Structured Grids

CFD General Notation System (CGNS)

Thomas Hauser

Utah State University, USA

Bruce Wedan

USA

Marc Poinot

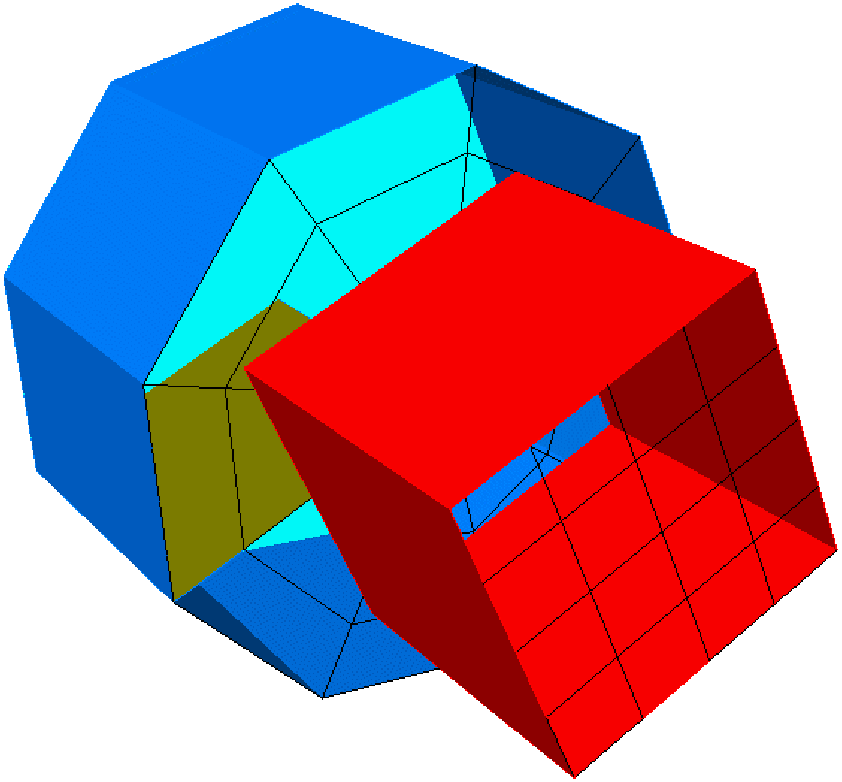
ONERA, France

Outline

- The CGNS data model top/down for structured grids
- Base
  - Zone
    - Structured Grids
    - Flow Solutions
    - Boundary Conditions
    - Connectivity between zones
- Add descriptions when needed

Example

- Cylinder attached to a cube



Example – initialize grid

```
include 'cgnslib_f.h'

!---- zone 1 - cube
do n=1,3
      idim1(n,1) = 5
      idim1(n,2) = 4
      idim1(n,3) = 0
end do
do i=1,5
      do j=1,5
            do k=1,5
                  r1(i,j,k,1) = i – 3
                  r1(i,j,k,2) = j – 3
                  r1(i,j,k,3) = k – 5
                  do n=1,5
                        q1(i,j,k,n) = n
                  enddo
            enddo
      enddo
enddo
```

```
!---- zone 2 – cylinder
do n=1,3
      idim2(n,1) = 5
      idim2(n,2) = 4
      idim2(n,3) = 0
enddo
idim2(2,1) = 10
idim2(2,2) = 9
do i=1,5
      do j=1,10
            do k=1,5
                  rad = i – 1
                  ang = 0.6981317*(j - 1)
                  r2(i,j,k,1) = rad * cos(ang)
                  r2(i,j,k,2) = rad * sin(ang)
                  r2(i,j,k,3) = k – 1
                  do n=1,5
                        q2(i,j,k,n) = n
                  enddo
            enddo
      enddo
enddo
```

The root of the tree

- The base is the computation highest structure
- Most information is contained in base
- Two bases may not share data

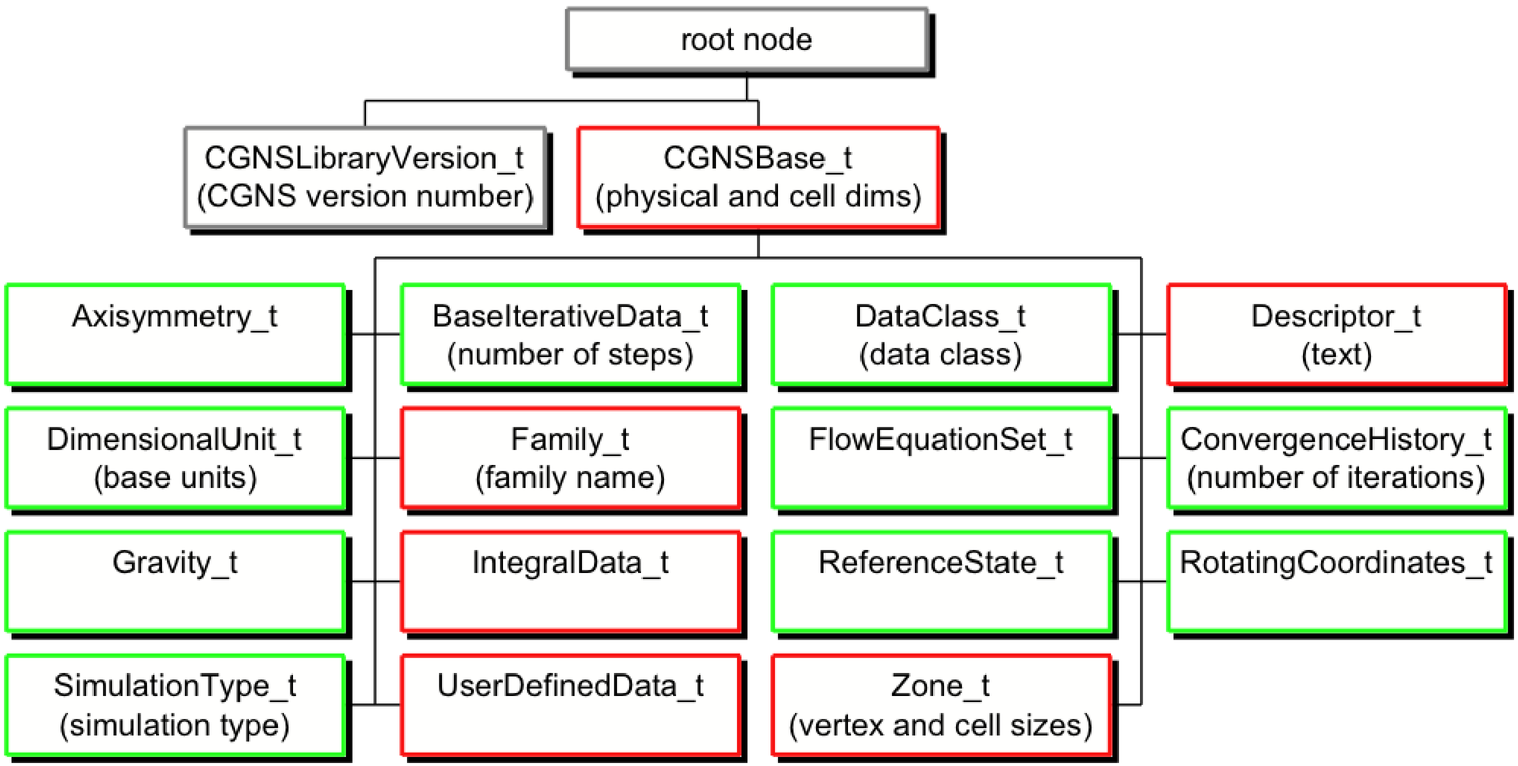
- A CGNS tree has a top node with
  - CGNSLibraryVersion
  - A list of Bases
    - Many tools only see the first base found !

CGNSBase_t

- The Base name is user defined
  - Our practice is to use the same name as filename

  - The base contains two integers within [1,2,3]
  - The physical dimension of computation
  - The topological dimension of computation
    - A 3D cube is pdim=3, cdim=3
    - A cylinder surface is pdim=3, cdim=2

Top Level Structure



MLL Base

- **Base creation**

```
cg_base_write_f(idfile, 'BaseName', cdim, pdim, idbase, errorcode)
```

```
errorcode=cg_base_write(idfile, 'BaseName', cdim, pdim, idbase)
```

- **Get number of bases in a tree**

```
errorcode=cg_nbases(idfile, nbases)
```

- **Get name, cell and physical dimensions of a base**

```
errorcode=cg_base_read(idfile, idbase, basename, cdim, pdim)
```

The Zone sub-tree

- A base can have a list of Zones
- Information related to a “space domain”:
  - Coordinates
  - Connectivity between Zones
  - Boundary conditions
  - Motion...
- Most information relative to this space domain is in the Zone sub-tree
- Other information may be found in...
  - Families

Zone

- Zone can be Structured or Unstructured
  - The CGNS data model insures a 'practical' reuse of data structures in structured or unstructured
  - You can mix structured/unstructured zones in a base, see example at the end of presentation
- Structured zone
  - No point connectivity information
  - Some unstructured data structures can be used, e.g. point list
- Zone size has strong impact on all Zone data

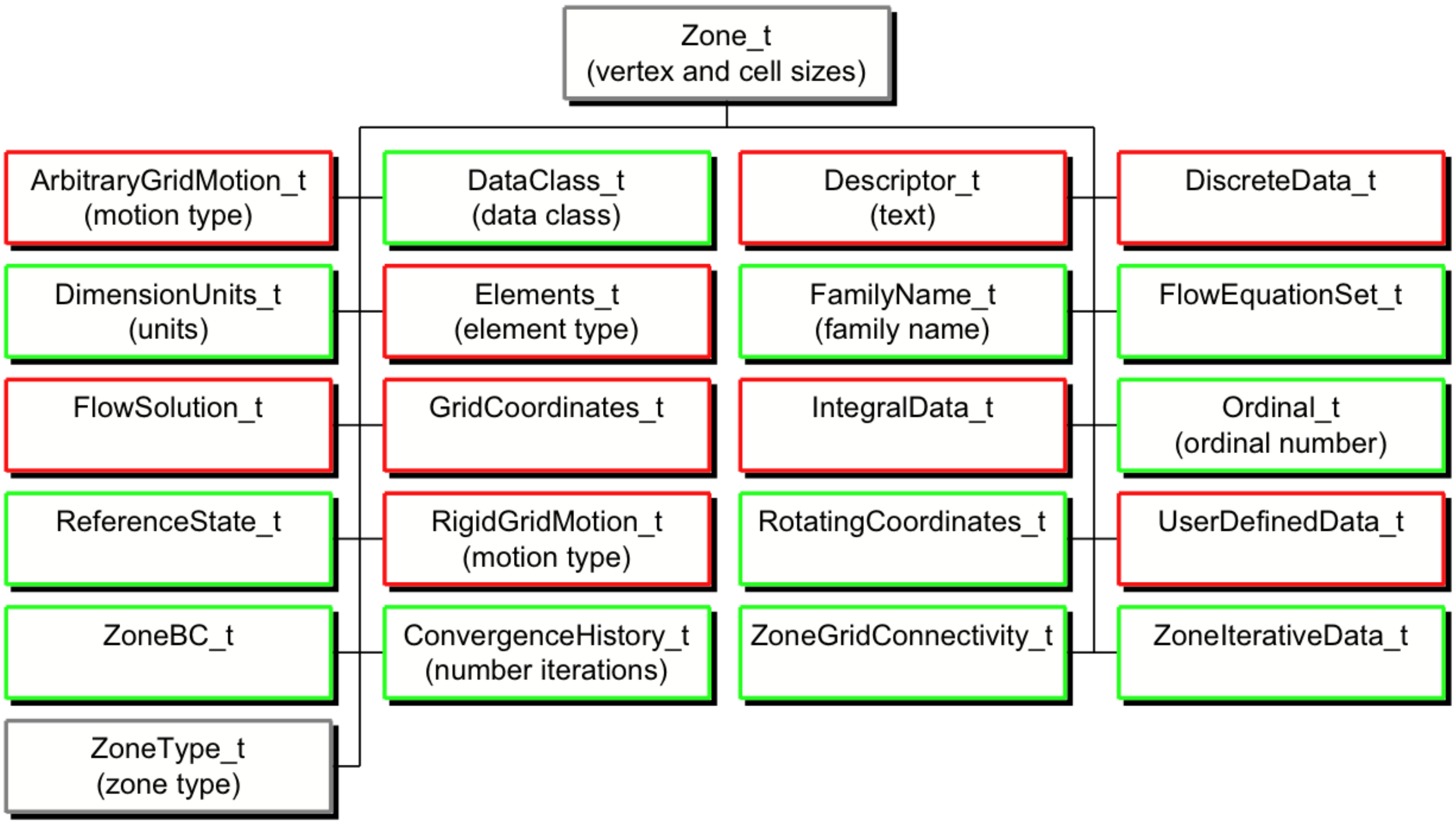
Zone_t

- Zone size information
- Related to Base dimensions
- Related to Zone type
  - Structured, Unstructured, UserDefined, Null
- Structured
  - VertexSize, CellSize, *Unused*

    `(`***`i,j,k`***`,`*`i-1,j-1,k-1,0,0,0`*`)`

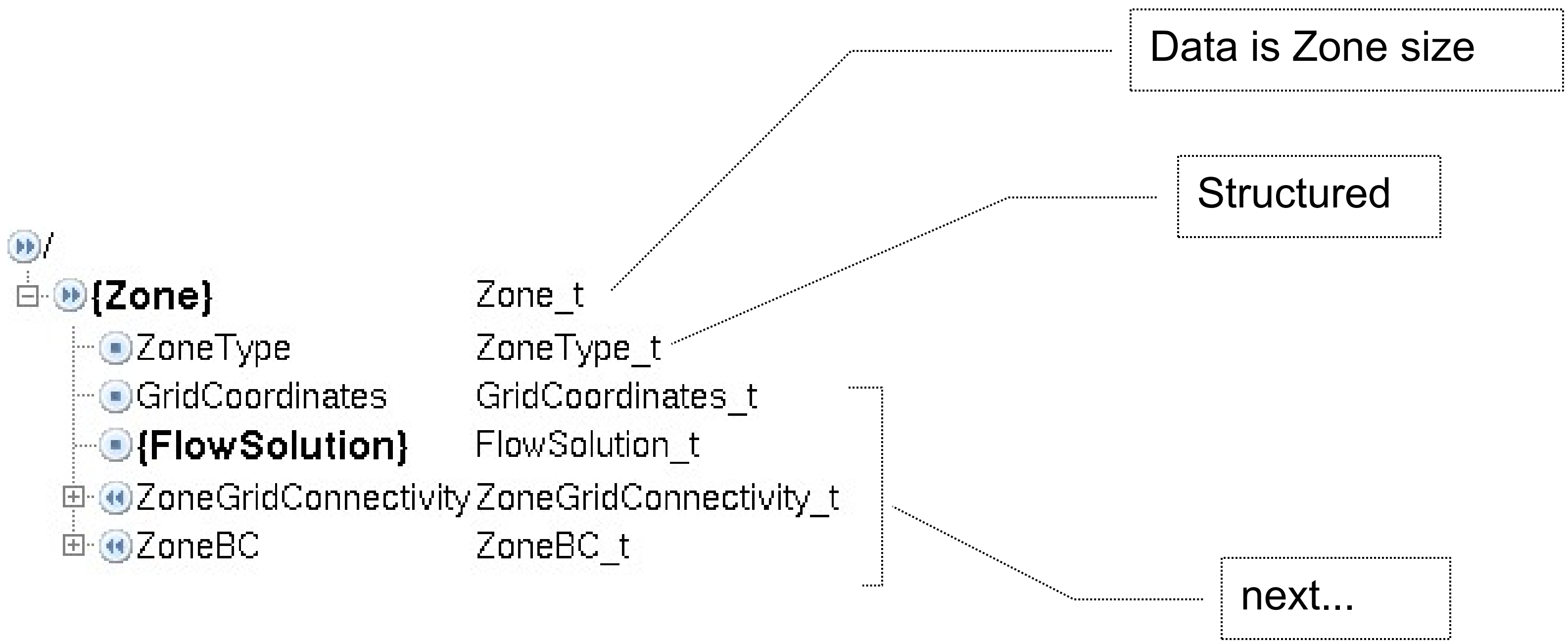
- Do not add the dummy cell size information (rind_t) in the size description

Zone_t Node



Zone_t
(vertex and cell sizes)
ArbitraryGridMotion_t
(motion type)
DataClass_t
(data class)
Descriptor_t
(text)
DiscreteData_t
DimensionUnits_t
(units)
Elements_t
(element type)
FamilyName_t
(family name)
FlowEquationSet_t
FlowSolution_t
GridCoordinates_t
IntegralData_t
Ordinal_t
(ordinal number)
ReferenceState_t
RigidGridMotion_t
(motion type)
RotatingCoordinates_t
UserDefinedData_t
ZoneBC_t
ConvergenceHistory_t
(number iterations)
ZoneGridConnectivity_t
ZoneIterativeData_t
ZoneType_t
(zone type)

Structured Zone simplified



MLL Zone

– Zone creation

```
err=cg_zone_write(idfile, idbase,'ZoneName',size,zonetype,idzone)
```

– Get Zone information

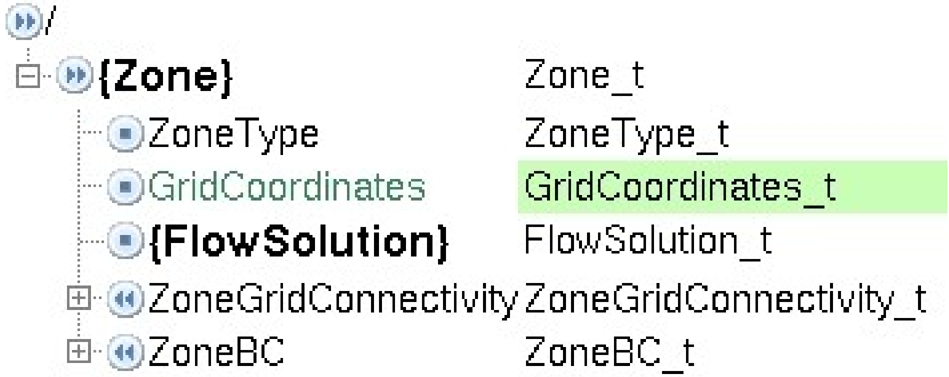
```
err=cg_nzones(idfile,idbase,nzones)

err=cg_zone_read(idfile,idbase,idzone,zonename,zonesize)

err=cg_zone_type(idfile,idbase,idzone,zonetype)
```

Example

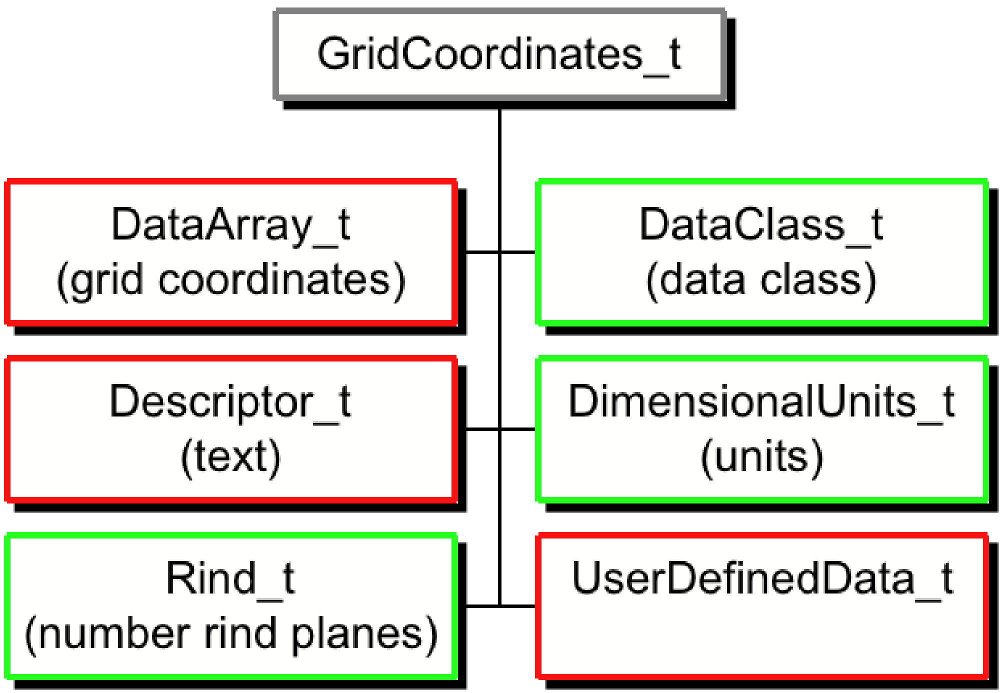
```
! ---- open file and create base
CALL  cg_open_f('example.cgns', MODE_WRITE,ifile,ierr)
IF (ierr .NE. CG_OK) CALL cg_error_exit_f

CALL cg_base_write_f(ifile,'Example',3,3,ibase,ierr)

! ----  zone 1 - cube
CALL cg_zone_write_f(ifile,ibase,'Cube',idim1,Structured, izone1,ierr)

! ----  zone 2 – cylinder
CALL  cg_zone_write_f(ifile,ibase,'Cylinder',idim2, &
    &Structured, izone2,ierr)
```

Zone mesh

- A Zone Grid is the node containing mesh points
  - Type is **`GridCoordinates_t`**
- The Grid node is a child of the Zone node
  - The default grid name is GridCoordinates
  - You can have more than one grid



GridCoordinates_t Node



Grid sub-tree

- The Grid is the mesh
  - Structured grid has no elements
    - Points connectivity is implicit
  - A grid contains set of coordinates
    - One separate array per coordinate
      - Use of Annex A of SIDS coordinates names is recommended
    - Loop ordering is Fortran (k,j,i)
      - All index ranges are (i,j,k)
    - Number of coordinates depends of *Base* dimensions
      - However no check is performed !
  - Size of coordinates array is enforced by Zone size
    - No rind data: CoordinateSize=VertexSize
    - RindData: CoordinateSize=VertexSize+RindPlaneSize

Grid coordinates example - 1

/	
GridCoordinates	DataArray_t
Rind	Rind_t
CoordinateX	DataArray_t
CoordinateY	DataArray_t
CoordinateZ	DataArray_t

Annex A:
Recommended Coordinates names w.r.t. Coordinate system
Coordinate system is not declared as a CGNS attribute

CoordinateX, CoordinateY, CoordinateZ
CoordinateR, CoordinateTheta, CoordinatePhi
CoordinateNormal

You SHOULD use these identifiers if you want to insure interoperability with pre/post tools

Rind node

- The Rind node indicates planes to count as dummy/ ghost cells
  - For each index
    - indexMin-indexRindMin
    - indexMax+indexRindMax
    - Size depends on Base CellDimensions

      `[0,0,0,0,1,1]`

      Rind planes kmin-1, kmax+1
  - Can be defined in the grid, flow solution or both
  - Default value for all Rind planes is 0

MLL GridCoordinates - 1

*These functions create/assume a "**GridCoordinates**" Grid*

– Grid & Coordinates creation

```
err=cg_coord_write(idfile,idbase,idzone,datatype,'CoordName',coordarray,idcoord)
```

– Get Coordinates information

```
err=cg_ncoords(idfile,idbase,idzone, ncoords)

err=cg_coord_info(idfile,idbase,idzone,idcoord, datatype, coordname)

err=cg_coord_read(idfile,idbase,idzone,idcoord, coordarray)
```

Example

```
! ---- write mesh for cube
 CALL cg_coord_write_f(ifile,ibase,izone1,RealSingle,'CoordinateX',&
     &rl(l,1,1,1),icoord,ierr)
 CALL cg_coord_write_f(ifile,ibase,izone1,RealSingle,'CoordinateY',&
     &rl(l,l,l,2),icoord,ierr)
 CALL cg_coord_write_f(ifile,ibase,izone1,RealSingle,'CoordinateZ',&
     &rl(l,l,l,3),icoord,ierr)

 ! ---- write mesh for cylinder
 DO n=l,3
   CALL cg_coord_write_f(ifile,ibase,izone2,RealSingle,cnames(n),&
       &r2(l,l,l,n),icoord,ierr)
 ENDDO
```

MLL GridCoordinates - 2

– Grid creation

```
err=cg_grid_write(idfile,idbase,idzone,'GridName',idgrid)
```

– Get Grid information

```
err=cg_ngrids(idfile,idbase,idzone, ngrids)
```

```
err=cg_grid_read(idfile,idbase,idzone,idgrid, gridname)
```

MLL positional nodes

- MLL knows two kinds of node types
  - Nodes with a fixed position in the data model
    - GridCoordinates is a child of Zone_t
    - Thus, a base id and a zone id are enough
  - Nodes that may be added in several places
    - A descriptor node can be a child of several types
    - Then you have to set a global cursor before access
      - the goto function
    - You can recognize the MLL functions that require a goto:
      - you have no id to pass as argument
- Usual “goto”-nodes
  - DataArray, Descriptor, UserDefinedData...

MLL Goto

– Using index and types

```
err=cg_goto(idfile,idbase,type1,index1,type2,index2,...,”end”)
```

– Using path string

```
err=cg_gopath(idfile,path)
```

```
err = cg_goto(idfile,idbase,"Zone_t",idzone,"FlowSolution_t",idflow,"end");
err = cg_gopath(idfile,"/Base-01/Zone-03/Solution-050");
```

MLL Rind – 2 ! Revise with userdefined data

- Requires a goto
- Node name is “**`Rind`**”

- Rind creation

```
err=cg_rind_write(rindarray)
```

- Rind retrieval

```
err=cg_rind_read(rindarray)
```

Array of Data

- The standard container for data

  `DataArray`

  - Often associated with dimensional information
  - Name may be fixed or user-defined
  - type can be I4, R4, R8
  - Size may depend on ancestor’s settings
  - DataArray is a leaf node
  - MLL:
    - Requires a goto
    - Midlevel library calls may create DataArrays

DataArrays everywhere !

- Usual data arrays:
  - Grid coordinates
  - Flow Solutions
  - BC local data
  - Rigid grid motion pointers
  - Convergence history
  - User defined data...

MLL DataArray

– Requires a goto

– DataArray creation (no id returned)

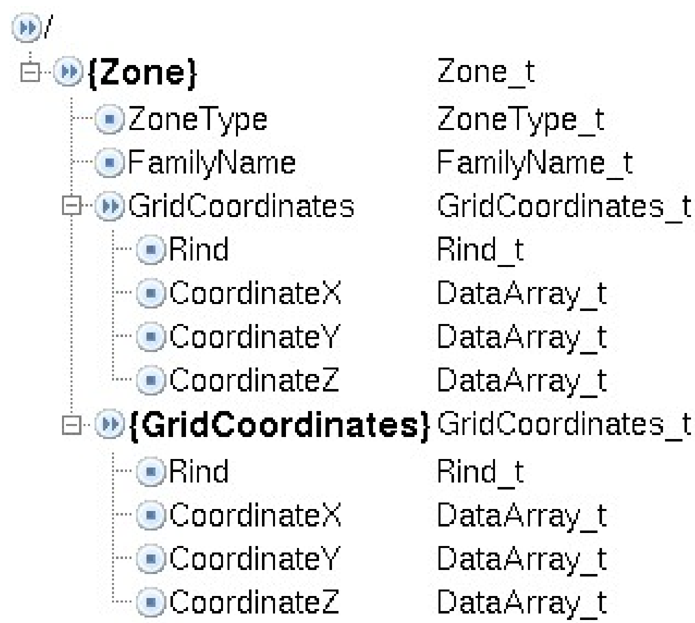
```
err=cg_array_write(arrayname,datatype,numberofdimensions,dimensions,actualda
  ta)
```

– DataArray retrieval (loop against array name)

```
err=cg_narrays(narrays)
err=cg_array_info(idarray,arrayname,datatype,numberofdimensions,dimensions)
err=cg_array_read(actualdata)
```

Coordinates at last !

- In the GridCoordinates_t
  - Coordinates are DataArrays



MLL two grids creation

```
cg_base_write(idfile, 'BaseName', cdim, pdim, idbase)
cg_zone_write(idfile, idbase, 'ZoneName', size, ZoneType_t, idzone)
cg_coord_write(idfile,idbase,idzone,DataType_t,'CoordinateX',arrayX,idcoord1
   )
cg_coord_write(idfile,idbase,idzone,DataType_t,'CoordinateY',arrayY,idcoord2
   )
cg_coord_write(idfile,idbase,idzone,DataType_t,'CoordinateZ',arrayZ,idcoord3
   )

cg_grid_write(idfile,idbase,idzone,'GridName',idgrid)
cg_goto(idfile,idbase,"Zone_t",idzone,"GridCoordinates_t",idgrid,"end");
cg_rind_write(rindarray)
cg_array_write('CoordinateX',datatype,numberofdimensions,dimensions,actualda
   ta)
cg_array_write('CoordinateY',datatype,numberofdimensions,dimensions,actualda
   ta)
cg_array_write('CoordinateZ',datatype,numberofdimensions,dimensions,actualda
   ta)
```

The Zone solutions

- Solutions nodes are children of Zone node

```
! --- write solution for cube
 CALL cg_sol_write_f(ifile,ibase,izone,'Cube Solution',Vertex,isol,ierr)

 CALL cg_field_write_f(ifile,ibase,izone,isol,RealSingle,'Density', &
     &                    q1(1,1,1,1),ifld,ierr)
 CALL cg_field_write_f(ifile,ibase,izone,isol,RealSingle,'MomentumX', &
     &                    q1(1,1,1,2),ifld,ierr)
 CALL cg_field_write_f(ifile,ibase,izone,isol,RealSingle,'MomentumY', &
     &                    q1(1,1,1,3),ifld,ierr)
 CALL cg_field_write_f(ifile,ibase,izone,isol,RealSingle,'MomentumZ', &
     &                    q1(1,1,1,4),ifld,ierr)
 CALL cg_field_write_f(ifile,ibase,izone,isol,RealSingle,'EnergyStagnationDensity', &
     &                    q1(1,1,1,5),ifld,ierr)
```

Solution 2

```
 ! --- write solution for cylinder
CALL cg_sol_write_f(ifile,ibase,izone,'Cylinder Solution',Vertex,isol,ierr)
  DO n=1,5
    CALL cg_field_write_f(ifile,ibase,izone,isol,RealSingle,snames(n),q2(1,1,1,n), &
        &                        ifld,ierr)
  ENDDO
```

Links between files

- Grid and solution are in one file
- But I really want separate files
  - Write the Grid File
    - Create Base, Zone and Write Coordinates
  - Write the Solution File
    - Create Base, Zone and Write Solution
    - Link to Coordinates in Grid File

Code for linking between files – add slide for links reading and path

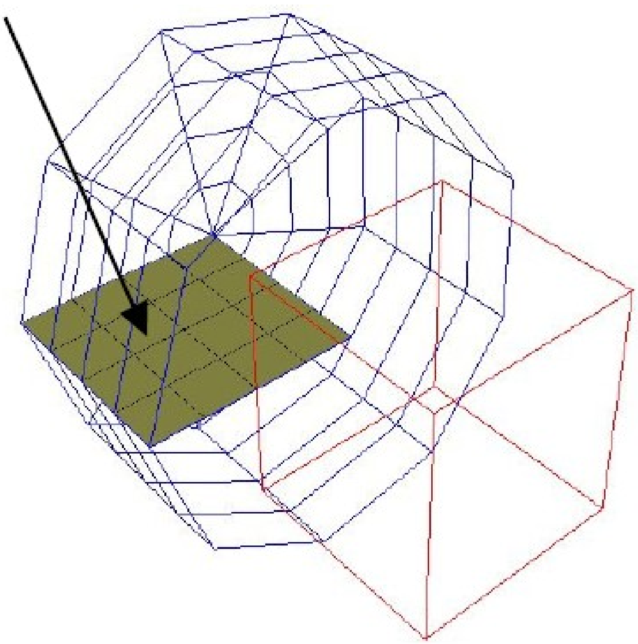
```
export ADF_LINK_PATH=$HOME/Simulations:/usr/local/data
call cg_zone_write_f(ifile,ibase,'Cube',idim1,Structured,izone,ierr)
call cg_goto_f(ifile,ibase,ierr,'Zone_t',izone,'end’)
call cg_link_write_f('GridCoordinates','grid.cgns','/Example/Cube/GridCoordinates’)
```

The Zone connectivities

- Connectivity nodes are children of Zone node
  - 1 to 1 grid connectivity
  - Mismatched and overset connectivity
  - Overset holes

Example - Connectivity

- Cylinder Cut as One to One Connection



```
! cylinder cut as one to one connection
 DO n=1,3
   transform(n) = n
   i_range(n,1) =1
   i_range(n,2) =5
   d_range(n,1) = 1
   d_range(n,2) = 5
 ENDDO
 i_range(2,2) =1
 d_range(2,1) = 10
 d_range(2,2) = 10
 CALL cg_1to1_write_f(ifile,ibase,izone,'Periodic',
 & 'Cylinder',i_range,d_range,transform,iconn,ierr)
```

The Index leaf

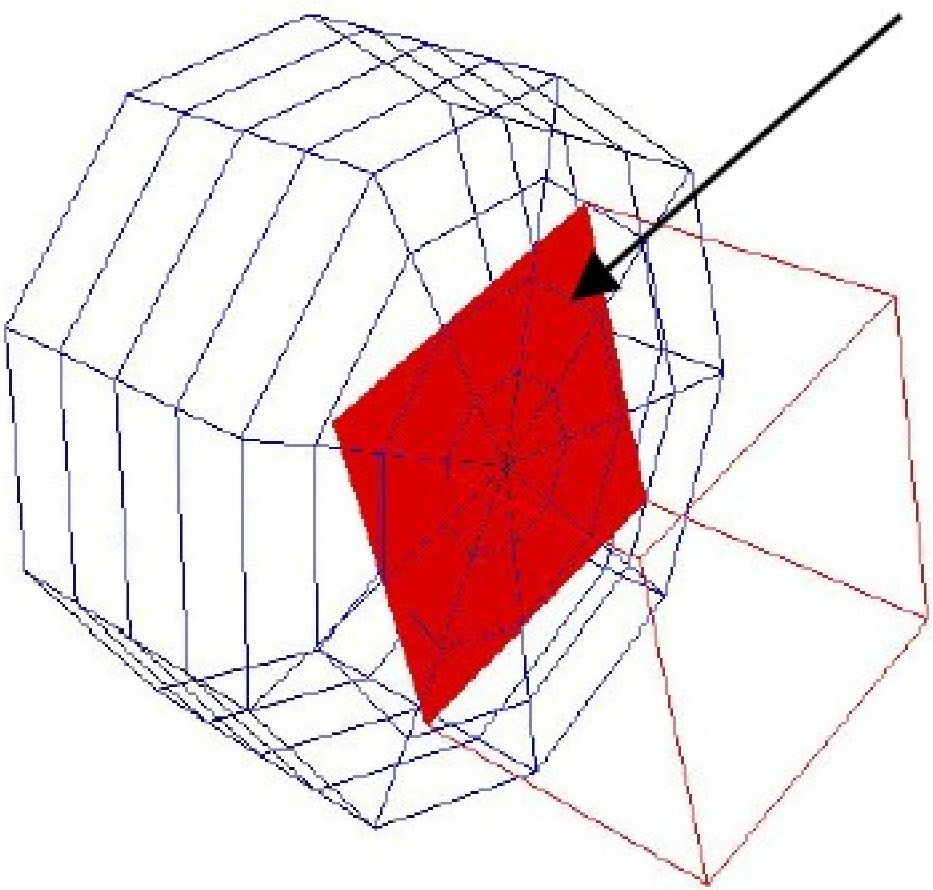
- CGNS uses a lot of index nodes
  - All of these are leaves in the data model
- IndexArray
  - A list of indices (PointList)

  `[i1,j1,k1,i2,j2,k2,...,ilast,jlast,klast]`
- IndexRange
  - A range of indices (PointRange)

  `[iBegin,jBegin,kBegin,iEnd,jEnd,kEnd]`
    - Does not require Begin>End
- int[IndexDimension]
  - List of values having CellDimension size (Transform)
    - For structured zones IndexDimension=CellDimension

Example Connectivity

- Cube to Cylinder Abbutting Connection



Abutting Connectivity

```
! cube to cylinder connectivity
  n = 0
  DO j=l,5
    DO i=l,5
      rad = SQRT(rl(i,j,5,1)**2 + rl(i,j,5,2)**2)
      ang = ATAN2(rl(i,j,5,2), rl(i,j,5,l))
      ic = rad
      IF (ic .GE. 4) ic = 3
      IF (ang .lt. 0.0) ang = ang + 6.2831853
      ang = ang / 0.6981317
      jc = ang
      IF (jc .GE. 9) jc = 8;
      pts(n+1) = i;
      pts(n+2) = j;
      pts(n+3) = 5;
      d_cell(n+l) = ic + 1 ;
      d_cell(n+2) = jc + 1 ;
      d_cell(n+3) = 1 ;
      interp(n+l) = rad - ic;
      interp(n+2) = ang - jc;
      interp(n+3) = 0.0;
      n = n + 3
    ENDDO
  ENDDO
  CALL cg_conn_write_f(ifile,ibase,izone,'Cube -> Cylinder’, Vertex,Abutting,PointList,n/3,pts, &
       'Cylinder',Structured,CellListDonor, INTEGER,n/3,d_cell,iconn,ierr)
  ! WRITE the interpolants
  CALL cg_goto_f(ifile,ibase,ierr,'Zone_t',izone, 'ZoneGridConnectivity_t',1, 'GridConnectivity_t',iconn,'end')
  dims(1) = 3 ;
  dims(2) = n / 3 ;
  CALL cg_array_write_f('InterpolantsDonor',RealSingle,2,dims,interp,ierr)
```

The Boundary conditions

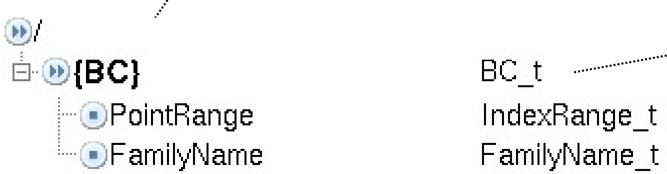
- BCs nodes are children of Zone node
  - All BC nodes are in ZoneBC
  - The ZoneBC is a mandatory node
    - Gathers all BC relative to parent Zone
  - BC are not complex
    - There are a lot of possibilities
    - You have to define your own level of use
  - You cannot map your solver BCs with CGNS Bcs
    - You have to add user defined data parts

Complete BC pattern

```
/
{BC}                                    BC_t
    PointRange                          IndexRange_t
    FamilyName                          FamilyName_t
    InwardNormalList                    IndexArray_t
    ReferenceState                      ReferenceState_t
    DataClass                           DataClass_t
    DimensionalUnits                    DimensionalUnits_t
        AdditionalUnits                 AdditionalUnits_t
    {UserDefinedData}                   UserDefinedData_t
    {Descriptor}                        Descriptor_t
    Ordinal                             Ordinal_t
    {BCDataSet}                         BCDataSet_t
        BCTypeSimple                    BCTypeSimple_t
        GridLocation                    GridLocation_t
        PointRange                      IndexRange_t
        PointList                       IndexArray_t
        {Descriptor}                    Descriptor_t
        DataClass                       DataClass_t
        DimensionalUnits                DimensionalUnits_t
            AdditionalUnits             AdditionalUnits_t
        ReferenceState                  ReferenceState_t
        {UserDefinedData}               UserDefinedData_t
        NeumannData                     BCData_t
            {Descriptor}                Descriptor_t
            {DataLocal}                 DataArray_t
            {DataGlobal}                DataArray_t
            DataClass                   DataClass_t
            DimensionalUnits            DimensionalUnits_t
                AdditionalUnits         AdditionalUnits_t
            {UserDefinedData}           UserDefinedData_t
        DirichletData                   BCData_t
            {Descriptor}                Descriptor_t
            {DataLocal}                 DataArray_t
            {DataGlobal}                DataArray_t
            DataClass                   DataClass_t
            DimensionalUnits            DimensionalUnits_t
                AdditionalUnits         AdditionalUnits_t
            {UserDefinedData}           UserDefinedData_t
```

Reasonable BC pattern

BC name is user defined



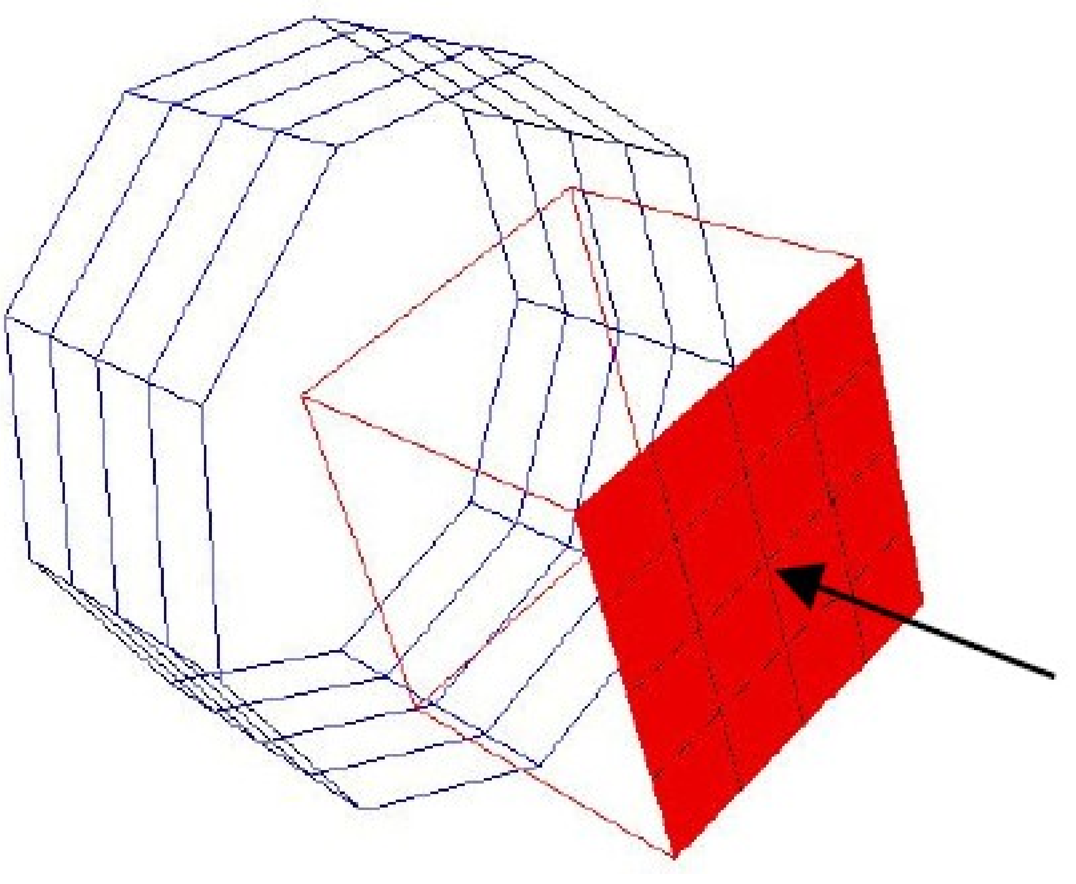
BC type is BC node data

BC patch related to its parent Zone and grid coordinates

You can gather all BC information not related to actual mesh in a family
You give the family name in the BC

Boundary Conditions

- Inlet on Cube Using Point Range

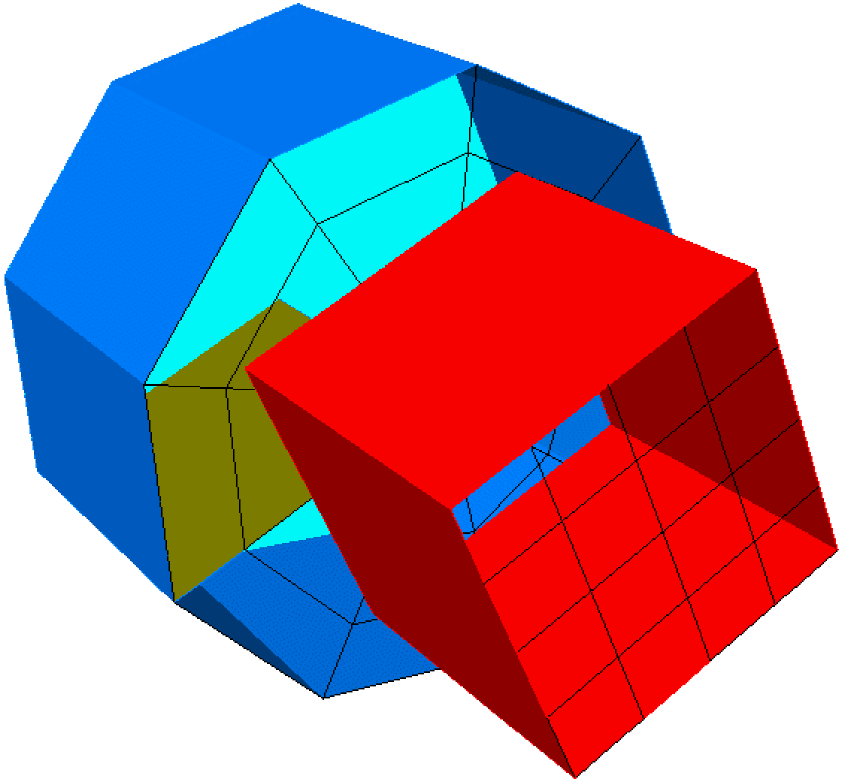


Boundary Conditions

```
! Boundary conditions
 ! ---- Inlet on Cube using point range
 DO n=l,3
   RANGE(n,1) = 1
   RANGE(n,2) = 5
 ENDDO
 RANGE(3,2) = 1
 CALL cg_boco_write_f(ifile,ibase,izone,'Inlet',BCInflow,&
     & PointRange,2,range,ibc, ierr)

 ! define inlet conditions
 CALL cg_dataset_write_f(ifile,ibase,izone,ibc, &
     & 'Inflow Conditions',BCInflowSubsonic,idset,ierr)
 CALL cg_bcdata_write_f(ifile,ibase,izone,ibc,idset, &
     & Dirichlet,ierr)

 CALL cg_goto_f(ifile,ibase,ierr,'Zone_t',izone,
 & 'ZoneBC_t ' , 1, ' BC_t ' , ibc, ' BCDataSet_t' , idset,
 & 'BCData_t',Dirichlet,'end')
 CALL cg_array_write_f('Density',RealSingle,1,1,0.9,ierr)
 CALL cg_array_write_f('VelocityX',RealSingle,1,1,1.5, ierr)
 CALL cg_array_write_f('VelocityY',RealSingle,1,1,0.0, ierr)
 CALL cg_array_write_f('VelocityZ',RealSingle,1,1,0.0, ierr)
```

Example

- Structured cylinder attached to unstructured cube



Example - Code

```
unlink("example.cgns");

cg_open("example.cgns", MODE_WRITE, &cgfile);

cg_base_write(cgfile, "Mismatched", CellDim, PhyDim, &cgbase);

cg_goto(cgfile, cgbase, "end");

cg_descriptor_write("Descriptor", "Mismatched Grid");

cg_dataclass_write(Dimensional);

cg_units_write(Kilogram, Meter, Second, Kelvin,   Radian);

/*----- zone 1 is unstructured cube -----*/

cg_zone_write(cgfile, cgbase, "UnstructuredZone",

    size, Unstructured, &cgzone);

/* write coordinates */

cg_coord_write(cgfile, cgbase, cgzone, RealSingle,    "CoordinateX",
      xcoord, &cgcoord);

cg_coord_write(cgfile, cgbase, cgzone, RealSingle,      "CoordinateY",
      ycoord, &cgcoord);

cg_coord_write(cgfile, cgbase, cgzone, RealSingle,      "CoordinateZ",
      zcoord, &cgcoord);

/* write elements and faces */

cg_section_write(cgfile, cgbase, cgzone, "Elements", HEXA_8, 1,
      num_element, 0, elements, &cgsect);

cg_section_write(cgfile, cgbase, cgzone, "Faces",     QUAD_4,
      num_element+1, num_element+num_face, 0,    faces, &cgsect);

cg_parent_data_write(cgfile, cgbase, cgzone, cgsect, parent);

/* write inflow and wall BCs */

cg_boco_write(cgfile, cgbase, cgzone, "Inlet", BCInflow, ElementRange,
      2, range, &cgbc);

cg_boco_write(cgfile, cgbase, cgzone, "Walls", BCWall, PointList, n,
      pts, &cgbc);

/*----- zone 2 is structured cylinder -----*/

cg_zone_write(cgfile, cgbase, "StructuredZone", size,       Structured,
      &cgzone);

/* write coordinates */

cg_coord_write(cgfile, cgbase, cgzone, RealSingle,   "CoordinateR",
      xcoord, &cgcoord);

cg_coord_write(cgfile, cgbase, cgzone, RealSingle,
      "CoordinateTheta", ycoord, &cgcoord);

cg_coord_write(cgfile, cgbase, cgzone, RealSingle,     "CoordinateZ",
      zcoord, &cgcoord);

/* write outlet and wall BCs */

cg_boco_write(cgfile, cgbase, cgzone, "Outlet", BCOutflow, PointRange,
      2, range, &cgbc);

cg_boco_write(cgfile, cgbase, cgzone, "Walls", BCWall, PointList, n/3,
      pts, &cgbc);

/* periodic 1to1 connectivity */

cg_1to1_write(cgfile, cgbase, 2, "Periodic", "StructuredZone", range,
      d_range, transform, &cgconn);

/*----- zone 1 -> zone 2 connectivity -----*/

cg_conn_write(cgfile, cgbase, 1, "Unstructured -> Structured", Vertex,
      Abutting, PointRange, 2, pts, "StructuredZone", Structured,
      CellListDonor, Integer, n/3, d_pts, &cgconn);

cg_goto(cgfile, cgbase, "Zone_t", 1, "ZoneGridConnectivity_t", 1,
      "GridConnectivity_t", cgconn, "end");

cg_array_write("InterpolantsDonor", RealSingle, 2, dims, interp);

/*----- zone 2 -> zone 1 connectivity similar -----*/

/* close file */

cg_close(cgfile);
```

Time Dependent Data - 1

- Means:
  - Unsteady, motion, code-coupling, polar curves...
- Overview:
    - add one node per data, use node name as key
    - add global structure to point-to data at given step and to order overall data
  - Base level: set global steps
    - Granularity should be the finest one found in the whole simulation
    - List of zones involved into the iterative change
  - Zone level: local nodes
- Pointers to zone children with respect to step

Time Dependent Data - 2

- RigidGridMotion_t
  - Actual grid unchanged, solver has to apply motion to have actual coordinates used for solution computation

Grid#001 + RigidMotion#001 = FlowSolution#001

Grid#001 stands with FlowSolution#001

Grid#002 stands with FlowSolution#002

Null used when there is no relevant data (empty cells below):

BaseIterativeData_t

IterationValues	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22
ZonePointers	A B C	A B C	A B C	A B C	A B C	A B C	A B C	A B C	A B C	A B C	B	B	B	B	B	A B	A B	A B	A B	A B	A B	A B

ZoneIterativeData_t

Grid	01	01	01	01	01	01	01	01	01	01	02	02	02	02	02	02	02	02	02	02	02	02
Motion			**01**	**01**	**01**	**02**	**02**	**02**	**02**	**02**	**03**	**03**	**03**	**03**	**03**	**03**	**03**	**03**	**04**	**04**	**04**	**04**
Flow	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**