```
    TimeAccurate,
    NonTimeAccurate ) ;
```

DataClass describes the global default for the class of data contained in the CGNS database. If the CGNS database contains dimensional data (e.g., velocity with units of m/s), DimensionalUnits may be used to describe the system of units employed.

FlowEquationSet contains a description of the governing flow equations associated with the entire CGNS database. This structure contains information on the general class of governing equations (e.g., Euler or Navier-Stokes), equation sets required for closure, including turbulence modeling and equations of state, and constants associated with the equations.

DataClass, DimensionalUnits, ReferenceState and FlowEquationSet have special function in the CGNS hierarchy. They are globally applicable throughout the database, but their precedence may be superseded by local entities (e.g., within a given zone). The scope of these entities and the rules for determining precedence are treated in Section 6.4.

Globally relevant convergence history information is contained in GlobalConvergenceHistory. This convergence information includes total configuration forces, moments, and global residual and solution-change norms taken over all the zones.

Miscellaneous global data may be contained in the IntegralData_t list. Candidates for inclusion here are global forces and moments.

The Family_t data structure, defined in Section 12.6, is used to record geometry reference data. It may also include boundary conditions linked to geometry patches. For the purpose of defining material properties, families may also be defined for groups of elements. The family-mesh association is defined under the Zone_t and BC_t data structures by specifying the family name corresponding to a zone or a boundary patch.

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.

6.3 Zone Structure Definition: Zone_t

The Zone_t structure contains all information pertinent to an individual zone. This information includes the zone type, the number of cells and vertices making up the grid in that zone, the physical coordinates of the grid vertices, grid motion information, the family, the flow solution, zone interface connectivity, boundary conditions, and zonal convergence history data. Zonal data may be recorded at multiple time steps or iterations. In addition, this structure contains a reference state, a set of flow equations and dimensional units that are all unique to the zone. For unstructured zones, the element connectivity may also be recorded.

```
  ZoneType_t := Enumeration(
    Null,
    Structured,
    Unstructured,
```

```
  UserDefined ) ;


ZoneElementsType_t := Enumeration(
  CellBased,
  FaceBased ) ;


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

  ZoneType_t ZoneType ;                                               (r)

  int[IndexDimension] VertexSize ;                                    (r)
  int[IndexDimension] CellSize ;                                      (r)
  int[IndexDimension] VertexSizeBoundary ;                            (o/d)

  List( GridCoordinates_t<IndexDimension, VertexSize>
        GridCoordinates MovedGrid1 ... MovedGridN ) ;                 (o)


  ZoneElementsType_t ZoneElementsType ;                               (o/d)


  List( Elements_t Elements1 ... ElementsN ) ;                        (o)

  List( RigidGridMotion_t RigidGridMotion1 ... RigidGridMotionN ) ;   (o)

  List( ArbitraryGridMotion_t
        ArbitraryGridMotion1 ... ArbitraryGridMotionN ) ;             (o)

  FamilyName_t FamilyName ;                                           (o)

  List( FlowSolution_t<IndexDimension, VertexSize, CellSize>
        FlowSolution1 ... FlowSolutionN ) ;                           (o)

  List( DiscreteData_t<IndexDimension, VertexSize, CellSize>
        DiscreteData1 ... DiscreteDataN ) ;                           (o)

  List( IntegralData_t IntegralData1 ... IntegralDataN ) ;            (o)

  ZoneGridConnectivity_t<IndexDimension, CellDimension>
     ZoneGridConnectivity ;                                           (o)
```

```
  ZoneBC_t<IndexDimension, PhysicalDimension> ZoneBC ;                 (o)

  ZoneIterativeData_t<NumberOfSteps> ZoneIterativeData ;               (o)

  ReferenceState_t ReferenceState ;                                    (o)

  RotatingCoordinates_t RotatingCoordinates ;                          (o)

  DataClass_t DataClass ;                                              (o)

  DimensionalUnits_t DimensionalUnits ;                                (o)

  FlowEquationSet_t<CellDimension> FlowEquationSet ;                   (o)

  ConvergenceHistory_t ZoneConvergenceHistory ;                        (o)

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

  int Ordinal ;                                                        (o)
  } ;
```

Notes

1. Default names for the Descriptor_t, Elements_t, RigidGridMotion_t, ArbitraryGridMotion_t, FlowSolution_t, DiscreteData_t, IntegralData_t, and UserDefinedData_t lists are as shown; users may choose other legitimate names. Legitimate names must be unique within a given instance of Zone_t and shall not include the names DataClass, DimensionalUnits, FamilyName, FlowEquationSet, GridCoordinates, Ordinal, ReferenceState, RotatingCoordinates, ZoneBC, ZoneConvergenceHistory, ZoneElementsType, ZoneGridConnectivity,ZoneIterativeData, or ZoneType.
2. The original grid coordinates should have the name GridCoordinates. Default names for the remaining entities in the GridCoordinates_t list are as shown; users may choose other legitimate names, subject to the restrictions listed in the previous note.
3. ZoneType, VertexSize, and CellSize are the only required fields within the Zone_t structure.

Zone_t requires the parameters CellDimension and PhysicalDimension. CellDimension, along with the type of zone, determines IndexDimension; if the zone type is Unstructured, IndexDimension = 1, and if the zone type is Structured, IndexDimension = CellDimension. These three structure parameters identify the dimensionality of the grid-size arrays. One or more of them are passed on to the grid coordinates, flow solution, interface connectivity, boundary condition and flow-equation description structures.

VertexSize is the number of vertices in each index direction, and CellSize is the number of cells in each direction. For example, for structured grids in 3-D, CellSize = VertexSize - [1,1,1], and for unstructured grids in 3-D, CellSize is simply the total number of 3-D cells. VertexSize is the number of vertices defining "the grid" or the domain (i.e., without rind points); CellSize is the number of cells on the interior of the domain. These two grid-size arrays are passed onto the grid-coordinate, flow-solution and discrete-data substructures.

If the nodes are sorted between internal nodes and boundary nodes, then the optional parameter VertexSizeBoundary must be set equal to the number of boundary nodes. If the nodes are sorted, the grid coordinate vector must first include the boundary nodes, followed by the internal nodes. By default, VertexSizeBoundary equals zero, meaning that the nodes are unsorted. This option is only useful for unstructured zones. For structured zones, VertexSizeBoundary always equals 0 in all index directions.

The GridCoordinates_t structure defines "the grid"; it contains the physical coordinates of the grid vertices, and may optionally contain physical coordinates of rind or ghost points. The original grid is contained in GridCoordinates. Additional GridCoordinates_t data structures are allowed, to store the grid at multiple time steps or iterations.

When the grid nodes are sorted, the DataArray_t in GridCoordinates_t lists first the data for the boundary nodes, then the data for the internal nodes.

The Elements_t data structure contains unstructured elements data such as connectivity, element type, parent elements, etc. Unstructured grid connectivity may be described in either one of two ways: cell- or element-based (ZoneElementsType = CellBased), where nodes of each cell are provided; or face-based (ZoneElementsType = FaceBased), where the nodes of each face in 3–D or edge in 2–D are provided along with the two adjacent cells. If ZoneElementsType is absent, then grid connectivity is cell based. See Section 7.3 for further details.

The RigidGridMotion_t and ArbitraryGridMotion_t data structures contain information defining rigid and arbitrary (i.e., deforming) grid motion.

FamilyName identifies to which family a zone belongs. Families may be used to define material properties.

Flow-solution quantities are contained in the list of FlowSolution_t structures. Each instance of the FlowSolution_t structure is only allowed to contain data at a single grid location (vertices, cell-centers, etc.); multiple FlowSolution_t structures are provided to store flow-solution data at different grid locations, to record different solutions at the same grid location, or to store solutions at multiple time steps or iterations. These structures may optionally contain solution data defined at rind points.

Miscellaneous discrete field data is contained in the list of DiscreteData_t structures. Candidate information includes residuals, fluxes and other related discrete data that is considered auxiliary to the flow solution. Likewise, miscellaneous zone-specific global data, other than reference-state data and convergence history information, is contained in the list of IntegralData_t structures. It is envisioned that these structures will be seldom used in practice but are provided nonetheless.

For unstructured zones only, the node-based DataArray_t vectors (GridLocation = Vertex) in FlowSolution_t or DiscreteData_t must follow exactly the same ordering as the GridCoordinates vector. If the nodes are sorted (VertexSizeBoundary $\neq 0$), the data on the boundary nodes must be listed first, followed by the data on the internal nodes. Note that the order in which the node-based data are recorded must follow exactly the node ordering in GridCoordinates_t, to be able to associate the data to the correct nodes. For element-based data (GridLocation =

CellCenter), the FlowSolution_t or DiscreteData_t data arrays must list the data values for each element, in the same order as the elements are listed in ElementConnectivity.

All interface connectivity information, including identification of overset-grid holes, for a given zone is contained in ZoneGridConnectivity.

All boundary condition information pertaining to a zone is contained in ZoneBC_t.

The ZoneIterativeData_t data structure may be used to record pointers to zonal data at multiple time steps or iterations.

Reference-state data specific to an individual zone is contained in the ReferenceState structure.

RotatingCoordinates may be used to specify the rotation center and rotation rate vector of a rotating coordinate system.

DataClass defines the zonal default for the class of data contained in the zone and its substructures. If a zone contains dimensional data, DimensionalUnits may be used to describe the system of dimensional units employed.

If a set of flow equations are specific to a given zone, these may be described in FlowEquationSet. For example, if a single zone within the domain is inviscid, whereas all other are turbulent, then this zone-specific equation set could be used to describe the special zone.

DataClass, DimensionalUnits, ReferenceState and FlowEquationSet have special function in the hierarchy. They are applicable throughout a given zone, but their precedence may be superseded by local entities contained in the zone's substructures. If any of these entities are present within a given instance of Zone_t, they take precedence over the corresponding global entities contained in database's CGNSBase_t entity. These precedence rules are further discussed in Section 6.4.

Convergence history information applicable to the zone is contained in ZoneConvergenceHistory; this includes residual and solution-change norms.

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.

Ordinal is user-defined and has no restrictions on the values that it can contain. It is included for backward compatibility to assist implementation of the CGNS system into applications whose I/O depends heavily on the numbering of zones. Since there are no restrictions on the values contained in Ordinal (or that Ordinal is even provided), there is no guarantee that the zones in an existing CGNS database will have sequential values from 1 to N without holes or repetitions. Use of Ordinal is discouraged and is on a user-beware basis.

6.4 Precedence Rules and Scope Within the Hierarchy

The dependence of a structure entity's information on data contained at higher levels of the hierarchy is typically explicitly expressed through structure parameters. For example, all arrays within Zone_t depend on the dimensionality of the computational grid. This dimensionality is passed down to a Zone_t entity through a structure parameter in the definition of Zone_t.

We have established an alternate dependency for a limited number of entities that is not explicitly stated in the structure type definitions. These special situations include entities for describing data

```
    DataArray_t<real, 1, 15> CoordinateX =
      {{
      Data(real, 1, 15) = (x(i), i=1,15) ;
      }} ;

    DataArray_t<real, 1, 15> CoordinateY =
      {{
      Data(real, 1, 15) = (y(i), i=1,15) ;
      }} ;

    DataArray_t<real, 1, 15> CoordinateZ =
      {{
      Data(real, 1, 15) = (z(i), i=1,15) ;
      }} ;
    }} ;
```

7.3 Elements Structure Definition: Elements_t

The Elements_t data structure is required for unstructured zones, and contains the element connectivity data, the element type, the element range, the parent elements data, and the number of boundary elements. The definition of Elements_t supports two different methods of describing the connectivity of a given unstructured grid. The first method is cell- or element-based, where in 3–D the nodes of each volume element are given; in 2–D the nodes of each area element are given. The second supported method is face-based connectivity in 3–D and edge-based in 2–D. In this method, the nodes of each face element are given; in addition, the two volume elements adjacent to the face are given. Any application should be aware of these two different methods for describing grid connectivity.

```
  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)
  } ;
```

Notes

1. Default names for the Descriptor_t and UserDefinedData_t lists are as shown; users may choose other legitimate names. Legitimate names must be unique within a given instance of Elements_t and shall not include the names ElementConnectivity, ElementRange, ParentData, or Rind.
2. IndexRange_t, ElementType_t, and ElementConnectivity_t are the required fields within the Elements_t structure. Rind has a default if absent; the default is equivalent to having a Rind structure whose RindPlanes array contains all zeros (see Section 4.8).

Rind is an optional field that indicates the number of rind elements included in the elements data. If Rind is absent, then the DataArray_t structure entities contain only core elements of a zone. If Rind is present, it will provide information on the number of rind elements, in addition to the core elements, that are contained in the DataArray_t structures.

Note that the usage of rind data with respect to the size of the DataArray_t structures is different under Elements_t than elsewhere. For example, when rind coordinate data is stored under GridCoordinates_t, the parameter VertexSize accounts for the core data only. The size of the DataArray_t structures containing the grid coordinates is determined by the DataSize function, which adds the number of rind planes or points to VertexSize. But for the element connectivity, the size of the DataArray_t structures containing the connectivity data is just ElementDataSize, which depends on ElementSize, and includes both the core and rind elements.

ElementRange contains the index of the first and last elements defined in ElementConnectivity. The elements are indexed with a global numbering system, starting at 1, for all element sections under a given Zone_t data structure. The global numbering insures that each element, whether it's a cell, a face, or an edge, is uniquely identified by its number. They are also listed as a continuous list of element numbers within any single element section. Therefore the number of elements in a section is:

```
ElementSize = ElementRange.end - ElementRange.start + 1
```

The element indices are used for the boundary condition and zone connectivity definition.

ElementSizeBoundary indicates if the elements are sorted, and how many boundary elements are recorded. By default, ElementSizeBoundary is set to zero, indicating that the elements are not sorted. If the elements are sorted, ElementSizeBoundary is set to the number of elements at the boundary. Consequently:

```
ElementSizeInterior = ElementSize - ElementSizeBoundary
```

ElementType_t is an enumeration of the supported element types:

```
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 );
```

Section 3.3 illustrates the convention for element numbering.

For all element types except type `MIXED`, `ElementConnectivity` contains the list of nodes for each element. If the elements are sorted, then it must first list the connectivity of the boundary elements, then that of the interior elements.

$$\texttt{ElementConnectivity = } \begin{array}{l} \texttt{Node1}_1\texttt{, Node2}_1\texttt{, ... NodeN}_1\texttt{,} \\ \texttt{Node1}_2\texttt{, Node2}_2\texttt{, ... NodeN}_2\texttt{,} \\ \texttt{...} \\ \texttt{Node1}_M\texttt{, Node2}_M\texttt{, ... NodeN}_M \end{array}$$

When the section `ElementType` is `MIXED`, the data array `ElementConnectivity` contains one extra integer per element, to hold each individual element type:

$$\texttt{ElementConnectivity = } \begin{array}{l} \texttt{Etype}_1\texttt{, Node1}_1\texttt{, Node2}_1\texttt{, ... NodeN}_1\texttt{,} \\ \texttt{Etype}_2\texttt{, Node1}_2\texttt{, Node2}_2\texttt{, ... NodeN}_2\texttt{,} \\ \texttt{...} \\ \texttt{Etype}_M\texttt{, Node1}_M\texttt{, Node2}_M\texttt{, ... NodeN}_M \end{array}$$

`ElementDataSize` indicates the size (number of integers) of the array `ElementConnectivity`. For all element types except type `MIXED`, the `ElementDataSize` is given by:

```
ElementDataSize = ElementSize * NPE[ElementType]
```

In the case of `MIXED` element section, `ElementDataSize` is given by:

$$\texttt{ElementDataSize = } \sum_{n=start}^{end} (\texttt{NPE[ElementType}_n\texttt{] + 1})$$

`NPE[ElementType]` is a function returning the number of nodes for the given `ElementType`. For example, `NPE[HEXA_8]=8`.

For face elements in 3D, or bar element in 2D, four more data may be saved for each element — the corresponding parents' element number, and the face position within these parent elements. At the boundaries, the second parent is set to zero.

`NGON_n` is used to express a polygon of n nodes. In order to record the number of nodes of any ngons, the `ElementType` must be set to `NGON_n + Nnodes`. For example, for an element type `NGON_n` composed of 25 nodes, one would set the `ElementType` to `NGON_n + 25`.

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.

7.4 Elements Examples

This section contains four examples of elements definition in CGNS. The first two examples depict element-based verses face-based connectivity for the simple grid displayed in Figure 3. The second two examples depict element-based connectivity for a grid with mixed tetrahedral and hexahedral elements.

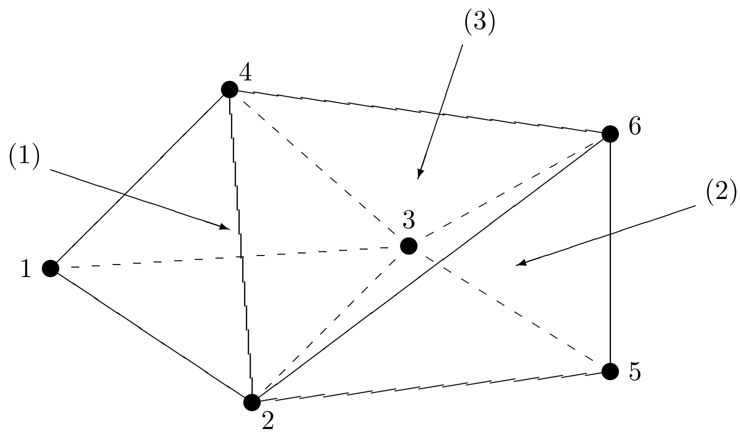

Figure 3: Example Tetrahedral Grid

Example 7-D: Tetrahedra Unstructured Elements, Element-Based Connectivity

This first example describes the element-based connectivity for the tetrahedral grid displayed in Figure 3. For each of the three tetraheral elements, the four nodes comprising each element are given. The data in `ElementConnectivity` is grouped by element; note that the parentheses are added here for presentation purposes only.

```
Zone_t UnstructuredZone =
  {{
  ZoneElementsType_t ZoneElementsType = CellBased ;

  Elements_t TetraElements =
    {{
    IndexRange_t ElementRange = [1,3] ;

    ElementType_t ElementType = TETRA_4 ;

    DataArray_t<int, 1, NPE[TETRA_4]×3> ElementConnectivity =
      {{
      Data(int, 1, NPE[TETRA_4]×3) =
        (1, 2, 3, 4), (2, 5, 3, 6), (2, 6, 3, 4) ;
      }} ;
    }} ;
  }} ;
```

Example 7-E: Tetrahedra Unstructured Elements, Face-Based Connectivity

This second example describes the alternate face-based connectivity for the tetrahedral grid displayed in Figure 3. The grid consists of three tetrahedral volume elements and ten triangle face elements. For each triangle face, the three nodes comprising that face are given. In addition, the two tetrahedral elements adjacent to the face are given in `ParentData`. Four integers are given in `ParentData` for each face; the first two are the adjacent volume elements, and the second two are the canonical face numbers in those two volume elements consistent with Section 3.3. Parentheses in `ElementConnectivity` and `ParentData` are for presentation purposes only.

```
Zone_t UnstructuredZone =
  {{
  ZoneElementsType_t ZoneElementsType = FaceBased ;

  Elements_t TriangleElements =
    {{
    IndexRange_t ElementRange = [1,10] ;

    ElementType_t ElementType = TRI_3 ;

    DataArray_t<int, 1, NPE[TRI_3]×10> ElementConnectivity =
      {{
      Data(int, 1, NPE[TRI_3]×10) =
        (1, 3, 2), (1, 2, 4), (2, 3, 4), (1, 4, 3),
        (2, 3, 5), (2, 5, 6), (3, 6, 5), (2, 6, 3),
        (2, 6, 4), (3, 4, 6) ;
      }} ;

    DataArray_t<int, 1, 4×10> ParentData =
      {{
      Data(int, 1, 4×10) =
        (1, 0, 1, 0), (1, 0, 2, 0), (1, 3, 3, 1), (1, 0, 4, 0),
        (2, 0, 1, 0), (2, 0, 2, 0), (2, 0, 3, 0), (2, 3, 4, 2)
        (3, 0, 4, 0), (3, 0, 3, 0) ;
      }} ;
    }} ;
  }} ;
```

Example 7-F: Unstructured Elements, Separate Element Types

This third example describes the element-based connectivity for a zone that contains 15 tetrahedral and 10 hexahedral elements. The elements are written in two separate sections, one for the tetrahedral elements and one for the hexahedral elements.

```
Zone_t UnstructuredZone =
  {{
```

```
    ZoneElementsType_t ZoneElementsType = CellBased ;

    Elements_t TetraElements =
      {{
      IndexRange_t ElementRange = [1,15] ;

      int ElementSizeBoundary = 10 ;

      ElementType_t ElementType = TETRA_4 ;

      DataArray_t<int, 1, NPE[TETRA_4]×15> ElementConnectivity =
        {{
        Data(int, 1, NPE[TETRA_4]×15) = (node(i,j), i=1,NPE[TETRA_4], j=1,15) ;
        }} ;
      }} ;
    Elements_t HexaElements =
      {{
      IndexRange_t ElementRange = [16,25] ;

      int ElementSizeBoundary = 0 ;

      ElementType_t ElementType = HEXA_8 ;

      DataArray_t<int, 1, NPE[HEXA_8]×10> ElementConnectivity =
        {{
        Data(int, 1, NPE[HEXA_8]×10) = (node(i,j), i=1,NPE[HEXA_8], j=1,10) ;
        }} ;
      }} ;
    }} ;
```

Example 7-G: Unstructured Elements, Element Type MIXED

In this forth example, the same unstructured zone described in Example 7-F is written in a single element section of type MIXED (i.e., an unstructured grid composed of mixed elements).

```
  Zone_t UnstructuredZone =
    {{
    ZoneElementsType_t ZoneElementsType = CellBased ;

    Elements_t MixedElementsSection =
      {{
      IndexRange_t ElementRange = [1,25] ;

      ElementType_t ElementType = MIXED ;
```

```
      DataArray_t<int, 1, ElementDataSize> ElementConnectivity =
        {{
        Data(int, 1, ElementDataSize) = (etype(j),(node(i,j),
             i=1,NPE[etype(j)]), j=1,25) ;
        }} ;
      }} ;
    }} ;
```

7.5 Axisymmetry Structure Definition: Axisymmetry_t

The Axisymmetry_t data structure allows recording the axis of rotation and the angle of rotation around this axis for a two-dimensional dataset that represents an axisymmetric database.

```
Axisymmetry_t :=
  {
  List( Descriptor_t Descriptor1 ... DescriptorN ) ;                    (o)

  DataArray_t<real,1,2> AxisymmetryReferencePoint ;                     (r)
  DataArray_t<real,1,2> AxisymmetryAxisVector ;                         (r)
  DataArray_t<real,1,1> AxisymmetryAngle ;                              (o)
  DataArray_t<char,2,[32,2]> CoordinateNames ;                          (o)

  DataClass_t DataClass ;                                               (o)

  DimensionalUnits_t DimensionalUnits ;                                 (o)

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

Notes

1. Default names for the Descriptor_t and UserDefinedData_t lists are as shown; users may choose other legitimate names. Legitimate names must be unique within a given instance of Axisymmetry_t and shall not include the names AxisymmetryAngle, AxisymmetryAxisVector, AxisymmetryReferencePoint, CoordinateNames, DataClass, or DimensionalUnits.
2. AxisymmetryReferencePoint and AxisymmetryAxisVector are the required fields within the Axisymmetry_t structure.

AxisymmetryReferencePoint specifies the origin used for defining the axis of rotation.

AxisymmetryAxisVector contains the direction cosines of the axis of rotation, through the AxisymmetryReferencePoint. For example, for a 2-D dataset defined in the (x, y) plane, if AxisymmetryReferencePoint contains $(0, 0)$ and AxisymmetryAxisVector contains $(1, 0)$, the x-axis is the axis of rotation.