REGIONS Proposal (version IX May 2008)

As proposed by CGNS Steering Subcommittee:

Chris Rumsey, Marc Poinot, Bob Bush, Mark Fisher, Steven Allmaras

Contact point:
Chris Rumsey, NASA Langley Research Center, Hampton, VA 23681
c.l.rumsey@nasa.gov

The need to be able to handle "regions" has been identified for some time. By "regions", we mean the ability to give flowfield or other information over a subset of the entire zone in a CGNS file. This subset may be over a portion of a boundary, or it may be over a portion of the volume field.

This version is a modification to an earlier amended proposal. The current proposal covers both structured and unstructured grids.

In addition to some relatively minor changes mostly involving naming conventions, the latest modification adds more detailed description, more examples, adds GridConnectivityRegionName, and simplifies `IndexRange_t` and `IndexArray_t` to be either PointList or PointRange. When used, PointList/Range are taken in combination with GridLocation to determine how and where the region is delineated. They can allow for edge-based specification among others.

The proposal is the following:

Zone Subregions Structure Definition: `ZoneSubRegion_t`

Under `Zone_t`, allow any number of the optional node:

```
ZoneSubRegion_t< int IndexDimension, int CellDimension > :=
  {
  List( Descriptor_t Descriptor1 ... DescriptorN ) ;                     (o)

  int RegionCellDimension ;                                              (o/d)

  GridLocation_t GridLocation ;                                          (o/d)

  IndexRange_t<IndexDimension> PointRange ;                              (r:o:o:o)
  IndexArray_t<IndexDimension, ListLength, int> PointList ;              (o:r:o:o)
  Descriptor_t BCRegionName  ;                                           (o:o:r:o)
  Descriptor_t GridConnectivityRegionName  ;                             (o:o:o:r)

  Rind_t<IndexDimension> Rind;                                           (o/d)

  List( DataArray_t<DataType, 1, ListLength[]> DataArray1 ... DataArrayN ) ;     (o)

  FamilyName_t FamilyName ;                                              (o)
  DataClass_t DataClass ;                                                    (o)

  DimensionalUnits_t DimensionalUnits ;                                      (o)

  List( UserDefinedData_t UserDefinedData1 ... UserDefinedDataN ) ;          (o)
  } ;
```

Notes

1. Default names for the `Descriptor_t`, `DataArray_t`, and `UserDefinedData_t` lists are as shown; users may choose other legitimate names. Legitimate names must be unique within a given instance of `ZoneSubRegion_t` and shall not include the names `RegionCellDimension`, `Rind`, `PointRange`, `PointList`, `BCRegionName`, `GridConnectivityRegionName`, `FamilyName`, `DataClass` or `DimensionalUnits`.
2. `RegionCellDimension` must be equal to or less than the `CellDimension` for the zone. If absent, then its default value is `CellDimension`.
3. `GridLocation` has a default value of `Vertex` if absent. Permissible values of `GridLocation` are determined by `RegionCellDimension` (see below). All data within a given instance of `ZoneSubRegion_t` must reside at the same grid location.
4. The extent of the region and distribution of its data is specified by one of `PointRange`, `PointList`, `BCRegionName`, or `GridConnectivityRegionName`. One and only one of these must be specified.

The extent of the subregion and the distribution of data within that subregion is determined by `RegionCellDimension`, `GridLocation`, and one of `PointRange/List`, `BCRegionName`, or `GridConnectivityRegionName`. For a 3–D subregion (`RegionCellDimension` = 3), data can be located at vertices, edges, face centers or cell centers. For a 2–D subregion (`RegionCellDimension` = 2), data can be located at vertices, edges or cell centers (i.e. area elements). It is anticipated that one of the widest uses for `ZoneSubRegion_t` will be to store specific boundary-only information. For example, in a 3–D simulation, one may wish to store additional data on surfaces. In this case, the `RegionCellDimension` would be set to 2.

`PointRange/List` refer to vertices, edges, faces or cell centers, depending on the values of `RegionCellDimension` and `GridLocation`. Note that it is the dimensionality of the subregion (i.e. `RegionCellDimension`), rather than the dimensionality of the zone in which it is embedded (i.e. `CellDimension` of the zone), that determines the types of elements permissible in `PointRange/List`; the following table shows these relations.

`RegionCellDimension`	`GridLocation`			
	`Vertex`	`EdgeCenter`	`*FaceCenter`	`CellCenter`
1	vertices	–	–	cells (line elements)
2	vertices	edges	–	cells (area elements)
3	vertices	edges	faces	cells (volume elements)

In the table, `*FaceCenter` stands for the possible types: `IFaceCenter`, `JFaceCenter`, `KFaceCenter`, or `FaceCenter`.

For both structured and unstructured grids, GridLocation = Vertex means that PointRange/List refers to vertex indices. For structured grids, edges, faces and cell centers are indexed using the minimum of the connecting vertex indices, as described in the section Structured Grid Notation and Indexing Conventions. For unstructured grids, edges, faces and cell centers are indexed using their element numbering, as defined in the Elements_t data structures.

If the vertices or elements of the subregion are continuously numbered, then PointRange may be used. Otherwise, PointList should be used to list the vertices/elements. Alternatively, if the data locations and range of the subregion coincide with an existing BC region or zone-to-zone GridConnectivity region, then BCRegionName or GridConnectivityRegionName may be used. BCRegionName is a string giving the name of an existing BC_t node of the current zone. GridConnectivityRegionName is a string giving the name of an existing GridConnectivity1to1_t or GridConnectivity_t node of the current zone. The name referred to should be unambiguous.

Consistent with FlowSolution_t, the subregion's solution data is stored in the list of DataArray_t entities; each DataArray_t structure entity contains a single quantity. Standardized data-name identifiers for solution quantities are described in Appendix A. As noted above, all solution data within a given subregion must reside at the same grid location.

DataClass defines the default class for data contained in the DataArray_t entities. For dimensional flow solution data, DimensionalUnits may be used to describe the system of units employed. If present, these two entities take precedence over the corresponding entities at higher levels of the CGNS hierarchy, following the standard precedence rules.

ZoneSubRegion_t requires the structure function ListLength[], which is used to specify the number of data points (e.g. vertices, cell centers, face centers, edge centers) corresponding to the given PointRange/List. If PointRange is specified, then ListLength is obtained from the number of points (inclusive) between the beginning and ending indices of PointRange. If PointList is specified, then ListLength is the number of indices in the list of points. In this situation, ListLength becomes a user input along with the indices of the list PointList. By "user" we mean the application code that is generating the CGNS database. The function ListLength is described in the SIDS section on Boundary Condition Structure Definition: BC_t.

Rind is an optional field that indicates the number of rind planes (for structured grids) or rind points (for unstructured grids). If Rind is absent, then the DataArray_t structure entities contain only core data of length ListLength, as defined for this region. If Rind is present, it will provide information on the number of rind elements, in addition to the ListLength, that are contained in the DataArray_t structures. The bottom line is that Rind simply adds a specified number to ListLength, as used by the DataArray_t structures.

There may be multiple instances of ZoneSubRegion_t in a given zone. These may simply be multiple regions defined for a single solution, or they may be associated with different times / different solutions in a time-dependent simulation (in which case ZoneIterativeData should be used to associate them).

Example 0-A: Volume Subregion for a Structured Grid

For this example, it is assumed that a 1-zone 3–D structured grid exists of size $(197\times97\times33)$. Inside of this zone, the user wishes to output a special subset region of interior data (say, temperature and kinematic viscosity) at the specific cell-center locations i = 121-149, j = 17-45, k = 21-23.

Even though this same data may possibly exist under FlowSolution_t (which holds the flowfield data for the entire zone), this particular location may represent a special region of interest where the user wants to focus attention or output different types of flowfield variables or user-defined data. Note that for structured grids, the location list always references grid nodes; in this case with GridLocation = Cellcenter the cell centers are indexed by the minimum i, j, and k indices of the connecting vertices.

Under Zone_t:

```
ZoneSubRegion_t<3,3>  Region1 =
  {{
  GridLocation_t  GridLocation = CellCenter ;
  int RegionCellDimension = 3;
  IndexRange_t<3>  PointRange =
    {{
    int[3] Begin = [121,17,21];
    int[3] End = [149,45,21];
    }};

  !  ListLength = (149-121+1)*(45-17+1)*(23-21+1) = 29*29*3 = 2523
  DataArray_t<real,1,2523>  Temperature =
    {{
    Data(real,1,2523) = temperature at the cell centers specified
    }} ;
  DataArray_t<real,1,2523>  ViscosityKinematic =
    {{
    Data(real,1,2523) = kinematic viscosity at the cell centers specified
    }} ;
  }} ;     ! end Region1
```

Example 0-B: Volume Subregion for an Unstructured Grid

This example is like the previous one, except it is for an unstructured zone. Inside of this zone, the user wishes to output a special subset region of data (say, temperature and kinematic viscosity) at a specific list of 2523 element cell-center locations, located somewhere within the (larger) field of elements. Recall that when GridLocation is anything other than Vertex in conjunction with unstructured grids, then the location list represents element numbers and not grid node numbers.

Under Zone_t:

```
ZoneSubRegion_t<1,3>  Region1 =
  {{
  GridLocation_t  GridLocation = CellCenter ;
  int RegionCellDimension = 3;
  IndexArray_t<1,2523,int>  PointList =
    {{
    int[1] ElementList = list of 3-D element numbers where region data given
    }} ;

  !  ListLength = length of the element list = 2523
  DataArray_t<real,1,2523>  Temperature =
    {{
    Data(real,1,2523) = temperature at the element cell centers specified
    }} ;
  DataArray_t<real,1,2523>  ViscosityKinematic =
    {{
    Data(real,1,2523) = kinematic viscosity at the element cell centers specified
    }} ;
  }} ;     ! end Region1
```

Example 0-C: Surface Subregion for an Unstructured Grid

In this example, boundary data is output on a 2–D surface subregion of a 3–D problem. Because this is data on a topologically 2–D boundary (in a 3–D simulation), `RegionCellDimension` is set to 2. `GridLocation` is specified as `FaceCenter`. Recall that when `GridLocation` is anything other than `Vertex` in conjunction with unstructured grids, then the location list represents element numbers and not grid node numbers. Thus, the `PointList/Range` indicates particular *surface elements* (or boundary elements) that need to have been defined in the file under their own `Elements_t` node(s), separate from the 3–D volume elements that make up the grid. In this case, we assume that the surface element numbers at which we are outputting data are 5568 through 5592 inclusive. Because the numbers occur in sequential order, we can make use of `PointRange`.

Under `Zone_t`:

```
  ZoneSubRegion_t<1,3>  Region1 =
    {{
    GridLocation_t  GridLocation = FaceCenter ;
    int RegionCellDimension = 2;
    IndexArray_t<1,25,int>  PointRange =
      {{
      int[1] Begin = [5568];
      int[1] End = [5592];
      }} ;

    !  ListLength = length of the element list = 25
    DataArray_t<real,1,25>  Temperature =
      {{
      Data(real,1,25) = temperature at the specific face element locations
                        specified
      }} ;
    DataArray_t<real,1,25>  ViscosityKinematic =
      {{
```

```
    Data(real,1,25) = kinematic viscosity at the specific face element
                      locations specified
    }} ;
  }} ;     ! end Region1
```

Example 0-D: Surface Subregion Utilizing BC Information

In this example, boundary data is output at the same locations where the BCs are specified in a particular `BC_t` node (in this case the `ListLength` is 25). Note that because this is data on a topologically 2–D boundary (in a 3–D simulation), `RegionCellDimension` is set to 2. `GridLocation` is not specified, because it is inherited from the `BC_t` node along with the `ListLength`.

Under `Zone_t`:

```
ZoneSubRegion_t<1,3>  Region1 =
  {{
  int RegionCellDimension = 2;
  Descriptor_t  BCRegionName = "name of a ZoneBC/BC_t node" ;

  !  ListLength = length of the point/element list from BC_t = 25
  DataArray_t<real,1,25>  Temperature =
    {{
    Data(real,1,25) = temperature at the specific BC locations specified
    }} ;
  DataArray_t<real,1,25>  ViscosityKinematic =
    {{
    Data(real,1,25) = kinematic viscosity at the specific BC locations specified
    }} ;
  }} ;     ! end Region1
```

Modification to Zone Iterative Data Structure Definition: `ZoneIterativeData_t`

The following optional `DataArray_t` node is added to `ZoneIterativeData_t`. Note, `ZoneIterativeData_t` is used to associate multiple `FlowSolution_t` and/or multiple `ZoneSubRegion_t` nodes for time-dependent flow.

```
DataArray_t<char, 2, [32, NumberOfSteps]> ZoneSubRegionPointers ;     (o)
```