REGIONS Proposal

submitted by:

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 proposal is a follow-on to an earlier one submitted by Alan Sayre (Element Regions), which was for unstructured grids only, and was not implemented. The current proposal attempts to cover both structured and unstructured grids.

Since people already are able to use (outside of its true intent) the BC node structure to store some of this type of data, the idea here is to create a new type of node patterned more or less in a similar way. (I.e., Under Zone_t, allow subset nodes patterned after the ZoneBC_t node and its children, but perhaps skipping the intermediate BCDataSet_t).

Thus, the nodes:

1. ZoneRegion_t would hold all the regions for a zone.
2. Region_t would essentially define the point range/list or element range/list of the region along with the grid location for the data storage.
3. RegionData_t would store the actual data at those locations.

The proposal is the following:

Under Zone_t, add the single (1) optional node:

ZoneRegion_t< int IndexDimension, int PhysicalDimension > :=
{
List(Descriptor_t Descriptor1 ... DescriptorN) ; (o)

List(Region_t<IndexDimension, int PhysicalDimension> Region1 ... RegionN) ; (o)

ReferenceState_t ReferenceState ; (o)

DataClass_t DataClass ; (o)

DimensionalUnits_t DimensionalUnits ; (o)

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

Notes

1. Default names for the Descriptor_t, ZoneRegion_t, and UserDefinedData_t lists are as shown; users may choose other legitimate names. Legitimate names must be unique within a given instance of ZoneRegion_t and shall not include the names DataClass, DimensionalUnits, or ReferenceState.
2. All lists within a ZoneRegion_t structure entity may be empty.

ZoneRegion_t requires two structure parameters, IndexDimension and PhysicalDimension, which are passed onto all Region_t substructures.

Information for a single region is contained in the Region_t structure.

Reference data applicable to all regions of a zone can be contained in the ReferenceState structure. DataClass defines the zonal default for the class of data contained in the regions of a zone. If the regions contain dimensional data, DimensionalUnits may be used to describe the system of dimensional units employed. If present, these three entities take precedence of all corresponding entities at higher levels of the hierarchy, following the standard precedence rules.

The UserDefinedData_t data structure allows arbitrary user-defined data to be stored in Descriptor_t and DataArray_t children without the restrictions or implicit meanings imposed on these node types at other node locations.

Region_t< int IndexDimension, int PhysicalDimension > :=
{
List(Descriptor_t Descriptor1 ... DescriptorN) ; (o)

GridLocation_t GridLocation ; (o/d)

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

List(RegionData_t<ListLength> RegionData1 ... RegionDataN) ; (o)

FamilyName_t FamilyName ; (o)

ReferenceState_t ReferenceState ; (o)

DataClass_t DataClass ; (o)

DimensionalUnits_t DimensionalUnits ; (o)

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

Notes

1. Default names for the Descriptor_t, RegionData_t, and UserDefinedData_t lists are as shown; users may choose other legitimate names. Legitimate names must be unique within a given instance of Region_t and shall not include the names (... need to define).
2. The area over which the region is defined is specified by one of PointRange or PointList. Only one of these may be specified.
3. PointRange and PointList refer to vertices, cell centers, or cell faces, depending on the value of GridLocation. GridLocation may be set to Vertex, CellCenter, IFaceCenter, JFaceCenter, KFaceCenter, or FaceCenter. If GridLocation is absent, then its default value is Vertex.

When GridLocation is set to Vertex, then PointList or PointRange refer to node indices, for both structured and unstructured grids.

When GridLocation is set to CellCenter, then PointList or PointRange refer to cell centers. Cell centers are indexed using different methods depending if the zone is structured or unstructured. For a structured zone, they are indexed using the minimum of the connecting vertex indices, as described in the section Structured Grid Notation and Indexing Conventions. For an unstructured zone, they are indexed using their element numbering, as defined in the Elements_t data structures.

When GridLocation is set to FaceCenter, then PointList or PointRange refer to face elements. Face elements are indexed using different methods depending if the zone is structured or unstructured. For a structured zone, they are indexed using the minimum of the connecting vertex indices, as described in the section Structured Grid Notation and Indexing Conventions. For an unstructured zone, they are indexed using their element numbering, as defined in the Elements_t data structures.

The user should *not* allow mixing of FaceCenter and CellCenter elements within the same Region_t structure.

The region may be specified by PointRange if it constitutes a logically rectangular region. In all other cases, PointList should be used to list the vertices, cell centers, or cell faces making up the region.

FamilyName can identify the family to which the region belongs. Family names can link the region to the CAD surfaces. (See the section on Family Data Structure Definition for more details.)

Reference data applicable to the particular conditions of a region can be contained in the ReferenceState structure. DataClass defines the default for the class of data contained. If the regions contain dimensional data, DimensionalUnits may be used to describe the system of dimensional units employed. If present, these three entities take precedence of all corresponding entities at higher levels of the hierarchy, following the standard precedence rules.

The UserDefinedData_t data structure allows arbitrary user-defined data to be stored in Descriptor_t and DataArray_t children without the restrictions or implicit meanings imposed on these node types at other node locations.

Rind elements are not allowed under Region_t.

```
RegionData_t< int ListLength > :=
   {
   List( Descriptor_t  Descriptor1 ... DescriptorN ) ;              (o)

   List( DataArray_t<DataType, 1, ListLength>
        DataLocal1 ... DataLocalN ) ;                          (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 RegionData_t and shall not include the names DataClass or DimensionalUnits.
2. There are no required elements; all lists may be empty.

The local data variables have size determined by the structure parameter ListLength. For DataArray_t entities with standardized data-name identifiers, DataType is determined by convention. For user-defined variables, DataType is a user input.

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

The UserDefinedData_t data structure allows arbitrary user-defined data to be stored in Descriptor_t and DataArray_t children without the restrictions or implicit meanings imposed on these node types at other node locations.

Example X.1:
For this example, it is assumed that a 1-zone 3-D structured grid exists of size (197x97x33). Inside of this zone, the user wishes to output a special subset region of 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 his attention, or perhaps where he wants to output different types of flowfield variables or UserDefined data.

Under Zone_t:

```
ZoneRegion_t<3,3>  ZoneRegion1 =
 {{
   Region_t<3,3>  Region1 =
    {{
       GridLocation_t  GridLocation = CellCenter ;
       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
       RegionData_t<2523>  RegionData1 =
        {{
          DataArray_t<real,1,2523>  Temperature =
           {{
             Data(real,1,2523) = temperature at the specific cell centers specified
           }} ;
          DataArray_t<real,1,2523>  ViscosityKinematic =
           {{
             Data(real,1,2523) = kinematic viscosity at the specific cell centers specified
           }} ;
        }} ;
    }} ;    ! end Region1
 }} ;    ! end ZoneRegion1
```

Example X.2:
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 cell-center locations, located somewhere within the (larger) field of elements.

Under Zone_t:

```
ZoneRegion_t<1,3>  ZoneRegion1 =
 {{
   Region_t<1,3>  Region1 =
    {{
       GridLocation_t  GridLocation = CellCenter ;
       IndexArray_t<1,2523,int>  PointList =
        {{
           int[1] ElementList = list of element numbers where region data given
        }} ;
       !  ListLength = length of the element list = 2523
       RegionData_t<2523>  RegionData1 =
        {{
          DataArray_t<real,1,2523>  Temperature =
           {{
             Data(real,1,2523) = temperature at the specific cell centers specified
           }} ;
          DataArray_t<real,1,2523>  ViscosityKinematic =
           {{
             Data(real,1,2523) = kinematic viscosity at the specific cell centers specified
           }} ;
        }} ;
    }} ;    ! end Region1
 }} ;    ! end ZoneRegion1
```