ONERA

THE FRENCH AEROSPACE LAB

Converter.Mpi Documentation

Release 2.9

/ELSA/MU-09020/V2.9

Jun 07, 2019

CONTENTS

CHAPTER

ONE

PREAMBLE

This module provides services to deal with distributed pyTrees.

A distributed pyTree is a tree where zones are distributed over different processes.

The number of a process is called the rank.

Three new concepts are introduced in addition to standard pyTrees: the **skeleton tree**, the **loaded skeleton tree** and the **partial tree**.

A **skeleton tree (S)** is a full pyTree where numpy arrays of DataArray_t type nodes are replaced by None.

A **loaded skeleton tree (LS)** is a skeleton tree for which zones attributed to the current rank are fully loaded.

A **partial tree (P)** is a pyTree with only zones attributed to the current rank fully loaded and no skeleton zones. It can be viewed as a *loaded skeleton tree* with skeleton zones suppressed.

Generally, Cassiopee functions will operate seamlessly on a partial tree, if the function doesn't requires data exchanges. If the function requires exchanges, then a specific version exists in the Module.Mpi module. For instance, for Converter, only center2Node requires exchanges and is redefined in Converter.Mpi.

To use the module:

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

To run a python script in parallel with two processes:

```
mpirun -np 2 python script.py
```

CHAPTER

TWO

LIST OF FUNCTIONS

– Input/output

`Converter.Mpi.convertFile2SkeletonTree`(fileName)	Read a file and return a skeleton tree.
`Converter.Mpi.convertFile2PyTree`(fileName[, . . .])	Read a file and return a full tree.
`Converter.Mpi.readZones`(t, fileName[, . . .])	Read some zones in a skeleton tree (by rank or name).
`Converter.Mpi.writeZones`(t, fileName[, . . .])	Write some zones in an existing file (adf or hdf).
`Converter.Mpi.readNodesFromPaths`(fileName, paths)	Read nodes from file given their paths.
`Converter.Mpi.readPyTreeFromPaths`(t, . . . [, . . .])	Read nodes from file given their path and complete t.
`Converter.Mpi.writeNodesFromPaths`(fileName, . . .)	Write nodes to file given their paths.
`Converter.Mpi.convertPyTree2File`(t, fileName)	Write a skeleton or partial tree.

– Conversion

`Converter.Mpi.convert2PartialTree`(t[, rank])	Convert a tree to a partial tree.
`Converter.Mpi.convert2SkeletonTree`(t)	Convert a tree to a skeleton tree.
`Converter.Mpi.createBBoxTree`(t[, method, . . .])	Return a bbox tree of t.

– Communication Graphs

`Converter.Mpi.getProc`(z)	Return the proc where zone is affected to.

Continued on next page

Table 2.3 – continued from previous page

`Converter.Mpi.setProc`(t, rank)	Set the proc number to a zone or a set of zones.
`Converter.Mpi.getProcDict`(t)	Return the dictionary proc['zoneName'].
`Converter.Mpi.computeGraph`(t[, type, t2, ...])	Return the communication graph for different block relation types.

– Exchanges

`Converter.Mpi.setCommunicator`(com)	Set MPI communicator to com.
`Converter.Mpi.addXZones`(t, graph[, ...])	Add zones specified in graph on current proc.
`Converter.Mpi.rmXZones`(t)	Remove zones added by addXZones.
`Converter.Mpi.allgatherTree`(t)	Gather a distributed tree on all processors.

– Actions

`Converter.Mpi.center2Node`(t[, var, ...])	Convert a zone or a field from centers to node.

CHAPTER THREE

CONTENTS

3.1 Input/output

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

Read a skeleton tree (**S**) from file (adf or hdf file format only). The loaded in memory skeleton tree is identical on all processors.

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.Mpi.**convertFile2PyTree**(*fileName*, *format=None*)

Read a full tree. The fully loaded in memory tree is identical on all processors.

Parameters

- **fileName** (string) – file name to read from
- **format** (string) – any converter format (optional)

Returns fully loaded tree

Return type pyTree node

Example of use:

- Read full tree (pyTree):

```
# - convertFile2PyTree (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()
# Identique sur tous les procs
t1 = Cmpi.convertFile2PyTree('in.cgns')
```

```
if Cmpi.rank == 1 or Cmpi.rank == 0:
    Internal.printTree(t1)
```

`Converter.Mpi.`**`readZones`**`(`*t*, *fileName*, *format=None*, *rank=None*, *zoneNames=None*`)`

Fill the data of skeleton zones of t following zone rank or zone name (adf or hdf).

If rank is not None, zone must have been attributed to ranks either with Distributor2.distribute or setProc. If the zone rank corresponds to process rank, the zone is filled with data read from file.

If zoneNames is not None, zone with corresponding name are filled with data read from file.

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

Parameters

- **t** (`[pyTree]`) – input data
- **fileName** (`string`) – file name to read from
- **format** (`string`) – bin_cgns, bin_adf, bin_hdf (optional)
- **rank** (`int`) – the processor of zones to read
- **zoneNames** (`list of strings`) – paths of zones to read (if rank is not set)

Returns modified reference copy of t

Return type Identical to t

Example of use:

- Read some zones in a skeleton tree (pyTree):

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

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

```
# Relit les zones par paths
t = Cmpi.convertFile2SkeletonTree('test.cgns')
Cmpi._readZones(t, 'test.cgns', zoneNames=['Base/cart'])

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

`Converter.Mpi.`**`writeZones`**(*t*, *fileName*, *format=None*, *rank=None*, *zoneNames=None*)

Write some zones in an existing file (adf or hdf) according to zone rank or zone name. If by rank, zone must have been attributed to processors either with Distributor2.distribute or setProc.

Parameters

- **t** ([pyTree]) – input data
- **fileName** (string) – file name to write to
- **format** (string) – bin_cgns, bin_adf, bin_hdf (optional)
- **rank** (int) – the processor of written zones
- **zoneNames** (list of strings) – paths of written zones (if rank is not set)

Example of use:

- Write some zones in an existing file (pyTree):

```
# - writeZones (pyTree) -
import Converter.PyTree as C
import Converter.Distributed as Distributed
import Distributor2.PyTree as Distributor2
import Generator.PyTree as G

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'])
C.convertPyTree2File(t, 'out.adf')
t[2][1][2] += [a,b]
(t, dic) = Distributor2.distribute(t, NProc=2, algorithm='fast')
Distributed.writeZones(t, 'out.adf', proc=0)
```

`Converter.Mpi.`**`convertPyTree2File`**(*t*, *fileName*, *format=None*)

Write a skeleton tree (**S**), a loaded skeleton tree (**LS**) or a partial tree (**P**) to a file (adf or hdf).

Parameters

- **`t`** (`[pyTree]`) – input data
- **`fileName`** (`string`) – file name to write to
- **`format`** (`string`) – bin_cgns, bin_adf, bin_hdf (optional)

Example of use:

- Write a tree to a file (pyTree):

```
# - convertPyTree2File (pyTree) -
import Converter.PyTree as C
import Converter.Mpi as Cmpi
import Generator.PyTree as G

if Cmpi.rank == 0:
    a = G.cart((0.,0.,0.),(0.1,0.1,0.1),(11,11,11))
    a[0] = 'cart0'
else:
    a = G.cart((10,0,0),(0.1,0.1,0.1),(11,11,11))
    a[0] = 'cart1'

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

`Converter.Mpi.`**`readNodesFromPaths`**(*fileName*, *paths*, *format=None*, *maxFloatSize=-1*, *maxDepth=-1*)

Read a node from a file (adf or hdf). 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.

Parameters

- **`fileName`** (`string`) – file name to write to
- **`paths`** (`string or list of strings`) – path of list of paths
- **`format`** (`string`) – bin_cgns, bin_adf, bin_hdf (optional)
- **`maxFloatSize`** – the maxSize of float array to load
- **`maxDepth`** (`int`) – max depth of load

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)
```

Note: new in version 2.6

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

Read nodes from a file (adf or hdf) and put them in pyTree. 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. Exists also as in place (_readPyTreeFromPaths) function that modifies t and returns None.

Parameters

- **t** ([pyTree]) – input data
- **fileName** (string) – file name to write to
- **paths** (string or list of strings) – path of list of paths
- **format** (string) – bin_cgns, bin_adf, bin_hdf (optional)
- **maxFloatSize** – the maxSize of float array to load
- **maxDepth** (int) – max depth of load

Example of use:

- Read nodes from file and put them to 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')
```

Note: new in version 2.6

`Converter.Mpi.`**`writeNodesFromPaths`**(*fileName*, *paths*, *nodes*, *format=None*)

Write nodes to a file (adf or hdf).

Parameters

- **`fileName`** (`string`) – file name to write to
- **`paths`** (`string or list of strings`) – path of list of paths
- **`nodes`** (`pyTree node or list of pyTree nodes`) – node or list of nodes
- **`format`** – bin_cgns, bin_adf, bin_hdf (optional)

Example of use:

- Write nodes from file (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)
```

Note: new in version 2.6

3.2 Conversions

`Converter.Mpi.`**`convert2SkeletonTree`**(*t*)

Convert a tree (**LS** or **P**) to a skeleton tree (S). In a skeleton tree, numpys in DataArray_t nodes are replaced by None.

Exists also as in place version (_convert2SkeletonTree) that modifies t and returns None.

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

Return type Identical to t

Example of use:

- Convert a tree to a skeleton tree (pyTree):

```
# - convert2SkeletonTree (pyTree) -
import Converter.PyTree as C
import Converter.Mpi as Cmpi
import Generator.PyTree as G

a = G.cart((0,0,0), (1,1,1), (10,10,10))
a = Cmpi.convert2SkeletonTree(a)
C.convertPyTree2File(a, 'out.cgns')
```

`Converter.Mpi.`**`convert2PartialTree`**(*t*, *rank=-1*)

Convert a loaded skeleton tree (**LS**) to a partial tree (**P**). If rank=-1, all skeleton zones are suppressed. If rank>=0, zones of given rank are suppressed.

Exists also as in place version (_convert2PartialTree) that modifies t and returns None.

Parameters

- **t** (`[pyTree, base, zone, list of zones]`) – input data
- **rank** – if rank=-1: suppress all skeleton zones, if rank>=0: suppress zones with proc=rank

Return type Identical to t

Example of use:

- Convert a tree to a partial tree (pyTree):

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

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

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

`Converter.Mpi.`**`createBBoxTree`**(*t*, *method='AABB'*)

From a partial tree (**P**) or a loaded skeleton tree (**LS**), create a full tree containing the bbox of zones. A bbox is a structured grid made of 8 points englobing zone. The returned tree is identical on all processors. Argument method can be 'AABB' (axis aligned bbox) or 'OBB' (oriented bbox).

Parameters

- **t** (`[pyTree, base, zone, list of zones]`) – input data
- **method** – 'AABB': axis aligned bbox, 'OBB': oriented bbox

Return type Identical to t

Example of use:

- Create a BBox tree (pyTree):

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

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

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

3.3 Graphs

`Converter.Mpi.`**`getProc`**`(z)`

Get the rank of zone. It only returns the value stored in .Solver#Param/proc node. You can use setProc or Distributor2 to create the .Solver#Param/proc node.

Parameters **z** ([zone]) – input zone

Return type int

Example of use:

- Get the proc of a zone (pyTree):

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

a = G.cart((0,0,0), (1,1,1), (10,10,10))
```

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

zones = Internal.getNodesFromType(t, 'Zone_t')
for z in zones: print(z[0]+' -> '+str(Cmpi.getProc(z)))
#>> cart -> 0
#>> cart.0 -> 1
```

`Converter.Mpi.`**`setProc`**(*t*, *rank*)

Set rank in t. t can be a skeleton (**S**), partial (**P**) or full tree. It only creates a .Solver#Param/proc node for the zones of t.

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

Parameters

- **t** (`[pyTree, base, zone, list of zones]`) – input data
- **rank** (`int`) – the rank value to set

Returns modified reference copy of t

Return type Identical to t

Example of use:

- Set the rank in zones (pyTree):

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

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

zones = Internal.getZones(t)
for z in zones: print z[0]+' -> '+str(Cmpi.getProc(z))
#>> cart -> 1
#>> cart.0 -> 1
```

`Converter.Mpi.`**`getProcDict`**(*t*)

Return the rank information stored in .Solver#Param/proc as a dictionary proc['zoneName'].

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

Return type Dictionary

Example of use:

- Get the dictionary of zone procs (pyTree):

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

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

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

procDict = Cmpi.getProcDict(t)
if Cmpi.rank == 0: print(procDict)
#>> {'cart.0': 1, 'cart': 0}
```

`Converter.Mpi.`**`computeGraph`**(*t*, *type='bbox'*, *t2=None*, *procDict=None*, *rank=0*, *intersectionsDict=None*)

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

- If type='bbox', a zone is connected to another if their bbox intersects. A must be a bbox tree.
- If type='bbox2', a zone is connected to another if their bbox intersects and are not in the same base. A must be a bbox tree.
- If type='bbox3', a zone is connected to another of another tree if their bbox intersects. A and t2 must be a bbox tree.
- If type='match' (S/LS/P), a zone is connected to another if they have a match between them. A can be a skeleton, loaded skeleton or a partial tree.
- If type='ID' (S/LS/P), a zone is connected to another if they have interpolation data between them. A can be a skeleton, a loaded skeleton or a partial tree.

- If type='IBCD' (S/LS/P), a zone is connected to another if they have IBC data between them. A can be a skeleton, a loaded skeleton or a partial tree.
- If type='ALLD' (S/LS/P), a zone is connected to another if they have Interpolation or IBC data between them. A can be a skeleton, a loaded skeleton or a partial tree.
- If type='proc', a zone is attributed to another proc than the one it is loaded on. A can be a skeleton, a loaded skeleton tree or a partial tree.

Parameters

- **t** (`[pyTree, base, zone, list of zones]`) – input data
- **type** (`string in 'bbox', 'bbox2', 'bbox3', 'match', 'ID', 'IBCD', 'ALLD', 'proc'`) – type of graph
- **t2** (`pyTree`) – optional second tree for type='bbox3'
- **procDict** (`dictionary`) – if provided, used for zone affectation
- **intersectionDict** (`python dictionary`) – dictionary of intersections

Return type python dictionary of communications

Example of use:

- Return the communication graph (pyTree):

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

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

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

```
# Cree le graph
graph = Cmpi.computeGraph(tb)
if Cmpi.rank == 0: print(graph)
```

3.4 Exchanges

`Converter.Mpi.`**`setCommunicator`**`(`*com*`)`

Set the MPI communicator for Cassiopee exchanges. By default, it is set to MPI.COMM_WORLD.

Parameters **com** (`MPI communicator`) – communicator to set

`Converter.Mpi.`**`addXZones`**`(`*t*, *graph*`)`

For a partial tree, add zones loaded on a different process that are connected to local zones through the graph.

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

Parameters

- **t** (`[pyTree]`) – input tree
- **graph** (`dictionary`) – communication graph as defined by computeGraph

Returns modified reference copy of t

Return type tree with connected zones added

Example of use:

- Add connected zones to a partial tree (pyTree):

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

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

# Relit des zones par procs
```

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

Converter.Mpi.**rmXZones**(*t*)

For a partial tree, remove zones created by addXZones.

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

Parameters t ([pyTree]) – input tree

Return type tree with connected zones suppressed

Example of use:

- Remove connected zones from a partial tree (pyTree):

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

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

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

```
t = Cmpi.rmXZones(t)
if Cmpi.rank == 0: C.convertPyTree2File(t, 'out.cgns')
```

Converter.Mpi.**allgatherTree**(*t*)

Gather a distributed tree on all processors. All processors then see the same tree.

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

Return type merged and gathered tree (identical on all processors)

Example of use:

- Gather a distributed tree (pyTree):

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

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

# Reread in parallel
t = Cmpi.convertFile2SkeletonTree('test.cgns')
(t, dic) = Distributor2.distribute(t, NProc=Cmpi.size, algorithm='fast')
t = Cmpi.readZones(t, 'test.cgns', rank=Cmpi.rank)
t = Cmpi.convert2PartialTree(t)
t = Cmpi.allgatherTree(t) # full tree on every processors
if Cmpi.rank == 0: Internal.printTree(t)
```

3.5 Actions

Converter.Mpi.**center2Node**(*t*, *var=None*, *cellNType=0*, *graph=None*)

Perform a center to node conversion for a distributed tree. If var is set, var can be a field name or a container name (Internal.__FlowSolutionNodes__, Internal.__FlowSolutionCenters__, . . .). Then, center2Node is performed in the given field. Otherwise, the zone with its coordinates is moved to node.

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

Return type merged and gathered tree (identical on all processors)

Example of use:

- Perform center2Node for a distributed tree (pyTree):

```
# - center2Node distributed -
import Converter.PyTree as C
import Distributor2.PyTree as Distributor2
import Converter.Mpi as Cmpi
import Transform.PyTree as T
import Connector.PyTree as X
import Converter.Internal as Internal
import Generator.PyTree as G

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

# Reread in parallel
sk = Cmpi.convertFile2SkeletonTree('in.cgns')
(sk, dic) = Distributor2.distribute(sk, NProc=Cmpi.size, algorithm='gradient0',
                                    useCom='match')
a = Cmpi.readZones(sk, 'in.cgns', rank=Cmpi.rank)

# center2Node
a = Cmpi.center2Node(a, 'centers:Density')
# a is now a partial tree
a = C.rmVars(a, 'centers:Density')

# Rebuild full tree in file
Cmpi.convertPyTree2File(a, 'out.cgns')
```

CHAPTER FOUR

INDEX

- genindex
- modindex
- search