Filter Documentation

Release 2.9

/ELSA/MU-09020/V2.9

Jun 07, 2019

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.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.loadZonesWoVars(a[, ...])	Load zones without loading variables.
Converter.Filter.Handle.loadVariables(a, var)	Load specified variables.

CHAPTER

THREE

CONTENTS

3.1 Low level layer

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

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

Returns Skeleton tree

Return type pyTree node

Example of use:

- Read skeleton tree (pyTree):

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

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

```
t1 = Cmpi.convertFile2SkeletonTree('in.cgns'); Internal.printTree(t1)
#>> ['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*)

Read nodes specified by their paths.

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

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)
```

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

Partially read nodes specified by a filter. Filter is a dictionary for each path to be read. For structured grids: [[imin,jmin,kmin], [1,1,1], [imax,jmax,kmax], [1,1,1]] For unstructured grids: [[istart], [1], [iend], [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** (int) – communicator if run with mpi

Returns dictionary of read nodes

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
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 = [[0]]
```

```
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]].ravel('k')
```

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

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

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 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')

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

# Complete t par leur paths
```

```
Filter._readPyTreeFromPaths(t, 'test.hdf', ['/Base/cart/GridCoordinates', 'Base/
↪cart.0/GridCoordinates'])
Internal.printTree(t)
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. 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.**_readPyTreeFromPaths**(*a*, *fileName*, *paths*, *format=None*, *maxFloatSize=-1*, *maxDepth=-1*)

Read nodes specified by their paths and stored it in a (in place).

Parameters

- **a** (`[pyTree, base, Zone, list of Zones]`) – input data
- **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)
- **maxFloatSize** (`int`) – the maxSize of float array to load
- **maxDepth** (`int`) – max depth of load

Example of use:

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

```
# - readPyTreeFromPaths (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')

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

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

`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')

# 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**()

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

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

# 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**()

Get the variables contained in file. This function minimal reads from file and store variable names in handle. This function must be called after loadSkeleton.

Returns list of variables 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.**loadZonesWoVars**(*a*, *znp=None*, *bbox=None*)

Load specified zones (coordinates, grid connectivity, boundary conditions). 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 zones without variable
h._loadZonesWoVars(a)
```

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

Load specified variables. 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))
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 zones without variable
h._loadZonesWoVars(a)

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

CHAPTER

FOUR

INDEX

- genindex
- modindex
- search