



CGNS Tutorial

Unstructured Grids

CFD General Notation System (CGNS)

Christopher L. Rumsey
NASA Langley Research Center

Outline

- Introductory grid example
 - Showing simple hex grid written both structured and unstructured (to compare the two methods)
- Unstructured Grid
- Element Connectivity
- Boundary Conditions
- Flow Solution
- Summary

Writing structured grids

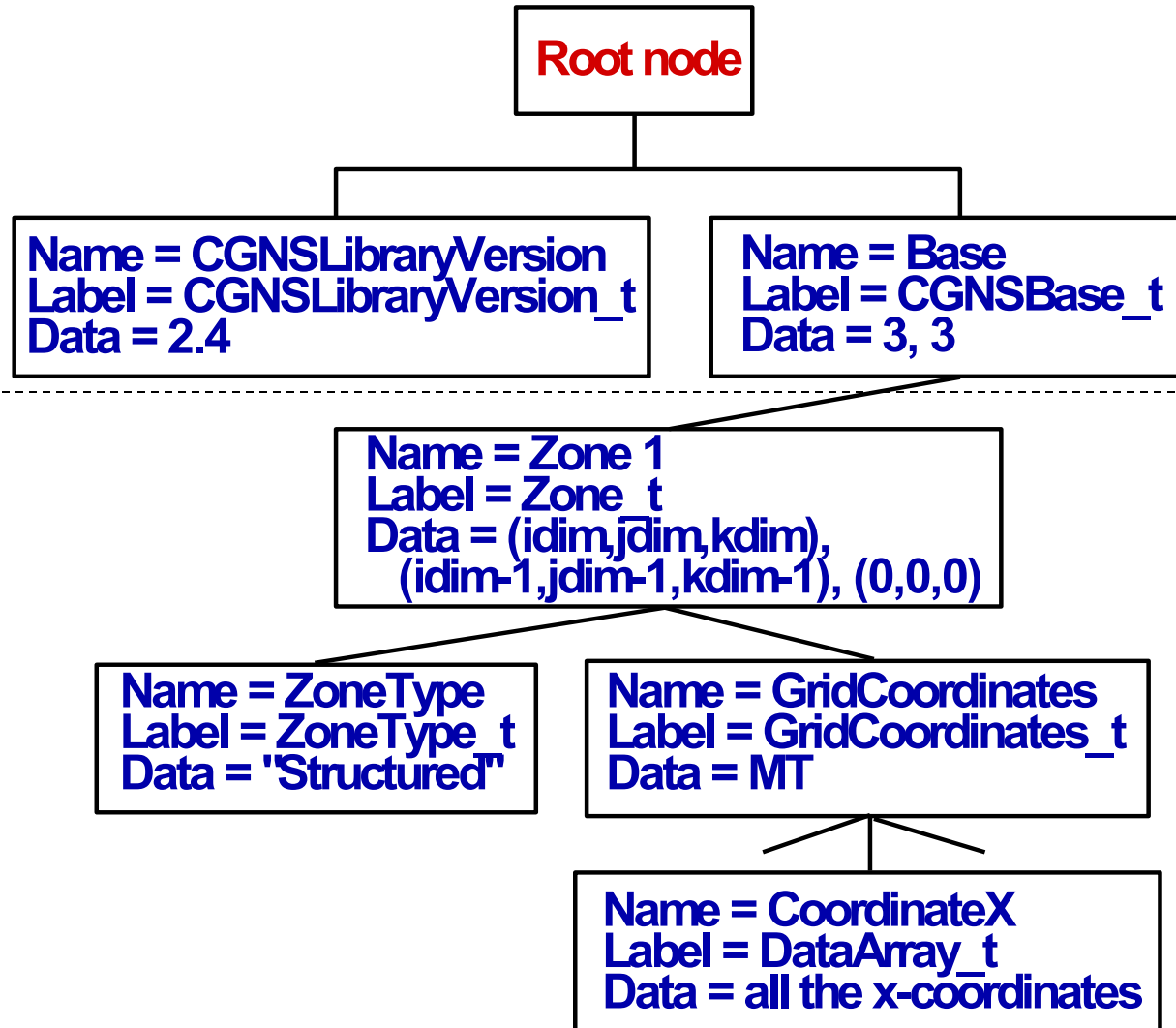
```
double x[kdim][kdim][idim], y[kdim][jdim][idim], z[kdim][jdim][idim];
int isize[3][3];
strcpy(zonename,"Zone 1");
/* vertex size (structured grid example) */
isize[0][0]=idim;
isize[0][1]=jdim;
isize[0][2]=kdim;
/* cell size (structured grid example) */
isize[1][0]=isize[0][0]-1;
isize[1][1]=isize[0][1]-1;
isize[1][2]=isize[0][2]-1;
/* boundary vertex size (always zero for structured) */
isize[2][0]=0;
isize[2][1]=0;
isize[2][2]=0;
```

Writing structured grids

(cont'd)

```
/* create zone */
cg_zone_write(indexf, indexb, zonename, isize[0], Structured, &indexz);
/* write grid coordinates */
cg_coord_write(indexf, indexb, indexz, RealDouble, "CoordinateX", x,
&indexcx);
cg_coord_write(indexf, indexb, indexz, RealDouble, "CoordinateY", y,
&indexcy);
cg_coord_write(indexf, indexb, indexz, RealDouble, "CoordinateZ", z,
&indexcz);
```

What the file looks like...



What the file looks like in adfviewer...

The screenshot shows the ADFviewer application window titled "ADFviewer : grid.cgns". The interface includes a menu bar (File, Config, Tree, Tools, Utilities, Help) and a toolbar with various icons. The main window is divided into several sections:

- Node Tree:** A hierarchical tree view showing the file structure. The path is: / > Base > Zone 1 > GridCoordinates > CoordinateX. The "CoordinateX" node is selected and highlighted in blue.
- Node Description:** A panel showing details for the selected node:
 - Parent Node: /Base/Zone 1/GridCoordinates
 - Node Name: CoordinateX (dropdown menu)
 - Node Label: DataArray_t (dropdown menu)
- Link Description:** A panel for linking to external files:
 - Link File: (text input field) [Browse]
 - Link Node: (text input field) [Browse]
- Data Description:** A panel showing data characteristics:
 - Data Type: R8 (dropdown menu)
 - Dimensions: 21 17 9 (text input field)
 - Bytes: 25704 (text input field)
- Actions:** A row of buttons: create, modify, read, clear, delete.
- Node Data:** A text area displaying the data values for the selected node. The data is organized in a grid-like format with 10 values per line. The first line is "0 1 2 3 4 5 6 7 8 9".
- Navigation:** A scrollbar on the right of the Node Data area and a status bar at the bottom showing "Line 1 (1) Values/Line 10".

Writing unstructured grids

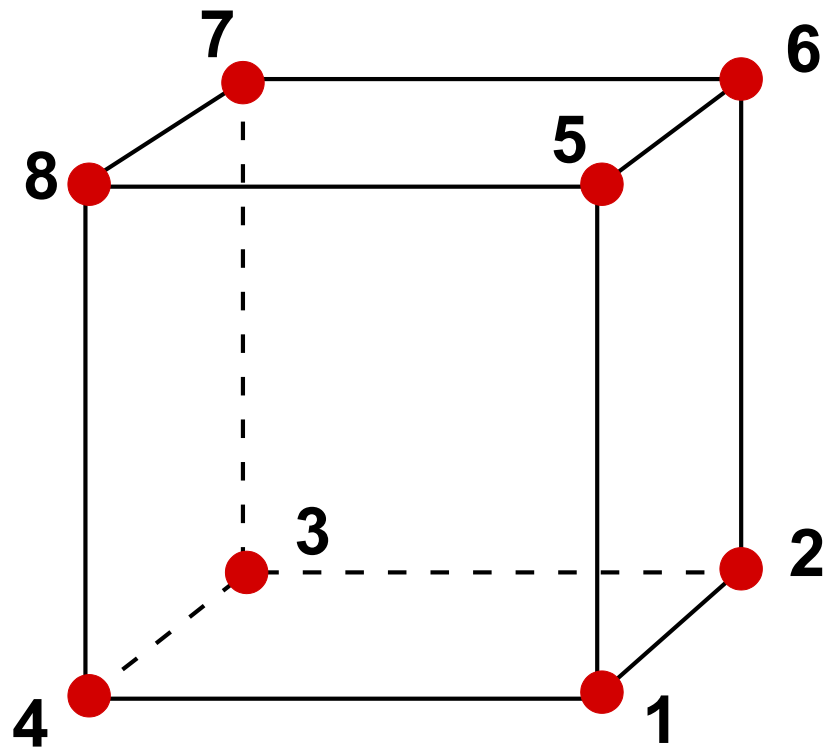
```
/* this is an example for HEXA_8 (cube-like) elements
double x[maxnodes], y[maxnodes], z[maxnodes];
int isize[3], ielem[maxelem][8];
strcpy(zonename,"Zone 1");
/* vertex size (unstructured grid example) */
isize[0]=inodedim;
/* cell size (unstructured grid example) */
isize[1]=icelldim;
/* boundary vertex size (zero if elements not sorted) */
isize[2]=ivbdy;
```

Writing unstructured grids

(cont'd)

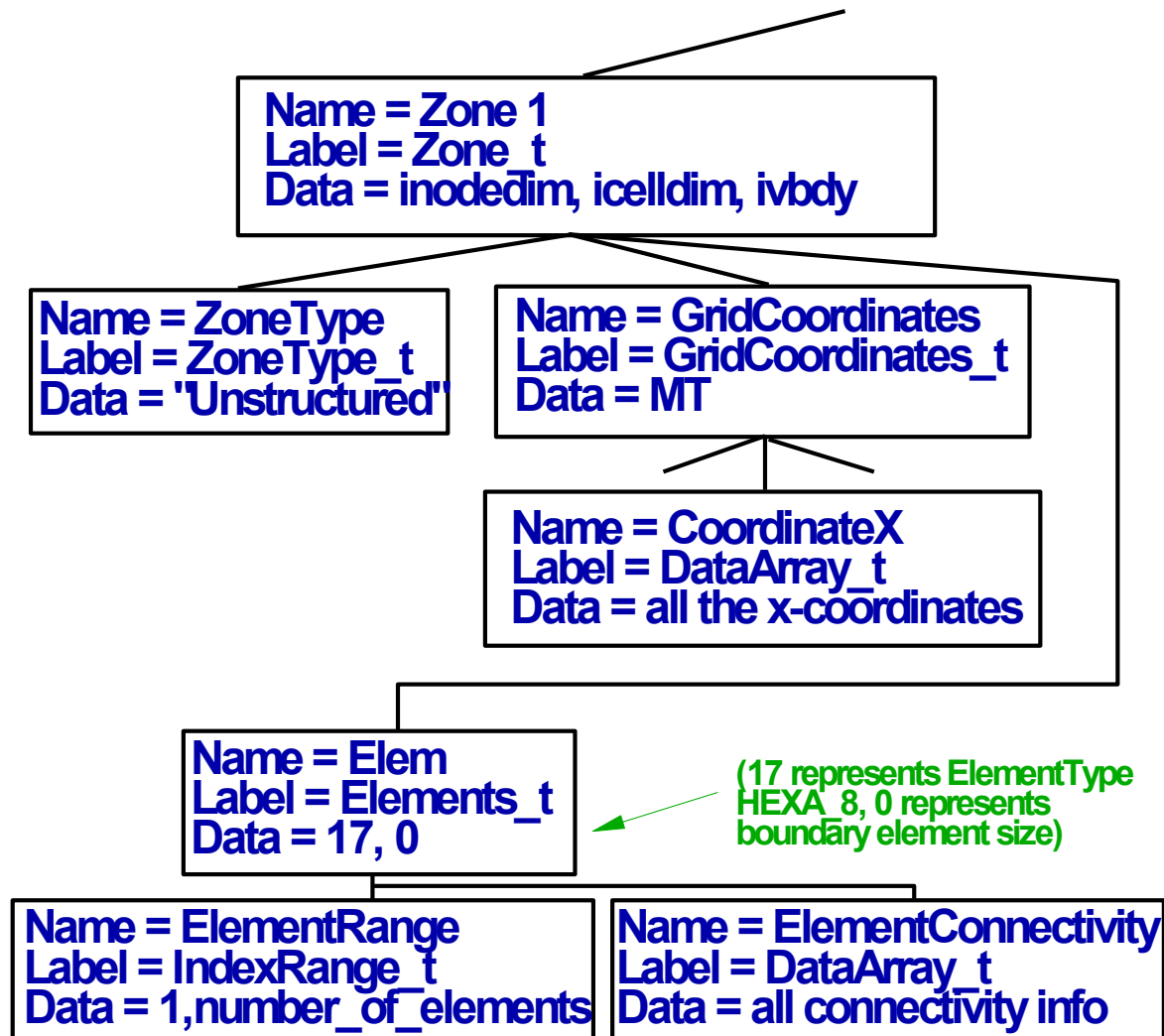
```
/* create zone */
cg_zone_write(indexf, indexb, zonename, isize, Unstructured, &indexz);
/* write grid coordinates */
cg_coord_write(indexf, indexb, indexz, RealDouble, "CoordinateX", x,
&indexcx);
cg_coord_write(indexf, indexb, indexz, RealDouble, "CoordinateY", y,
&indexcy);
cg_coord_write(indexf, indexb, indexz, RealDouble, "CoordinateZ", z,
&indexcz);
/* write element connectivity */
cg_section_write(indexf, indexb, indexz, "Elem", HEXA_8, nelem_start,
nelem_end, nbdyelem, ielem[0], &indexe);
```


Element connectivity for HEXA_8



What the file looks like...

(below Base)



What the file looks like in adfviewer...

The screenshot shows the ADFviewer application window titled "ADFviewer : grid_c.cgns". The interface includes a menu bar (File, Config, Tree, Tools, Utilities, Help) and a toolbar with various icons. The main area is divided into two panels:

- Node Tree:** A hierarchical tree view showing the structure of the CGNS file. The selected node is "ElementConnectivity" under "Elem".
- Node Description:** A panel showing details for the selected node. It includes fields for Parent Node, Node Name, and Node Label. Below this are sections for Link Description (Link File, Link Node) and Data Description (Data Type, Dimensions, Bytes). At the bottom of this panel are buttons for "create", "modify", "read", "clear", and "delete".

The Node Data panel displays the following data:

```
1 2 23 22 358 359 380 379
2 3 24 23 359 360 381 380
3 4 25 24 360 361 382 381
4 5 26 25 361 362 383 382
5 6 27 26 362 363 384 383
6 7 28 27 363 364 385 384
7 8 29 28 364 365 386 385
8 9 30 29 365 366 387 386
9 10 31 30 366 367 388 387
10 11 32 31 367 368 389 388
```

At the bottom of the Node Data panel, it shows "Line 1 (1)" and "Values/Line 8".

Unstructured Grid

```
GridCoordinates_t< int IndexDimension, int VertexSize[IndexDimension] > :=  
{  
  List( Descriptor_t Descriptor1 ... DescriptorN ) ;           (o)  
  Rind_t<IndexDimension> Rind ;                               (o/d)  
  List( DataArray_t<DataType, IndexDimension, DataSize[]>  
    DataArray1 ... DataArrayN ) ;                             (o)  
  DataClass_t DataClass ;                                    (o)  
  DimensionalUnits_t DimensionalUnits ;                     (o)  
  List( UserDefinedData_t UserDefinedData1 ... UserDefinedDataN ) ; (o)  
};
```

- Overall SIDS definition same for both structured and unstructured grids
- For unstructured: IndexDimension always 1, VertexSize=number of nodes (excluding any rind points)

Unstructured Grid, MLL

For 3-D structured:

```
cg_zone_write(indexf, indexb, zonename, isize, Structured, &indexz);
```

isize is dimensioned [3][3]:

idim, jdim, kdim

idim-1, jdim-1, kdim-1

0, 0, 0

For 3-D unstructured:

```
cg_zone_write(indexf, indexb, zonename, isize, Unstructured, &indexz);
```

isize is dimensioned [1][3]:

#nodes

#cells

optional boundary vertex size

Element Connectivity

- Grid points alone are not enough for unstructured grids: element connectivity information is also required
 - Uses global numbering system under a given Zone_t (i.e., all elements in a zone must have a unique number, beginning at 1)

Elements_t :=

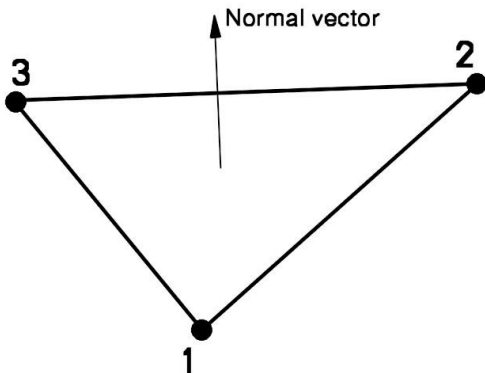
```
{  
List( Descriptor_t Descriptor1 ... DescriptorN ) ;           (o)  
Rind_t<IndexDimension> Rind ;                               (o/d)  
IndexRange_t ElementRange ;                               (r)  
int ElementSizeBoundary ;                                 (o/d)  
ElementType_t ElementType ;                               (r)  
dataArray_t<int, 1, ElementDataSize> ElementConnectivity ; (r)  
dataArray_t<int, 2, [ElementSize, 4]> ParentData;         (o)  
List( UserDefinedData_t UserDefinedData1 ... UserDefinedDataN ) ; (o)  
};
```

Element Connectivity, cont'd

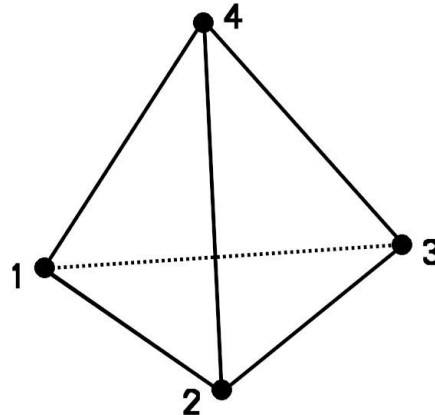
- Current element types supported:

```
ElementType_t := Enumeration(  
  Null, NODE, BAR_2, BAR_3,  
  TRI_3, TRI_6, QUAD_4, QUAD_8, QUAD_9,  
  TETRA_4, TETRA_10, PYRA_5, PYRA_14,  
  PENTA_6, PENTA_15, PENTA_18,  
  HEXA_8, HEXA_20, HEXA_27, MIXED, NGON_n, UserDefined );
```

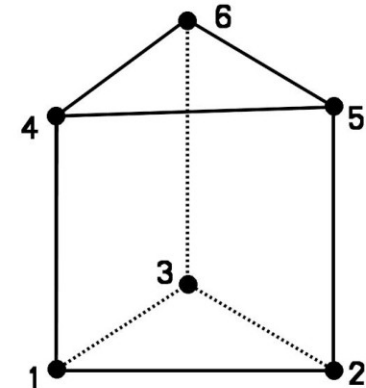
TRI_3



TETRA_4



PENTA_6



Element Connectivity, cont'd

For most element types:

ElementConnectivity =

Node11, Node21, ... NodeN1,	← nodes for element 1
Node12, Node22, ... NodeN2,	← nodes for element 2
...	
Node1M, Node2M, ... NodeNM	← nodes for element M

For MIXED:

ElementConnectivity =

Etype1, Node11, Node21, ... NodeN1,
Etype2, Node12, Node22, ... NodeN2,
...
EtypeM, Node1M, Node2M, ... NodeNM

Element Connectivity = MIXED

```
count=1
```

```
! first element is HEXA_8
```

```
element(count)=HEXA_8
```

```
  do i=1,8
```

```
    count = count + 1
```

```
    element(count)=...
```

```
  enddo
```

```
! second element TETRA_4
```

```
  count = count + 1
```

```
  element(count)=TETRA_4
```

```
  do i=1,4
```

```
    count = count + 1
```

```
    element(count)=...
```

```
  enddo
```

```
! write element connectivity
```

```
call cg_section_write_f(indexf, indexb, indexz, 'Elem', MIXED, nelem_start, nelem_end,  
  nbdyelem, ielem, indexe, ier)
```

Element Connectivity, cont'd

NEW... For arbitrary polyhedra:

The NGON_n element type is used to specify all the faces in the grid, and the NFACE_n element type is then used to define the polyhedral elements as a collection of these faces. Except for boundary faces, each face of a polyhedral element must be shared by another polyhedral element.

I.e., for NGON_n, the data array ElementConnectivity contains a list of nodes making up each face in the grid, with the first value for each face defining the number of nodes making up that face:

```
ElementConnectivity = Nnodes1, Node11, Node21, ... NodeN1,  
                     Nnodes2, Node12, Node22, ... NodeN2,  
                     ...  
                     NnodesM, Node1M, Node2M, ... NodeNM
```

where here M is the total number of faces, and Ni is the number of nodes in face i. The ElementDataSize is the total number of nodes defining all the faces, plus one value per face specifying the number of nodes making up that face.

Element Connectivity, cont'd

NEW... For arbitrary polyhedra (cont'd):

Then for NFACE_n, ElementConnectivity contains the list of face elements making up each polyhedral element, with the first value for each element defining the number of faces making up that element.

ElementConnectivity = Nfaces1, Face11, Face21, ... FaceN1,
Nfaces2, Face12, Face22, ... FaceN2,
...
NfacesM, Face1M, Face2M, ... FaceNM

where now M is the total number of polyhedral elements, and N_i is the number of faces in element i . The sign of the face number determines its orientation (i.e., the direction of the face normal)

Notes

- BEST PRACTICE: When writing unstructured grids, **always** include boundary elements and **try** to give each boundary type as a separate node if possible
 - Helps others decipher the file
 - Extremely useful when using plotting packages like Tecplot
 - Also helps establish boundary condition locations when reading the file

Element Connectivity, MLL example

```
/* write element connectivity */
```

```
cg_section_write(indexf, indexb, indexz, "Elem", HEXA_8, nelem_start1,  
    nelem_end1, nbdyelem1, ielem, &indexe);
```

```
/* add element connectivity for all boundaries */
```

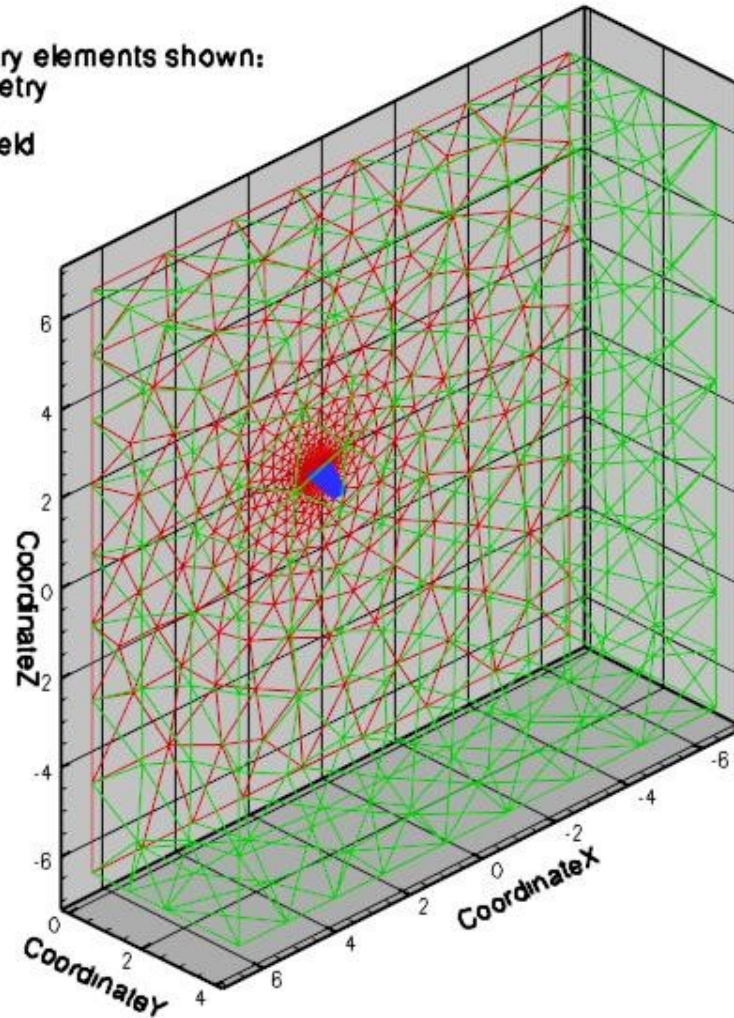
```
cg_section_write(indexf,indexb,indexz,"BdyWalls",QUAD_4,nelem_start2,  
    nelem_end2, nbdyelem2, kelem, &indexe);
```

```
cg_section_write(indexf,indexb,indexz,"BdyFarfield",QUAD_4,nelem_start3,  
    nelem_end3, nbdyelem3, lelem, &indexe);
```

```
cg_section_write(indexf,indexb,indexz,"BdySymm",QUAD_4,nelem_start4,  
    nelem_end4, nbdyelem4, melem, &indexe);
```

Example viewed with Tecplot

Only boundary elements shown:
Red = symmetry
Blue = wing
Green = farfield



Boundary Conditions, MLL

- BC implementation essentially identical for Structured and Unstructured grids
- However, although it is allowed to list the BCs as a function of the boundary **node points** (if desired), for Unstructured it often makes more sense to associate the BCs with boundary **elements** (ElementList or ElementRange) instead
 - It seems that this method of association is the most common practice currently in the unstructured grid community

```
call cg_boco_write_f(index_file,index_base,index_zone,'llo',  
+ BCTunnelInflow,ElementList,icount,ipnts,index_bc,ier)
```

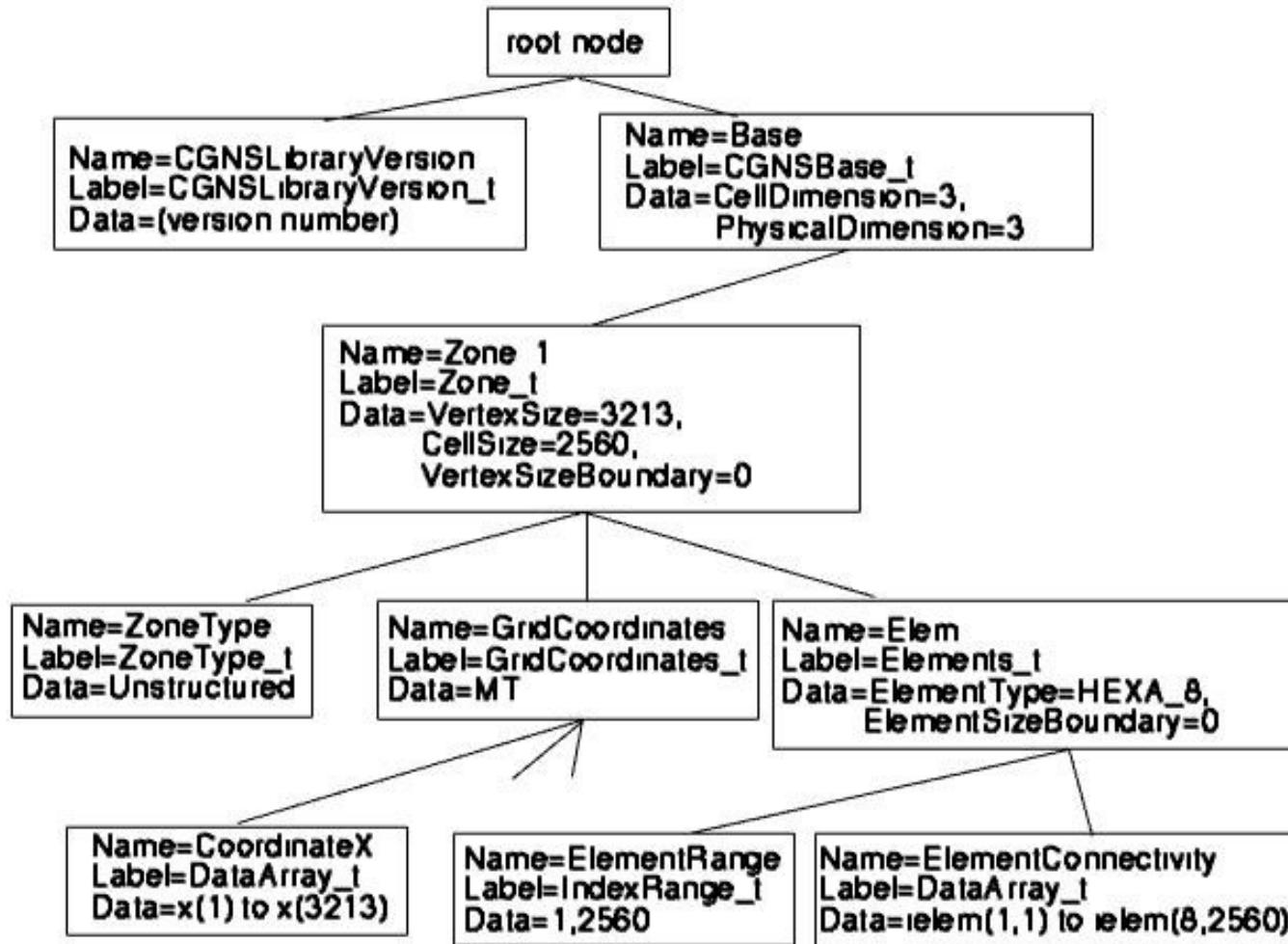
Boundary Conditions (cont'd)

```
ZoneBC_t< int IndexDimension, int PhysicalDimension > :=  
{  
  List( Descriptor_t Descriptor1 ... DescriptorN ) ;           (o)  
  List( BC_t<IndexDimension, int PhysicalDimension> BC1 ... BCN ) ; (o)  
  ReferenceState_t ReferenceState ;                             (o)  
  DataClass_t DataClass ;                                       (o)  
  DimensionalUnits_t DimensionalUnits ;                         (o)  
  List( UserDefinedData_t UserDefinedData1 ... UserDefinedDataN ) ; (o)  
};
```


Boundary Conditions (cont'd)

```
BC_t< int IndexDimension, int PhysicalDimension > :=  
{  
  List( Descriptor_t Descriptor1 ... DescriptorN ) ;      (o)  
  BCType_t BCType ;                                     (r)  
  GridLocation_t GridLocation ;                        (o/d)  
  IndexRange_t<IndexDimension> PointRange ;           (r:o:o:o)  
  IndexArray_t<IndexDimension, ListLength, int> PointList ; (o:r:o:o)  
  IndexRange_t<IndexDimension> ElementRange ;         (o:o:r:o)  
  IndexArray_t<IndexDimension, ListLength, int> ElementList ; (o:o:o:r)  
  int[IndexDimension] InwardNormalIndex ;             (o)  
  IndexArray_t<PhysicalDimension, ListLength, real> InwardNormalList ; (o)  
  List( BCDataSet_t<ListLength> BCDataSet1 ... BCDataSetN ) ; (o)  
  BCProperty_t BCProperty ;                           (o)  
  FamilyName_t FamilyName ;                           (o)  
  ReferenceState_t ReferenceState ;                   (o)  
  DataClass_t DataClass ;                             (o)  
  DimensionalUnits_t DimensionalUnits ;               (o)  
  List( UserDefinedData_t UserDefinedData1 ... UserDefinedDataN ) ; (o)  
  int Ordinal ;                                       (o)  
};
```

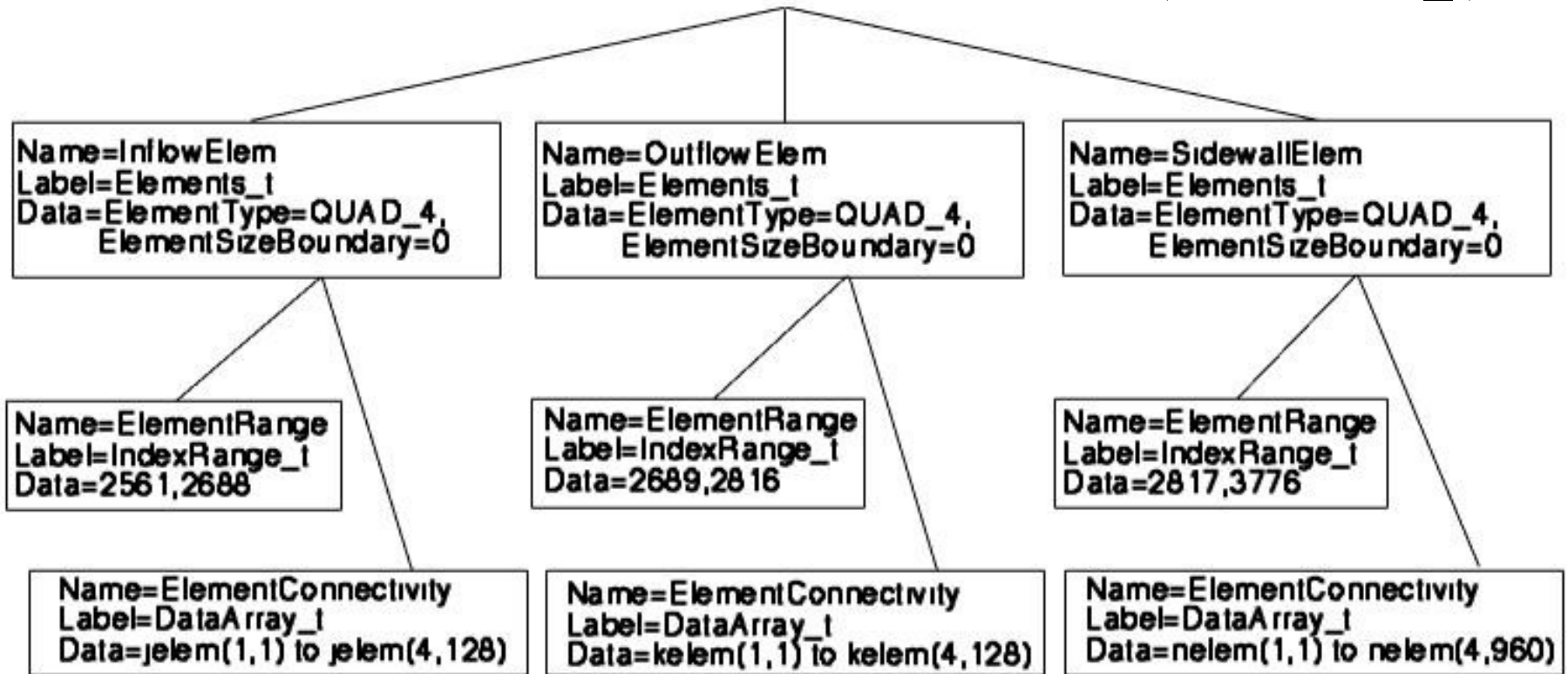
Example Unstructured Tree



Example Unstructured Tree

(cont'd)

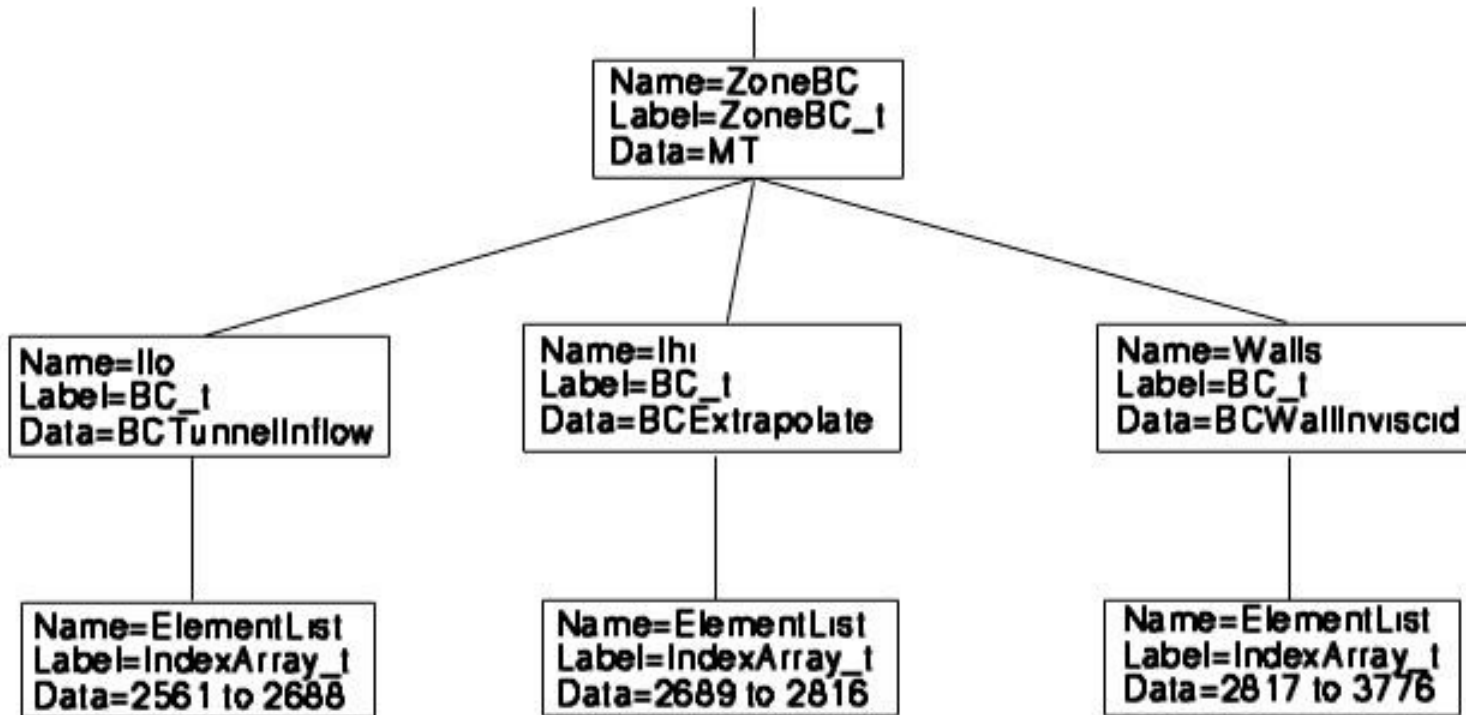
(under Zone_t)



Example Unstructured Tree

(cont'd)

(under Zone_t)



Flow Solution, MLL

- Flow Solution implementation essentially identical for Structured and Unstructured grids

```
/* define flow solution node name (user can give any name) */
strcpy(solname,"FlowSolution");
/* create flow solution node */
cg_sol_write(index_file,index_base,index_zone,solname,Vertex,&indexf);
/* write flow solution (user must use SIDS-standard names here) */
cg_field_write(index_file,index_base,index_zone,index_flow,
    RealDouble,"Density",r,&index_field);
```

Flow Solution (cont'd)

```
FlowSolution_t< int IndexDimension, int VertexSize[IndexDimension],
               int CellSize[IndexDimension] > :=
{
  List( Descriptor_t Descriptor1 ... DescriptorN ) ;           (o)
  GridLocation_t GridLocation ;                               (o/d)
  Rind_t<IndexDimension> Rind ;                               (o/d)
  List( DataArray_t<DataType, IndexDimension, DataSize[]>
        DataArray1 ... DataArrayN ) ;                       (o)
  DataClass_t DataClass ;                                    (o)
  DimensionalUnits_t DimensionalUnits ;                     (o)
  List( UserDefinedData_t UserDefinedData1 ... UserDefinedDataN ) ; (o)
};
```

Summary

- Unstructured grids written/read using same MLL calls as those for structured grids, except:
 - only one index dimension (instead of three for 3D structured)
 - element connectivity written using `cg_section_write`
 - arbitrary element types can be specified
 - 2D boundary elements (for 3D geometry) are included as separate element entities