Filter Documentation

Release 3.5

/ELSA/MU-09020/V3.5

Nov 21, 2022

CONTENTS

CHAPTER

ONE

PREAMBLE

This module provides services for partially reading or writing cgns files (HDF or ADF).

To use the module:

```
import Converter.Filter as Filter
```

CHAPTER

TWO

LIST OF FUNCTIONS

– Low level layer

Converter.Filter.convertFile2SkeletonTree(...)	Read a file and return a skeleton tree.
Converter.Filter.readNodesFromPaths(... [, ...])	Read nodes from file given their paths.
Converter.Filter.readNodesFromFilter(... [, ...])	Read nodes from file given a filter.
Converter.Filter.readPyTreeFromPaths(t, ...)	Read nodes from file given their path and complete t.
Converter.Filter.writeNodesFromPaths(... [, ...])	Write nodes to file given their paths.
Converter.Filter.writePyTreeFromPaths(t, ...)	Write some nodes of the pyTree given their paths.
Converter.Filter.writePyTreeFromFilter(t, ...)	Write nodes to file given a filter.
Converter.Filter.deletePaths(fileName, paths)	Delete nodes in file given their paths.

– High level layer

Converter.Filter.Handle(fileName)	Handle for partial reading.
Converter.Filter.Handle.loadSkeleton([...])	Load a skeleton tree.
Converter.Filter.Handle.getVariables([a, cont])	Return the variable names contained in file.
Converter.Filter.Handle.loadZones(a[, znp])	Fully load zones.
Converter.Filter.Handle.loadZonesWoVars(a[, ...])	Load zones without loading variables.

Continued on next page

Table 2 – continued from previous page

Converter.Filter.Handle.loadVariables(a, var)	Load specified variables.
Converter.Filter.Handle.writeZones(a[, ...])	Write specified zones.
Converter.Filter.Handle.writeZonesWoVars(a)	Write specified zones without variables.
Converter.Filter.Handle.writeVariables(a, var)	Write specified variables.
Converter.Filter.Handle.loadFromProc([...])	Load and distribute zones from proc node.
Converter.Filter.Handle.loadAndDistribute([...])	Load and distribute zones.
Converter.Filter.Handle.loadAndSplit([...])	Load and split a file.

CHAPTER

THREE

CONTENTS

3.1 Low level layer

Converter.Filter.**convertFile2SkeletonTree**(*fileName*, *format=None*, *maxFloatSize=5*, *maxDepth=-1*, *links=None*)

Read a skeleton tree. If float data array size of DataArray_t type nodes is lower than maxFloatSize then the array is loaded. Otherwise it is set to None. If maxDepth is specified, load is limited to maxDepth levels.

Parameters

- **fileName** (string) – file name to read from
- **format** (string) – bin_cgns, bin_adf, bin_hdf (optional)
- **maxFloatSize** (int) – the maxSize of float array to load
- **maxDepth** (int) – max depth of load
- **links** (list of list of 4 strings) – if not None, return a list of links in file

Returns Skeleton tree

Return type pyTree node

For documentation on links, see Converter read options.

Example of use:

- Read skeleton tree (pyTree):

```
# - convertFile2SkeletonTree (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Filter as Filter
import Converter.Internal as Internal
```

(continues on next page)

(continued from previous page)

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

t = Filter.convertFile2SkeletonTree('in.cgns'); Internal.printTree(t)
#>> ['CGNSTree',None,[2 sons],'CGNSTree_t']
#>>   |_['CGNSLibraryVersion',array([3.0999999046325684],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
↪']
#>>           |_['ZoneType',array('Structured',dtype='|S1'),[0 son],'ZoneType_t
↪']
#>>           |_['GridCoordinates',None,[3 sons],'GridCoordinates_t']
#>>               |_['CoordinateX',None,[0 son],'DataArray_t']
#>>               |_['CoordinateY',None,[0 son],'DataArray_t']
#>>               |_['CoordinateZ',None,[0 son],'DataArray_t']
```

Converter.Filter.**readNodesFromPaths**(*fileName*, *paths*, *format=None*, *maxFloatSize=-1*, *maxDepth=-1*, *dataShape=None*, *skipTypes=None*, *com=None*)

Read nodes specified by their paths. If maxFloatSize=-1, all data are loaded, otherwise data are loaded only if the number of elements is lower that maxFloatSize. If maxDepth=-1, the read is fully recursive. Otherwise, load is limited to maxDepth levels. If skipTypes is specified, children of nodes of given type are not loaded (HDF only).

Parameters

- **fileName** (string) – file name to read from
- **paths** (list of strings) – paths to read
- **format** (string) – bin_cgns, bin_adf, bin_hdf (optional)
- **maxFloatSize** (int) – the maxSize of float array to load
- **maxDepth** (int) – max depth of load
- **dataShape** (None or dictionary of shapes of data) – dictionary of returned data shapes if not None
- **skipTypes** (None or list of strings) – list of CGNS types to skip
- **com** (MPI communicator) – optional MPI communicator. If set, triggers parallel IO

Returns read nodes

Return type pyTree node list

Example of use:

- Read nodes from file (pyTree):

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

# Cree le fichier test
a = G.cart((0,0,0), (1,1,1), (10,10,10))
b = G.cart((12,0,0), (1,1,1), (10,10,10))
t = C.newPyTree(['Base',a,b])
C.convertPyTree2File(t, 'test.hdf')

# Relit les noeuds par leur paths
nodes = Filter.readNodesFromPaths('test.hdf', ['CGNSTree/Base/cart/
↪GridCoordinates'])
Internal.printTree(nodes)
#>> ['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']
```

Converter.Filter.**readNodesFromFilter**(*fileName*, *filter*, *format='bin_hdf'*, *com=None*)

Partially read nodes from a file specified by a filter. Filter is a dictionary for each path to be read. It specifies the slice of array you want to load from file and write to memory. Each slice information is made of [offset, stride, count, block]. Offset specifies the starting index of slice. Stride specifies the stride of slice (for example 1 means all elements, 2 means one over two, ...). Count is the number of element to read. Block is always 1.

For example, for structured grids: [[imin,jmin,kmin], [1,1,1], [imax-imin+1,jmax-jmin+1,kmax-kmin+1], [1,1,1]].

For example, for unstructured grids: [[istart], [1], [iend-imax+1], [1]].

Only for HDFfile format.

Parameters

- **fileName** (string) – file name to read from
- **filter** (dictionary of lists) – paths and indices to be read
- **format** (string) – bin_hdf
- **com** (MPI communicator) – optional MPI communicator. If set, tirggers parralel IO

Returns dictionary of read node data

Return type dictionary of numpys

Example of use:

- Partially read nodes from file (pyTree):

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

# Cree le fichier test
a = G.cart((0,0,0), (1,1,1), (10,10,10))
b = G.cart((12,0,0), (1,1,1), (10,10,10))
t = C.newPyTree(['Base',a,b])
C.convertPyTree2File(t, 'test.hdf')

# Relit les noeuds par un filtre
path = ['/Base/cart/GridCoordinates/CoordinateX',
        '/Base/cart/GridCoordinates/CoordinateY']

# start/stride/count (nbre d'entrees)/block
# Read 2x2x2 data from file, starting from 1,1,1
DataSpaceMMRY = [[0,0,0], [1,1,1], [2,2,2], [1,1,1]]
DataSpaceFILE = [[1,1,1], [1,1,1], [2,2,2], [1,1,1]]
DataSpaceGLOB = [[10,10,10]]

f = {}
f[path[0]] = DataSpaceMMRY+DataSpaceFILE+DataSpaceGLOB
f[path[1]] = DataSpaceMMRY+DataSpaceFILE+DataSpaceGLOB

# Lit seulement les chemins fournis, retourne un dictionnaire des chemins lus
a = Filter.readNodesFromFilter('test.hdf', f)
print(a[path[0]])
#>> [1. 2. 1. 2. 1. 2. 1. 2.]
```

Converter.Filter.**readPyTreeFromPaths**(*t*, *fileName*, *paths*, *format=None*, *maxFloatSize=-1*, *maxDepth=-1*, *dataShape=None*, *skipTypes=None*, *com=None*)

Read nodes of t specified by their paths. Exists also as in place function (_readPyTreeFromPaths) that modifies t and returns None.

Parameters

- **t** (pyTree) – input tree
- **fileName** (string) – file name to read from
- **paths** (list of strings) – paths to read
- **format** (string) – bin_cgns, bin_adf, bin_hdf (optional)
- **maxFloatSize** (int) – the maxSize of float array to load
- **maxDepth** (int) – max depth of load
- **dataShape** (None or dictionary of shapes of data) – dictionary of returned data shapes if not None
- **skipTypes** (None or list of strings) – list of CGNS types to skip
- **com** (MPI communicator) – optional MPI communicator. If set, triggers parallel IO

Return type modified tree

Example of use:

- Read nodes from file and modify tree (pyTree):

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

# Cree le fichier test
a = G.cart((0,0,0), (1,1,1), (10,10,10))
b = G.cart((12,0,0), (1,1,1), (10,10,10))
t = C.newPyTree(['Base',a,b])
C.convertPyTree2File(t, 'test.hdf')

t = Filter.convertFile2SkeletonTree('test.hdf', maxDepth=3)

# Complete t par leur paths
Filter._readPyTreeFromPaths(t, 'test.hdf', ['/Base/cart/GridCoordinates', 'Base/
↪cart.0/GridCoordinates'])
C.convertPyTree2File(t, 'out.hdf')
```

Converter.Filter.**writeNodesFromPaths**(*fileName*, *paths*, *nodes*, *format=None*, *maxDepth=-1*, *mode=0*)

Write given nodes to specified paths in file. If mode=0 (append), nodes are appened to path location. Nevertheless, if a node with identical name already exists in the path node children, it will be replaced by the appended ones. If mode=1 (replace), nodes are replaced to path location. If maxDepth>0, replace mode kill children of replaced node. If maxDepth=0, replace mode replaces value and type of node (not the name).

Parameters

- **fileName** (string) – file name to write to
- **paths** (list of strings) – paths to write to
- **nodes** (list of pyTree nodes) – nodes to write
- **format** (string) – bin_cgns, bin_adf, bin_hdf (optional)
- **maxDepth** (int) – max depth to write
- **mode** (int) – writing mode (0: append, 1: replace)

Example of use:

- Write nodes from paths (pyTree):

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

t = C.newPyTree(['Base'])
C.convertPyTree2File(t, 'out.hdf')

# Append
a = G.cart((0,0,0), (1,1,1), (10,10,10))
Filter.writeNodesFromPaths('out.hdf', 'CGNSTree/Base', a)

# Append and replace
a = G.cart((1,1,1), (1,1,1), (10,10,10)); a[0] = 'cart'
Filter.writeNodesFromPaths('out.hdf', 'CGNSTree/Base', a)
```

Converter.Filter.**writePyTreeFromPaths**(*t*, *fileName*, *paths*, *format=None*, *maxDepth=-1*)

Write given paths of tree to the same specified paths in file.

Parameters

- **t** (pyTree node) – input pyTree
- **fileName** (string) – file name to write to
- **paths** (list of strings) – paths to write to
- **format** (string) – bin_cgns, bin_adf, bin_hdf (optional)
- **maxDepth** (int) – max depth to write

Example of use:

- Write pyTree from paths (pyTree):

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

t = C.newPyTree(['Base'])
C.convertPyTree2File(t, 'out.hdf')

a = G.cart((0,0,0), (1,1,1), (10,10,10))
t[2][1][2] += [a]
Filter.writePyTreeFromPaths(t, 'out.hdf', 'CGNSTree/Base/cart')
```

Converter.Filter.**writePyTreeFromFilter**(*t*, *fileName*, *filter*, *format='bin_hdf'*, *com=None*)

Write partial data from tree t to file specified by a filter.

Parameters

- **t** (pyTree node) – input pyTree
- **fileName** (string) – file name to write to
- **filter** (dictionary of lists) – paths and indices to be read
- **format** (string) – bin_cgns, bin_adf, bin_hdf (optional)
- **com** (MPI communicator) – optional MPI communicator. If set, triggers parallel IO

Example of use:

- Write pyTree from filter (pyTree):

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

(continues on next page)

(continued from previous page)

```
import Converter.Filter as Filter

# Ecrit la zone en entier
t = C.newPyTree(['Base'])
a = G.cart((0,0,0), (1,1,1), (10,10,10))
t[2][1][2] += [a]
C.convertPyTree2File(t, 'out.hdf')

# Prend une subzone et la remplace dans le fichier
t = C.newPyTree(['Base'])
b = T.subzone(a, (2,2,2), (5,5,5)); b[0] = 'cart'
C._initVars(b, 'CoordinateX', 1.)
t[2][1][2] += [b]

DataSpaceMMRY = [[0,0,0], [1,1,1], [4,4,4], [1,1,1]]
DataSpaceFILE = [[2,2,2], [1,1,1], [4,4,4], [1,1,1]]
DataSpaceGLOB = [[10,10,10]]

f = {}
f['/Base/cart/GridCoordinates/CoordinateX'] =␣
↪DataSpaceMMRY+DataSpaceFILE+DataSpaceGLOB

# skelData != None since node already exists
Filter.writePyTreeFromFilter(t, 'out.hdf', f, skelData=[])
```

Converter.Filter.**deletePaths**(*fileName*, *paths*, *format=None*)

Delete paths in file.

Parameters

- **fileName** (string) – file name to read from
- **paths** (list of strings) – paths to read (relative to a)
- **format** (string) – bin_cgns, bin_adf, bin_hdf (optional)

Example of use:

- Delete some nodes in file from paths (pyTree):

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

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

(continues on next page)

(continued from previous page)

```
# Delete paths
Filter.deletePaths('out.hdf', 'CGNSTree/Base')
```

3.2 High level layer

Converter.Filter.**Handle**(*fileName*)

Create a handle on a file to enable partial reading. The file must be a CGNS/ADF or CGNS/HDF file.

Parameters **fileName** (string) – file name to read from

Return type handle class

Example of use:

- Create a handle on a file (pyTree):

```
# - Filter.Handle -
import Converter.Filter as Filter

# Create a handle on a CGNS file
h = Filter.Handle('file.hdf')
```

Converter.Filter.Handle.**loadSkeleton**(*maxDepth=3*, *readProcNode=False*)

Load a skeleton tree from file (a tree of depth maxDepth where no data are loaded).

Parameters

- **maxDepth** (int) – the depth you want to load (-1 means all tree)
- **readProcNode** (boolean) – if true, force reading of proc node. Usefull for maxDepth <= 4.

Return type a skeleton pyTree

Example of use:

- Load a skeleton tree from file (pyTree):

```
# - loadSkeleton -
import Converter.Filter as Filter
import Converter.PyTree as C
import Generator.PyTree as G
```

(continues on next page)

(continued from previous page)

```
# Create test case
a = G.cart((0,0,0), (1,1,1), (10,10,10))
C.convertPyTree2File(a, 'file.hdf')

# Create a handle on a CGNS file
h = Filter.Handle('file.hdf')

# Load skeleton
a = h.loadSkeleton()
```

Converter.Filter.Handle.**getVariables**(*cont=None*)

Get the names of variables contained in file. This function must be called after loadSkeleton.

Parameters **cont** – container name. Can be a CGNS name 'FlowSolution', ... or 'centers' or 'nodes'

Returns list of variable names contained in file

Return type list of strings

Example of use:

- Read variable list from file (pyTree):

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

# Create test case
a = G.cart((0,0,0), (1,1,1), (10,10,10))
C._initVars(a, 'F', 0)
C._initVars(a, 'centers:G', 1)
C.convertPyTree2File(a, 'file.hdf')

# Create a handle on a CGNS file
h = Filter.Handle('file.hdf')

# Load skeleton
a = h.loadSkeleton()

# Get variables from file
vars = h.getVariables(); print(vars)
#>> ['FlowSolution/F', 'FlowSolution#Centers/G']
```

Converter.Filter.Handle.**loadZones**(*a*, *znp=None*)

Fully load specified zones (coordinates, fields, grid connectivity, boundary conditions) in tree. This function must be called after loadSkeleton.

Parameters

- **a** (pyTree) – modified pyTree
- **znp** (list of strings) – paths of zones to load (must be a list of 'BaseName/ZoneName')

Example of use:

- Fully load zones (pyTree):

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

# Create test case
a = G.cart((0,0,0), (1,1,1), (10,10,10))
b = G.cart((10,0,0), (1,1,1), (10,10,10))
C.convertPyTree2File([a,b], 'file.hdf')

# Create a handle on a CGNS file
h = Filter.Handle('file.hdf')

# Load skeleton
a = h.loadSkeleton()

# Fully load all zones
h._loadZones(a)

# Fully load two zones
h._loadZones(a, znp=['Base/cart', 'Base/cart.0'])
```

Converter.Filter.Handle.**loadZonesWoVars**(*a*, *znp=None*, *bbox=None*)

Load specified zones (coordinates, grid connectivity, boundary conditions) in tree. If bbox=[xmin,ymin,zmin,xmax,ymax,zmax] is specified, load only zones intersecting this bbox. This function must be called after loadSkeleton.

Parameters

- **a** (pyTree) – modified pyTree
- **znp** (list of strings) – path of zones to load from (starting from top)
- **bbox** (list of 6 floats) – optional bbox

Example of use:

- Load zones without variable (pyTree):

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

# Create test case
a = G.cart((0,0,0), (1,1,1), (10,10,10))
C.convertPyTree2File(a, 'file.hdf')

# Create a handle on a CGNS file
h = Filter.Handle('file.hdf')

# Load skeleton
a = h.loadSkeleton()

# Load all zones without variable
h._loadZonesWoVars(a)

# Load one zone without variable
h._loadZonesWoVars(a, znp=['Base/cart'])
```

Converter.Filter.Handle.**loadVariables**(*a*, *var*, *znp=None*)

Load specified variables in tree. This function must be called after loadSkeleton.

Parameters

- **a** (pyTree) – modified pyTree
- **var** (string or list of strings) – variables to load
- **znp** (list of strings) – path of zones to load from (starting from top)

Example of use:

- Load given variables (pyTree):

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

# Create test case
a = G.cart((0,0,0), (1,1,1), (10,10,10))
```

(continues on next page)

(continued from previous page)

```
C._initVars(a, 'F', 0)
C._initVars(a, 'centers:G', 1)
C.convertPyTree2File(a, 'file.hdf')

# Create a handle on a CGNS file
h = Filter.Handle('file.hdf')

# Load skeleton
a = h.loadSkeleton()

# Load all zones without variable
h._loadZonesWoVars(a)

# Load given variables for all zones
h._loadVariables(a, var=['F', 'centers:G'])

# Load given variables for given zone
h._loadVariables(a, var=['F', 'centers:G'], znp=['Base/cart'])
```

Converter.Filter.Handle.**writeZones**(*a*, *fileName=None*, *znp=None*)

Fully write specified zones (coordinates, fields, grid connectivity, boundary conditions) in file.

Parameters

- **a** (pyTree or list of zones) – input pyTree
- **fileName** (string) – file name if different of handle name
- **znp** (list of strings) – path of zones to write to in file (starting from root)

Example of use:

- Fully write zones (pyTree):

```
# - writeZones (pyTree) -
# with Filter
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Filter as Filter

a = G.cart((0,0,0), (1,1,1), (10,10,10))
C._initVars(a, 'centers:Density=1.')
b = G.cart((11,0,0), (1,1,1), (10,10,10))
C._initVars(b, 'centers:Density=1.')
```

(continues on next page)

(continued from previous page)

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

h = Filter.Handle('out.hdf')

# Interface sur arbre (mais a doit etre mis dans t)
t[2][1][2] = [a,b]
h.writeZones(t, znp='Base/cart')

# Interface sur zones
h.writeZones(a, znp='Base/cart')
h.writeZones([a,b], znp=['Base/cart','Base/cart.0'])
```

Converter.Filter.Handle.**writeZonesWoVars**(*a*, *fileName=None*, *znp=None*)

Write specified zones without fields (coordinates, grid connectivity, boundary conditions) in file.

Parameters

- **a** (pyTree or list of zones) – input pyTree
- **fileName** (string) – file name if different of handle name
- **znp** (list of strings) – path of zones to write to in file (starting from root)

Example of use:

- Write zones without fields (pyTree):

```
# - writeZones (pyTree) -
# with Filter
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Filter as Filter

a = G.cart((0,0,0), (1,1,1), (10,10,10))
C._initVars(a, 'centers:Density=1.')
b = G.cart((11,0,0), (1,1,1), (10,10,10))
C._initVars(b, 'centers:Density=1.')
t = C.newPyTree(['Base'])
C.convertPyTree2File(t, 'out.hdf')

h = Filter.Handle('out.hdf')
h.writeZonesWoVars(a, znp='Base/cart')
h.writeZonesWoVars([a,b], znp=['Base/cart','Base/cart.0'])
```

Converter.Filter.Handle.**writeVariables**(*a*, *var*, *fileName=None*, *znp=None*)

Write specified variables in file.

Parameters

- **a** (pyTree or list of zones) – input pyTree
- **var** (string or list of strings) – variables to write
- **fileName** (string) – file name if different of handle name
- **znp** (list of strings) – path of zones to write to in file (starting from root)

Example of use:

- Write variables (pyTree):

```
# - writeVariables (pyTree) -
# with Filter
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Filter as Filter

a = G.cart((0,0,0), (1,1,1), (10,10,10))
C._initVars(a, 'centers:Density=1.')
b = G.cart((11,0,0), (1,1,1), (10,10,10))
C._initVars(b, 'centers:Density=1.')
t = C.newPyTree(['Base'])
C.convertPyTree2File(t, 'out.hdf')

h = Filter.Handle('out.hdf')
h.writeZonesWoVars(a, znp='Base/cart')
h.writeVariables(a, 'centers:Density', znp='Base/cart')
```

Converter.Filter.Handle.**loadFromProc**(*loadVariables=True*)

Load on each processor the zones with the corresponding proc node. The zones in file must have a .Solver#Param/proc node.

Parameters **loadVariables** (Boolean) – If true, load all variables in file. Otherwise load only coordinates

Return type partial tree on each processor

Example of use:

- Load tree from proc node (pyTree):

```
# - loadFromProc (pyTree) -
import Converter.Filter as Filter
import Converter.Internal as Internal
import Converter.Mpi as Cmpi

# Build case
if Cmpi.rank == 0:
    import Converter.PyTree as C
    import Generator.PyTree as G
    import Transform.PyTree as T
    import Distributor2.PyTree as D2
    a = G.cart((0,0,0), (1,1,1), (100,50,50))
    a = T.splitNParts(a, Cmpi.size)
    D2._distribute(a, Cmpi.size)
    C.convertPyTree2File(a, 'case1.cgns')
Cmpi.barrier()

h = Filter.Handle('case1.cgns')
a = h.loadFromProc()
Internal.printTree(a)
Cmpi.convertPyTree2File(a, 'out.cgns')
```

Converter.Filter.Handle.**loadAndDistribute**(*strategy=None, algorithm='graph', loadVariables=True*)

Load and distribute zones of file on the different processors.

Parameters

- **strategy** (string) – strategy for distribution. Can be None (only the number of points of block is considered), 'match' (use matching boundaries to optimize distribution)
- **algorithm** (string) – algorithm for distribution. Can be 'graph', 'fast', 'gradient'. See Distributor2 documentation.
- **loadVariables** (Boolean) – If true, load all variables in file. Otherwise load only coordinates

Return type partial tree on each processor

Example of use:

- Load and distribute tree (pyTree):

```
# - loadAndDistribute (pyTree) -
import Converter.Filter as Filter
import Converter.Internal as Internal
import Converter.Mpi as Cmpi
```

(continues on next page)

(continued from previous page)

```
# Build case
if Cmpi.rank == 0:
    import Converter.PyTree as C
    import Generator.PyTree as G
    import Transform.PyTree as T
    a = G.cart((0,0,0), (1,1,1), (100,50,50))
    a = T.splitNParts(a, Cmpi.size)
    C.convertPyTree2File(a, 'case1.cgns')
Cmpi.barrier()

h = Filter.Handle('case1.cgns')
a = h.loadAndDistribute()
Internal.printTree(a)
Cmpi.convertPyTree2File(a, 'out.cgns')
```

Converter.Filter.Handle.**loadAndSplit**(*NParts=None, NProc=Cmpi.size, splitByBase=False, algorithm='graph'*)

Load and split zones of file on the different processors (only for structured zones).

Parameters

- **NParts** (int) – number of parts to split file into. If None, use NProc instead.
- **NProc** (int) – number of target processors.
- **splitByBase** (boolean) – if true, split each base in NParts.
- **algorithm** (string) – algorithm for distribution. Can be 'graph', 'fast', 'gradient'. See Distributor2 documentation.

Return type partial tree on each processor

Example of use:

- Load and split tree (pyTree):

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

# Build case
if Cmpi.rank == 0:
    a = G.cart((0,0,0), (1,1,1), (100,100,100))
```

(continues on next page)

(continued from previous page)

```
    C.convertPyTree2File(a, 'out.hdf')
Cmpi.barrier()

# With skeleton
h = Filter.Handle('out.hdf')
t = h.loadAndSplitSkeleton(NParts=3)
h._loadContainerPartial(t, variablesN=['GridCoordinates/CoordinateX',
↪'GridCoordinates/CoordinateY','GridCoordinates/CoordinateZ'])
Cmpi.convertPyTree2File(t, 'out.cgns')

# In one go
h = Filter.Handle('out.hdf')
t = h.loadAndSplit(NParts=3)
Cmpi.convertPyTree2File(t, 'out.cgns')
```

CHAPTER

FOUR

INDEX

- genindex
- modindex
- search