthal sectors to define 360 degrees
- **isChimera** (Boolean) – to say that a zone is involved in a periodic Chimera configuration

Returns modified reference copy of t

Return type same as input

Example of use:

- 'Add periodic data (rotation) in '.Solver#Param' nodes for elsA <Examples/Converter/addPeriodicDataInSolverParamPT.py>'_:

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

a = G.cylinder((0.,0.,0.), 0.1, 1., 0., 90., 5., (11,11,11))
t = C.newPyTree(['Base', a])
tp = elsAProfile.addPeriodicDataInSolverParam(t, rotationAngle=[0.,0.,90.],␣
↪isChimera=True)
C.convertPyTree2File(tp, 'out.cgns')
```

Converter.elsAProfile.**addOutput**(*a*, *Dict*, *name=''*, *update=False*)

Add a '.Solver#Output' node suffixed by name in order to perform some extractions defined in a dictionary Dict. Dict is a python dictionary containing all output information. For example: >>> Dict={}

```
>>> Dict["var"]="convflux_rou convflux_rov convflux_row"
```

```
>>> Dict["loc"]=1
```

```
>>> Dict["fluxcoef"]=1.
```

```
>>> Dict["period"]=10
```

Name is optional to suffix the '.Solver#Output' name. If the node 'Solver#Output' already exists, it can be cleared and recreated (update=True). In the other case, if the value of the dictionary key is already set in the '.Solver#Output', it is updated. Exists also as in place version (_addOutput) that modifies the node a and returns None.

Parameters

- **a** (standard node (zone, bc,..)) – input node defining topologically the extraction (e.g. a bc node)
- **Dict** (python dictionary) – dictionary containing all the output information.
- **name** (string) – to suffix the node '.Solver#Output' by name
- **update** (Boolean) – if True, removes the existing node of name '.Solver#Output'+name in a

Returns modified reference copy of a

Return type identical to input

Example of use:

- Add a '.Solver#Output' node for elsA extraction (pyTree):

```
# - addOutput(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))
C._addBC2Zone(a, 'wall', 'BCWall', 'imin')
bcs = Internal.getNodesFromName(a, 'wall')
```

(continues on next page)

(continued from previous page)

```
dict = {}
dict['var'] = 'convflux_rou convflux_rov convflux_row'
dict['loc'] = 1
dict['fluxcoef'] = 1.
dict['period'] = 10
for b in bcs: elsAProfile._addOutput(b,dict)
C.convertPyTree2File(a,'out.cgns')
```

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*)

Add a .Solver#Output:Forces node in order to extract required value. name is a suffix that can be appened to the '.Solver#Output:Forces' name. loc is the location (integer value for elsA) for extraction. writingmode value is the integer corresponding to elsA. period is the extraction frequency

Exists also as an in-place function (_addOutputForces) that modifies node a and returns None.

Parameters

- **a** (pyTree node) – input node
- **name** (string) – suffix node name to add to '.Solver#Output:Forces'
- **var** (list of strings) – list of variables names
- **loc** (string) – suffix node name to add to '.Solver#Output:Forces'
- **writingmode** (integer) – writingmode value for elsA
- **period** (integer) – period of extraction
- **pinf** (float) – value of farfield pressure
- **fluxcoef** (float) – coefficient to correct the forces
- **torquecoef** (float) – coefficient to correct the torques
- **xyztorque** (list of float) – coordinates of the torque origin
- **frame** (string) – writing frame ('relative','absolute')
- **governingEquations** (string) – governing equations of the model
- **xtorque** (float) – X-coordinate of the torque origin
- **ytorque** (float) – Y-coordinate of the torque origin

- **ztorque** (float) – Z-coordinate of the torque origin

Returns modified reference copy of a

Return type same as input node type

Example of use:

- Add a .Solver#Output:Forces node for elsA extraction (pyTree):

```
# - 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')
```

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

Add a .Solver#Output:Friction node in order to extract required value. name is a suffix that can be appened to the '.Solver#Output:Friction' name. loc is the location (integer value for elsA) for extraction. writingmode value is the integer corresponding to elsA. period is the extraction frequency. Exists also as in place version (_addOutputFriction) that modifies t and returns None.

Parameters

- **a** (pyTree node) – input node
- **name** (string) – suffix node name to add to '.Solver#Output:Forces'
- **var** (list of strings) – list of variables names
- **loc** (string) – suffix node name to add to '.Solver#Output:Forces'
- **writingmode** (integer) – writingmode value for elsA
- **period** (integer) – period of extraction
- **fluxcoef** (float) – coefficient to correct the forces
- **torquecoef** (float) – coefficient to correct the torques
- **writingframe** (string) – writing frame ('relative','absolute')

Returns modified reference copy of a

Return type identical to a

Example of use:

- Add a .Solver#Output:Friction node for elsA solver extraction (pyTree):

```
# - 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')
```

Converter.elsAProfile.**addGlobalConvergenceHistory**(*t*, *normValue=0*)

Add a convergence node in each base of the CGNS/Python tree. The type of norm used in residual computation can be specified (0: L0, 1: L2). Exists also as an in-place function (_addGlobalConvergenceHistory) modifying t and returning None.

Parameters

- **t** (pyTree) – input tree
- **normValue** (integer (0 or 1)) – an optional integer, specifying the type of norm as value of the GlobalConvergenceHistory node.

Returns modified reference copy of t

Return type pyTree

Example of use:

- Add a global convergence history node for elsA solver (pyTree):

```
# - 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')
```

Converter.elsAProfile.**addReferenceState**(*t*, *conservative=None*, *temp=None*, *turbmod='spalart'*, *name='ReferenceState'*, *comments=None*)

Add a ReferenceState node in each base. Turbulence model, reference variables constants are mandatory. Reference temperature can also be defined. Exists also as in place version (_addReferenceState) that modifies t and returns None. Depending on the turbulence model, one, two or 7 additional constants must be defined in conservative list, that define turbulence model variables (according to the CGNS standard names):

- For turbmod='spalart':'TurbulentSANuTildeDensity' constant.
- For turbmod='komega': ['TurbulentEnergyKineticDensity','TurbulentDissipationRateDensity'] constants
- For turbmod='keps' or 'chien' or 'asm': ['TurbulentEnergyKineticDensity','TurbulentDissipationDensity']
- For turbmod='smith': ['TurbulentEnergyKineticDensity','TurbulentLengthScaleDensity']
- For turbmod='kkl' or 'earsm': ['TurbulentEnergyKineticDensity','TurbulentKineticPLSDensity']
- For turbmod='rsm': ["ReynoldsStressXX","ReynoldsStressXY","ReynoldsStressXZ","ReynoldsS

Parameters

- **t** (pyTree) – input tree
- **conservative** (list of floats) – list of constant values of reference variables [conservative + turbulence model variables].
- **temp** (float) – reference temperature (optional).
- **turbmod** (string (possible values: 'spalart','komega', 'keps','chien','asm','smith','kkl','earsm','rsm')) – turbulence model name.
- **name** (string) – name of the ReferenceState node.
- **comments** (string) – optional comments to describe the reference state (added to ReferenceStateDescription node).

Returns modified reference copy of t

Return type pyTree

Example of use:

- Add a reference state for elsA solver (pyTree):

```
# - 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')
```

Converter.elsAProfile.**addFlowSolution**(*t*, *name=''*, *loc='CellCenter'*, *variables=None*, *governingEquations=None*, *writingMode=None*, *writingFrame='relative'*, *period=None*, *output=None*, *addBCExtract=False*, *protocol="end"*)

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 value 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. Note that if governingEquations is set to None, then the governing equations set in the GoverningEquations_t node in the pyTree are chosen. If variables=[], it does not create any variable in the FlowSolution node. Exists also as in place version (_addFlowSolution) that modifies t and returns None.

Parameters

- **t** (pyTree) – input tree
- **name** (string) – suffix to append to the name 'FlowSolution'
- **loc** (string) – location of extraction ('CellCenter', 'Vertex' or 'cellfict')
- **variables** (list of string) – list of variables (elsA names)
- **governingEquations** (string) – kind of governing equations (None or standard governing equations name ('Euler', 'NSTurbulent'...)). Optional: if already defined in the pyTree in the GoverningEquations_t node, its value is chosen.
- **writingMode** (integer) – writingmode value for elsA
- **writingFrame** (string) – frame of FlowSolution ('absolute','relative')

- **period** (integer) – period of extraction
- **output** (dictionary) – an optional dictionary of node to add to '.Solver#Output' if necessary
- **addBCExtract** (Boolean) – True to add extractions from BC windows (pseudo CellFict)
- **protocol** (string) – Protocol node value ('iteration','end','after') for extractions

Returns modified reference copy of t

Return type pyTree: pyTree

Example of use:

- Add the FlowSolution nodes for elsA solver (pyTree):

```
# - 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])
t = elsAProfile.addFlowSolution(t, governingEquations='Euler')
import Converter.Internal as Internal
Internal.printTree(t)
C.convertPyTree2File(t, 'out.cgns')
```

Converter.elsAProfile.**addFlowSolutionEoR**(*t*, *name=''*, *variables=None*, *governingEquations=None*, *writingFrame='relative'*, *addBCExtract=False*, *protocol="end"*)

Add a FlowSolution#EndOfRun node located at cell centers for each zone. Parameter name is a suffix that can be appended to the 'FlowSolution#EndOfRun' name. Parameter governingEquations is the value of the 'GoverningEquations' node. If set to None, then its value is obtained from the 'GoverningEquations_t' node in the pyTree. If variables=[], it does not create any variable in the FlowSolution node. If addBCExtract is True, the boundary windows are also extracted. Parameter protocol is an optional string in 'iteration', 'end', 'after', specifying when extraction is performed. Exists also as an in-place function (_addFlowSolutionEoR) that modifies t and returns None.

Parameters

- **t** (pyTree) – input tree
- **name** (string) – suffix to append to 'FlowSolution#EndOfRun'

- **variables** (list of strings) – list of variables (elsA names)
- **governingEquations** (string) – kind of governing equations ('Euler','NSLaminar','NSTurbulent')
- **writingFrame** (string) – frame of FlowSolution ('absolute','relative')
- **addBCExtract** (Boolean) – if True, extractions are also achieved from BC windows (pseudo CellFict)
- **protocol** (string) – Protocol node value ('iteration','end','after') for extractions

Returns modified reference copy of t

Return type pyTree: pyTree

Example of use:

- Add the FlowSolution#EndOfRun nodes for elsA solver (pyTree):

```
# - addFlowSolutionEoR (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])
t = elsAProfile.addFlowSolutionEoR(t, governingEquations='Euler')
C.convertPyTree2File(t, 'out.cgns')
```

Converter.elsAProfile.**addNeighbours__**(*t*, *sameBase=0*)

Fill the NeighbourList nodes if needed with bounding-box domains intersection between bases. If sameBase=1, the intersecting domains are also searched in the current base.

Parameters

- **t** (pyTree) – input tree
- **sameBase** (integer) – choice for keeping the same current base in the bounding-box intersection (1 for yes, 0 else)

Returns modified reference copy of t

Return type pyTree

Example of use:

- Fill the NeighbourList nodes with bounding-box domains intersection (pyTree):

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

a = G.cylinder((0,0,0), 1., 1.5, 360., 0., 1., (30,30,10))
a = C.addBC2Zone(a, 'overlap1', 'BCOverlap', 'jmax')
a = C.addBC2Zone(a, 'wall1', 'BCWall', 'jmin')
b = G.cart((-10.,-10.,-10.),(0.4,0.4,0.4), (50,50,50))

t = C.newPyTree(['Cyl',a,'Cart',b])
t = X.applyBCOverlaps(t)
tp = elsAProfile.buildBCOverlap(t)
tp = elsAProfile.rmGCOverlap(tp)
tp = elsAProfile.addNeighbours__(tp)
C.convertPyTree2File(tp, 'out.cgns')
```

Converter.elsAProfile.**addTurbulentDistanceIndex**(*t*)

Add a node 'TurbulentDistanceIndex' initialized with -1 (float) in the container of flow solution at centers if TurbulentDistance node exists in the same container. To set that node in the 'FlowSolution#Init' container, be sure that the 'TurbulentDistance' node is in that container too and set: Internal.__FlowSolutionCenters__='FlowSolution#Init'. Exists also as in place version (_addTurbulentDistanceIndex) that modifies t and returns None.

Parameters **t** (pyTree, base, zone, list of zones) – input data

Returns modified reference copy of t

Return type Identical to t

Example of use:

- Add the TurbulentDistance index node for elsA solver (pyTree):

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

Internal.__FlowSolutionCenters__='FlowSolution#Init'
a = G.cart((0.,0.,0.),(0.1,0.1,0.1),(10,10,10))
t = C.newPyTree(['Base',a])
C._initVars(t, "centers:TurbulentDistance", 1.)
```

(continues on next page)

(continued from previous page)

```
tp = elsAProfile.addTurbulentDistanceIndex(t)
C.convertPyTree2File(tp, 'out.cgns')
```

Converter.elsAProfile.**createElsaHybrid**(*t*, *method=0*, *axe2D=0*, *methodPE=0*)

Add nodes required by elsA Hybrid solver for NGON zones. For elsA < 3.8.01, use method=0, for newer versions use method=1. If the mesh is 2D, use axe2D to specify 2D plane (1: (y,z), 2: (x,z), 3: (x,y)). If the mesh has poor quality cells (concave...) use methodPE=1 to build the ParentElement node in a topological manner. Exists also as in place version (_createElsaHybrid) that modifies t and returns None.

Parameters

- **t** (pyTree, base, zone, list of zones) – input data
- **method** (0 or 1) – 0 (if for elsA < 3.8.01), 1 otherwise
- **axe2D** (int) – 1 if (y,z), 2 if (x,z), 3 if (x,y)
- **methodPE** (0 or 1) – 0 (for regular mesh), 1 otherwise

Returns modified reference copy of t

Return type identical to t

Example of use:

- Create specific nodes for elsA Hybrid (pyTree):

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

# Must be NGon
a = G.cartNGon((0,0,0), (1,1,1), (10,10,10))
a = C.fillEmptyBCWith(a, 'farfield', 'BCFarfield')
elsAProfile._createElsaHybrid(a)
C.convertPyTree2File(a, 'out.cgns')
```

3.3 Miscellaneous

Converter.elsAProfile.**getCGNSkeys**(*key*, *verbose=True*)

Return the CGNS name corresponding to an elsA key.

Parameters **key** (string) – elsA key

Returns CGNS name of elsA key

Return type string

Example of use:

- Return the CGNS name corresponding to an elsA key:

```
# - getCGNSKey (pyTree) -
import Converter.elsAProfile as elsAProfile

for elsAkey in elsAProfile.keyselsA2CGNS:
    print('elsA %s corresponds to CGNS name %s'%(elsAkey,elsAProfile.
↪getCGNSkeys(elsAkey)))
```

Converter.elsAProfile.**buildMaskFiles**(*t*, *keepOversetHoles=True*, *fileDir='.'*, *prefixBase=False*)

Write the mask files in bin_v3d format. It can also keep or delete the OversetHoles nodes in the tree. The fileDir variable allows to choose the directory where the v3d hole files will be written. OversetHoles nodes can be created first by Connector.PyTree.cellN2OversetHoles. Exists also as an in-place function (_buildMaskFiles) that modifies t and returns None.

Parameters

- **t** (pyTree) – input tree
- **keepOversetHoles** (boolean) – choice for keeping or not the OversetHoles in the tree
- **fileDir** (string) – path of directory for file writing
- **prefixBase** (Boolean) – if True, add the base name to the zone name in the created file of name 'hole_*'. Results in 'hole_mybase_myzone.v3d' or 'hole_myzone.v3d'

Returns modified reference copy of t

Return type pyTree

Example of use:

- Build the mask files for elsA solver (pyTree):

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

a = G.cart((-1.,-1.,-1.),(0.1,0.1,0.1), (20,20,20))
t = C.newPyTree(['Cart',a])
```

(continues on next page)

(continued from previous page)

```
C._initVars(t, 'centers:cellN=({centers:CoordinateX}>0.)')
t = X.cellN2OversetHoles(t)
C.convertPyTree2File(t, "in.cgns")
tp = elsAProfile.buildMaskFiles(t, keepOversetHoles=False, prefixBase=True)
C.convertPyTree2File(tp, 'out.cgns')
```

Converter.elsAProfile.**convert2elsAxdt**(*t*)

Macro-function that converts some Cassiopee data stored in the pyTree into elsA-compliant nodes. It performs the following functions available as single functions:

- addTurbulentDistanceIndex(t)
- buildMaskFiles(t)
- adaptNearMatch(t)
- adaptPeriodicMatch(t)
- overlapGC2BC(t)
- rmGCOverlap(t)
- fillNeighbourList(t)
- prefixDnrInSubRegions(t)

Exists also as in place version (_convert2elsAxdt) that modifies t and returns None.

Parameters **t** (pyTree) – input tree

Returns modified reference copy of t

Return type pyTree: pyTree

Example of use:

- Convert a tree for elsA solver (pyTree):

```
# - convert2elsAxdt (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Connector.PyTree as X
import Converter.elsAProfile as CE
a = G.cylinder((0,0,0),1.,3.,360,0,1,(200,30,2)); a[0] = 'cylindre1'
C._addBC2Zone(a, 'wall1', 'BCWall', 'jmin')
C._addBC2Zone(a, 'ov1', 'BCOverlap', 'jmax')
b = G.cylinder((4,0,0),1.,3.,360,0,1,(200,30,2)); b[0] = 'cylindre2'
C._addBC2Zone(b, 'wall1', 'BCWall', 'jmin')
C._addBC2Zone(b, 'ov1', 'BCOverlap', 'jmax')
c1 = G.cart((-5.,-7.5,0), (15./200,15./200,1), (100,200,2))
```

(continues on next page)

(continued from previous page)

```
c2 = G.cart((2.425,-7.5,0), (15./200,15./100,1), (100,100,2))
t = C.newPyTree(['Corps1', 'Corps2', 'Bgrd'])
t[2][1][2].append(a); t[2][2][2].append(b);t[2][3][2]+=[c1,c2]
t = X.connectMatch(t, dim=2)
t = X.connectNearMatch(t, dim=2)
t = X.applyBCOverlaps(t, depth=1)

# blanking
C._initVars(t[2][3],'{centers:cellN}={centers:cellN}*(1.-({centers:CoordinateX}
↪<2)*({centers:CoordinateX}>-2)*({centers:CoordinateY}<2)*(
↪{centers:CoordinateY}>-2))')
t = X.setInterpolations(t, loc='cell',storage='inverse')
C._initVars(t,"centers:TurbulentDistance", 1.)
CE._convert2elsAxdt(t)
C.convertPyTree2File(t,"out.cgns")
```

CHAPTER

FOUR

INDEX

- genindex
- modindex
- search