CFD General Notation System
SIDS-to-ADF File Mapping Manual

Document Version 2.4.2

CGNS Version 2.4.4

Contents

List of Figures

1 Brief Description of CGNS

The CFD General Notation System (CGNS) is a set of standards, together with software implementing those standards, for the recording of data associated with Computational Fluid Dynamics (CFD). Physical data storage is accomplished via a general database manager, either ADF (Advanced Data Format) or HDF (Hierarchical Data Format). The present document describes the mapping from the Standard Interface Data Structures (SIDS) to the file structure provided by the ADF database manager. A similar document, the *SIDS-to-HDF File Mapping Manual*, describes the mapping for the HDF database manager.

2 CGNS Documentation

Documentation of CGNS is found in a number of related publications. These are maintained separately for several reasons. First, they describe logically independent aspects of the CGNS system. Second, many users will find that a subset of the documentation is sufficient for their needs. And last, some portions of the system can be viewed as independent entities useful outside the context of CGNS.

The main documents currently available are:

- CGNS Overview and Entry-Level Document
- A User's Guide to CGNS
- ADF User's Guide
- HDF Documentation
- Standard Interface Data Structures
- SIDS-to-ADF File Mapping Manual (this document)
- SIDS-to-HDF File Mapping Manual
- CGNS Mid-Level Library

2.1 Description of the Documents

In order to make the current document as self-contained as possible, basic information regarding all components of the CGNS standard has been included. However, the reader should be aware of the documents describing these other components. These should be regarded as the authoritative and complete descriptions of their respective components of CGNS. Each of these documents is described briefly in this section.

2.1.1 CGNS Overview and Entry-Level Document

This is an introductory document which provides an overall view of the purpose and components of CGNS. The Overview is intended as an entry point into CGNS. Prospective users completely unfamiliar with CGNS should consult the Overview to determine whether CGNS provides the capabilities they seek.

2.1.2 A User's Guide to CGNS

The *User's Guide to CGNS* has been written to aid users in the implementation of CGNS. It is intended as a tutorial: light in content, but heavy in examples, advice, and guidelines. The guide provides a concise overview of many of the most commonly-used features of the SIDS, and gives coding examples using the CGNS Mid-Level Library to write and read simple CGNS files.

2.1.3 ADF User's Guide

The *ADF User's Guide* describes one of the underlying database managers, ADF (Advanced Data Format), which may be used to create CGNS files. ADF is a binary format, based on a simple tree structure. In principal, nearly any kind of data could be stored in ADF format. ADF was, however, especially designed for the storage of large quantities of scientific data in a platform-independent and randomly-accessible manner. The "ADF Core" is a set of portable routines which store and retrieve

data in ADF format. These routines are written in C, but Fortran versions are also provided. The *ADF User's Guide* describes both the ADF format and the ADF Core routines in detail.

Because the CGNS File Mapping depends intimately on the format of the underlying database manager, a summary of ADF data structures is provided in Section 4.

It should be emphasized that ADF, with its Core routines, constitutes a very general stand-alone database manager which is not directly related to CFD. It can therefore be used to store any kind of data *once it has been specified where to place that data within the ADF format*. This File Mapping document describes the CGNS conventions governing that placement for CFD-specific data.

2.1.4 HDF Documentation

HDF5 (Hierarchical Data Format) is another underlying database manager that may be used to create CGNS files, and is developed and maintained by the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign. Documentation for HDF5 is available at the HDF5 Home Page, at http://hdf.ncsa.uiuc.edu/HDF5/.

Like ADF, HDF5 is a binary format, based on a tree structure, designed for the storage of large quantities of scientific data in a platform-independent and randomly-accessible manner. Also like ADF, HDF5 can be used to store any kind of data. The *SIDS-to-HDF File Mapping Manual* describes the CGNS conventions governing the placement of CFD-specific data within the HDF5 format.

2.1.5 Standard Interface Data Structures

The *Standard Interface Data Structures*, usually abbreviated as "SIDS," define the "intellectual content" of CFD data. They detail the data which must be stored to completely characterize each CFD entity. They describe, for example, what exactly is meant by a "grid" or a "boundary condition". They also establish a system of nomenclature which gives standard meaning to certain names, such as "Density" and "SubsonicInflow".

The SIDS description of the CFD data is hierarchical in nature, in that complex entities are built up out of simpler ones. In the SIDS document, this is reflected in a syntax which uses C-like structures to define the various entities. The result is a tree-like structure which maps naturally onto the ADF format. Consequently, the File Mapping described by the current document exactly parallels the SIDS. Thus in terms of basic structure, the Mapping itself summarizes the SIDS. However, there are many conventions regarding the nomenclature and meaning of data which are not summarized in the current document, and for these the SIDS is the authoritative document.

It is worth emphasizing that the SIDS may be regarded as a stand-alone definition of the data associated with CFD, and that these data could be stored in any sufficiently general format, given a mapping onto that format.

2.1.6 The SIDS-to-ADF File Mapping Manual

The *SIDS-to-ADF File Mapping Manual* specifies the exact manner in which, under CGNS conventions, CFD data structures (the SIDS) are to be stored in, i.e., mapped onto, the file structure provided by the ADF database manager. Adherence to the mapping conventions guarantees uniform meaning and location of CFD data within ADF files, and thereby allows the construction of universal software to read and write the data.

2.1.7 The SIDS-to-HDF File Mapping Manual

The *SIDS-to-HDF File Mapping Manual* is similar to the current document, and specifies the mapping from the SIDS to the file structure provided by the HDF5 database manager.

The document describes the *node level* system, whereas the Mid-Level Library can be understood as the *tree level* system. A description is given for every Mid-Level Library type, or sub-tree, in terms of sets of atomic nodes. Moreover, it specifies the exact manner in which, under CGNS conventions, CFD data structures (the SIDS) are to be mapped onto the data structures provided by the low level layer (HDF5).

Some system-specific mechanisms, such as *link* management are detailed.[1]

Adherence to the mapping conventions guarantees uniform meaning and location of CFD data within HDF5 files, and thereby allows the construction of universal software to read and write the data.

2.1.8 The CGNS Mid-Level Library

The *CGNS Mid-Level Library* document describes a set of routines which store and retrieve the CFD data objects defined in the SIDS. Their purpose is to provide CGNS compliant I/O without the need for detailed programming in the ADF or HDF Core. These "mid-level routines" are designed to be inserted directly into applications codes, such as flow solvers and grid generators.

2.2 Which Documents Do You Need?

Ideally, all users of the CGNS system will want to have all the documents available for reference. However, many will find it possible to begin to use the system effectively without reading all the documents beforehand. In fact, since the CGNS system is intended to minimize interaction with underlying data structures, some users will find they need very little knowledge of the system's internal workings. We distinguish four classes of users who may wish to consult the CGNS Documentation.

2.2.1 Prospective Users

Prospective users are presumably unfamiliar with CGNS. They will probably wish to begin with the Overview, or, if they require more detailed information, one or more of the various papers that have been written describing CGNS. Beyond that, most will find a quick read of this file mapping document (or the SIDS-to-HDF file mapping document) enlightening as to the logical form of the contents of CGNS files. Browsing the figures in Appendix A of this document, as well as the SIDS itself, will provide some feel for the scope of the system. The *User's Guide to CGNS*, and the *CGNS Mid-Level Library* document should give an indication of what might be required to implement CGNS in a given application. Prospective users should probably not concern themselves with the details of ADF or HDF.

[1] Please note that only the *final* representation of these *links* are relevant, in other words the sequence of HDF5 calls required to obtain such a representation is not important.

2.2.2 End Users

The end user is the practitioner of CFD who generates the grids, runs the flow codes and/or analyzes the results. For this user, CGNS provides a mechanism for accumulating the output of the various processes related to CFD, e.g., grid generation and flow solution, and for making this output available to subsequent processes or for archiving final results. For this user, a scan of the Overview will sufficiently explain the overall workings of the system. This includes end user responsibilities for matters not governed by CGNS, such as the maintenance of files and directories. The end user will also find useful those portions of the SIDS which deal with standard nomenclature. AIAA 98-3007 may also be useful if more details about the capabilities of CGNS are desired.

The end user is, by definition, not involved in the building of CGNS-compliant applications code.

2.2.3 Applications Code Developers

The applications code developer builds or maintains code to support the various sub-processes encountered in CFD, e.g., grid generation, flow solution, post-processing, or flow visualization. The code developer must be able to install CGNS compliant I/O. The most convenient method for doing so is to utilize the CGNS Mid-Level Library. The *User's Guide to CGNS* is the starting point for learning to use the Mid-Level Library to create and use CGNS files. The *CGNS Mid-Level Library* document itself should also be considered essential. This library of routines will perform the most common I/O operations in a CGNS-compliant manner. However, even when the CGNS Library suffices to implement all necessary I/O, an understanding of the SIDS and the file mapping (either SIDS-to-ADF or SIDS-to-HDF) will be useful. It will likely be necessary to consult the SIDS to determine the precise meaning of the nomenclature.

2.2.4 CGNS System Developers

CGNS System development can be kept somewhat compartmentalized. Developers responsible for the maintenance or building of supplements to the ADF or HDF Core, need not concern themselves with documentation other than the *ADF User Guide* or the HDF5 documentation. System developers wishing to add to the CGNS Mid-Level Library will need all the documents. Theoretical developments, such as extensions to the SIDS, may possibly be undertaken with a knowledge of the SIDS alone, but such contributions must also be added to the SIDS-to-ADF and SIDS-to-HDF file mappings before they can be implemented.

2.3 How to Use This Document

Those wishing to do more than simply browse this document will find that the detailed information begins with the Summary of ADF in Section 4, and continues with the General File Mapping Concepts in Section 5. The detailed textual node descriptions in Section 6 are more useful as reference than as sequential literature. The best overall technical view of the layout of CGNS files can be acquired by reference to the figures in Appendix A.

3 CGNS Background

The information in this section is supplied for the sake of completeness. It is identical with the information found in the Overview.

3.1 Purpose

The purpose of CGNS is to provide a standard for recording and recovering computer data associated with the numerical solution of the equations of fluid dynamics. The format implemented by this standard is (1) general, (2) portable, (3) expandable, and (4) durable.

The CGNS system consists of a collection of conventions, and software implementing those conventions, for the storage and retrieval of CFD (computational fluid dynamics) data. The system consists of two parts: (1) a standard format for recording the data, and (2) software that reads, writes and modifies data in that format. The format is a conceptual entity established by the documentation; the software is a physical product supplied to enable developers to access and produce data recorded in that format. The CGNS standard, applied through the use of the supplied software, is intended to do the following:

- facilitate the exchange of CFD data
  - between sites.
  - between applications codes.
  - across computing platforms.
- stabilize the archiving of CFD data.

3.2 Participation and Brief History

The CGNS project originated around 1994–1995 through a series of meetings between Boeing and NASA that addressed improved means for transferring NASA technology to industrial use. It was held that a principal impediment to technology transfer was the disparity in I/O formats employed by various flow codes, grid generators, and so forth. The CGNS system was conceived as a means to promote "plug-and-play" CFD.

Agreement was reached to develop CGNS at Boeing, under NASA Contract NAS1-20267, with active participation by a team of CFD researchers from

- NASA Langley Research Center
- NASA Glenn Research Center
- NASA Ames Research Center
- Boeing St-Louis (McDonnell-Douglas Corporation).
- Boeing Commercial Airplane Group Aerodynamics
- Boeing Commercial Airplane Group Propulsion
- ICEM CFD Engineering Corporation of Berkeley, California

Also participating in the discussions at various times have been researchers from

- Defense and Space Group and Environmental Systems
- Arnold Engineering Development Center, representing the NPARC Alliance
- Wright-Patterson Air Force Base

3.3 Scope

The principal target of CGNS is the data normally associated with compressible viscous flow (i.e., the Navier-Stokes equations), but the standard is also applicable to subclasses such as Euler and potential flows.

CGNS Version 1.0, released 5/15/98, was limited to problems described by multiblock structured grids. Version 1.1 addresses grids, flowfields, boundary conditions, and block-to-block connection information. Also included are a number of auxiliary items, including nondimensionalization, reference state, and equation set specifications. The extension to time-dependent flows and unstructured grids is addressed in Version 2. Also included are links between CGNS data and CAD geometry. Any mix of the following types of field data can be recorded:

- nodal.
- edge-centered.
- face-centered.
- cell-centered.

Block connections can be of the following types:

- contiguous (one-to-one).
- abutting (patched mismatched).
- overlapping (Chimera).

Much of the standard and the software is applicable to computational field physics in general. Disciplines other than fluid dynamics would need to augment the data definitions and storage conventions, but the fundamental database software, which provides platform independence, is not specific to fluid dynamics.

4 Summary Description of ADF (Advanced Data Format)

The purpose of the current document is to describe the way in which CFD data is to be stored in an ADF file. To do this, it is necessary to first describe the structure of the ADF file itself in some detail. Therefore, a conceptual summary of ADF is given here in order to make the current document relatively independent, and to allow the reader to focus on those aspects of ADF which are essential to understanding the file mapping. The *ADF User's Guide* should be used as the authoritative reference to resolve any issues not covered by this summary.

4.1 General Description of ADF

The ADF, or Advanced Data Format, together with its access software known as the ADF Core, constitutes a general database manager particularly suited to the storage of numerical data. Its use is not restricted to data connected with CFD, and ADF contains no built-in references to concepts from CFD.

4.1.1 ADF Files and the ADF Core

Files created by the ADF Core are referred to as ADF files. These are binary files whose precise physical form on the external storage medium is completely controlled by the Core routines. ADF files are not intelligible or accessible except through the ADF Core routines, and their physical form is of interest only to ADF Core programmers.

The ADF Core routines perform typical operations on ADF files: open, close, create, delete, read, write, and so on. They are written in ANSI C and are thus themselves portable to any platform supporting an ANSI C compiler. Because the Core completely determines the physical form of the ADF files, the files themselves can be read on those platforms as well.[2] In addition to portability, this arrangement provides integrity of data across both space and time. In particular, it is never necessary to know more about an ADF file (other than that it is one) in order to open it and find out what it contains.

The ADF Core implements the minimal set of procedures required to fully manipulate the database. The Core itself is written in C, but each Core call is also provided in Fortran. This enables the user to access ADF data from either of these languages.

4.1.2 The Conceptual Structure of ADF Files

Although the physical structure of an ADF file in storage is (or should be) of little concern to users of ADF, an understanding of its logical or conceptual structure is essential. This structure determines its suitability for the type of data at hand and is reflected in all of the ADF Core calling sequences.

An ADF file consists entirely of a collection of elements called nodes. These nodes are arranged in a tree structure which is logically similar to a UNIX file system. The nodes are said to be connected in a "child-parent" relationship according to the following simple rules:

1. Each node may have any number of child nodes.

[2] There are necessarily some issues relating to the retention of precision on platforms of varying word length.

2. Except for one node, called the root, each node is the child of exactly one other node, called its parent.

3. The root node has no parent.

Every node in an ADF file has exactly the same internal structure. Each node contains identifying information, pointers to any children, and, optionally, data.

When an ADF file is opened (by the appropriate Core routine), information is returned to the calling program which is sufficient to access the root node. It is then the responsibility of the program to search the tree for whatever information is required, or to add to the tree any information it wishes to store.

There is a special kind of node called a link, which serves as a pointer to a node in another ADF file, or in another part of the same ADF file. The tree structure at and below the node to which the link points is available as if that node were present instead of the link. This allows an ADF file to span multiple physical files, and also allows a portion of one ADF file to be referenced by several other ADF files.

4.1.3 The ADF Mid-Level Library (projected)

The ADF Core routines access the data at a very fundamental level. Since by definition the Core implements a minimum number of basic functions, it necessarily deals with the data at a very fundamental level. While skilled programmers may find this acceptable, most programs define higher level routines which coalesce oft-repeated sequences of Core calls. We envision that these routines will eventually be gathered into an ADF "Mid-Level" Library.

At this time, there are approximately four such routines. However, there has been no coordinated effort to gather, organize, or distribute such a Library.

(These remarks apply only to routines designed to access general ADF data. There does exist a set of higher level routines used to access CFD related data, namely, the CGNS Mid-Level Library.)

4.2 The Structure of an ADF Node

The File Mapping specifies not only the location of the node at which a particular kind of data is to be stored, but also how the internal structure of the node is to be used. Each node contains a number of fields into which data may be entered directly via ADF Core calls. They are:

- The Node Name
- The Label
- The Data Type
- The Number of Dimensions
- The Dimension Values
- The Data

In addition, two other entities associated with the nodes are managed by ADF itself. These are:

- The Node ID
- The Child Table

4.2.1 The Node ID

The node ID is a unique identifier assigned to each existing node by ADF when the file containing it is opened, and to new nodes as they are created. ADF Core inquiries generally return node IDs as a result and accept node IDs as input. By building a table of IDs, calling software can subsequently access specific nodes without further search. The Node ID is real and is not under user control.

4.2.2 The Node Name

The node Name is a 32-byte character field which is user controllable. Its general use is to distinguish among the children of a given node; consequently, no two children of the same parent may have the same Name.

4.2.3 The Label

The Label is a 32-byte character field which is user controllable. ADF assigns no formal role to the Label, but the intent was to identify the structure of the included data. It is common for the various children of a single parent to store different instances of the same structure. Therefore, there is no prohibition against more than one child of the same parent having the same Label.

4.2.4 The Data Type

The Data Type is a 32-byte character field which specifies the type and precision of any data which is stored in the data field. Types provided by ADF are:

Table 1: Data Types

Data Type	Notation
No Data	`MT`
Integer 32	`I4`
Integer 64	`I8`
Unsigned Integer 32	`U4`
Unsigned Integer 64	`U8`
Real 32	`R4`
Real 64	`R8`
Complex 64	`X4`
Complex 128	`X8`
Character	`C1`
Byte	`B1`
Link	`LK`

If the data type is `MT` or `LK`, the node attributes which described the data may be left undefined.

4.2.5 The Number of Dimensions

The Data portion of a node is designed to store multi-dimensional arrays of data, each element of which is presumed to be of the Data Type specified. The Number of Dimensions specifies the number of integers required to reference a single datum within the array.

4.2.6 The Dimension Values

The Dimension Values are a list of integers expressing the actual sizes of the stored array in each of the dimensions specified.

4.2.7 The Data

The portion of the node holding the actual stored data array.

4.2.8 The Child Table

ADF maintains a table recording the number of, and pointers to, the children of each node. The table is adjusted when children are added or deleted by ADF Core calls.

Children may be identified by their names and labels, and, thence, by their node IDs once these have been determined. ADF provides no notion of order among children. In particular, the order of a list of children returned by ADF has nothing to do with the order in which they were inserted in the file. However, the order returned is consistent from call to call provided the file has not been closed and the node structure has not been modifed.

Note that there is no *parent* table; that is, a node has no direct knowledge of its parent. Since calling software must open the file from the root, it presumably cannot access a child without having first accessed the parent. It is the responsibility of the calling software to record the node ID of the parent if this information will be required.

5 General CGNS File Mapping Concepts

This section describes the general philosophy underlying the use of the ADF tree structure by CGNS. Section 6 describes the exact conventions for each type of data.

In Section 5.1, we first describe the roles of the various ADF node attributes as they are specifically applied within CGNS. Section 5.3 describes the overall layout of the tree structure itself.

5.1 Use of ADF Nodes in CGNS

Section 4.2 described the general role of each of the ADF node attributes without reference to CFD. Here we note any additional information regarding their use within CGNS.

Attributes described in 5.1.1 through 5.1.8 are those recognized by both ADF and CGNS. In 5.1.9 through 5.1.11 we describe certain attributes of nodes which are derived from context, i.e., which the node possesses by virtue of its location within a CGNS database. These notions, namely, Cardinality, Parameters and Functions, are unique to CGNS.

5.1.1 The Node ID

The Node ID is completely controlled by ADF, and thus its role is exactly the same for CGNS as it is for ADF. CGNS software accesses the Node ID only through calls to ADF. ADF itself guarantees that Node IDs are unique and constant within any ADF file (or collection of files) while the file(s) are open.

5.1.2 The Node Name

In CGNS, the Name may be left to the choice of the user, or it may be specified by the SIDS. At the levels of the tree nearest the root, the (end-)user is free to set the Name to distinguish among like objects in the case at hand. For example, in a multizone problem, nodes associated with different zones might be named "`UnderLeftWing`" or "`AboveForwardFuselage`". At this level, it is generally not possible to identify a collection of names which are likely to recur. This means that the naming of high level objects does not require standardization, and the SIDS are silent regarding the naming convention.

Because every ADF node must be given a name when it is opened, default names, based on the node Label, are provided by convention. The CGNS Midlevel Library will record the default names if none is provided by the user. The precise formula is given in the Label section below.

At levels of the tree farther from the root, the SIDS often specify the name. There is, for example, a commonly encountered collection of flow variables whose general meaning is widely understood. In this case, standardizing the name conveys precise information. Thus the SIDS specify, for instance, that a node containing static internal energy per unit mass should have the Name "`EnergyInternal`". Adherence to these naming conventions guarantees uniform meaning of the data from site to site and user to user. Of course, there may be a desire to store quantities for which no naming convention is specified. In this case any suitable name can be used, but there is no guarantee of proper interpretation by anyone unaware of the choice.

5.1.3 The Label

Within CGNS, nearly all labels reflect C-style type definitions ("typedefs") specified by the SIDS, and end in the characters "`_t`". Some "Leaf" nodes (i.e. nodes that have no children) do not represent higher level CGNS structures and therefore have labels that do not follow the "`_t`" convention. At this writing, all such nodes have the type `int[]`, i.e., integer array, a type already recognized in C, for which a separate type definition would be artificial. Such nodes are generally located by the software through their names, which are specified by the SIDS, rather than through their labels.

The Label generally indicates the role of the data at and below the node in the context of CFD. Nodes which are entry points to data for a particular zone, for example, have the Label "`Zone_t`".

Parent nodes often have a number of children each containing data for different instances of the same structure. Multiple children of the same parent therefore often have the same Label. It is customary for software to conduct searches which depend on the Label, e.g., to determine the number of zones in a problem. The software will fail if the conventions regarding Labels are not observed.

Labels are also used to build default node Names. These are derived from the Label by dropping the characters "`_t`" and substituting the smallest positive integer resulting in a unique name among children of the same parent. For example, the first default Name for a node of type `Zone_t` will be "`Zone1`"; the second will be "`Zone2`"; and so on.

5.1.4 The Data Type

Data Types are completely specified by the file mapping. Although ADF provides a number of types, in CGNS the only types used are `MT` (No Data), `I4` (Integer), `R4` and `R8` (Real), `C1` (Character), and `LK` (Link).

The specification of data types within the File Mapping allows for the probability that files written under different circumstances may differ in precision. The issue is complicated by the ability of ADF to sense the capabilities of the platform on which it is running and interpret or record data accordingly. The general rule is that, although the user of ADF can specify the precision in which it is desired to read or write the data, ADF knows both the precision available at the source and the precision acceptable to the destination and will mitigate accordingly. Thus to specify the precison of real data as `R4`, for example, has no meaning unless both `R4` and `R8` are available. Therefore, the generic specification "`DataType`" is used to allow for all possibilities.

For all integer data specified by the SIDS, `I4` provides sufficient precision. Since ADF software handles conversion to the external environment, all integer data is specified by the File Mapping as `I4`.

5.1.5 The Number of Dimensions

Whenever data is stored at a node, it is in the form of a single array of elements of a single date type. Insofar as possible, the dimension specified by CGNS is the natural underlying dimension; for example, a rectangular array of pressures is stored with dimension equal to the physical dimension of the problem.

There are situations in which this representation is not feasible. For instance, a list of points which do not form a rectangular array in physical space may be compacted into a one-dimensional array in ADF.

Frequently the data is of type `C1` (character data). In some cases, the data holds additional information in the form of a name specified by the SIDS, and in some cases holds user comment. All such data is generally represented as a one-dimensional array (or list) of characters.

5.1.6 The Dimension Values

These are used exactly as specified by ADF. In the case of rectangular arrays of physical data, the dimension values are set to the actual sizes in physical space. Note that these sizes often depend on whether the values are associated with grid nodes, cell centers or other physical locations with respect to the grid. In any event, they refer to the amount of data actually stored, not to any larger array from which it may have been extracted.

In the case of list data, the dimension value is the length of the list. Lists of characters may contain termination bytes such as "`\n`"; by this means an entire document can be stored in the data field.

5.1.7 The Data

CGNS imposes no conventions on the data itself beyond those specified by ADF. Note that it is a responsibility of the CGNS software to ensure that the amount and type of stored data agrees with the specification of the data type, number of dimensions, and dimension values.

5.1.8 The Child Table

The Child Table is completely controlled by ADF, and thus its role is exactly the same for CGNS as it is for ADF. CGNS software accesses and modifies the child table only through calls to ADF.

In addition to the meaning of attributes of individual ADF nodes, the File Mapping specifies the relations between nodes in a CGNS database. Consequently, the File Mapping determines what kinds of nodes will lie in the child table.

It is important to reemphasize that ADF provides no notion of order among children. This means children can be identified only by their names, labels and system-provided node IDs. In particular, the order of a list of children returned by ADF has nothing to do with the order in which they were inserted in the file. However, the order returned is consistent from call to call provided the file has not been closed and the node structure has not been modifed.

5.1.9 Cardinality

The *cardinality* of a CGNS node is the number of nodes of the same label permitted at one point in the tree, i.e., as children of the same parent. It consists of both lower and upper limits.

Since the notion of a CGNS database allows for work in progress, the lower limit is generally zero (although the database may be of little use until certain nodes are filled). The upper limit is usually either one or many (N).

5.1.10 Parameters

CGNS relies on the fact that ADF nodes cannot be found except by following the pointers from their parents. This means that software accessing a node has had an opportunity to note all the

data above that node in the tree. Therefore, nodes do not repeat within themselves information which is necessary for their interpretation but which is available at a higher level.

A datum which is necessary for the proper interpretation of a node but which is derived from its ancestors is referred to as a *structure parameter*.

5.1.11 Functions

Occasionally the proper interpretation of a node depends on an implicitly understood *function* of its structure parameters. Usually these relate to the actual amount of data stored. Several of these functions are defined in the SIDS and referenced in this document.

5.2 CGNS Databases

5.2.1 Definition of a CGNS Database

By definition, a CGNS database is created when, within an ADF file, a node is created which conforms to the specifications given below for a node of type "`CGNSBase_t`". This node is conceptually the root of the CGNS database. Because it is created and controlled by the user, it cannot be the root of the ADF file. Current CGNS conventions require that it be located directly below the ADF root node.

Further, by the mechanism of links, a CGNS database may span multiple files. Thus there is no notion of a CGNS *file*, only of a CGNS *database* implemented within one or more *ADF files*.

By virtue of its intended use, a CGNS database is dynamic in that its content at any time reflects the current state of a CFD problem of interest. For example, after the completion of a grid generation procedure, a CGNS file may contain a grid but no boundary conditions. Therefore, beyond the occurrence of a `CGNSBase_t` node, there is no minimum content required in a CGNS database.

Conversely, there is no proscription against the inclusion, anywhere within an ADF file containing a CGNS database, of nodes of any form whatsoever, provided only that their naming and labeling does not mimic CGNS conventions. Such "non-CGNS" nodes, and those below them in the ADF tree, are not regarded as part of the CGNS database. CGNS software will not detect the existence of non-CGNS nodes.

We may therefore take the following as a definition of a CGNS database:

> *A CGNS database is a subtree of an ADF file or files which is rooted at a node with label "`CGNSBase_t`" and which conforms to the SIDS data model as implemented by the SIDS-to-ADF File Mapping.*

5.2.2 Location of CGNS Databases within ADF Files

An ADF file may contain more than one `CGNSBase_t` node; i.e., there may be more than one CGNS database rooted within the same ADF file. CGNS software accepts the *name* of the desired database as an argument, and will locate the correct `CGNSBase_t` node within the specified ADF file. Obviously, each `CGNSBase_t` node in a single ADF file must have a unique name.

A CGNS database may link to CGNS nodes in the same or other ADF files. Thus, for example, a CGNS database may reference the grid from another CGNS database without physically copying

the the information. In this case, the structure of the ADF file into which the link is made is invisible except below the node to which the link is made.

5.2.3 File Management

Beyond *Open* and *Close* neither ADF nor CGNS provides any file management facilities. The user is responsible for ensuring that:

- The ADF file containing the root of the required database is available and its permissions are properly set at runtime.
- If links are made to other ADF files, including any not under the user's direct control, these are also available at runtime.
- No file is opened for writing by more than one program at a time.

It is possible, within CGNS, to protect files from inadvertent writing by opening them as "read only".

5.3 Internal Organization of a CGNS Database

5.3.1 The `CGNSBase_t` Node

At the highest level of the tree defining a CGNS database there is always a node labeled "`CGNSBase_t`". The name of this node is user defined, and serves essentially as the name of the database itself. This name is used by the CGNS software to open the database.

5.3.2 The `CGNSLibraryVersion_t` Node

An ADF file may also contain other nodes below the root node beside `CGNSBase_t`, but these are *not* officially part of the CGNS database and will not be recognized by most CGNS software. One exception to this is a node called `CGNSLibraryVersion_t`, which is a child of the ADF root node. This node stores the version number of the CGNS standard with which the file is consistent, and is created automatically when the file is created or modified using the CGNS Mid-Level Library. Officially, the CGNS version number is not a part of the CGNS database (because it is not located below `CGNSBase_t`). But because the Mid-Level Library software makes use of it, the node is included in this document.

5.3.3 Topological Basis of CGNS Database Organization

Below the root, the organization of a CGNS database reflects the problem topology. Omitting detail, Figure 1, Appendix A shows the overall structure of the ADF file. Below the ADF root node is the `CGNSLibraryVersion_t` node, and one or more `CGNSBase_t` nodes. Each `CGNSBase_t` node is the root of a CGNS database.

At the next level below a `CGNSBase_t` node are general specifications which apply to the problem globally, such as reference states, units, and so on. At this level we also find a collection of nodes labeled "`Zone_t`". The tree below each of these holds all the data local to one of the various zones or subdomains which constitute the problem.

Beneath each `Zone_t` node there are nodes whose subtrees store: the grid (labeled `GridCoordinates_t`); flowfields (`FlowSolution_t`); boundary conditions (`ZoneBC_t`); information about the geometrical connection to other zones (`GridConnectivity_t`); and information defining time-dependent data. Below these there may be additional nodes containing yet more geometrically local information. For example, under the `ZoneBC_t` node there are nodes defining individual boundary conditions on portions of faces of the zone (`BC_t`).

Certain types of nodes originally specified at a high level are optionally repeated below. For example, immediately below a `Zone_t` node we may find another `ReferenceState_t` node (see Appendix A, Figure 4). The CGNS convention is that such a node overrides (for the associated portion of the topology only) any data found at a higher level.

5.3.4 Topics Not Currently Covered

No specification of the kind represented by this file mapping can ever be complete. However, it is worth noting that there are certain entities common in CFD which are not currently specified by the file mapping.

Within nodes of type `FlowSolution_t`, the current file mapping permits the storage of fields of any number of dependent variables. In addition to those whose names are specified in the SIDS the user may add any desired quantities, naming them appropriately. Names that are not currently codified in the SIDS will not be common between practitioners without separate communication.

Obviously any sort of physical field could be stored in a `FlowSolution_t` node. The problem with using CGNS for such applications lies in the probable need to specify additional physical information. Standardizing this information is tantamount to extending the SIDS and File Mapping to the disciplines in question.

Similarly, if a reacting flow problem requires the specification of rate tables or catalytic wall boundary conditions, extensions to the SIDS and File mapping will be needed.

6 Detailed CGNS Node Descriptions

This section, together with the figures in Appendix A, constitutes a complete description of the CGNS database structure, together with detailed descriptions of the contents of each attribute of each node. It is intended to be suitable as a reference for anyone implementing CGNS using the ADF Core, but should also be of interest to those wishing to understand exactly where information is stored within a CGNS database.

Note that it is the advertised purpose of the CGNS Mid-Level Library to store and retrieve information in conformity with the mapping herein described. Therefore, anyone accessing a CGNS database through the CGNS routines alone does not need a detailed understanding of the file mapping per se. However, this document should still prove useful in ascertaining the meaning of some of the arguments which must be supplied to the Library routines. Further, it will be necessary to consult the SIDS themselves to determine some of the naming conventions.

The node descriptions in this section are closely coupled to and cross-referenced with the figures in Appendix A, which show all the nodes defined in the SIDS that have child nodes. In the current section, the "**Children:**" entry in the list of *Node Attributes* is a reference to the figure showing that node with its children.

The nodal hierarchy of the CGNS database directly reflects that of the SIDS. Certain sections of the SIDS, notably Sections 4, 5, and 12, describe basic data structures which appear repeatedly as children of nodes representing more complex structures. In order to simplify the presentation and avoid the introduction of undefined terms, this section has been divided into two parts. Section 6.1 defines a number of basic types which recur often in the structure, and Section 6.2 describes higher level nodes of more specific function built from those in Section 6.1.[3]

6.1 Basic CGNS Nodes

In this section we describe CGNS nodes which hold fundamental *types* of information. Their structure and function, which are the same everywhere, are described here. However, the *meaning* of the data they hold at any particular point in a CGNS database depends on the context, i.e., the parent node. Therefore, where necessary, any special context-dependent meaning is elaborated in the paragraph devoted to the parent.

6.1.1 Descriptor Group

These are user-assigned nodes designed to further describe the user's intent. Their data is meant for human perusal or other user-designated purposes.

6.1.1.1 `Descriptor_t`

The purpose of a node of type `Descriptor_t` is to store textual data. It is not intended to be read by any system software, except to return the text for human examination.

The intent of the `Descriptor_t` node is to hold comment sufficient to allow someone other than the originator to understand what the node contains. This may consist of a problem description, reference documents, personal notes, etc.

[3] For convenience, we group the node descriptions according to the type of data the nodes contain. These groupings only roughly correspond to the chapters in the SIDS.

Any node may have zero to many `Descriptor_t` children, with the uses differentiated by the Name field. At this time, there are *no* conventions for these names or for the form of the associated text. It is expected that a standard set such as `README`, `TimeStamp`, etc. will evolve as a matter of practice.

Node Attributes

Name:	User defined
Label:	`Descriptor_t`
DataType:	`C1`
Dimension:	1
Dimension Values:	Length of string to be stored, including any carriage control or null bytes.
Data:	String - Since line terminators can be stored within the data, the user could conceptually store an entire document in this area, read it into a program and then print it out. For example, an entire PostScript document describing the problem (and maybe results) could be stored in the Data field, read by a program and then sent to a printer.
Children:	None
Cardinality:	0,*N*

6.1.1.2 `Ordinal_t`

Because there is no notion of order among children, there is occasionally a desire to order children in a way that survives from one opening of a CGNS database to another. The current CGNS Library provides means of doing this. However, another early method was to place the node "number" in a child of type `Ordinal_t`.

Like `Descriptor_t`, the `Ordinal_t` node is completely under the control of the user, who takes full responsibility for its content. Unlike `Descriptor_t`, CGNS conventions do not encourage the use of `Ordinal_t`, as it usually encodes information which is redundant with the name. It is not read or written by standard CGNS software, and so there is no assurance that sibling nodes will be differently, consecutively, or consistently numbered by `Ordinal_t`. Clearly, if `Ordinal_t` must be used, no node should have more than one `Ordinal_t` child, and no two siblings should have `Ordinal_t` children containing the same data.

It is worth noting that, if consistent numbering is desired, one way of achieving it is to make the desired integer either the name or part of the name. In fact, if, for example, individual zones are left unnamed, the default convention will provide names of `Zone1`, `Zone2`, etc. Alternatively, the character strings "`1`", "`2`", ..., are legal names. ADF or CGNS software, of course, will return these as strings. This may necessitate type conversion or parsing before the names can be used as integer indices.

Node Attributes

Name:	`Ordinal`
Label:	`Ordinal_t`
DataType:	`I4`
Dimension:	1
Dimension Values:	1
Data:	The user-defined ordinal number (an integer).
Children:	None
Cardinality:	0,1

6.1.2 Physical Data Group

6.1.2.1 `DataClass_t`

A `DataClass_t` node specifies the dimensional nature of the data in and below its parent. It overrides any `DataClass_t` information higher up in the tree. There are six recognized string values. It is necessary to consult the SIDS to determine the precise meaning.

Node Attributes

Name:	`DataClass`
Label:	`DataClass_t`
DataType:	`C1`
Dimension:	1
Dimension Values:	The length of the string
Data:	One of: `DataClassNull`, `DataClassUserDefined`, `Dimensional`, `NormalizedByDimensional`, `NormalizedByUnknownDimensional`, `NondimensionalParameter`, or `DimensionlessConstant`
Children:	None
Cardinality:	0,1

6.1.2.2 `DimensionalUnits_t`

A `DimensionalUnits_t` node specifies dimensional units which apply to data in and below its parent. It overrides any `DimensionalUnits_t` information higher up in the tree. There are five strings to specify, corresponding, respectively, to units for mass, length, time, temperature, and angular measure. The number of recognized string values varies with the physical property.

Units for three additional types of data are specified in a child node, `AdditionalUnits_t`.

Node Attributes

Name:	`DimensionalUnits`	
Label:	`DimensionalUnits_t`	
DataType:	`C1`	
Dimension:	2	
Dimension Values:	(32,5)	
Data:	For mass, one of:	`MassUnitsNull`, `MassUnitsUserDefined`, `Kilogram`, `Gram`, `Slug`, `Pound-Mass`
	For length, one of:	`LengthUnitsNull`, `LengthUnitsUserDefined`, `Meter`, `Centimeter`, `Millimeter`, `Foot`, `Inch`
	For time, one of:	`TimeUnitsNull`, `TimeUnitsUserDefined`, `Second`
	For temperature, one of:	`TemperatureUnitsNull`, `TemperatureUnitsUserDefined`, `Kelvin`, `Celsius`, `Rankine`, `Fahrenheit`
	For angles, one of:	`AngleUnitsNull`, `AngleUnitsUserDefined`, `Degree`, `Radian`
Children:	See Figure 40	
Cardinality:	0,1	

6.1.2.3 `AdditionalUnits_t`

An `AdditionalUnits_t` node specifies dimensional units for additional types of data. To maintain compatibility with earlier CGNS versions, this is an optional child node of `DimensionalUnits_t`. The specified units apply to data in and below the parent of the corresponding `DimensionalUnits_t` node, and override any `AdditionalUnits_t` information higher up in the tree. There are three strings to specify, corresponding, respectively, to units for electric current, substance amount, and luminous intensity. The number of recognized string values varies with the physical property.

Node Attributes

Name:	`AdditionalUnits`	
Label:	`AdditionalUnits_t`	
DataType:	`C1`	
Dimension:	2	
Dimension Values:	(32,3)	
Data:	For electric current, one of:	`ElectricCurrentUnitsNull`, `ElectricCurrentUnitsUserDefined`, `Ampere`, `Abampere`, `Statampere`, `Edison`, `auCurrent`
	For substance amount, one of:	`SubstanceAmountUnitsNull`, `SubstanceAmountUnitsUserDefined`, `Mole`, `Entities`, `StandardCubicFoot`, `StandardCubicMeter`
	For luminous intensity, one of:	`LuminousIntensityUnitsNull`, `LuminousIntensityUnitsUserDefined`, `Candela`, `Candle`, `Carcel`, `Hefner`, `Violle`
Children:	None	
Cardinality:	0,1	

6.1.2.4 `DataConversion_t`

A `DataConversion_t` node specifies a non-homogeneous linear function which converts non-dimensional data in its parent to raw dimensional data. Although in principle it overrides any `DataConversion_t` information higher up in the tree, it is generally not meaningful for it to apply to more than one kind of physical data. Therefore, CGNS specifies its use only as a child of a node which actually contains a single type of real data.

There are two values to specify, corresponding to the scale factor and offset. The SIDS contain the exact conversion formula.

Node Attributes

Name:	`DataConversion`
Label:	`DataConversion_t`
DataType:	`R4` or `R8`
Dimension:	1
Dimension Values:	2
Data:	`ConversionScale`, `ConversionOffset`
Children:	None
Cardinality:	0,1

6.1.2.5 DimensionalExponents_t

A DimensionalExponents_t node specifies the powers of mass, length, time, temperature, and angular measure which characterize dimensional data in its parent. Although in principle it overrides any DimensionalExponents_t information higher up in the tree, it is generally not meaningful for it to apply to more than one kind of physical data. Therefore, CGNS specifies its use only as a child of a node which actually contains a single type of real data. There are five values to specify, corresponding to the five types of units specified using DimensionalUnits_t. The data type is real, not integer.

Exponents for three additional types of data are specified in a child node, AdditionalExponents_t.

Node Attributes

Name:	DimensionalExponents
Label:	DimensionalExponents_t
DataType:	R4 or R8
Dimension:	1
Dimension Values:	5
Data:	MassExponent, LengthExponent, TimeExponent, TemperatureExponent, AngleExponent
Children:	See Figure 41
Cardinality:	0,1

6.1.2.6 AdditionalExponents_t

An AdditionalExponents_t node specifies the powers of the units for additional types of data, which characterize the corresponding dimensional data. There are three values to specify, corresponding to the three types of units specified using AdditionalUnits_t. The data type is real, not integer.

Node Attributes

Name:	AdditionalExponents
Label:	AdditionalExponents_t
DataType:	R4 or R8
Dimension:	1
Dimension Values:	3
Data:	ElectricCurrentExponent, SubstanceAmountExponent, LuminousIntensityExponent
Children:	None
Cardinality:	0,1

6.1.2.7 DataArray_t

A DataArray_t node is a very general type of node meant to hold large arrays of data, such as grids and flowfields. Often, some of the attributes of a DataArray_t node depend on the context in which the node is found; that is, they are structure parameters.

For example, the SIDS specify that the Data Type of DataArray_t is a structure parameter, "DataType", which may assume any of the values "I*n*", "R*n*", "C*n*", or "bit".

The other two attributes of DataArray_t, Dimensions and DataSize, also depend on the context where they are being used. Dimensions is a function of the underlying dimensionality of the data

being described (often `IndexDimension`, defined in the `CGNSBase_t` node), and the `DataSize` may be inferred from detailed descriptions of the grid.

A node may have any number of `DataArray_t` children. The meaning of their contents is differentiated by Name, often according to conventions specified by the SIDS. SIDS names are usually precise and descriptive, such as `CoordinateTheta` or `EnergyInternal`. (For a current list of sanctioned names, see the SIDS, Appendix A.) Conversely, quantities not specified by the SIDS can be stored in `DataArray_t` nodes, but should be given names other than those specified in the SIDS. In other words, to comply with the SIDS requires that one give a quantity the SIDS-defined name *if and only if* it is one of the SIDS-defined quantities.

Node Attributes

Name:	Context dependent
Label:	`DataArray_t`
DataType:	Context dependent
Dimension:	Context dependent
Dimension Values:	Context dependent
Data:	The array of data values
Children:	See Figure 39
Cardinality:	0,*N*
Parameters:	`DataType`, dimension of the data, size of the data

6.1.2.8 Integer Arrays

Integer array nodes perform the same function as nodes of type `DataArray_t`, but store integer instead of real arrays. They are always of type `int[]`, with the dimensions and values given either explicitly in the appropriate fields, or as parameters or functions.

Node Attributes

Name:	Context dependent
Label:	`int`, `int[IndexDimension]`, `int[2*IndexDimension]`, or `int[1 + ... + IndexDimension]`
DataType:	`I4`
Dimension:	1
Dimension Values:	1, `IndexDimension`, `2*IndexDimension`, or `(1 + ... + IndexDimension)`
Data:	The array of integer values
Cardinality:	0,1
Children:	None
Parameters:	`IndexDimension` or none (context dependent)

6.1.3 Location and Position Group

6.1.3.1 `GridLocation_t`

A `GridLocation_t` node specifies the physical location, with respect to the underlying grid, with which the field data below its parent is associated. The value (data field) is a character string of enumeration type, i.e., it must take one of a number of predefined values. These values are: `Vertex`, `CellCenter`, `FaceCenter`, `IFaceCenter`, `JFaceCenter`, `KFaceCenter`, or `EdgeCenter`. The strings are case sensitive, and an exact match is required. The `GridLocation_t` node is optional, and the default is `Vertex`.

Node Attributes

Name:	`GridLocation`
Label:	`GridLocation_t`
DataType:	`C1`
Dimension:	1
Dimension Values:	Length of the string value
Data:	`Vertex`, `CellCenter`, `FaceCenter`, `IFaceCenter`, `JFaceCenter`, `KFaceCenter`, or `EdgeCenter`
Children:	None
Cardinality:	0,1

6.1.3.2 `Rind_t`

The presence of a `Rind_t` node indicates that field data stored below its parent includes values associated with a spatial extent beyond that of the basic underlying grid. Such data often arise from the use of ghost cells, or from the copying of information from adjacent zones.

Within a single zone, the size of the basic grid is found in the data field of the `Zone_t` node (see Section 6.2.8.1). The data field of a `Rind_t` node contains integers specifying the number of planes (for structured grids) or number of rind points or elements (for unstructured grids) of included extra data. The planes for structured grids correspond to the low and high values in the i-direction, low and high values in the j-direction, and low and high values in the k-direction (if needed), in that order. Note that the actual size of the field data, which is stored in a `DataArray_t` sibling node, is a `DataSize` structure parameter which depends on the basic grid size, the `GridLocation`, and the `Rind`.

The `Rind_t` node is optional, and the default is no rind.

Node Attributes

Name:	`Rind`
Label:	`Rind_t`
DataType:	`I4`
Dimension:	1
Dimension Values:	`2*IndexDimension`
Data:	Number of planes of extra data in low i, high i, low j, high j, etc. (for structured grids) or number of points or elements of extra data (for unstructured grids)
Children:	None
Cardinality:	0,1
Parameters:	`IndexDimension`

6.1.3.3 `IndexRange_t`

An `IndexRange_t` node describes a subregion of a zone. This may be, for example, a sub-block or a portion of a face of a zone. It may be used to describe the locations of boundary condition patches and holes for overset grids.

Node Attributes

Name:	`PointRange`, `PointRangeDonor`, `ElementRange`, or user defined
Label:	`IndexRange_t`
DataType:	`I4`
Dimension:	2

Dimension Values:	`IndexDimension`, 2
Data:	First indices, last indices
Children:	None
Cardinality:	Context dependent
Parameters:	`IndexDimension`

6.1.3.4 `IndexArray_t`

An `IndexArray_t` node describes a general subregion of a zone. Unlike `IndexRange_t`, it lists all the elements of the subregion, rather then only the first and last ones. Its use is similar to `IndexRange_t`.

Node Attributes

Name:	`PointList`, `PointListDonor`, `CellListDonor`, or `InwardNormalList`
Label:	`IndexArray_t`
DataType:	`I4`, or (for `InwardNormalList`) `R4` or `R8`
Dimension:	2
Dimension Values:	`IndexDimension`, number of items in the list; or (for `InwardNormalList`) `PhysicalDimension`, number of items in the list
Data:	Index coordinates of each point or element in the list, or (for `InwardNormalList`) physical-space normal vectors at each point or element in the list
Children:	None
Cardinality:	0,1
Parameters:	`IndexDimension`, either `PointListSize` or `ListLength`, and `DataType`; or (for `InwardNormalList`) `PhysicalDimension`, `ListLength`, and `DataType`

6.1.4 Auxiliary Data Group

6.1.4.1 `ReferenceState_t`

The appearance of a `ReferenceState_t` node is optional. It is used to specify the values of flow quantities at reference conditions, e.g., at freestream or stagnation. This is typically done for the whole database, in which case the `ReferenceState_t` node is a child of the `CGNSBase_t` node.

`ReferenceState_t` nodes follow the usual convention that information specified lower in the tree overrides higher level specifications. Such overrides are therefore specified if a `ReferenceState_t` node appears as a child of a `Zone_t`, `ZoneBC_t`, or `BCDataSet_t` node.

The actual values are stored in one or more `DataArray_t` children whose names identify the quantities being stored. If present, the units specified in the `DimensionalUnits_t` child apply to all `DataArray_t` children, subject to the usual override convention. (I.e., if one of the `DataArray_t` children itself has a `DimensionalUnits_t` child, it takes precedence over the higher level specification.)

Node Attributes

Name:	`ReferenceState`
Label:	`ReferenceState_t`
DataType:	`MT`
Children:	See Figure 38
Cardinality:	0,1

6.1.4.2 ConvergenceHistory_t

ConvergenceHistory_t nodes are intended for the storage of lists of quantities accumulated during calculations associated with either the entire CGNS database or with a single zone.

In the former case, they are called Global convergence histories, and appear as children of the CGNSBase_t node. In the latter, they are called Local and stored below, with the zones to which they correspond.

Each ConvergenceHistory_t node is a parent of a collection of one-dimensional DataArray_t nodes, each of which contains a list of values of a quantity defined by the user. These quantities are differentiated by their user-assigned Names. User definitions of the names are recorded in a Descriptor_t child node with Name NormDefinitions. Children of types DataClass_t and DimensionalUnits_t modify the meaning of the DataArray_t children in the usual manner.

Node Attributes

Name:	GlobalConvergenceHistory if under a CGNSBase_t node; ZoneConvergenceHistory if under a Zone_t node
Label:	ConvergenceHistory_t
DataType:	I4
Dimension:	1
Dimension Values:	1
Data:	Number of iterations
Children:	See Figure 36
Cardinality:	0,1

6.1.4.3 IntegralData_t

IntegralData_t nodes are intended for the storage of integrated flow quantities such as mass flows, forces and moments. These are kept in DataArray_t children just as in the ConvergenceHistory_t nodes, except that these nodes hold only one real number each.

Node Attributes

Name:	User defined
Label:	IntegralData_t
DataType:	MT
Children:	See Figure 37
Cardinality:	0,*N*

6.1.4.4 UserDefinedData_t

UserDefinedData_t nodes are intended as a means of storing arbitrary user-defined data in Descriptor_t and DataArray_t children without the restrictions or implicit meanings imposed on these node types at other node locations.

Multiple Descriptor_t and DataArray_t children may be stored below a UserDefinedData_t node, and the DataArray_t children may be of any dimension and size.

Node Attributes

Name:	User defined
Label:	UserDefinedData_t
DataType:	MT
Children:	See Figure 49

Cardinality: 0,N

6.1.4.5 `Gravity_t`

An optional `Gravity_t` node may be used to define the gravitational vector.

Node Attributes

Name: `Gravity`
Label: `Gravity_t`
DataType: `MT`
Children: See Figure 50
Cardinality: 0,1

6.2 Specialized Nodes

In this section we describe nodes whose use is specialized to certain types of CFD-related data. Although these nodes may appear in multiple places in a CGNS DataBase, they play a single role in the description of the data.

6.2.1 Grid Specification

CGNS recognizes the notion of a collection of subdomains called zones, within each of which there is a single structured or unstructured grid. Mathematically, the grid is an assignment of a location in physical space to each element in a discrete computational space. An essential feature of the grid is the connection structure it inherits from the underlying computational space.

It is possible, given a grid, to create others from it, by translation to cell centers, for example. However, CGNS views these as new field structures associated with the original grid, and the File Mapping specifies that they be stored as `FlowSolution_t` or `DiscreteData_t` nodes (see Section 6.2.2).

6.2.1.1 `GridCoordinates_t`

A `GridCoordinates_t` node describes a grid associated with a single zone. For a structured zone, the connection structure of the underlying computational space is that of a rectangular array, and its dimension is the `IndexDimension`, that is, the number of integers required to identify a point in the grid. The physical dimension is the number of real coordinates assigned at each grid point and need not be the same. Thus CGNS can store a grid, for example, with `IndexDimension` equal to two and a physical dimension of three, that is, a structured grid on a curved surface.

`IndexDimension` is a zone dependent parameter. For an unstructured grid, it always equals one, meaning that a unique index is required to specified a node location. For a structured grid, `IndexDimension` varries with the `CellDimension` of the mesh. For a mesh composed of 3D cells, `IndexDimension` equals 3, while for a mesh composed of surface or shell elements, `IndexDimension` equals 2. The values of the physical coordinates of the grid points are stored in `DataArray_t` children of `GridCoordinates_t`. The names of the coordinates are stored in the Name field of the corresponding `DataArray_t` node. For common coordinate systems, i.e., Cartesian, polar, cylindrical, and spherical, the names are specified by the SIDS.

Unlike `FlowSolution_t` and `DiscreteData_t` nodes (see Section 6.2.2), `GridCoordinates_t` nodes are not permitted to have `GridLocation_t` children, because all grid points are at vertices by definition.

Coordinate arrays may also contain rind data. If they do, the `GridCoordinates_t` node must have a `Rind_t` child node describing the amount of rind. All `DataArray_t` nodes under `GridCoordinates_t` must have the same size. Because the number of field quantities to be stored depends on the number of rind, the actual dimension values are functions, specified in this document by the generic term `DataSize[]`.

Under each node of type `Zone_t`, the original grid is contained in a node named `GridCoordinates`. Additional `GridCoordinates_t` data structures are allowed, with user-defined names, to store the grid at multiple time steps or iterations.

Node Attributes

Name:	`GridCoordinates` or user defined
Label:	`GridCoordinates_t`
DataType:	`MT`
Children:	See Figure 5
Cardinality:	0,*N*
Parameters:	`IndexDimension`, `VertexSize`
Functions:	`DataSize`

6.2.1.2 `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.

Node Attributes

Name:	User defined
Label:	`Elements_t`
DataType:	`I4`
Dimension:	1
Dimension Values:	2
Data:	`ElementType` value, `ElementSizeBoundary`
Children:	See Figure 6
Cardinality:	0,*N*

6.2.1.3 `Axisymmetry_t`

The `Axisymmetry_t` data structure may be included as a child of the `CGNSBase_t` node to record the axis of rotation and the angle of rotation around this axis for an axisymmetric database.

Node Attributes

Name:	`Axisymmetry`
Label:	`Axisymmetry_t`
DataType:	`MT`
Children:	See Figure 7
Cardinality:	0,1

6.2.1.4 `RotatingCoordinates_t`

The `RotatingCoordinates_t` data structure may be included as a child of either the `CGNSBase_t` node or a `Zone_t` node to record the center of rotation and the rotation rate vector for a rotating coordinate system.

Node Attributes

Name:	`RotatingCoordinates`
Label:	`RotatingCoordinates_t`
DataType:	`MT`
Children:	See Figure 8
Cardinality:	0,1

6.2.2 Field Specification

The object of computational field physics is to compute fields of physical data associated with points in space.

6.2.2.1 `FlowSolution_t`

A `FlowSolution_t` node describes a field of physical data associated with the grid for a single zone. It is intended for the storage of computed flowfield data such as densities and pressures. There is no convention as to how many or what kind of quantities must or may be stored. In particular, it is not specified that the quantities need in any sense be either complete or non-redundant.

The data are stored in `DataArray_t` children of `FlowSolution_t`. These `DataArray_t` nodes are dimensioned by the same underlying `IndexDimension` parameter as the grid, and the order of storage within the `DataArray_t` nodes is presumed the same as it is for the grid. The names of the physical quantities are stored in the Name field of the corresponding `DataArray_t` node. For common fluid dynamic quantities the names are specified by the SIDS.

The relationship between the locations of the field quantities and the vertices of the grid is specified by a `GridLocation_t` child node. If this node is absent, the field quantities are assumed to be associated with the grid vertices. Field arrays may also contain rind data. If they do, the `FlowSolution_t` node must have a `Rind_t` child node describing the amount of rind. All `DataArray_t` nodes under a single `FlowSolution_t` must have the same size. Field arrays containing different numbers of rind must be stored under different `FlowSolution_t` nodes. There may be any number of nodes of type `FlowSolution_t` under a `Zone_t`.

Because the number of field quantities to be stored depends on the number of rind and on the location with respect to the grid, the actual dimension values are functions, specified in this document by the generic term `DataSize[]`.

The meaning of the field arrays is modified in the usual way by any `DataClass_t` or `DimensionalUnits_t` children of the `FlowSolution_t` node.

Node Attributes

Name:	User defined
Label:	`FlowSolution_t`
DataType:	`MT`
Children:	See Figure 9
Cardinality:	0,N
Parameters:	`IndexDimension`, `VertexSize`, `CellSize`
Functions:	`DataSize`

6.2.2.2 `DiscreteData_t`

`DiscreteData_t` nodes are identical to `FlowSolution_t` nodes, but are intended for the storage

of fields of real data not usually identified as part of the field solution, such as cell-centered grids.

Node Attributes

Name:	User defined
Label:	`DiscreteData_t`
DataType:	`MT`
Children:	See Figure 9
Cardinality:	0,*N*
Parameters:	`IndexDimension`, `VertexSize`, `CellSize`
Functions:	`DataSize`

6.2.3 Connectivity Group

6.2.3.1 `Transform` Node

The `Transform` node is a node of type `int[]` which is identified by its name rather than its label. Thus the name must be "`Transform`". It appears only as a child of a node of type `GridConnectivity1to1_t`.

This node stores the transformation matrix relating the indices of two adjacent zones. Its data field contains a list of `IndexDimension` signed integers, each within the range `[-IndexDimension, ..., +IndexDimension]`, and no two of which have the same absolute value. Thus in 3-D allowed components are 0, ±1, ±2, and ±3. Each component of the array shows the image in the adjacent zone of a positive index increment in the current zone. The SIDS contain complete details.

Node Attributes

Name:	`Transform`
Label:	"`int[IndexDimension]`"
DataType:	`I4`
Dimension:	1
Dimension Values:	`IndexDimension`
Data:	Transformation matrix (shorthand)
Children:	None
Cardinality:	0,1
Parameters:	`IndexDimension`

6.2.3.2 `GridConnectivityType_t`

The purpose of this node is to describe the type of zone-to-zone connectivity specified by its parent, which is always a `GridConnectivity_t` node. The connectivity type is given in the data field as a character string which may take one of three specific values: `Abutting`, `Abutting1to1`, or `Overset`.

There is a shorthand form of the `GridConnectivity_t` node, namely, `GridConnectivity1to1_t`, which incorporates the assumption that the connection is `Abutting1to1`. Nodes of type `GridConnectivity1to1_t` do not have `GridConnectivityType_t` subnodes. However, `GridConnectivity1to1_t` nodes can only be used to specify zone-to-zone connections on rectangular subregions between two structured zones. So the use of `GridConnectivityType_t` subnodes to specify `Abutting1to1` is required if the connecting regions are not rectangular, or if the connectivity involves a least one unstructured zone.

Node Attributes

Name:	`GridConnectivityType`
Label:	`GridConnectivityType_t`
DataType:	`C1`
Dimension:	1
Dimension Values:	Length of string
Data:	`Abutting`, `Abutting1to1`, or `Overset`
Children:	None
Cardinality:	0,1

6.2.3.3 `GridConnectivity1to1_t`

This node is a shorthand format of `GridConnectivity_t` (see Section 6.2.3.4) capable of describing only `Abutting1to1` connections between two structured zones. The underlying subregion must have rectangular data structure.

Each `GridConnectivity1to1_t` node describes a subregion of a face of a zone whose vertices are coincident in a 1-to-1 fashion with those of a corresponding subregion of a face of another zone. Each `ZoneGridConnectivity_t` node may have as many `GridConnectivity1to1_t` (or `GridConnectivity_t`) children as are required to describe the connection structure.

The location of the connected subregion of a face of the current zone is given in a single child of type `IndexRange_t`, whose name is specified by the mapping as "`PointRange`". The location of the corresponding subregion on a face of the other zone is given in a single child of type `IndexRange_t`, whose name is specified by the mapping as "`PointRangeDonor`". The first (i.e., beginning) points in these `IndexRange_t` nodes are presumed to be coincident. The specification of the correspondence is completed by the inclusion of a `Transform` child node which describes the relative orientation of the two systems of indices. The second (i.e., end) point of the `PointRange` subnode specifies the extant of the connection.

In general, the File Mapping seeks to avoid the storage of redundant data. However, there are two redundancies associated with `GridConnectivity1to1_t`. First, for the sake of symmetry, the information recorded here is duplicated (in reverse) in a corresponding node under the donor zone. It is expected that these two specifications will agree.

Second, the end point of the `PointRangeDonor` can be calculated from the other three points specified, along with the transform. However, the transform cannot be inferred from the four points. Therefore, the end point of the `PointRangeDonor` is considered to be redundant, and the three points and the transform are designated as the primary specification.

Node Attributes

Name:	User defined
Label:	`GridConnectivity1to1_t`
DataType:	`C1`
Dimension:	1
Dimension Values:	Length of string
Data:	`ZoneDonorName`
Children:	See Figure 11
Cardinality:	0,N
Parameters:	`IndexDimension`

6.2.3.4 GridConnectivity_t

The GridConnectivity_t node is the most general format for describing grid connectivity. It can describe one-to-one, mismatched, and overset connectivity, and the underlying subregions of the connecting zones need not be rectangular.

Each GridConnectivity_t node describes a subregion of a zone which corresponds to a subregion of another zone. Each ZoneGridConnectivity_t node may have as many GridConnectivity_t (or GridConnectivity1to1_t) children as are required to describe the connection structure.

The location of the connected subregion of the current zone is given in a single child of type either IndexRange_t or IndexArray_t, whose name is specified by the mapping as "PointRange" or "PointList", respectively.

If the grid connectivity is one-to-one, the corresponding subregion is defined with a single child of type IndexArray_t, whose name is specified by the mapping as "PointListDonor". Otherwise, the corresponding subregion is defined by two child nodes, one defining the cells and the other the interpolation factors within the cells. See the SIDS for the complete description.

Node Attributes

Name:	User defined
Label:	GridConnectivity_t
DataType:	C1
Dimension:	1
Dimension Values:	Length of string
Data:	ZoneDonorName
Children:	See Figure 12
Cardinality:	0,*N*
Parameters:	IndexDimension, CellDimension
Functions:	PointListSize

6.2.3.5 GridConnectivityProperty_t

An optional GridConnectivityProperty_t node may be used to record special properties associated with particular connectivity patches.

Node Attributes

Name:	GridConnectivityProperty
Label:	GridConnectivityProperty_t
DataType:	MT
Children:	See Figure 13
Cardinality:	0,1

6.2.3.6 Periodic_t

A Periodic_t node may be used as a child of GridConnectivityProperty_t to record data associated with a periodic interface.

Node Attributes

Name:	Periodic
Label:	Periodic_t
DataType:	MT
Children:	See Figure 14

Cardinality: 0,1

6.2.3.7 `AverageInterface_t`

An `AverageInterface_t` node is used as a child of `GridConnectivityProperty_t` when data at the current connectivity interface is to be averaged in some way prior to passing it to a neighboring interface.

Node Attributes

Name: `AverageInterface`
Label: `AverageInterface_t`
DataType: `MT`
Children: See Figure 15
Cardinality: 0,1

6.2.3.8 `OversetHoles_t`

A node of type `OversetHoles_t` describes a region in a grid in which solution values are to be ignored because the data in the region is to be represented by values associated with other "overlapping" zones (equivalent to that specified by `IBLANK` = 0 in the PLOT3D format). Each `ZoneGridConnectivity_t` node may have as many `OversetHoles_t` children as are required to describe the affected region.

Each hole is described either by a single child of type `IndexArray_t` or by any number of children of type `IndexRange_t`. The latter is provided as a means of specifying holes which are unions of small numbers of logically rectangular subregions. However, if the region is irregular, the intent is that it should be specified by a single child of type `IndexArray_t` which lists the points.

Node Attributes

Name: User defined
Label: `OversetHoles_t`
DataType: `MT`
Children: See Figure 16
Cardinality: 0,N
Parameters: `IndexDimension`

6.2.3.9 `ZoneGridConnectivity_t`

Each `Zone_t` node may have at most one child of type `ZoneGridConnectivity`. It holds no data, but serves as the point below which all connectivity data associated with the zone can be found.

Node Attributes

Name: `ZoneGridConnectivity`
Label: `ZoneGridConnectivity_t`
DataType: `MT`
Children: See Figure 10
Cardinality: 0,1
Parameters: `IndexDimension`, `CellDimension`

6.2.4 Boundary Condition Group

Nodes in this group are used to specify the physical boundary conditions. Each boundary condition is associated with a subregion of a zone. For brevity below, we use the word "domain" to refer to the region on which a boundary condition is to be enforced.

The domain is usually, but not necessarily, a subregion of a face of the zone. The mapping is sufficiently general to permit the description of internal boundary conditions and boundary conditions which do not lie on a constant coordinate plane.

Mathematical boundary conditions are generally applied on subregions of physical dimension one less than the corresponding field problem. This condition, however, is neither defined nor enforced by the File Mapping.

A large number of standard boundary condition types are named by the SIDS. In addition, it is possible to define new types as collections of Dirichlet and Neumann conditions. It is not possible to describe the entire array of possibilities within this document, and the reader should consult the SIDS for a full description.

6.2.4.1 `InwardNormalIndex`

An `InwardNormalIndex` node is a node of type `int[IndexDimension]` which is identified by its Name. It applies to structured grids only, and its function is to specify on which side of the domain the condition is to be enforced.

`InwardNormalIndex` may have only one nonzero element, whose sign indicates the computational-coordinate direction of the boundary condition patch normal; this normal points into the interior of the zone. For example, if the domain lies on the face of a three-dimensional zone where the second index is a maximum, the inward normal index values are $[0, -1, 0]$.

The `InwardNormalIndex` node must apply to the entire domain of the boundary condition.

For a boundary condition on a face of a zone, the `InwardNormalIndex` can be calculated from other data and need not be specified. Its purpose is to define the normal direction for internal boundary conditions and other cases where the direction is ambiguous.

Node Attributes

Name:	`InwardNormalIndex`
Label:	"`int[IndexDimension]`"
DataType:	`I4`
Dimension:	1
Dimension Values:	`IndexDimension`
Data:	Index of inward normal
Children:	None
Cardinality:	0,1
Parameters:	`IndexDimension`

6.2.4.2 `InwardNormalList`

An `InwardNormalList` node is a node of type `IndexArray_t` identified by its Name. Its data field contains an array of physical (real) vectors which point into the region on which the boundary condition is to be applied. It may be used for boundary conditions on complex domains for which `InwardNormalIndex` is not defined, or to store vectors orthogonal to the domain of the boundary condition where these are not easily calculated from the domain itself.

Node Attributes

Name:	`InwardNormalList`
Label:	`IndexArray_t`
DataType:	`R4` or `R8`
Dimension:	2
Dimension Values:	`PhysicalDimension`, `ListLength`
Data:	Inward normal vectors
Children:	None
Cardinality:	0,1
Parameters:	`PhysicalDimension`, `ListLength`

6.2.4.3 `BCData_t`

When global or local Dirichlet or Neumann boundary conditions are defined, a node of type `BCData_t` is introduced to store the numerical data. For global data, this consists of a single quantity kept in a `DataArray_t` child. For local data, e.g., a pressure profile, it is a vector of quantities stored in an order corresponding to that defining the domain and kept in a child node of type `DataArray_t`.

Node Attributes

Name:	`DirichletData` or `NeumannData`
Label:	`BCData_t`
DataType:	`MT`
Children:	See Figure 20
Cardinality:	0,1
Parameters:	`ListLength`

6.2.4.4 `BCDataSet_t`

The function of a `BCDataSet_t` node is to specify the equations to be applied at the boundary, including any actual data values which may be required. The type of the equation is specified by the SIDS and recorded in the data field. For some types, the data is implicit or empty. For others, the data is specified in `BCData_t` children.

If the locations at which the boundary conditions are to be applied are specified in `BCDataSet_t`, using `PointRange` or `PointList`, the structure function `ListLength` is used. Otherwise, the structure parameter `ListLength` is required.

Node Attributes

Name:	User defined
Label:	`BCDataSet_t`
DataType:	`C1`
Dimension:	1
Dimension Values:	Length of string
Data:	`BCTypeSimple` value
Children:	See Figure 19
Cardinality:	0,N
Functions:	`ListLength`
Parameters:	`ListLength`

6.2.4.5 BC_t

A BC_t node specifies a single boundary condition to be applied on a single zone. It specifies the domain on which the condition is to be applied and the equations to be enforced. All the BC_t nodes for a single zone are found under that zone's ZoneBC_t node. A ZoneBC_t node may have as many BC_t children as are required to describe the physical boundary conditions on the corresponding zone.

The domain on which the boundary condition is to be enforced is specified by a single node of type either IndexRange_t or IndexArray_t. The equations are specified in one or more BCDataSet_t children.

The type of the boundary condition, which may be either simple or compound, is specified in the data field. For a complete description, it is necessary to consult the SIDS.

Node Attributes

Name:	User defined
Label:	BC_t
DataType:	C1
Dimension:	1
Dimension Values:	Length of string
Data:	BCType value
Children:	See Figure 18
Cardinality:	0,*N*
Parameters:	IndexDimension, PhysicalDimension

6.2.4.6 ZoneBC_t

The ZoneBC_t node occurs at most once for each zone and serves as the location under which all boundary conditions on that zone are collected.

Node Attributes

Name:	ZoneBC
Label:	ZoneBC_t
DataType:	MT
Children:	See Figure 17
Cardinality:	0,1
Parameters:	IndexDimension, PhysicalDimension

6.2.4.7 BCProperty_t

An optional BCProperty_t node may be used to record special properties associated with particular boundary condition patches.

Node Attributes

Name:	BCProperty
Label:	BCProperty_t
DataType:	MT
Children:	See Figure 21
Cardinality:	0,1

6.2.4.8 WallFunction_t

A WallFunction_t node may be used as a child of BCProperty_t to record data associated with the use of wall function boundary conditions.

Node Attributes

Name:	WallFunction
Label:	WallFunction_t
DataType:	MT
Children:	See Figure 22
Cardinality:	0,1

6.2.4.9 Area_t

An Area_t node may be used as a child of BCProperty_t to record data associated with area-related boundary conditions such as bleed.

Node Attributes

Name:	Area
Label:	Area_t
DataType:	MT
Children:	See Figure 23
Cardinality:	0,1

6.2.5 Equation Specification Group

Nodes in this group serve to identify the physical model associated with the data being recorded. Nearly always, the data is of enumeration type and is selected from a collection of terms defined in detail in the SIDS. The names are largely self explanatory, and the detailed definitions will not be repeated here. Numerical values associated with the physical model depend on the type of modeling being chosen and are generally stored in child nodes of type DataArray_t.

6.2.5.1 GoverningEquations_t

This node names the equation set being solved, for example, FullPotential or NSTurbulent. If Navier-Stokes, the diffusion terms retained may be specified in a DiffusionModel subnode.

Node Attributes

Name:	GoverningEquations
Label:	GoverningEquations_t
DataType:	C1
Dimension:	1
Dimension Values:	Length of string
Data:	GoverningEquationsType value
Children:	See Figure 25
Cardinality:	0,1
Parameters:	CellDimension

6.2.5.2 GasModel_t

A node of type GasModel_t names the gas model used, for example, Ideal or VanderWaals.

Node Attributes

Name:	`GasModel`
Label:	`GasModel_t`
DataType:	`C1`
Dimension:	1
Dimension Values:	Length of string
Data:	`GasModelType` value
Children:	See Figure 26
Cardinality:	0,1

6.2.5.3 `ViscosityModel_t`

A node of type `ViscosityModel_t` names the molecular viscosity model used to relate the viscosity to the temperature, for example, `PowerLaw` or `SutherlandLaw`.

Node Attributes

Name:	`ViscosityModel`
Label:	`ViscosityModel_t`
DataType:	`C1`
Dimension:	1
Dimension Values:	Length of string
Data:	`ViscosityModelType` value
Children:	See Figure 27
Cardinality:	0,1

6.2.5.4 `EquationDimension`

A node named `EquationDimension`, of type `int[]`, gives the number of dependent variables required for a complete solution description, or the number of equations being solved. For example, for `NSTurbulent` with the k-ϵ turbulence model in three dimensions, it is 7.

Node Attributes

Name:	`EquationDimension`
Label:	"`int`"
DataType:	`I4`
Dimension:	1
Dimension Values:	1
Data:	`EquationDimension` value
Children:	None
Cardinality:	0,1

6.2.5.5 `ThermalConductivityModel_t`

A node of type `ThermalConductivityModel_t` names the model used to relate the thermal conductivity to the temperature, for example, `ConstantPrandtl`, `PowerLaw`, or `SutherlandLaw`. These closely parallel the viscosity model.

Node Attributes

Name:	`ThermalConductivityModel`
Label:	`ThermalConductivityModel_t`
DataType:	`C1`

Dimension: 1
Dimension Values: Length of string
Data: `ThermalConductivityModelType` value
Children: See Figure 28
Cardinality: 0,1

6.2.5.6 `TurbulenceClosure_t`

A node of type `TurbulenceClosure_t` names the method of closing the Reynolds stress equations when the governing equations are turbulent, for example, `EddyViscosity` or `ReynoldsStressAlgebraic`.

Node Attributes

Name: `TurbulenceClosure`
Label: `TurbulenceClosure_t`
DataType: `C1`
Dimension: 1
Dimension Values: Length of string
Data: `TurbulenceClosureType` value
Children: See Figure 29
Cardinality: 0,1

6.2.5.7 `TurbulenceModel_t`

A node of type `TurbulenceModel_t` names the equation set used to model the turbulence quantities, for example, `Algebraic_BaldwinLomax` or `OneEquation_SpalartAllmaras`.

Node Attributes

Name: `TurbulenceModel`
Label: `TurbulenceModel_t`
DataType: `C1`
Dimension: 1
Dimension Values: Length of string
Data: `TurbulenceModelType` value
Children: See Figure 30
Cardinality: 0,1
Parameters: `CellDimension`

6.2.5.8 `ThermalRelaxationModel_t`

A node of type `ThermalRelaxationModel_t` names the equation set used to model the thermal relaxation quantities, for example, `Frozen` or `ThermalEquilib`.

Node Attributes

Name: `ThermalRelaxationModel`
Label: `ThermalRelaxationModel_t`
DataType: `C1`
Dimension: 1
Dimension Values: Length of string
Data: `ThermalRelaxationModelType` value
Children: See Figure 31
Cardinality: 0,1

6.2.5.9 ChemicalKineticsModel_t

A node of type `ChemicalKineticsModel_t` names the equation set used to model the chemical kinetics quantities, for example, `Frozen` or `ChemicalEquilibCurveFit`.

Node Attributes

Name:	`ChemicalKineticsModel`
Label:	`ChemicalKineticsModel_t`
DataType:	`C1`
Dimension:	1
Dimension Values:	Length of string
Data:	`ChemicalKineticsModelType` value
Children:	See Figure 32
Cardinality:	0,1

6.2.5.10 EMElectricFieldModel_t

A node of type `EMElectricFieldModel_t` names the electric field model used for electromagnetic flows, for example, `Constant` or `Voltage`.

Node Attributes

Name:	`EMElectricFieldModel`
Label:	`EMElectricFieldModel_t`
DataType:	`C1`
Dimension:	1
Dimension Values:	Length of string
Data:	`EMElectricFieldModelType` value
Children:	See Figure 33
Cardinality:	0,1

6.2.5.11 EMMagneticFieldModel_t

A node of type `EMMagneticFieldModel_t` names the magnetic field model used for electromagnetic flows, for example, `Constant` or `Interpolated`.

Node Attributes

Name:	`EMMagneticFieldModel`
Label:	`EMMagneticFieldModel_t`
DataType:	`C1`
Dimension:	1
Dimension Values:	Length of string
Data:	`EMMagneticFieldModelType` value
Children:	See Figure 34
Cardinality:	0,1

6.2.5.12 EMConductivityModel_t

A node of type `EMConductivityModel_t` names the conductivity model used for electromagnetic flows, for example, `Constant` or `Equilibrium_LinRessler`.

Node Attributes

Name:	`EMConductivityModel`

Label:	`EMConductivityModel_t`
DataType:	`C1`
Dimension:	1
Dimension Values:	Length of string
Data:	`EMConductivityModelType` value
Children:	See Figure 35
Cardinality:	0,1

6.2.5.13 `FlowEquationSet_t`

A node of type `FlowEquationSet_t` appears either at the highest level of the tree (under `CGNSBase_t`), to indicate the equation set whose solution is recorded throughout the database, or below a `Zone_t` node, to indicate the set of equations solved in that zone. The usual convention applies, i.e., specifications at the local (zone) level override global specifications.

Node Attributes

Name:	`FlowEquationSet`
Label:	`FlowEquationSet_t`
DataType:	`MT`
Children:	See Figure 24
Cardinality:	0,1
Parameters:	`CellDimension`

6.2.6 Family Group

Because there is rarely a 1-to-1 connection between mesh regions and geometric entities, it is often desirable to set geometric associations indirectly in a CGNS file. That is, rather than setting the geometry data for each mesh entity (nodes, edges, and faces), it's useful to associate them with intermediate objects. The intermediate objects are in turn linked to nodal regions of the computational mesh. This intermediate object is defined as a *CFD family*.

Each mesh surface may linked to the geometric entities of one or more CAD databases by a user-defined CFD family name. The CFD family corresponds to one or more CAD geometric entities on which the mesh face is projected. Each one of these geometric entities is described in a CAD file and is not redefined within the CGNS file.

6.2.6.1 `Family_t`

This node, a child of the `CGNSBase_t` node, contains the definition of a single CFD family. Multiple `Family_t` nodes are allowed.

Node Attributes

Name:	User defined
Label:	`Family_t`
DataType:	`MT`
Children:	See Figure 42
Cardinality:	0,*N*

6.2.6.2 `FamilyName_t`

This node is used to identify a family to which a particular zone or boundary belongs. Note that the name of the family is defined by the "Name" of the `Family_t` node, and is stored as data in the

`FamilyName_t` node.

Node Attributes

Name:	`FamilyName`
Label:	`FamilyName_t`
DataType:	`C1`
Dimension:	1
Dimension Values:	Length of string
Data:	Name of CFD family
Children:	None
Cardinality:	0,1

6.2.6.3 `FamilyBC_t`

This node contains a boundary condition type for a particular CFD family.

Node Attributes

Name:	`FamilyBC`
Label:	`FamilyBC_t`
DataType:	`C1`
Dimension:	1
Dimension Values:	Length of string
Data:	`BCType` value
Children:	See Figure 43
Cardinality:	0,1

6.2.6.4 `GeometryReference_t`

`GeometryReference_t` nodes are used to associate a CFD family with one or more CAD databases.

Node Attributes

Name:	User defined
Label:	`GeometryReference_t`
DataType:	`MT`
Children:	See Figure 44
Cardinality:	0,N

6.2.6.5 `GeometryFile_t`

This node contains the name of the CAD geometry file.

Node Attributes

Name:	`GeometryFile`
Label:	`GeometryFile_t`
DataType:	`C1`
Dimension:	1
Dimension Values:	Length of string
Data:	Name of the CAD geometry file
Children:	None
Cardinality:	1

6.2.6.6 GeometryFormat_t

This enumeration node defines the format of the CAD geometry file.

Node Attributes

Name:	GeometryFormat
Label:	GeometryFormat_t
DataType:	C1
Dimension:	1
Dimension Values:	Length of string
Data:	Name of the CAD geometry format
Children:	None
Cardinality:	1

6.2.6.7 GeometryEntity_t

GeometryEntity_t nodes define the names of the entities in CAD geometry file that make up a CFD family.

Node Attributes

Name:	User defined
Label:	GeometryEntity_t
DataType:	MT
Children:	None
Cardinality:	0,N

6.2.7 Time-Dependent Group

Nodes in this section are used for information related to time-dependent flows, and include specification of grid motion and storage of time-dependent or iterative data.

6.2.7.1 BaseIterativeData_t

Located directly under the CGNSBase_t node, the BaseIterativeData_t node contains information about the number of time steps or iterations being recorded, and the time and/or iteration values at each step. In addition, it may include the list of zones and families for each step of the simulation, if these vary throughout the simulation.

Node Attributes

Name:	User defined
Label:	BaseIterativeData_t
DataType:	I4
Dimension:	1
Dimension Values:	1
Data:	NumberOfSteps
Children:	See Figure 45
Cardinality:	0,1

6.2.7.2 ZoneIterativeData_t

The ZoneIterativeData_t node is a child of the Zone_t node, and is used to store pointers to zonal data for each recorded step of the simulation.

Node Attributes

Name:	User defined
Label:	`ZoneIterativeData_t`
DataType:	`MT`
Children:	See Figure 46
Cardinality:	0,1
Parameters:	`NumberOfSteps`

6.2.7.3 `RigidGridMotion_t`

`RigidGridMotion_t` nodes are used to store data defining rigid translation and/or rotation of the grid coordinates. Multiple `RigidGridMotion_t` nodes may be associated with different iterations or time steps in the computation. This association is recorded under the `ZoneIterativeData_t` node.

Node Attributes

Name:	User defined
Label:	`RigidGridMotion_t`
DataType:	`C1`
Dimension:	1
Dimension Values:	Length of string
Data:	`RigidGridMotionType` value
Children:	See Figure 47
Cardinality:	0,*N*

6.2.7.4 `ArbitraryGridMotion_t`

`ArbitraryGridMotion_t` nodes are used to store grid velocities for each grid point in a zone (i.e., for deforming grids). Multiple `ArbitraryGridMotion_t` nodes may be associated with different iterations or time steps in the computation. This association is recorded under the `ZoneIterativeData_t` node.

Note that instantaneous grid coordinates at different iterations or time steps may be recorded using multiple `GridCoordinates_t` nodes under a `Zone_t` node.

Node Attributes

Name:	User defined
Label:	`ArbitraryGridMotion_t`
DataType:	`C1`
Dimension:	1
Dimension Values:	Length of string
Data:	`ArbitraryGridMotionType` value
Children:	See Figure 48
Cardinality:	0,*N*
Parameters:	`IndexDimension`, `VertexSize`, `CellSize`
Functions:	`DataSize`

6.2.8 Structural Nodes

In this section we describe the highest levels of the hierarchy. Nodes in this section store only quantities which refer to all the entities below them. Therir primary function is to provide organization to the data below.

6.2.8.1 Zone_t

Directly below the highest level node in the database, which is by definition of type CGNSBase_t, are found nodes of type Zone_t providing entry into the data specific to each zone. There are as many Zone_t nodes as there are zones. Their children, in turn, record grid, field, connectivity, and boundary conditions, and a variety of auxiliary data.

Node Attributes

Name:	User defined
Label:	Zone_t
DataType:	I4
Dimension:	2
Dimension Values:	IndexDimension, 3
Data:	VertexSize[IndexDimension], CellSize[IndexDimension], VertexSizeBoundary[IndexDimension]
Children:	See Figure 4
Cardinality:	0,*N*
Parameters:	CellDimension, PhysicalDimension

6.2.8.2 CGNSBase_t

The CGNSBase_t node is by definition the highest level node in the database, and is located directly below the ADF root node. It provides entry into all other data. Multiple CGNSBase_t nodes are allowed in an ADF file. The particular database being accessed is determined by the name of the CGNSBase_t node.

The only data stored in the node itself are CellDimension, the dimensionality of a cell in the mesh (i.e., 3 for a volume cell and 2 for a face cell), and PhysicalDimension, the number of indices required to specify a unique physical location in the field data being recorded. However, a variety of global information concerning the entire database may be stored in children of the CGNSBase_t node. In particular, a Descriptor_t node at this level can store user commentary on the entire history of the development of the database.

Other information typically stored directly below the CGNSBase_t node includes convergence histories, reference states, dimensional units, integrated quantities, and information on the flow equations being solved.

Node Attributes

Name:	User defined
Label:	CGNSBase_t
DataType:	I4
Dimension:	1
Dimension Values:	2
Data:	CellDimension, PhysicalDimension
Children:	See Figure 3
Cardinality:	0,*N*

6.2.8.3 SimulationType_t

This enumeration-type node is a child of the CGNSBase_t node, and specifies whether or not the data below CGNSBase_t is time-accurate. Nodes for describing time-dependent data are presented in Section 6.2.7.

Node Attributes

Name:	`SimulationType`
Label:	`SimulationType_t`
DataType:	`C1`
Dimension:	1
Dimension Values:	Length of string
Data:	`TimeAccurate` or `NonTimeAccurate`
Children:	None
Cardinality:	0,1

6.2.8.4 `ZoneType_t`

This enumeration-type node is a required child of the `Zone_t` node, and specifies whether the grid in that zone is structured or unstructured.

Node Attributes

Name:	`ZoneType`
Label:	`ZoneType_t`
DataType:	`C1`
Dimension:	1
Dimension Values:	Length of string
Data:	`Structured` or `Unstructured`
Children:	None
Cardinality:	1

6.2.8.5 `CGNSLibraryVersion_t`

An ADF file containing a CGNS database also contains, directly below the ADF root node, a `CGNSLibraryVersion_t` node. This node contains the version number of the CGNS standard with which the file is consistent, and is created automatically when the file is created or modified using the CGNS Mid-Level Library. Note that this node is not actually part of the CGNS database, since it is not located below a `CGNSBase_t` node.

Note that an ADF file may contain multiple CGNS databases, but there is only one `CGNSLibraryVersion_t` node. It is assumed that the version number in the `CGNSLibraryVersion_t` node is applicable to all the CGNS databases in the file.

Note also that some CGNS nodes may actually be links to CGNS nodes in other files. In this case, it is assumed that the `CGNSLibraryVersion_t` node in the "top-level" file is applicable to the file(s) containing the linked nodes.

Node Attributes

Name:	`CGNSLibraryVersion`
Label:	`CGNSLibraryVersion_t`
DataType:	`R4`
Dimension:	1
Dimension Values:	1
Data:	CGNS version number
Children:	None
Cardinality:	1

A CGNS File Mapping Figures

This Appendix contains figures illustrating the nodal structure of a CGNS database. An example ADF file hierarchy is shown in Figure 1, but without any internal detail of the individual nodes. Just below the ADF root node is a `CGNSLibraryVersion_t` node, and one or more `CGNSBase_t` nodes. Each `CGNSBase_t` node represents the root node of a CGNS database.

Figure 2 shows the layout of succeeding figures. At the top of each page is a single parent node, and below it are boxes containing detailed descriptions of the nodes which the mapping specifies as children. Also within the parent's box are references to the figures corresponding to its parents in turn.

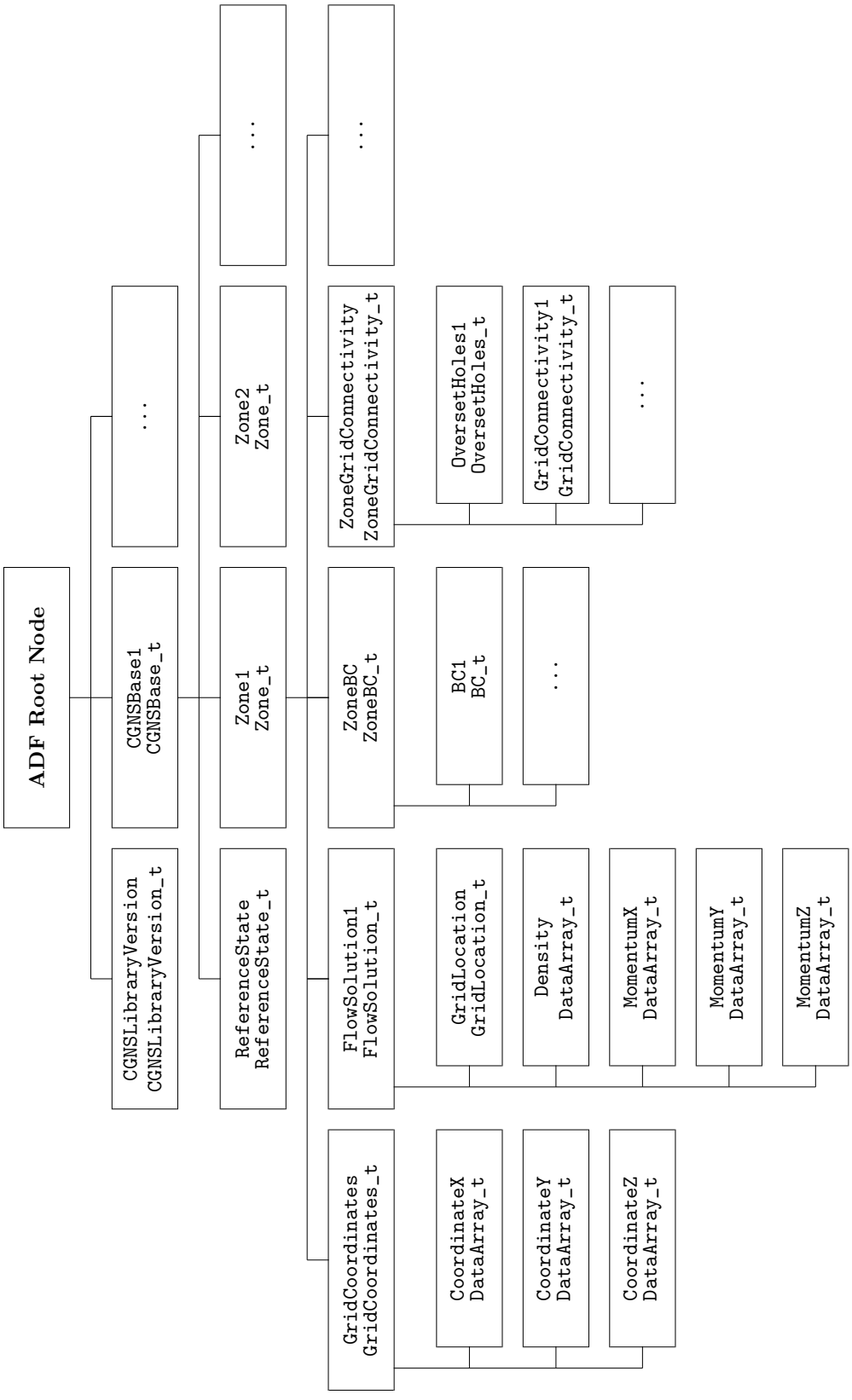


Figure 1: Example Hierarchy

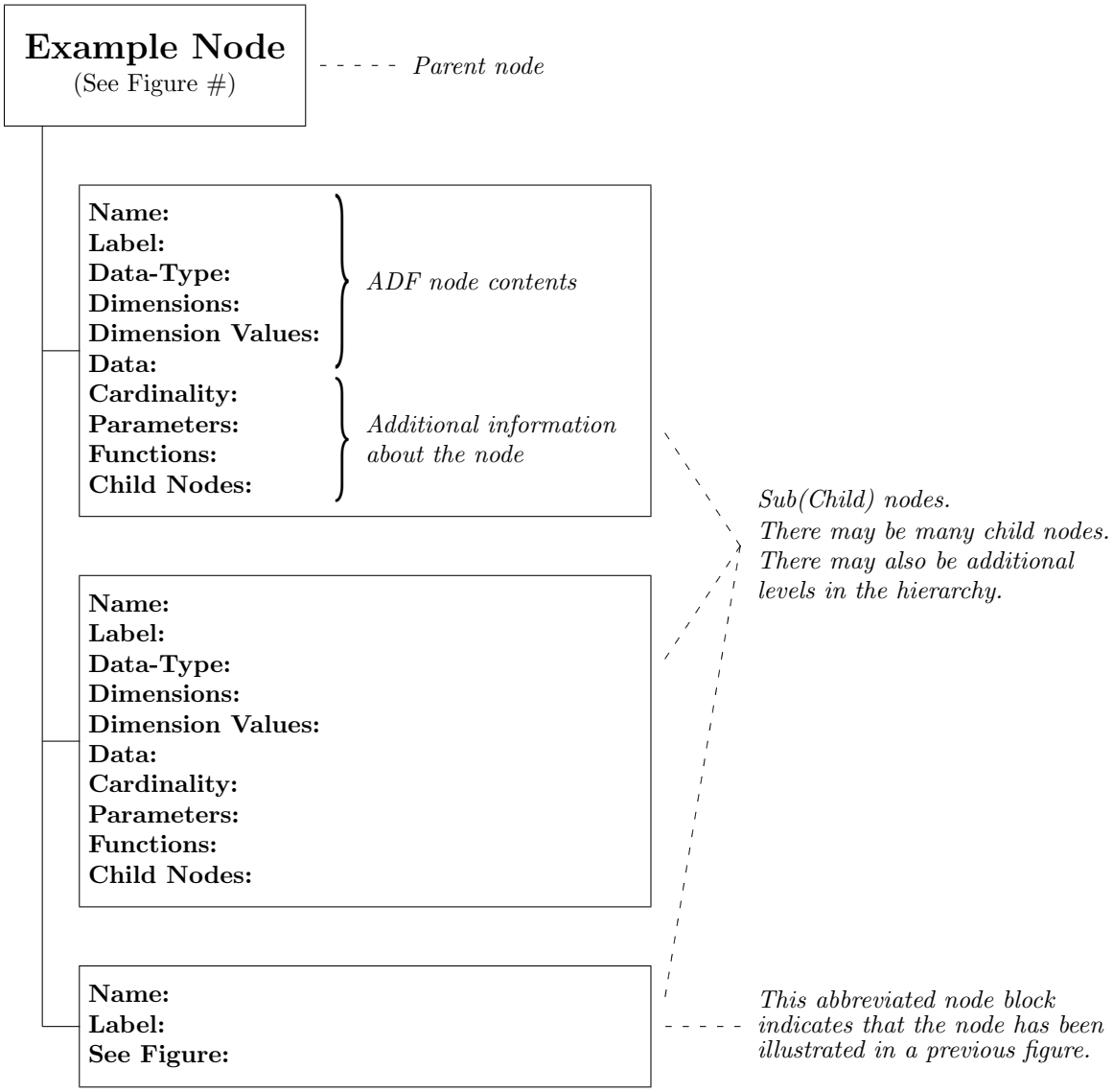


Figure 2: Example Node Structure

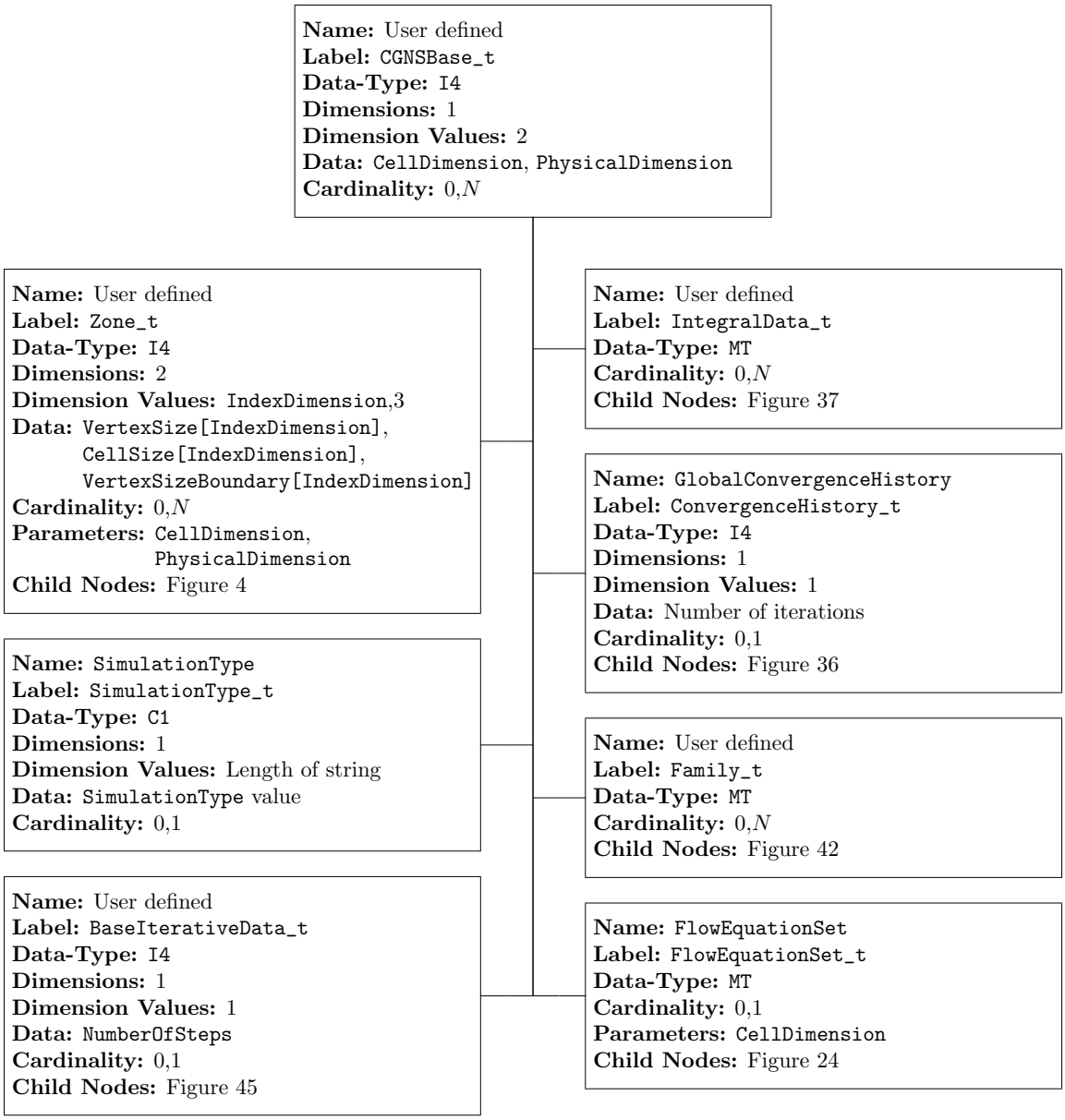


Figure 3: `CGNSBase_t` Data Structure (*Continued on next page*)

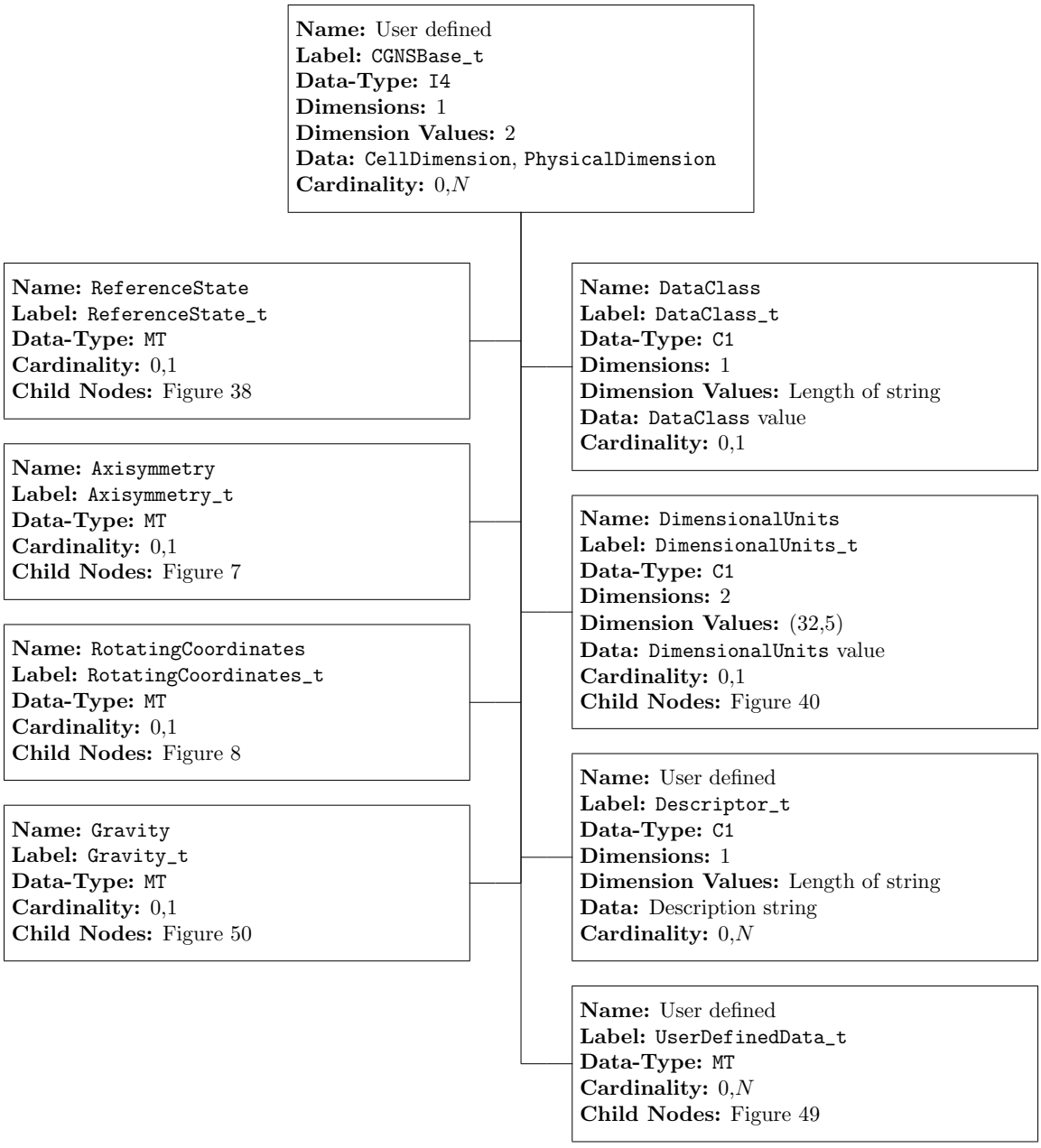


Figure 3: CGNSBase_t Data Structure (*Continued from previous page*)

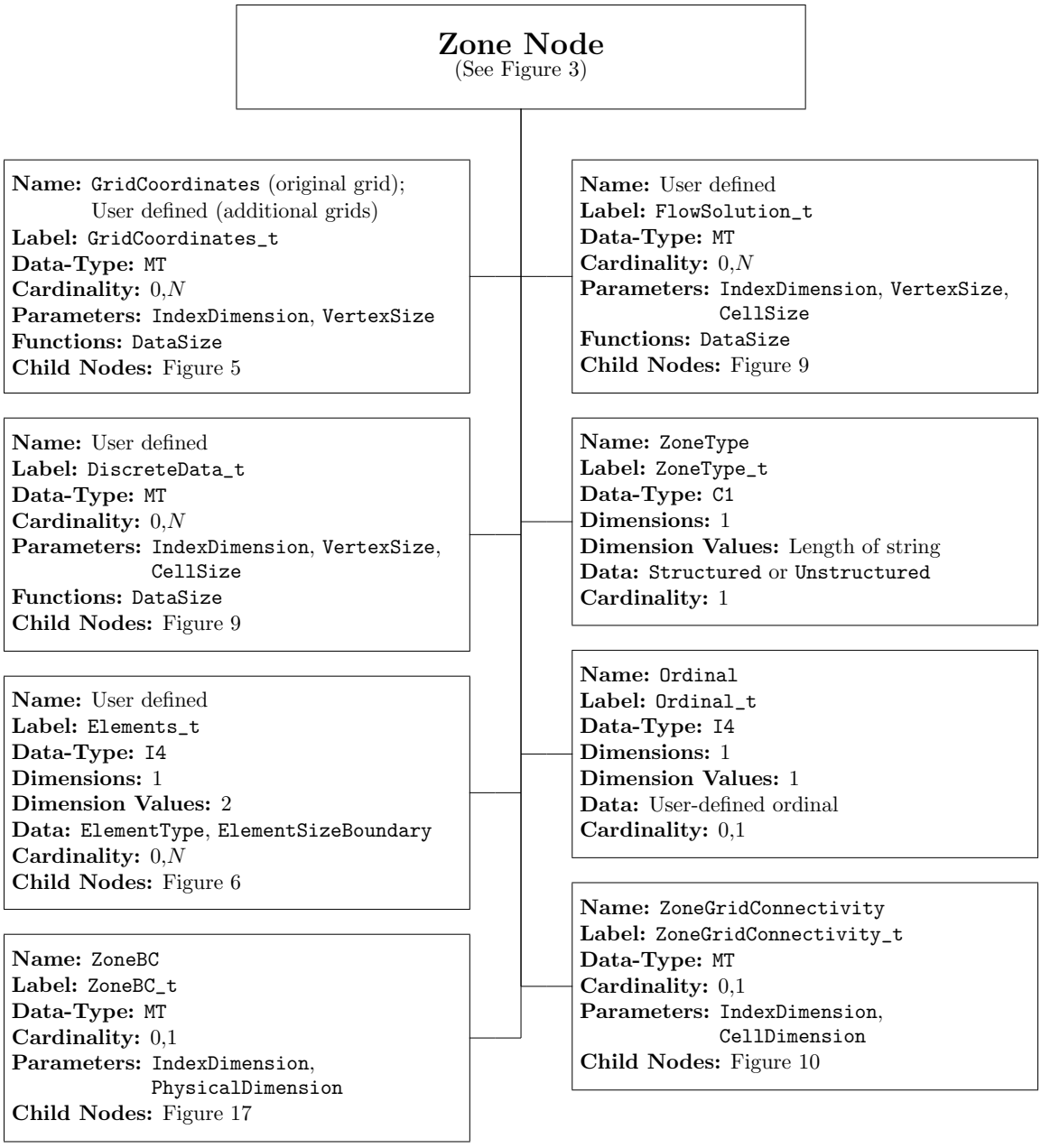


Figure 4: `Zone_t` Data Structure (*Continued on next page*)

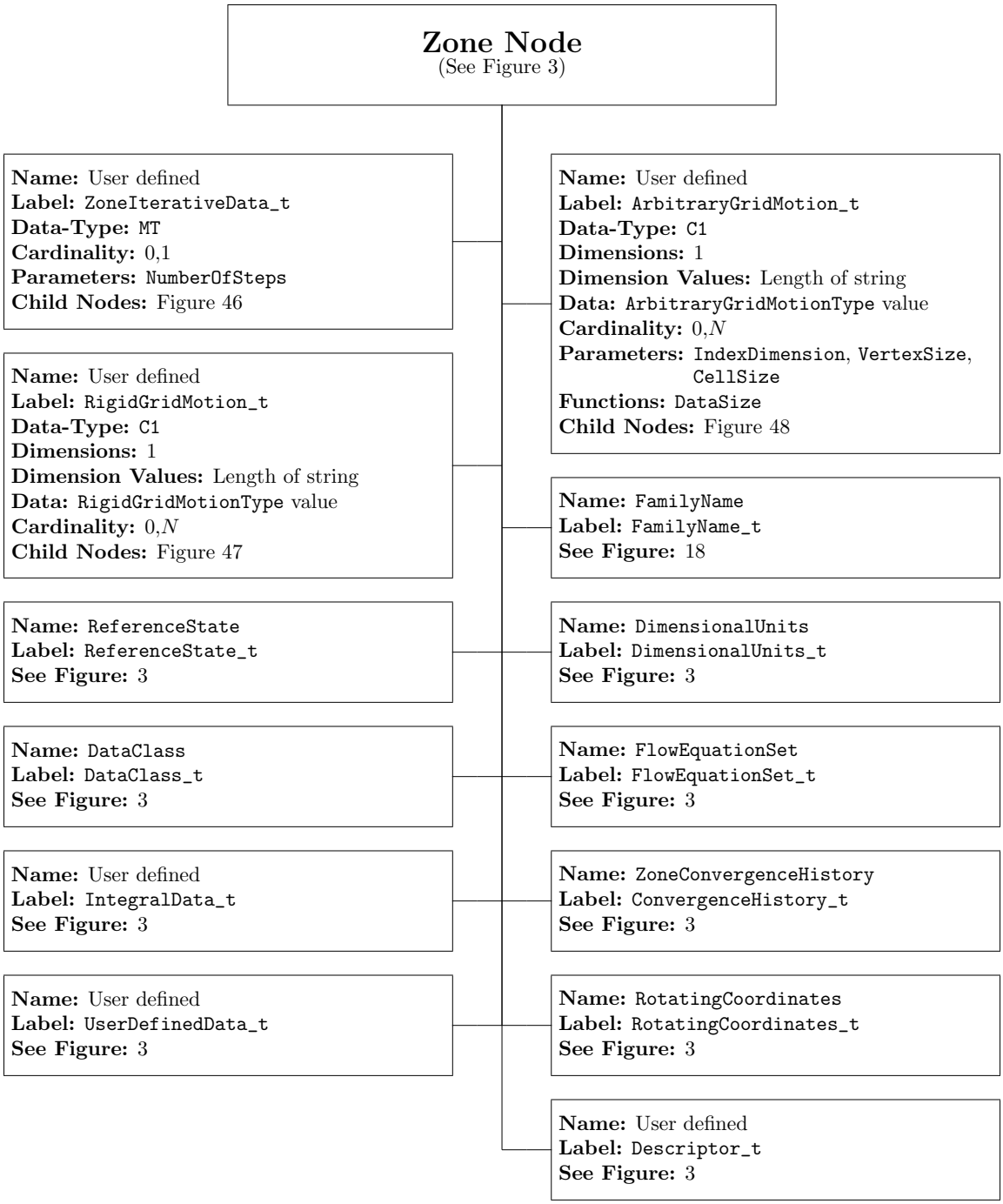
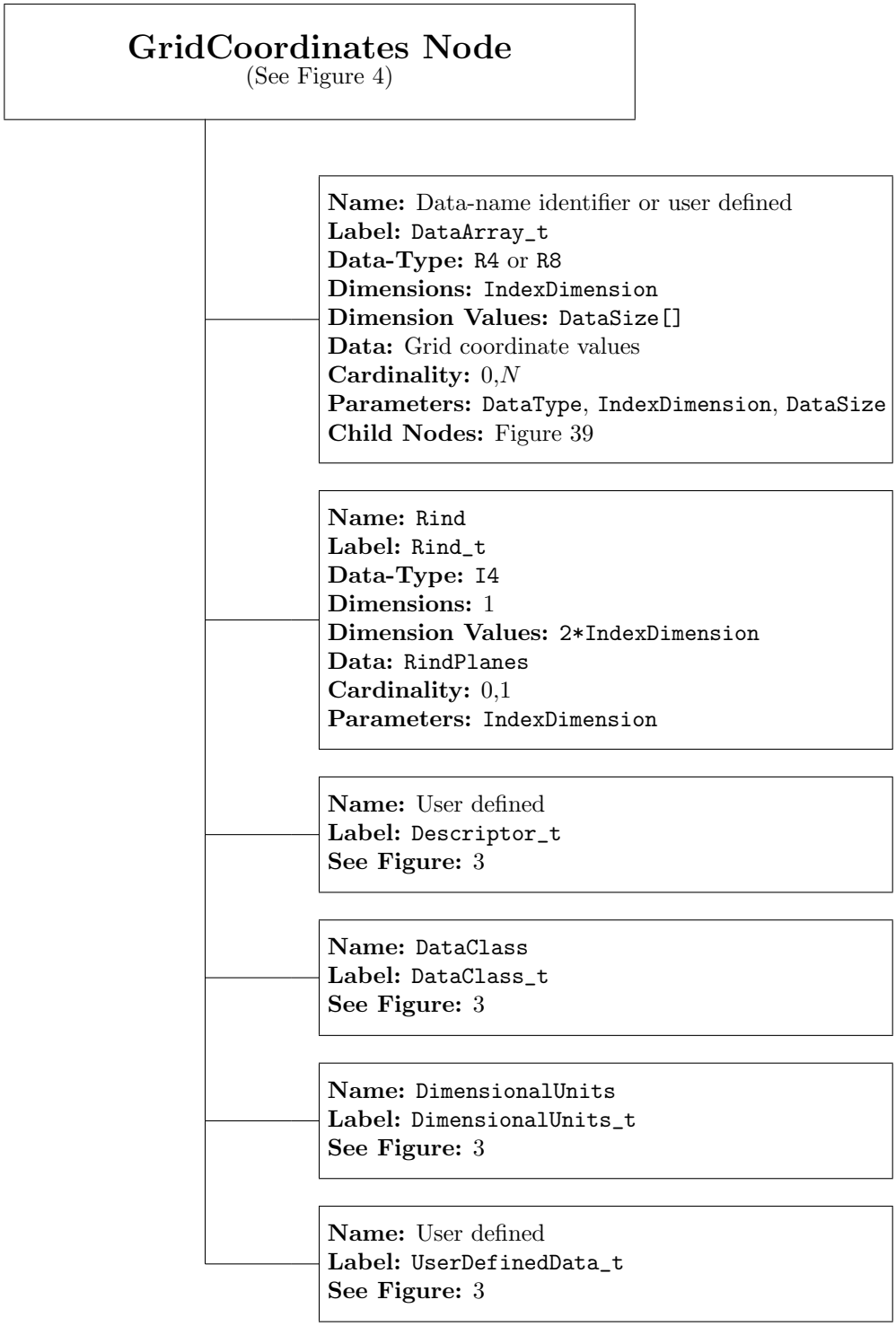


Figure 4: `Zone_t` Data Structure (*Continued from previous page*)



Note:

• The data labeled `RindPlanes` under `Rind_t` refers to number of rind planes for structured grids, and number of rind points for unstructured grids.

Figure 5: `GridCoordinates_t` Data Structure

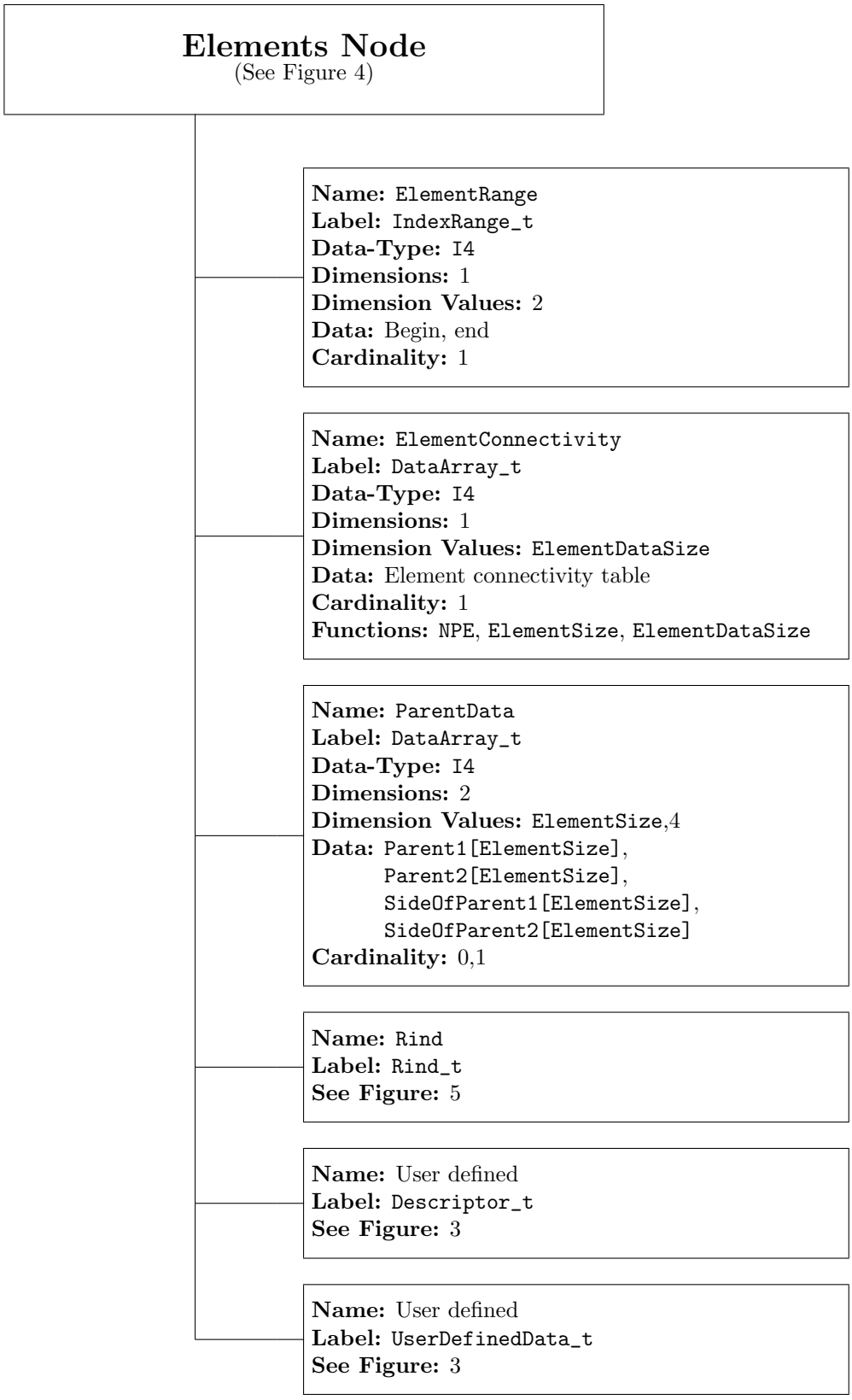


Figure 6: `Elements_t` Data Structure

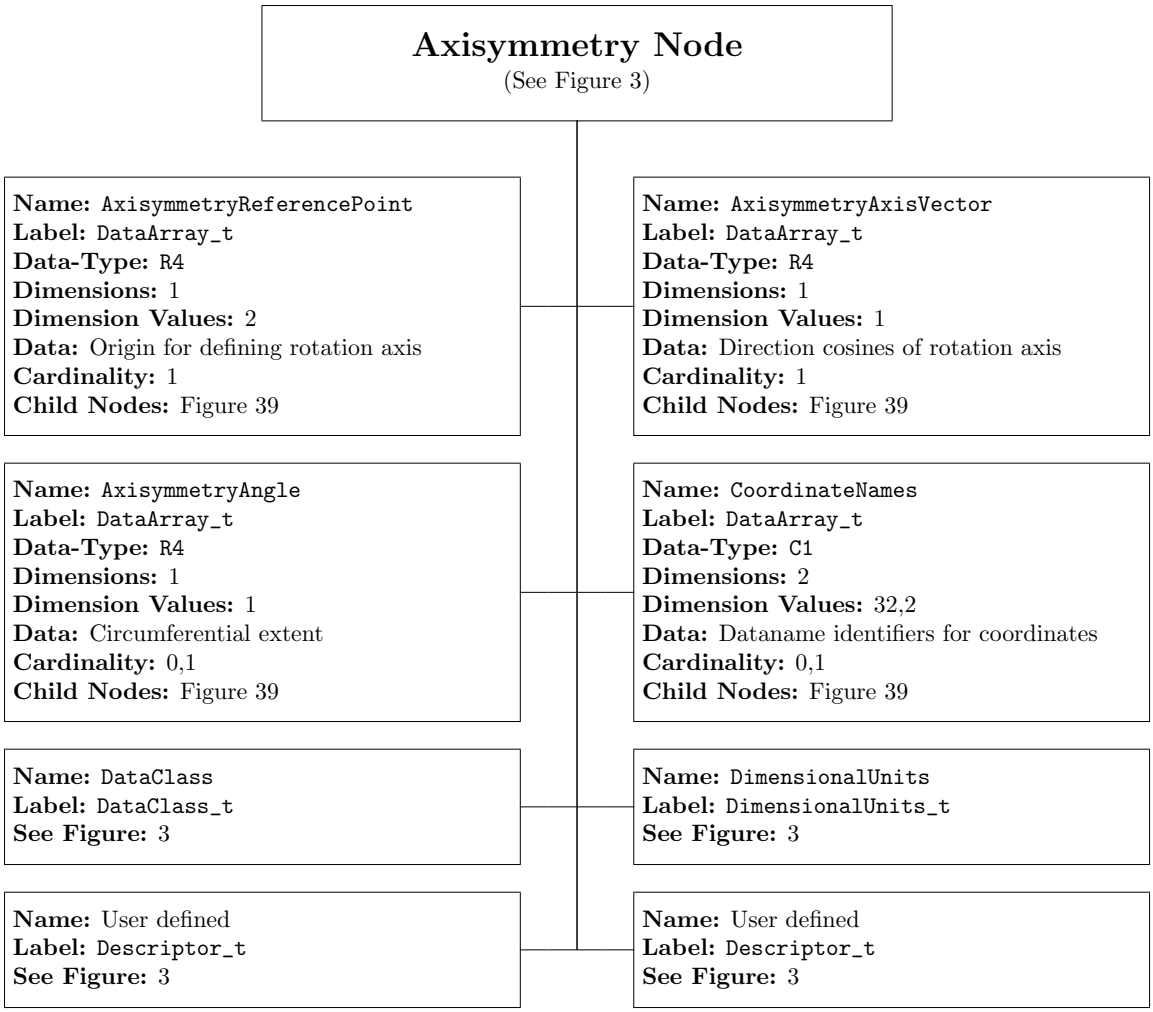


Figure 7: `Axisymmetry_t` Data Structure

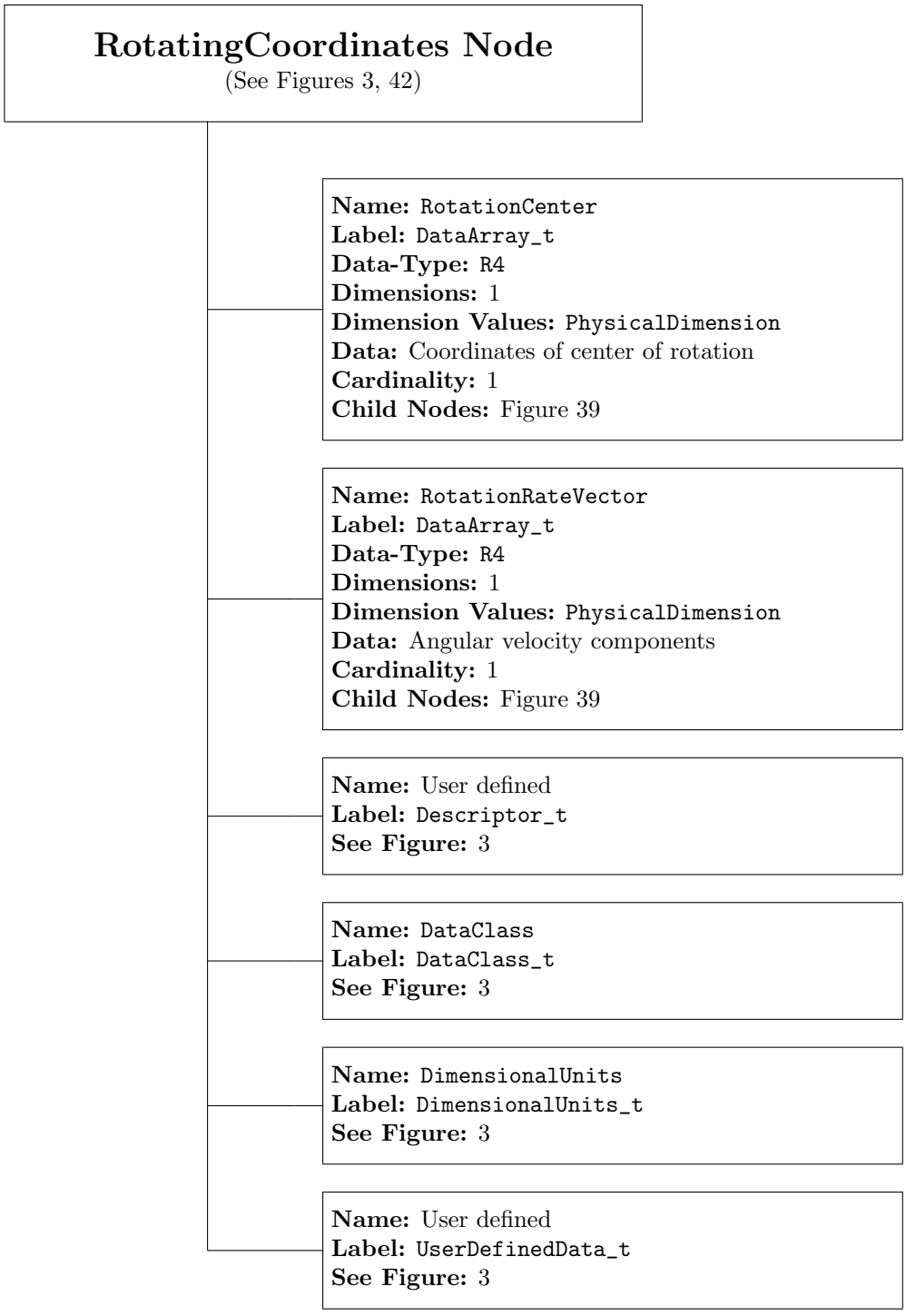


Figure 8: `RotatingCoordinates_t` Data Structure

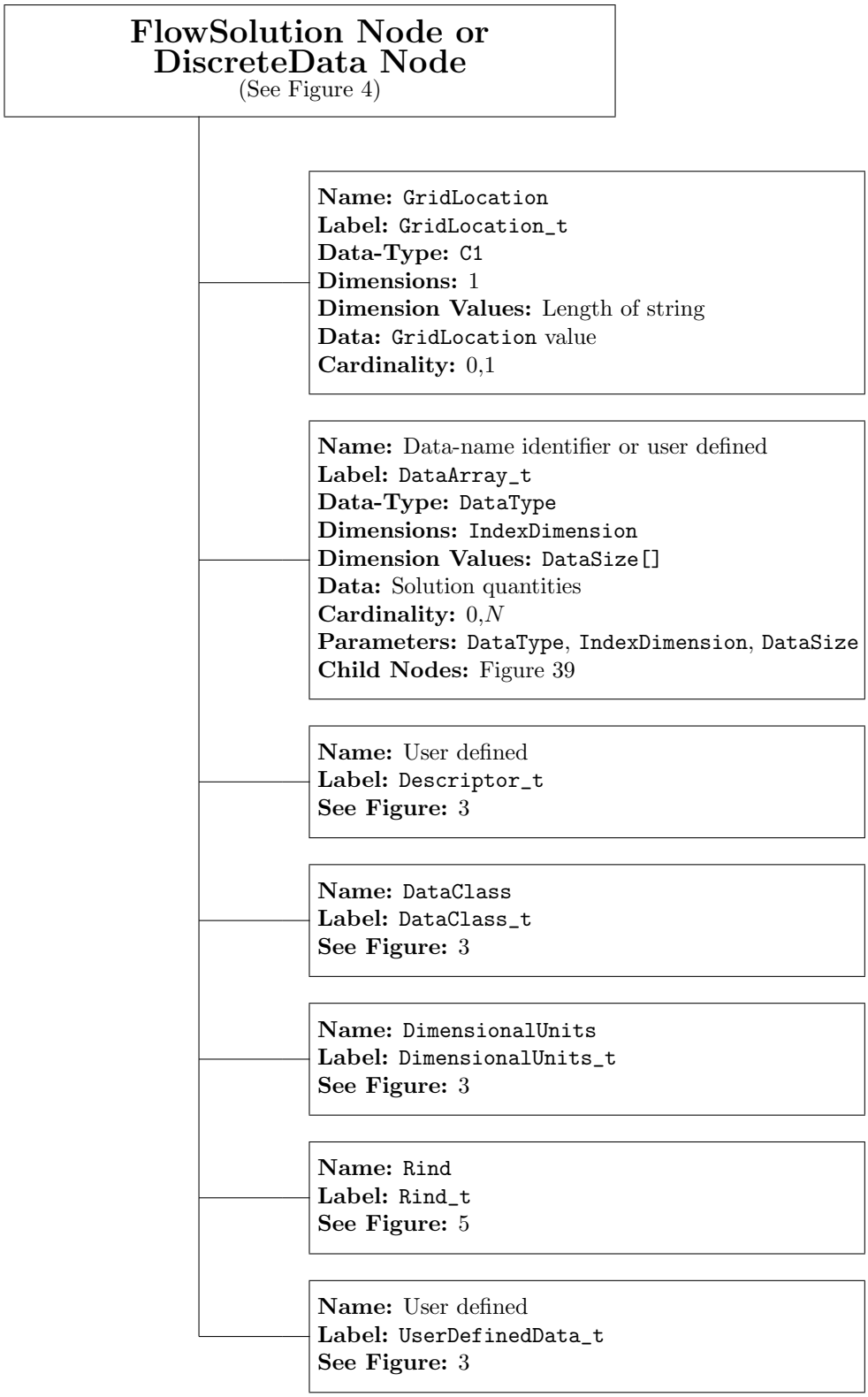
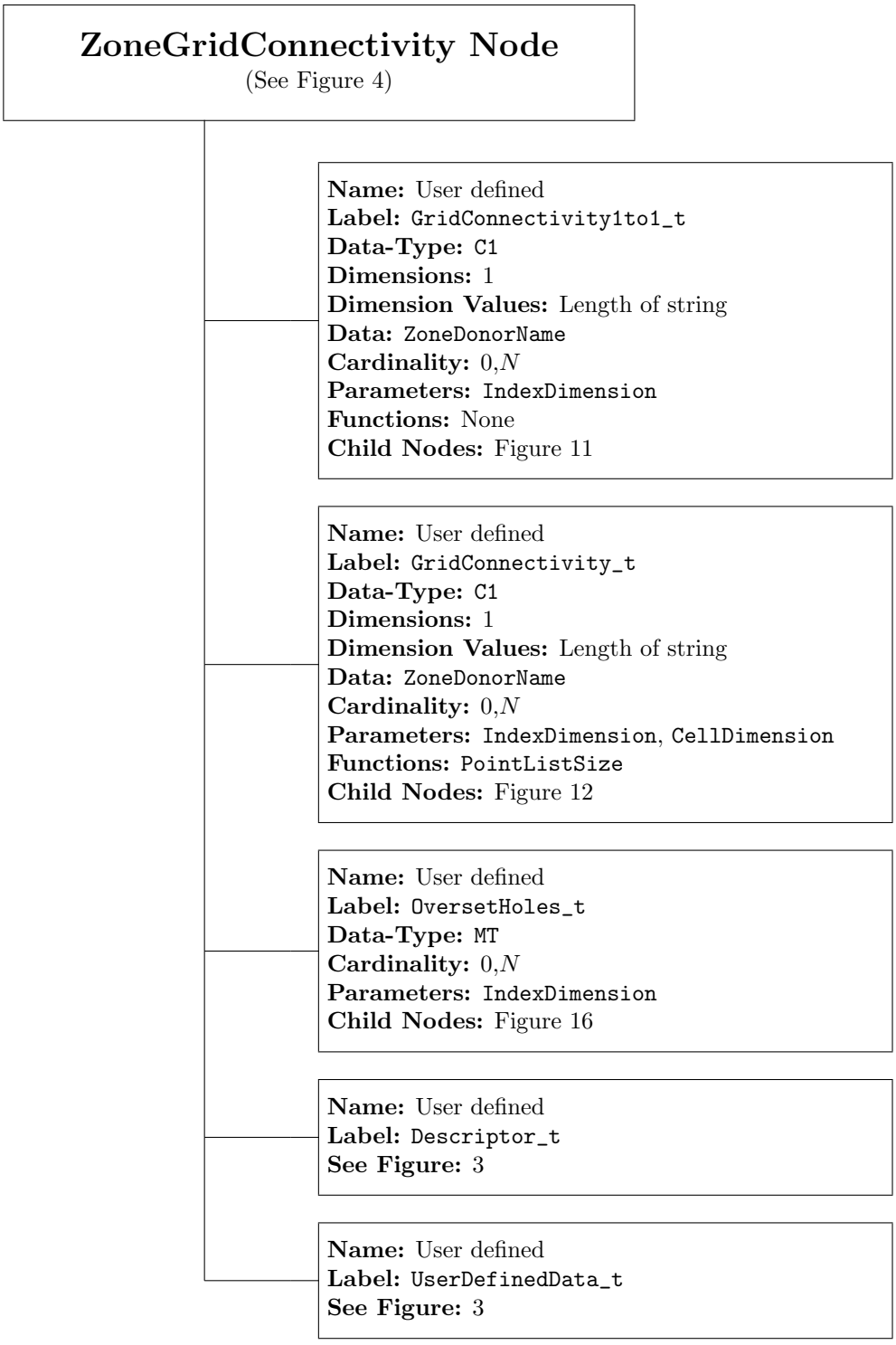


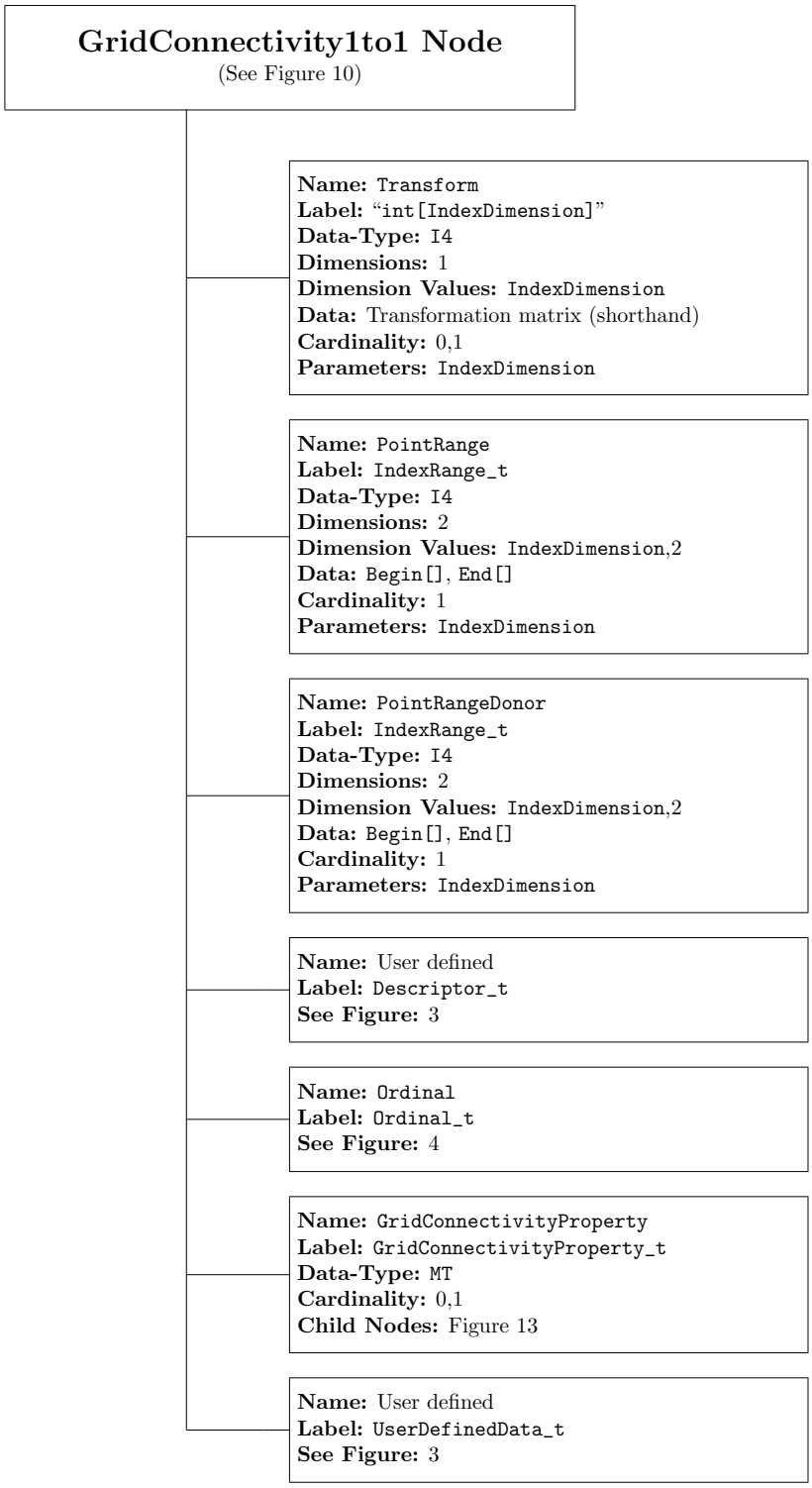
Figure 9: `FlowSolution_t` and `DiscreteData_t` Data Structures



Note:

- `GridConnectivity1to1` is only applicable to structured-to-structured 1-to-1 mesh connectivity.

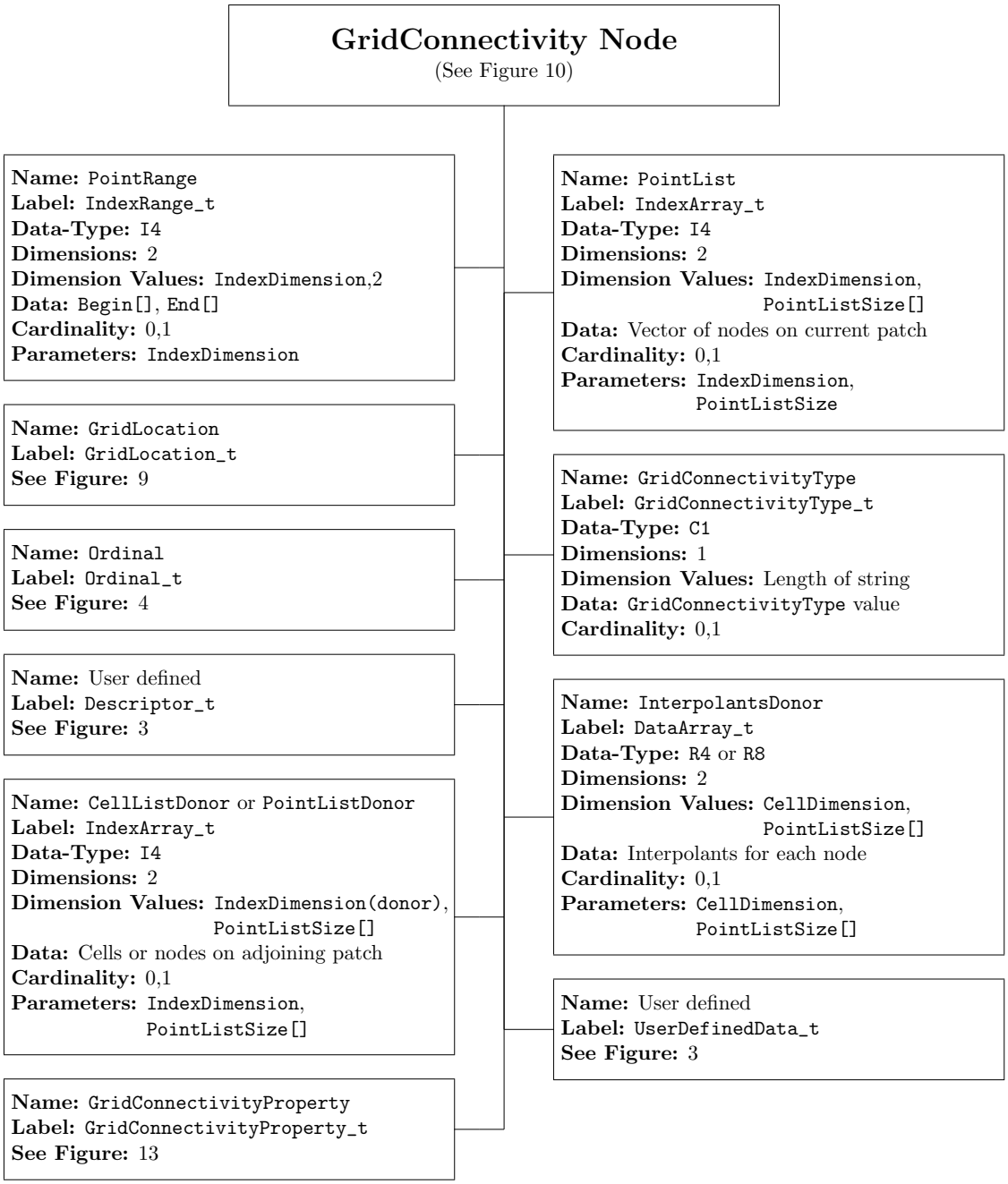
Figure 10: `ZoneGridConnectivity_t` Data Structure



Note:

- `GridConnectivity1to1` is only applicable to structured-to-structured 1-to-1 mesh connectivity.

Figure 11: `GridConnectivity1to1_t` Data Structure



Notes:

- The nodes `PointRange` and `PointList` are mutually exclusive.
- For mismatched interfaces, the combination of `CellListDonor` and `InterpolantsDonor` is used to define the position of each receiver point in the donor zone.

Figure 12: `GridConnectivity_t` Data Structure

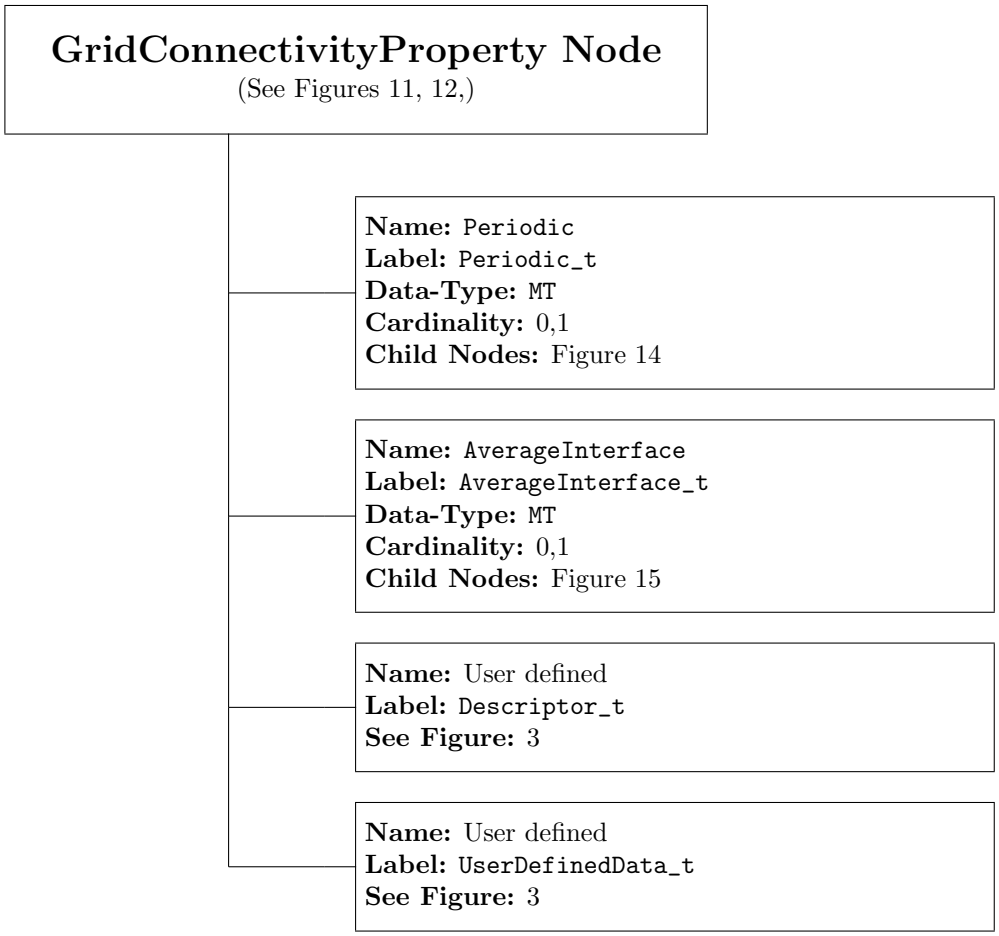


Figure 13: `GridConnectivityProperty_t` Data Structure

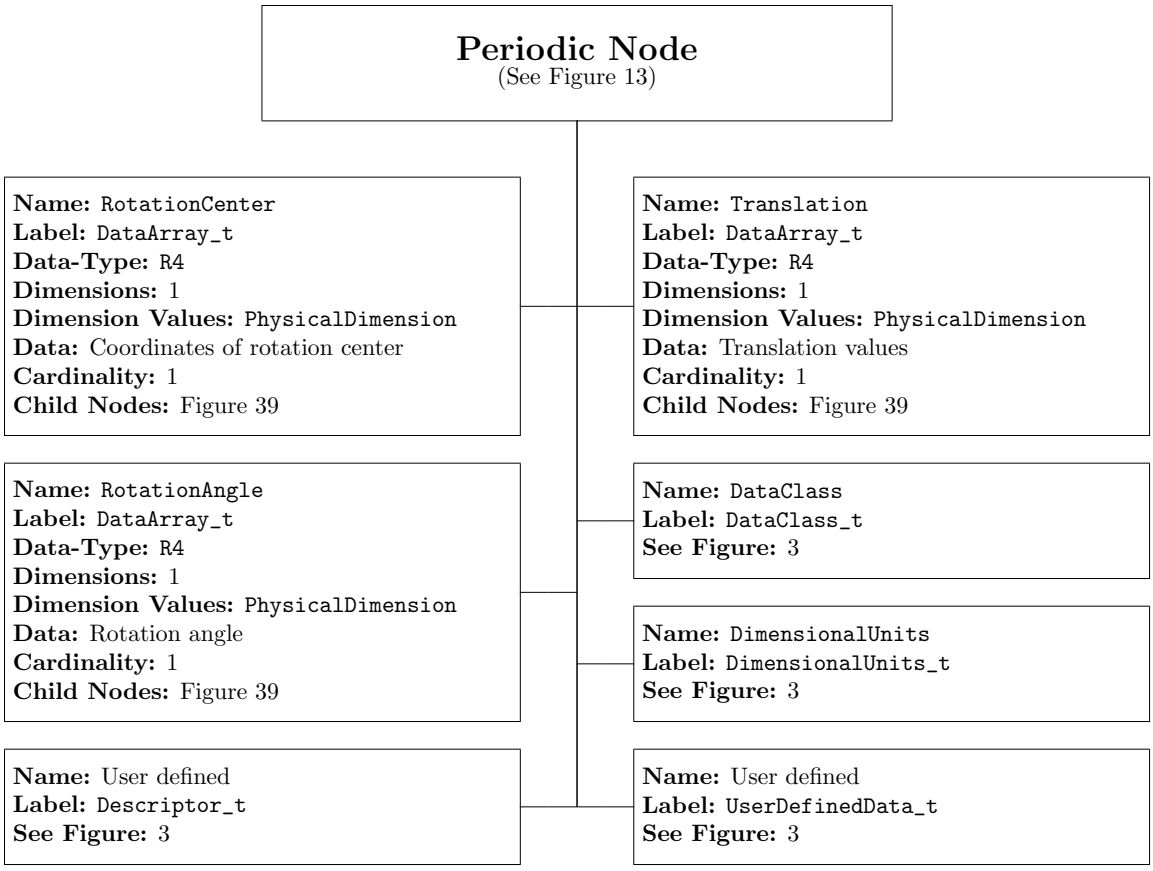


Figure 14: `Periodic_t` Data Structure

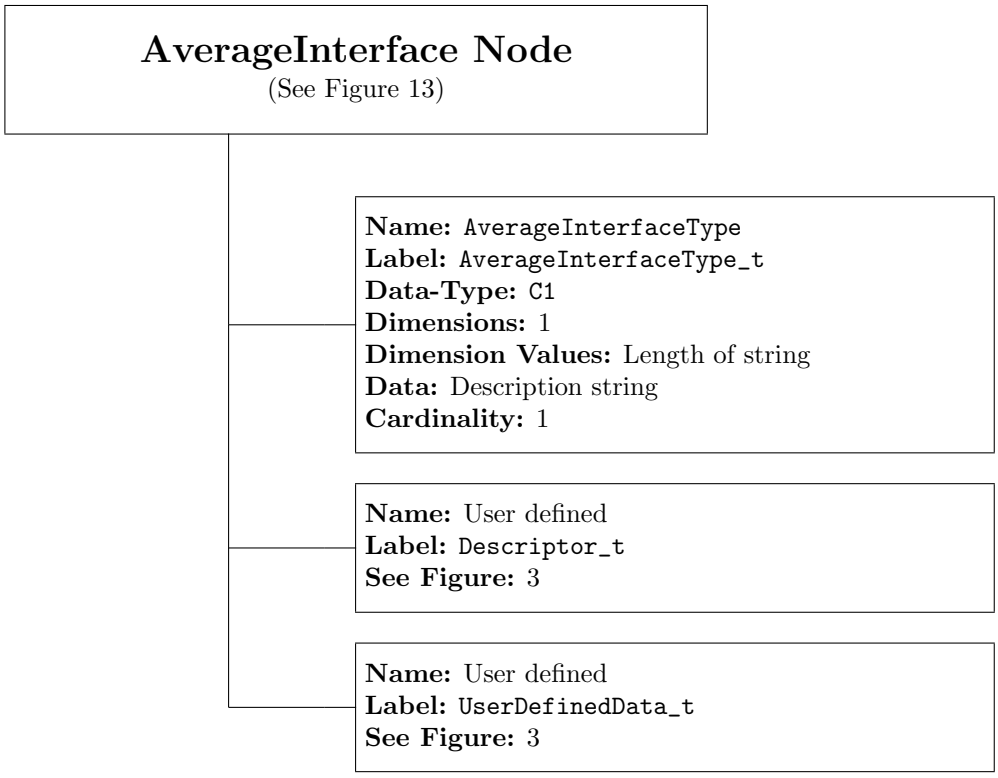
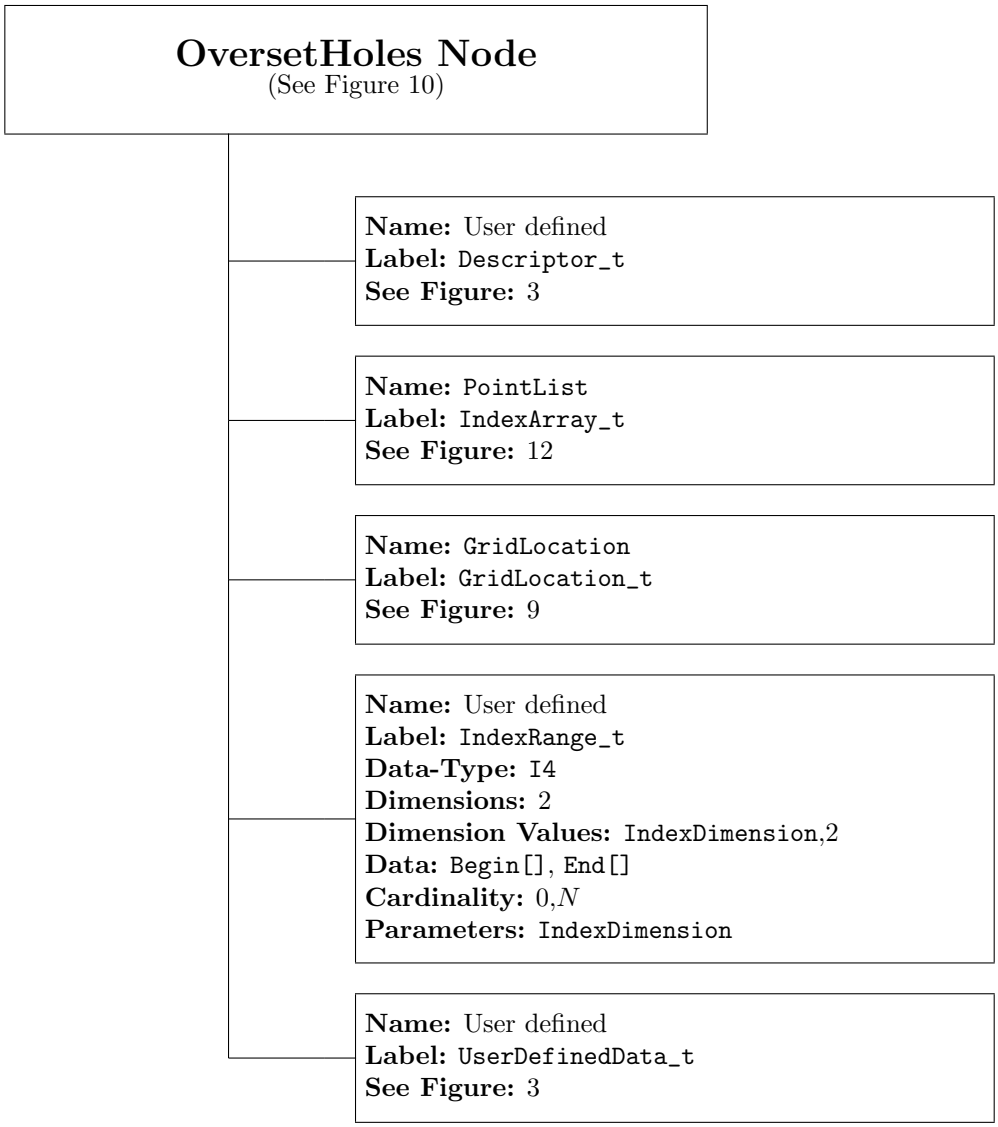


Figure 15: `AverageInterface_t` Data Structure



Note:

- The nodes `PointRange` and `PointList` are mutually exclusive. Use one format or the other.

Figure 16: `OversetHoles_t` Data Structure

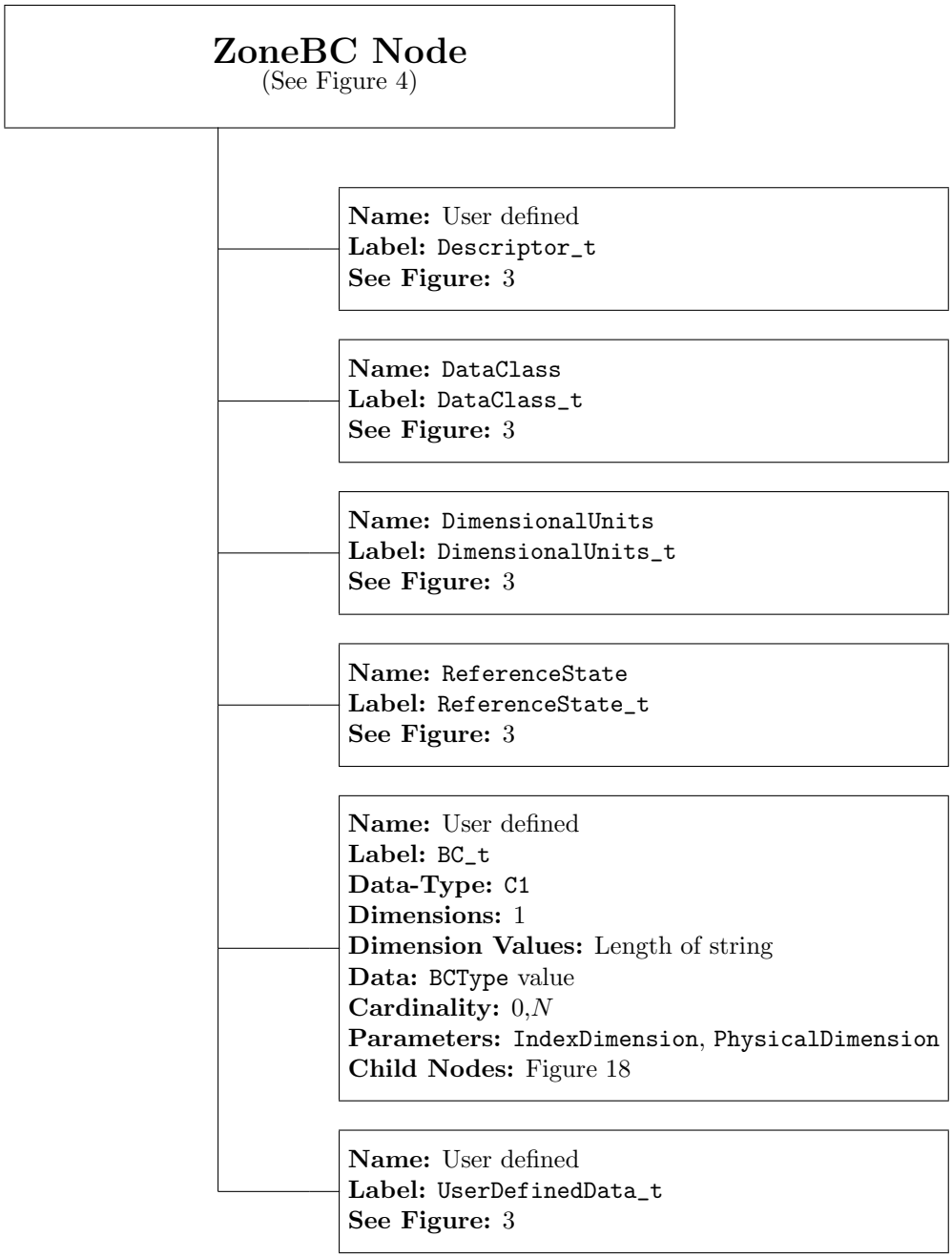


Figure 17: `ZoneBC_t` Data Structure

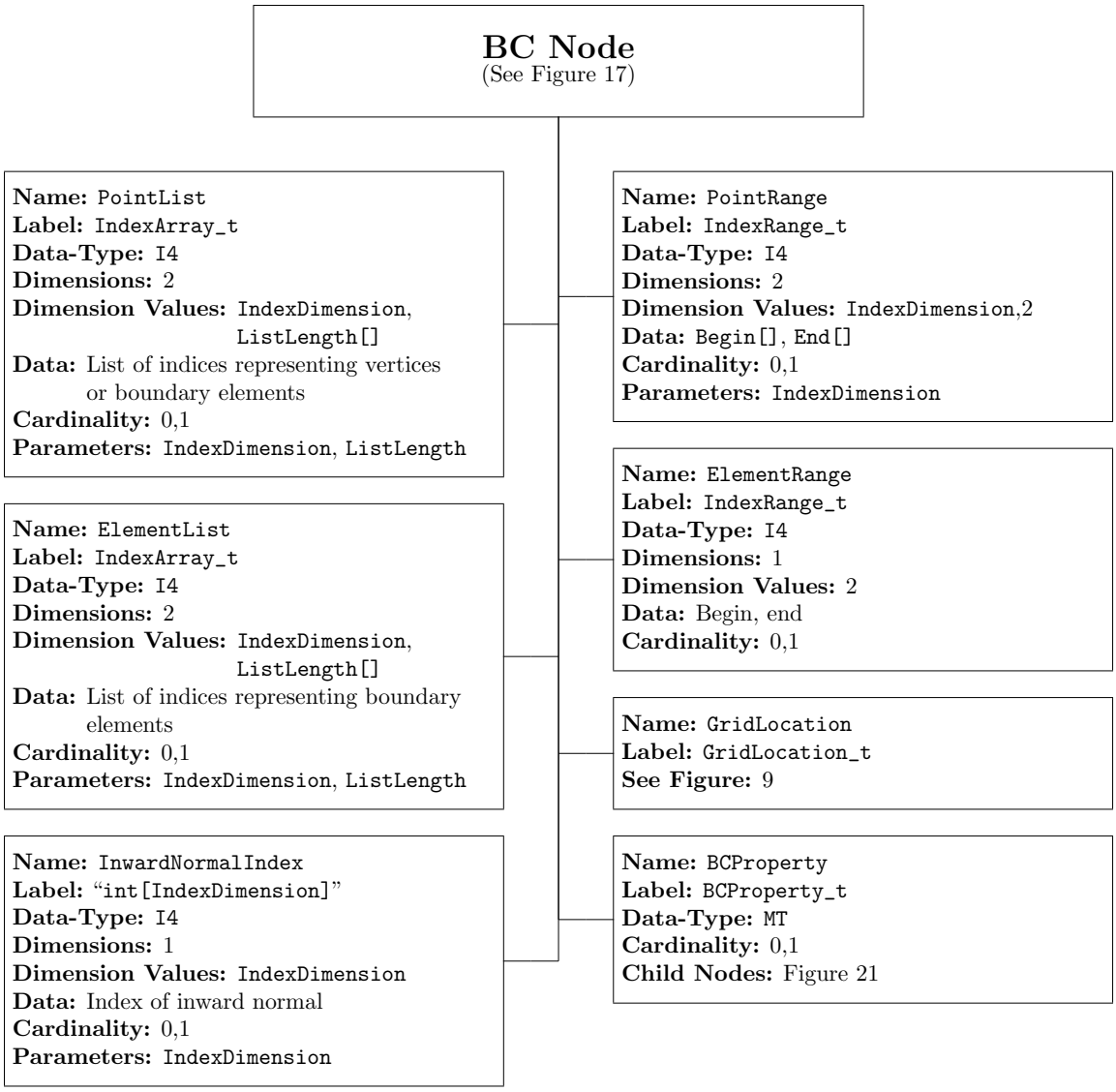


Figure 18: `BC_t` Data Structure (*Continued on next page*)

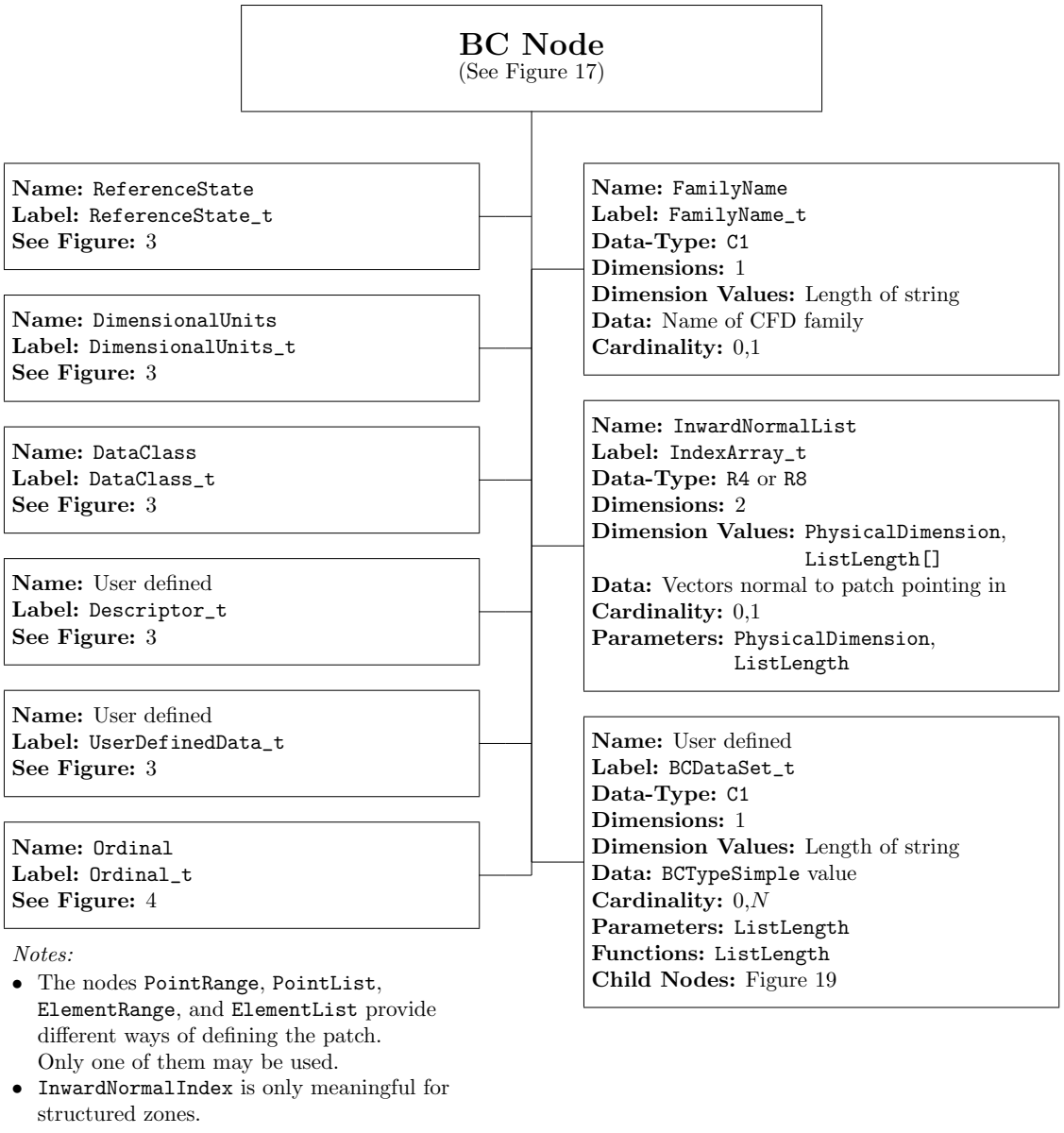


Figure 18: `BC_t` Data Structure (*Continued from previous page*)

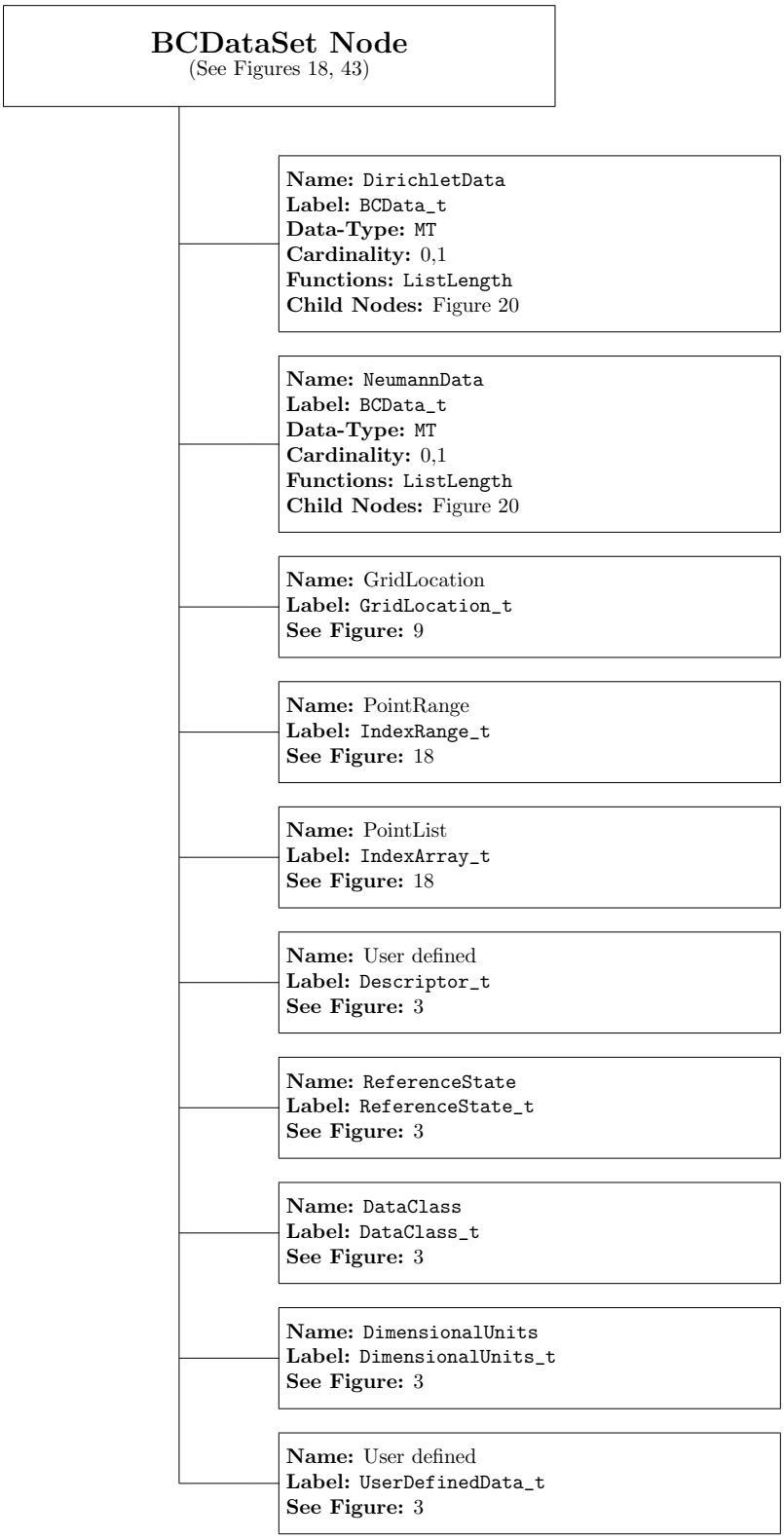


Figure 19: `BCDataSet_t` Data Structure

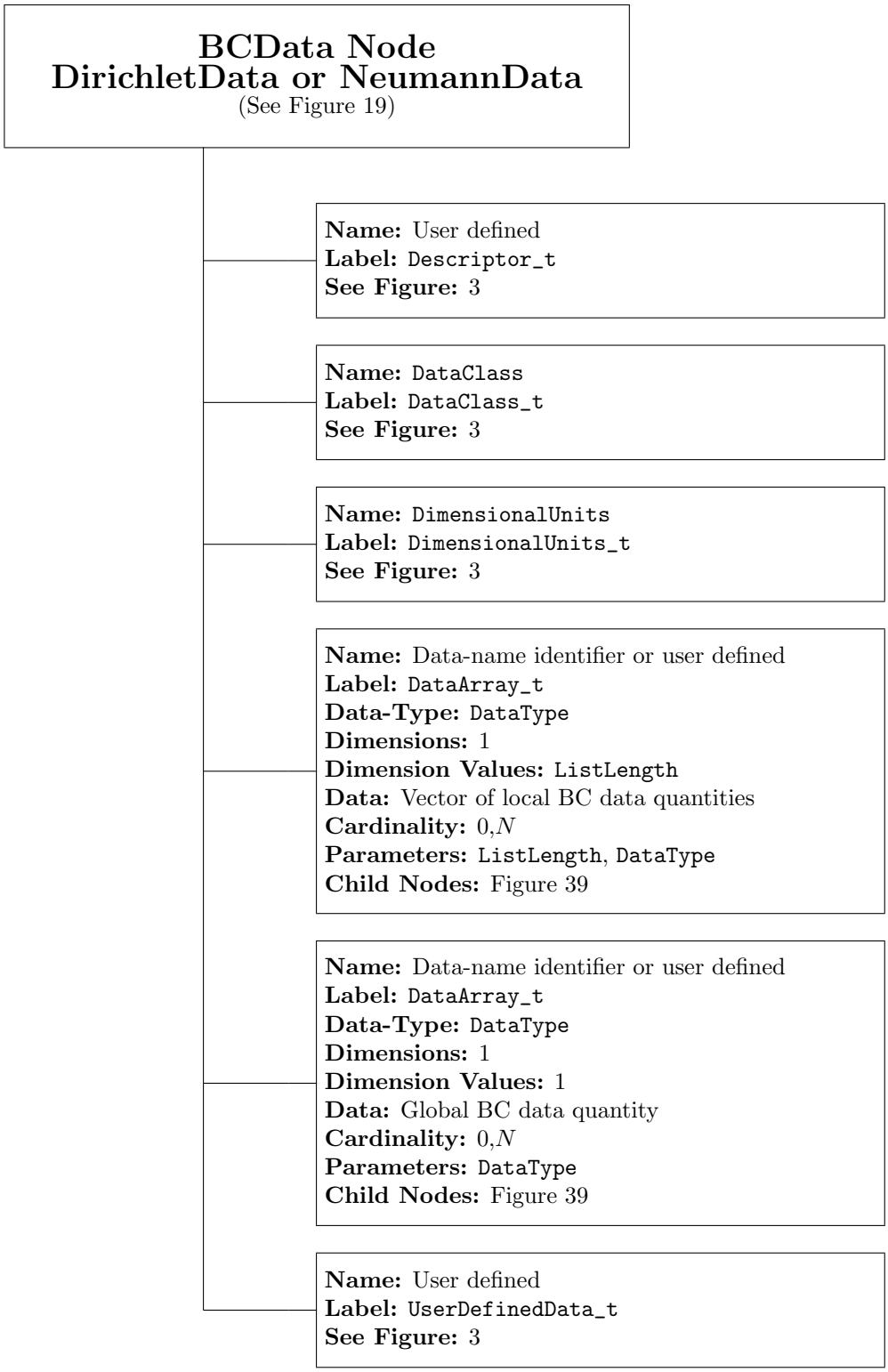


Figure 20: `BCData_t` Data Structure

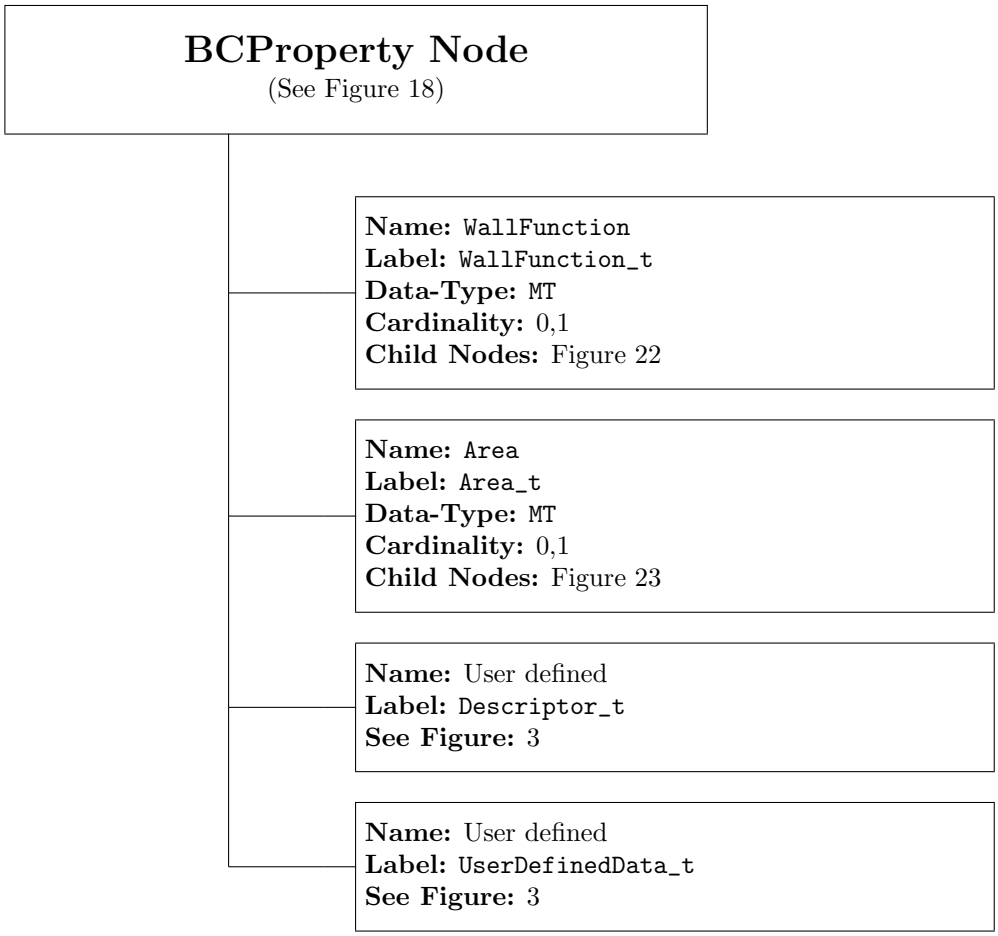


Figure 21: `BCProperty_t` Data Structure

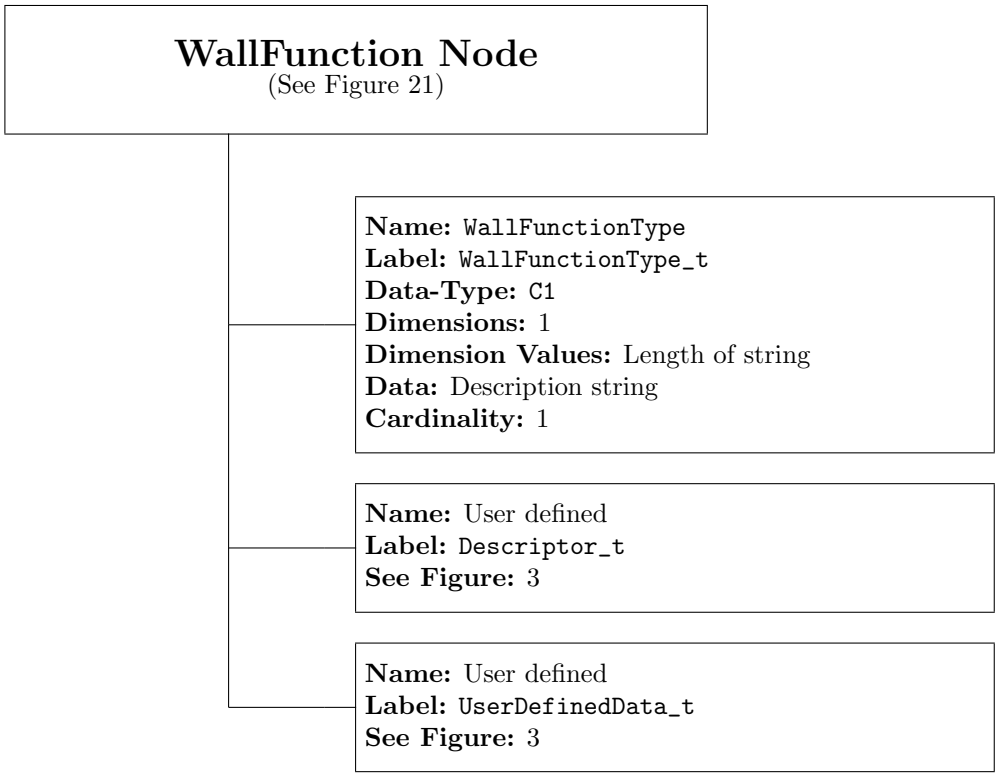


Figure 22: `WallFunction_t` Data Structure

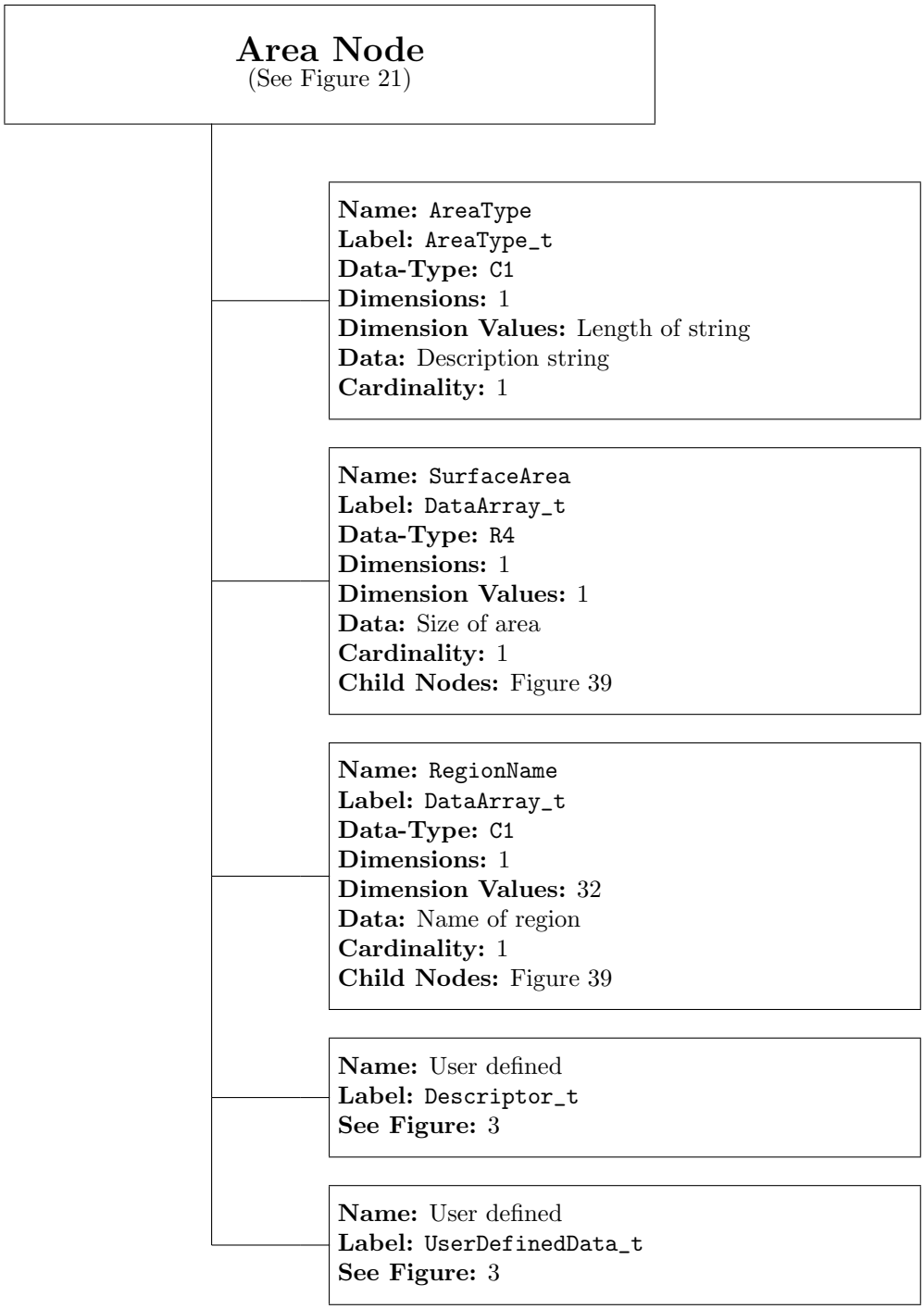


Figure 23: `Area_t` Data Structure

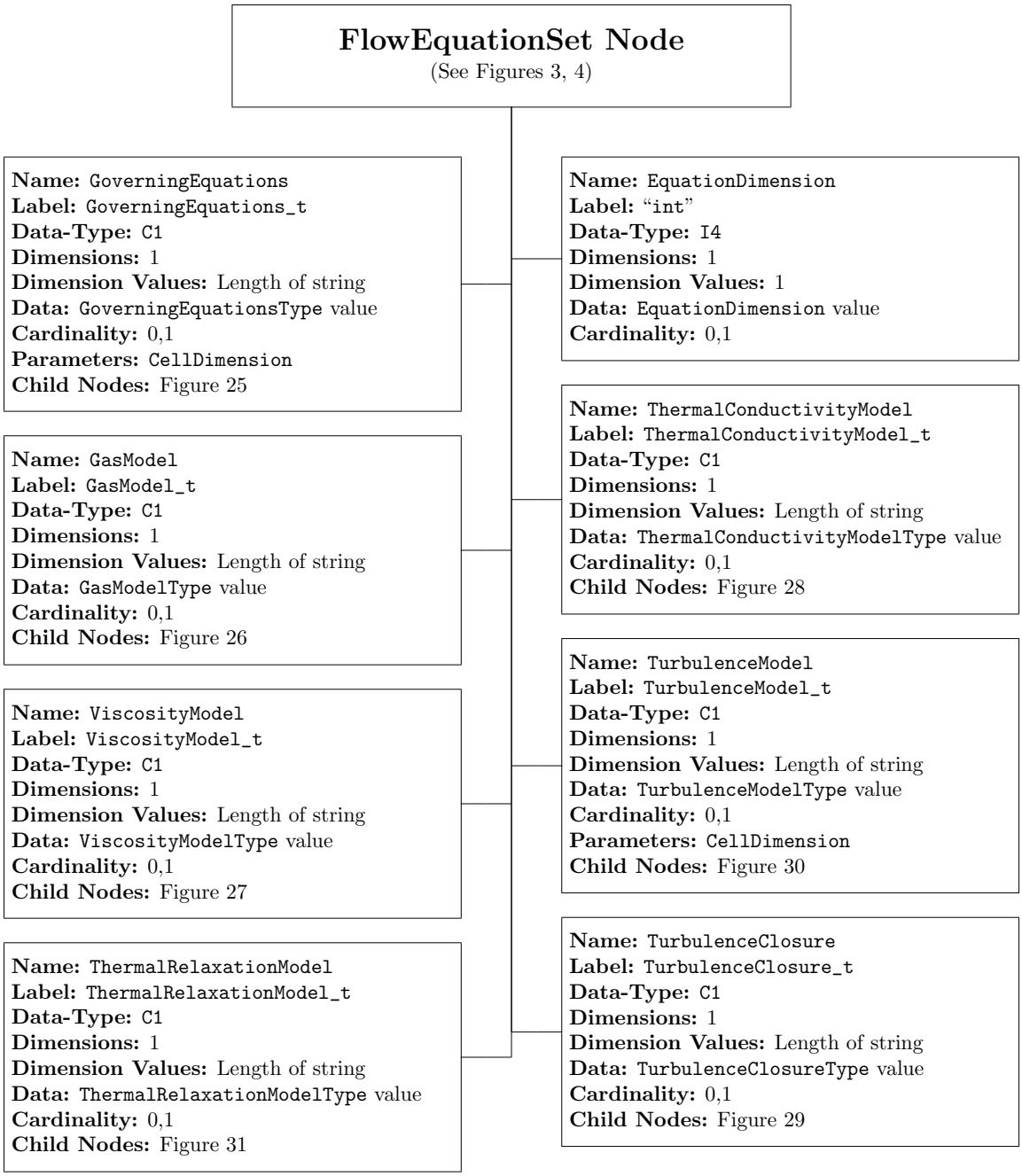


Figure 24: `FlowEquationSet_t` Data Structure (*Continued on next page*)

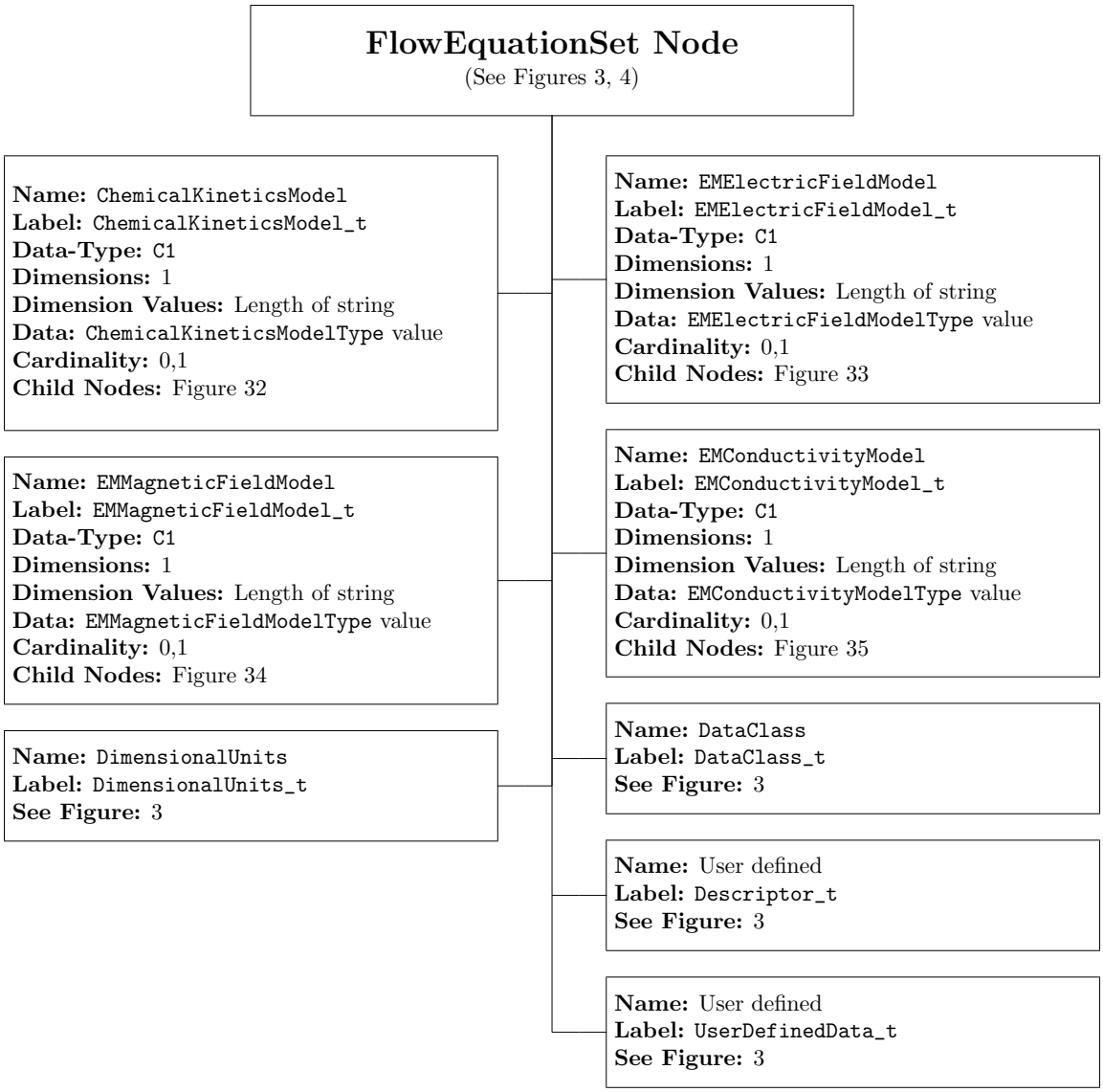
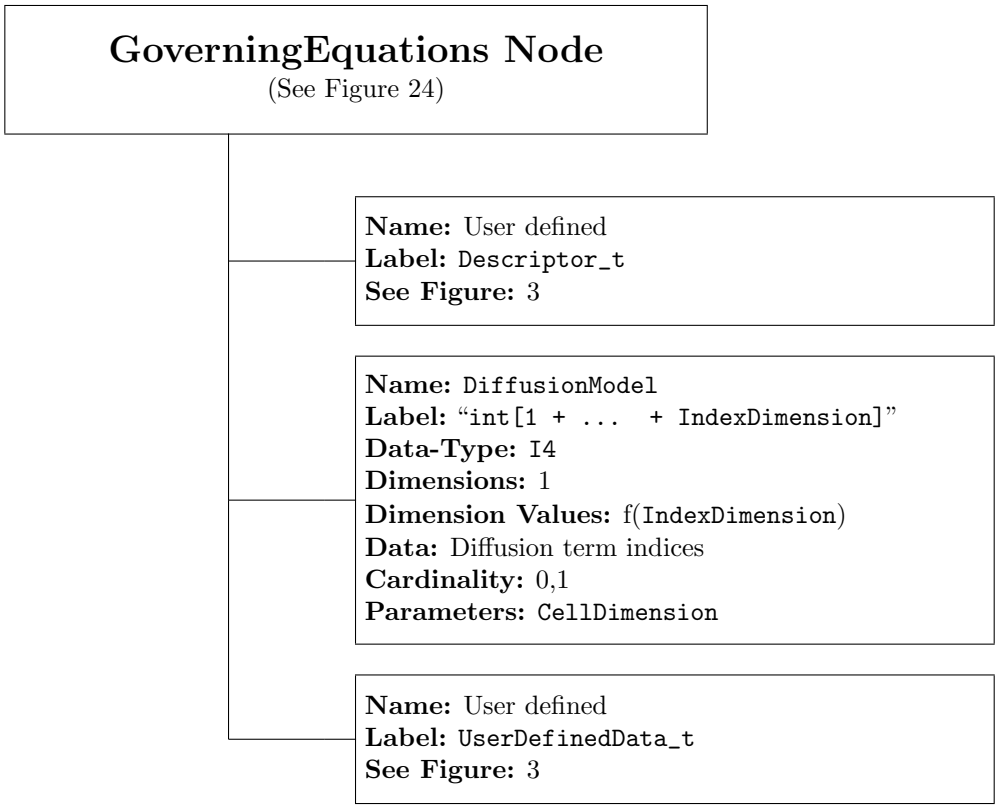


Figure 24: `FlowEquationSet_t` Data Structure (*Continued from previous page*)



Note:

- The diffusion model defined here is only applicable to structured zones.
- For a structured zone, `IndexDimension = CellDimension`.

Figure 25: `GoverningEquations_t` Data Structure

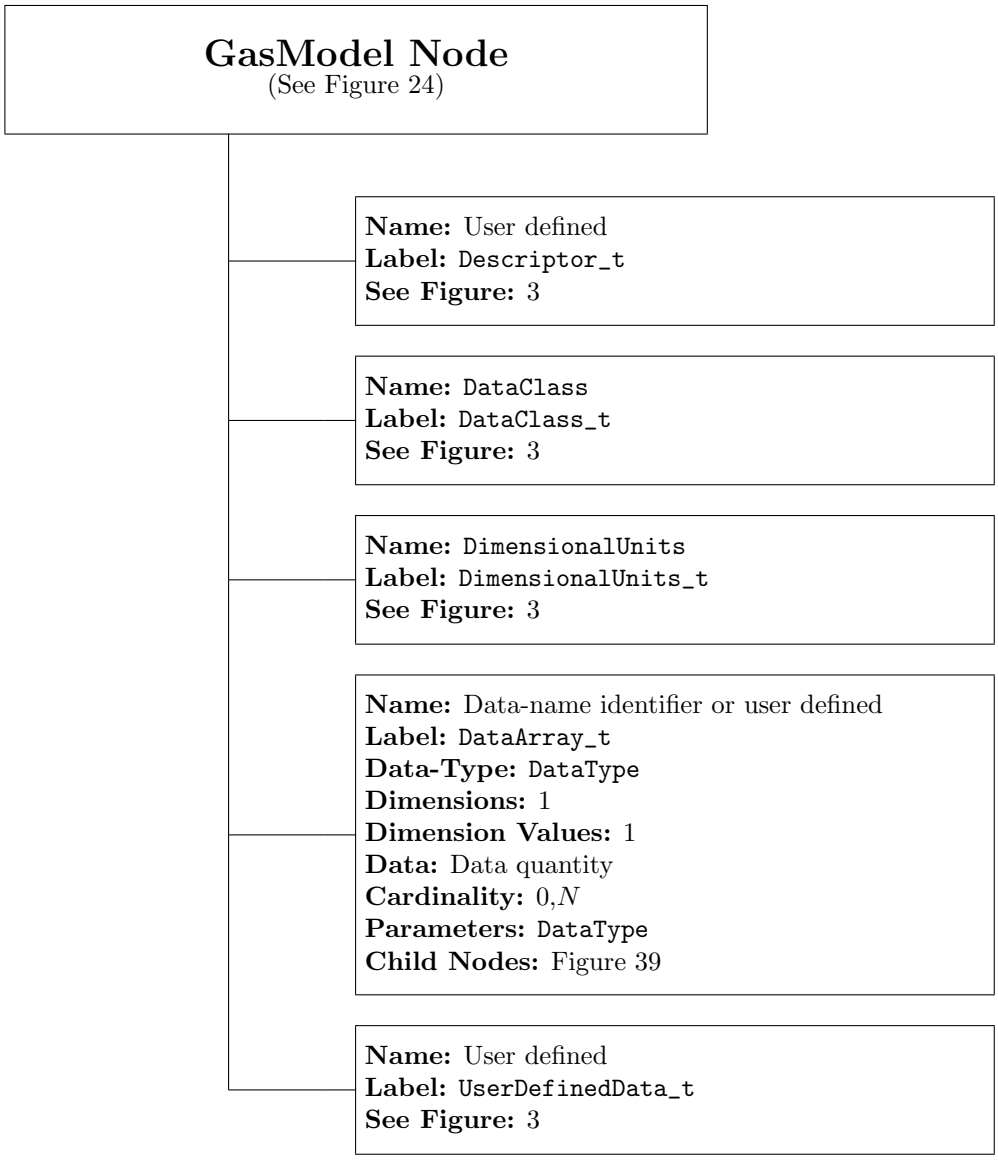


Figure 26: `GasModel_t` Data Structure

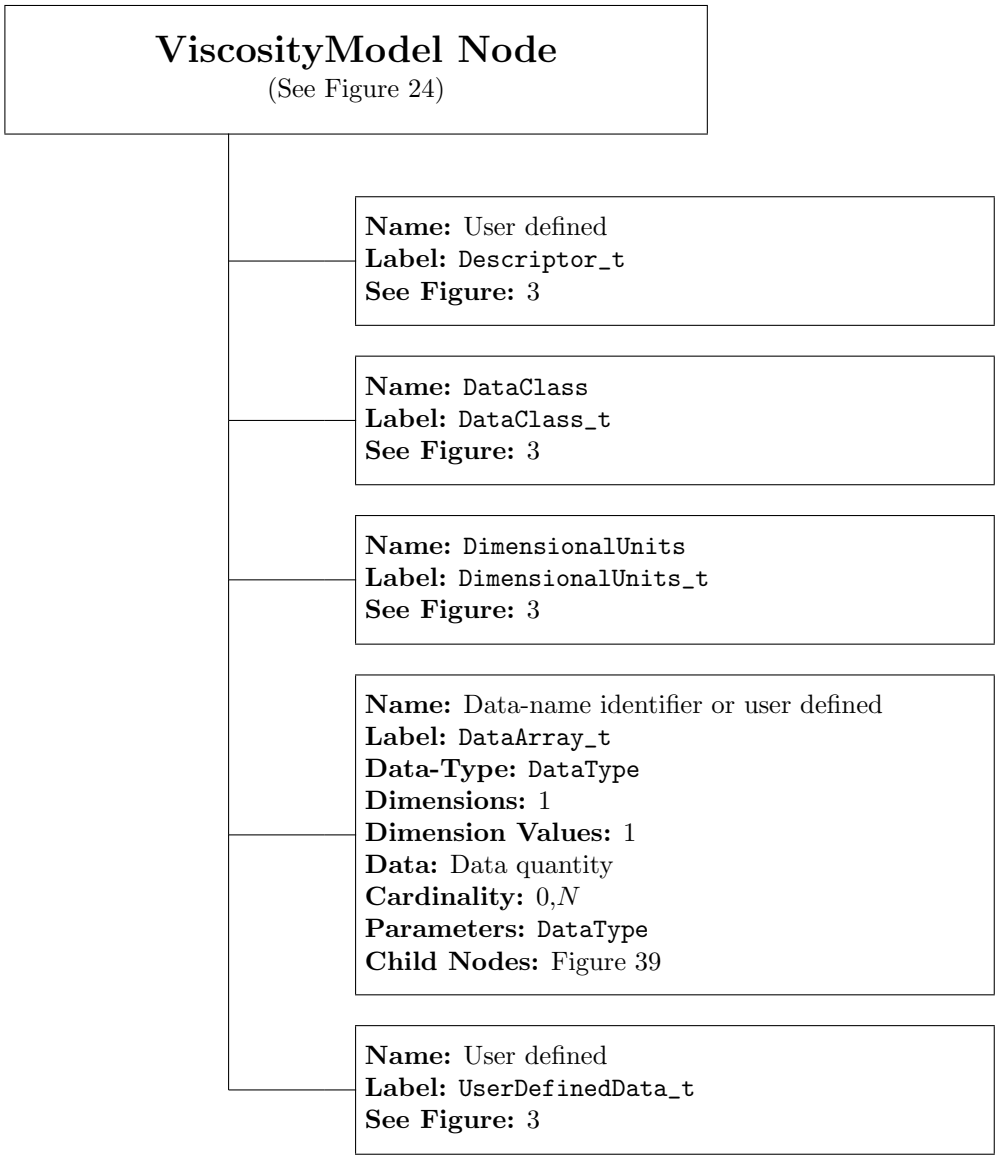


Figure 27: `ViscosityModel_t` Data Structure

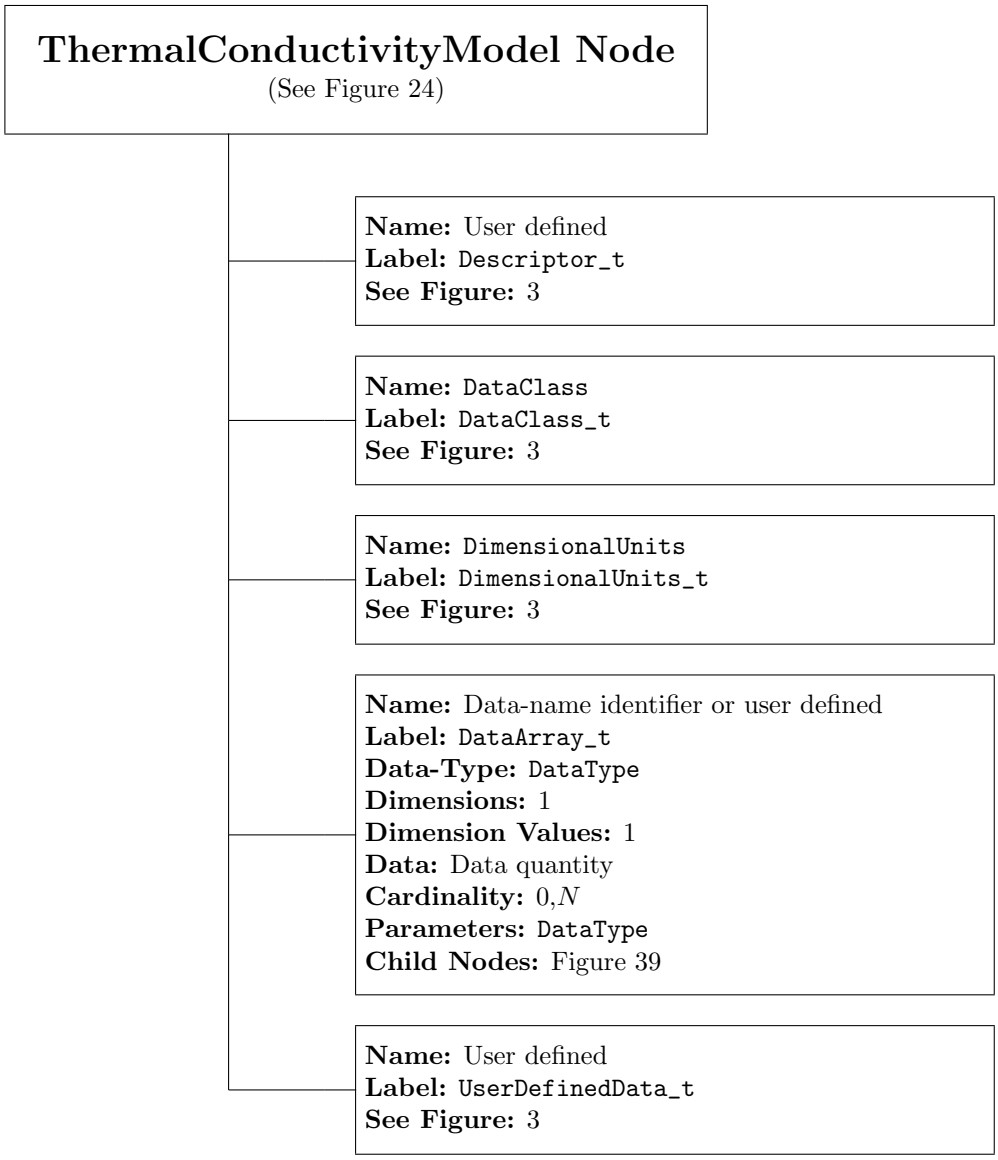


Figure 28: `ThermalConductivityModel_t` Data Structure

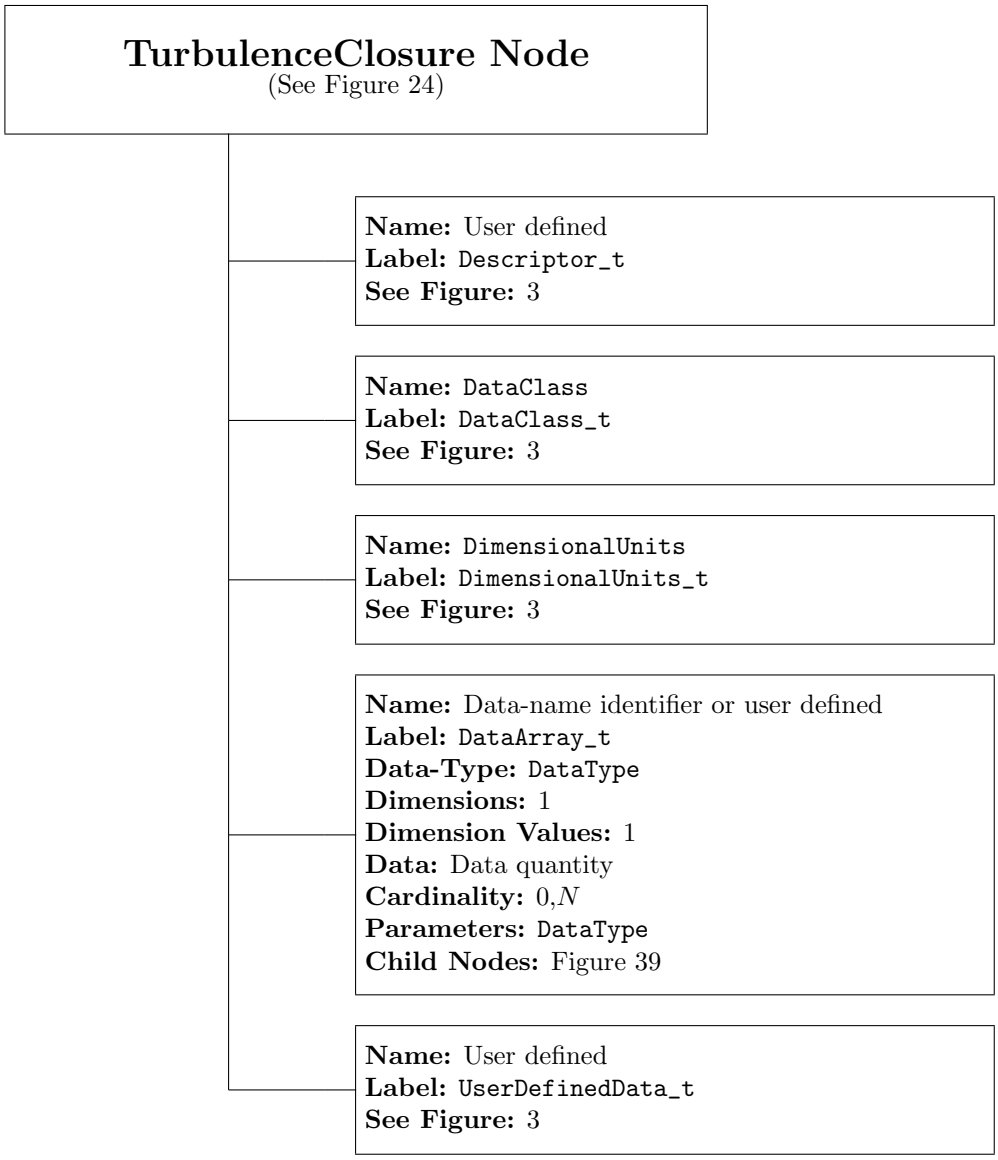
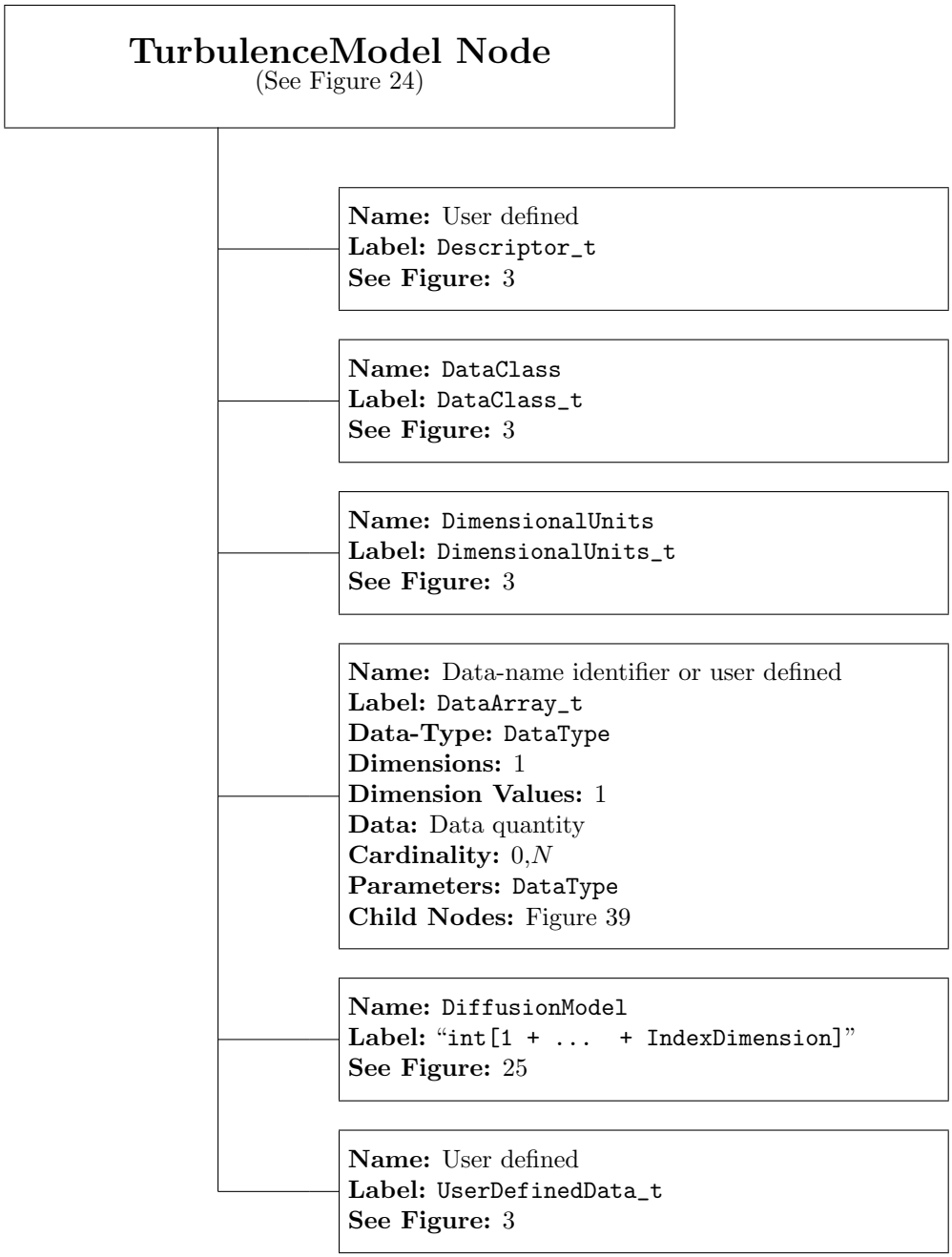


Figure 29: `TurbulenceClosure_t` Data Structure



Note:

- The diffusion model as defined here is only applicable to structured zones.
- For a structured zone, `IndexDimension = CellDimension`.

Figure 30: `TurbulenceModel_t` Data Structure

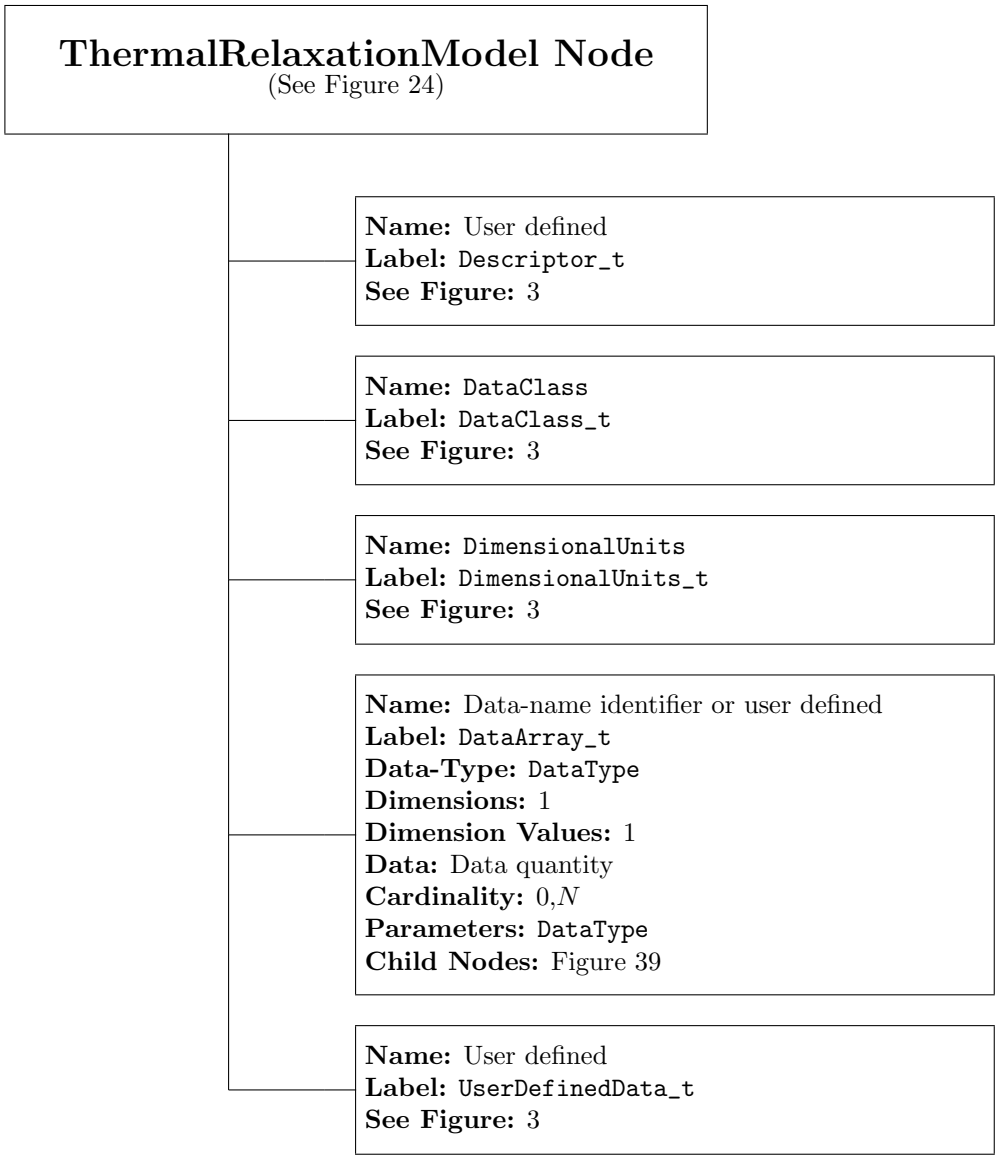


Figure 31: `ThermalRelaxationModel_t` Data Structure

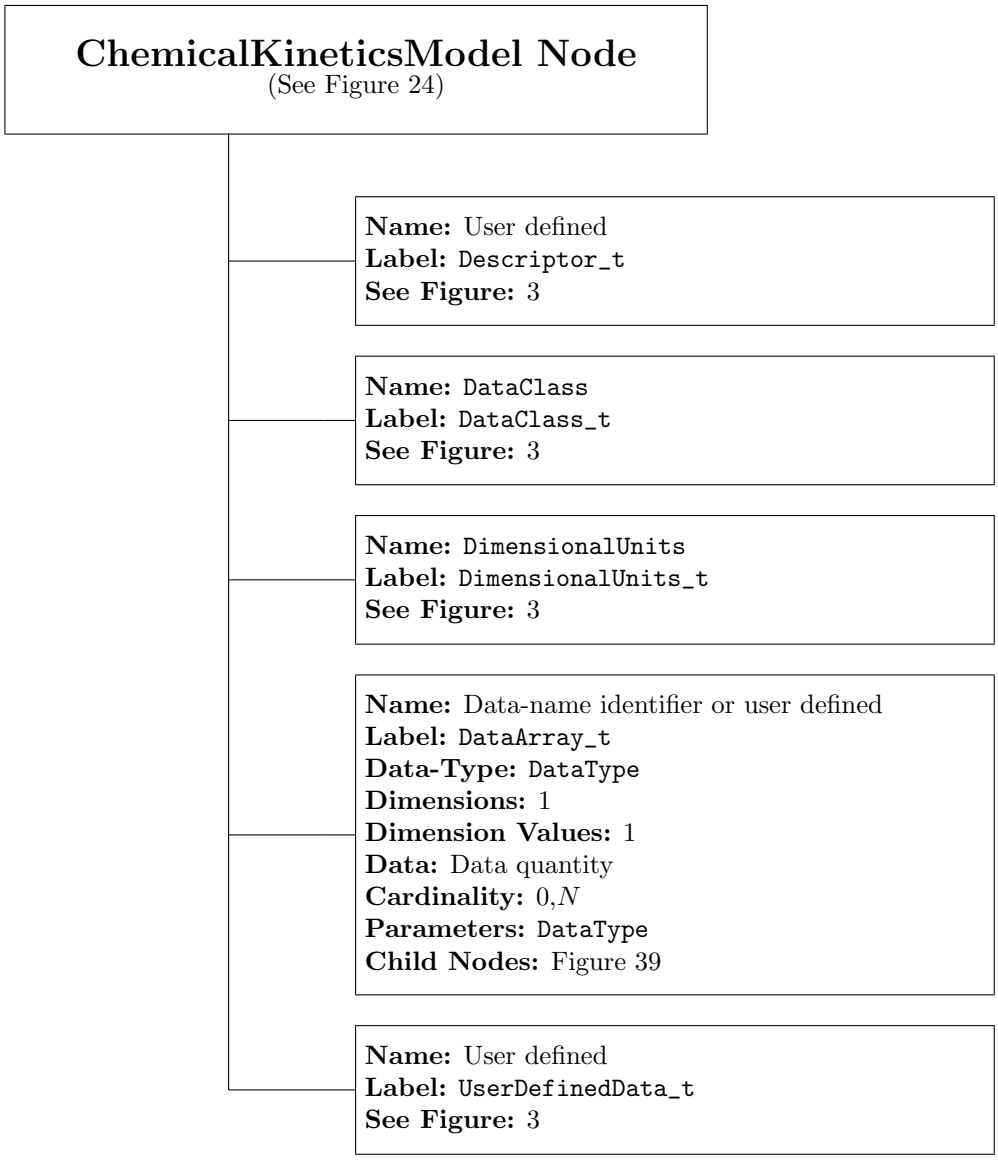


Figure 32: `ChemicalKineticsModel_t` Data Structure

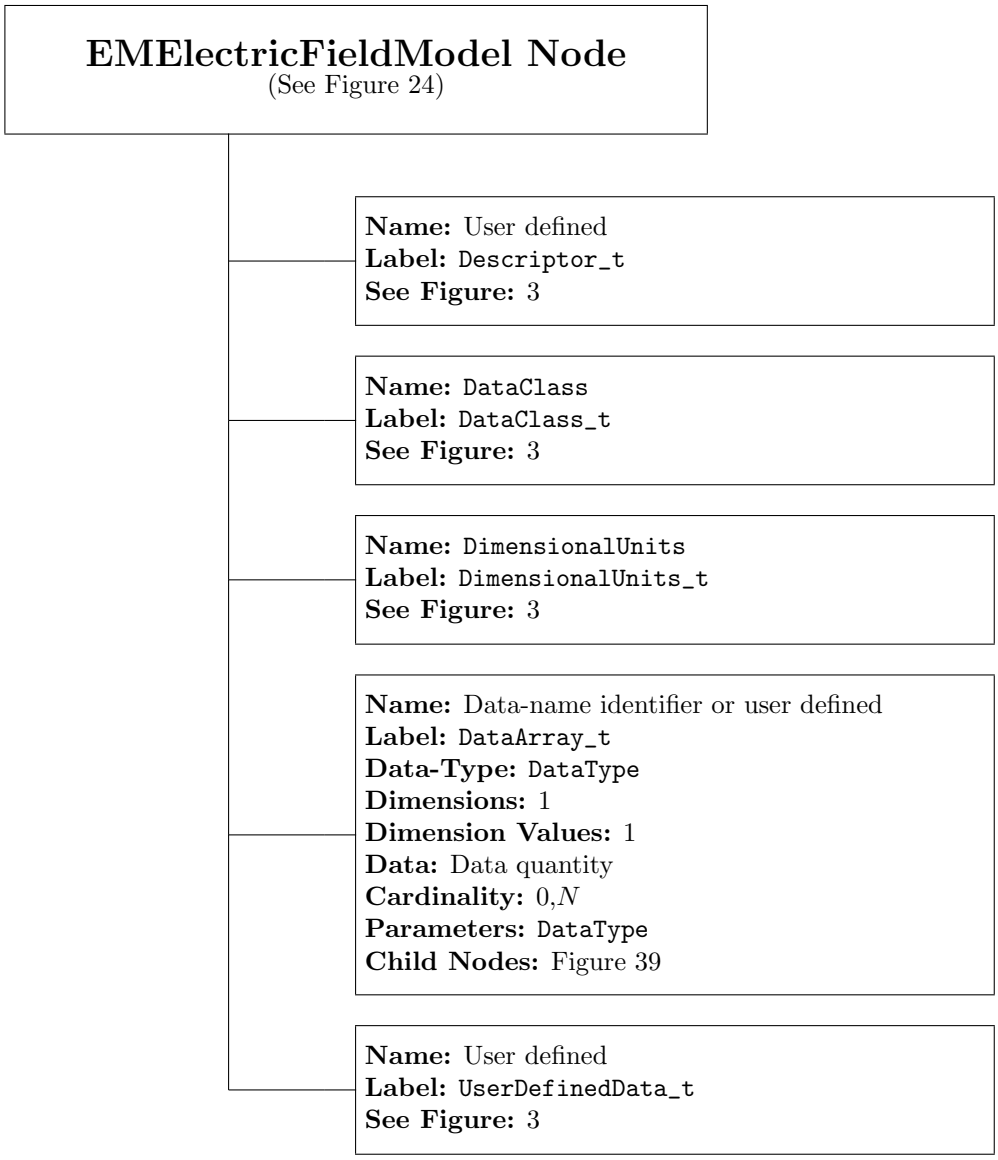


Figure 33: `EMElectricFieldModel_t` Data Structure

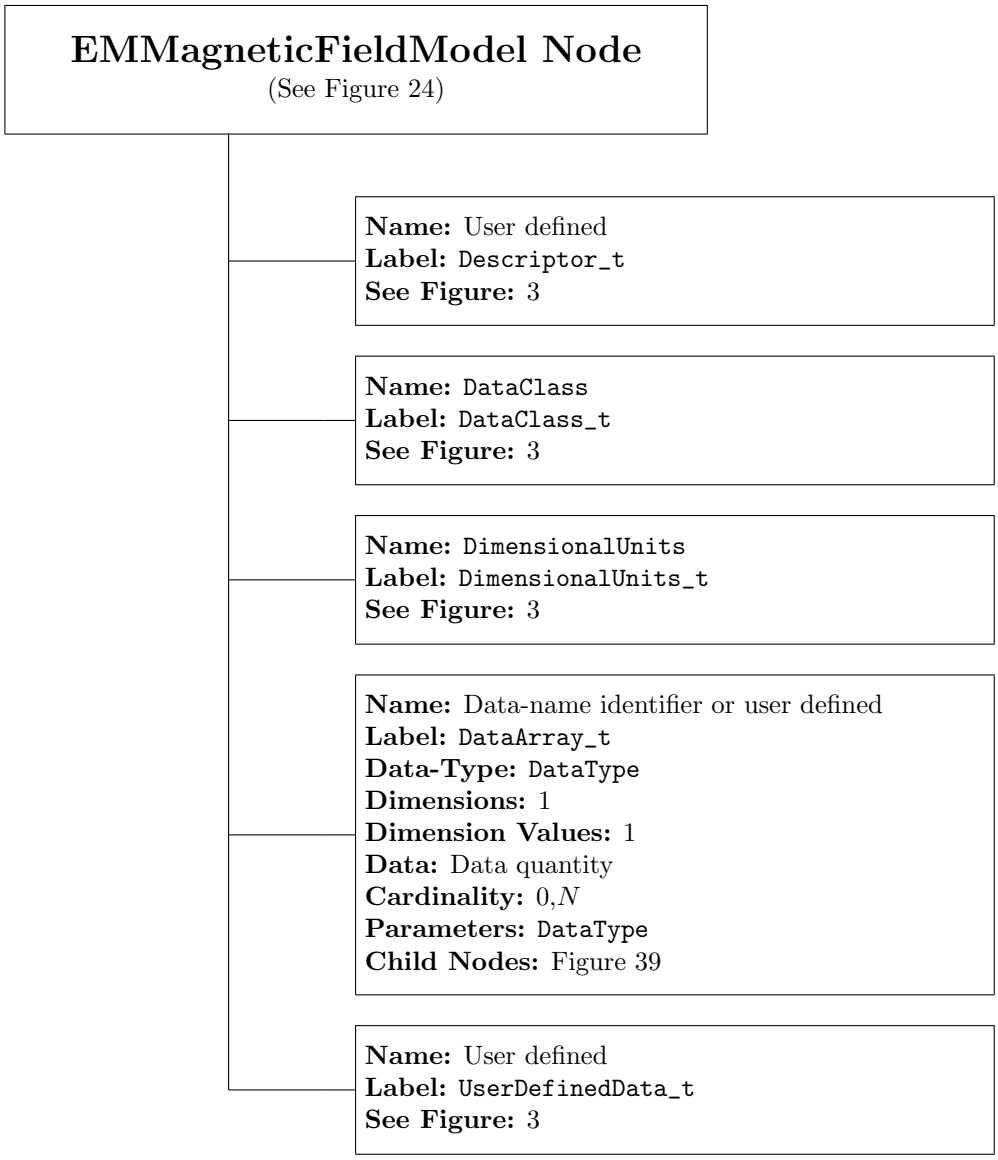


Figure 34: `EMMagneticFieldModel_t` Data Structure

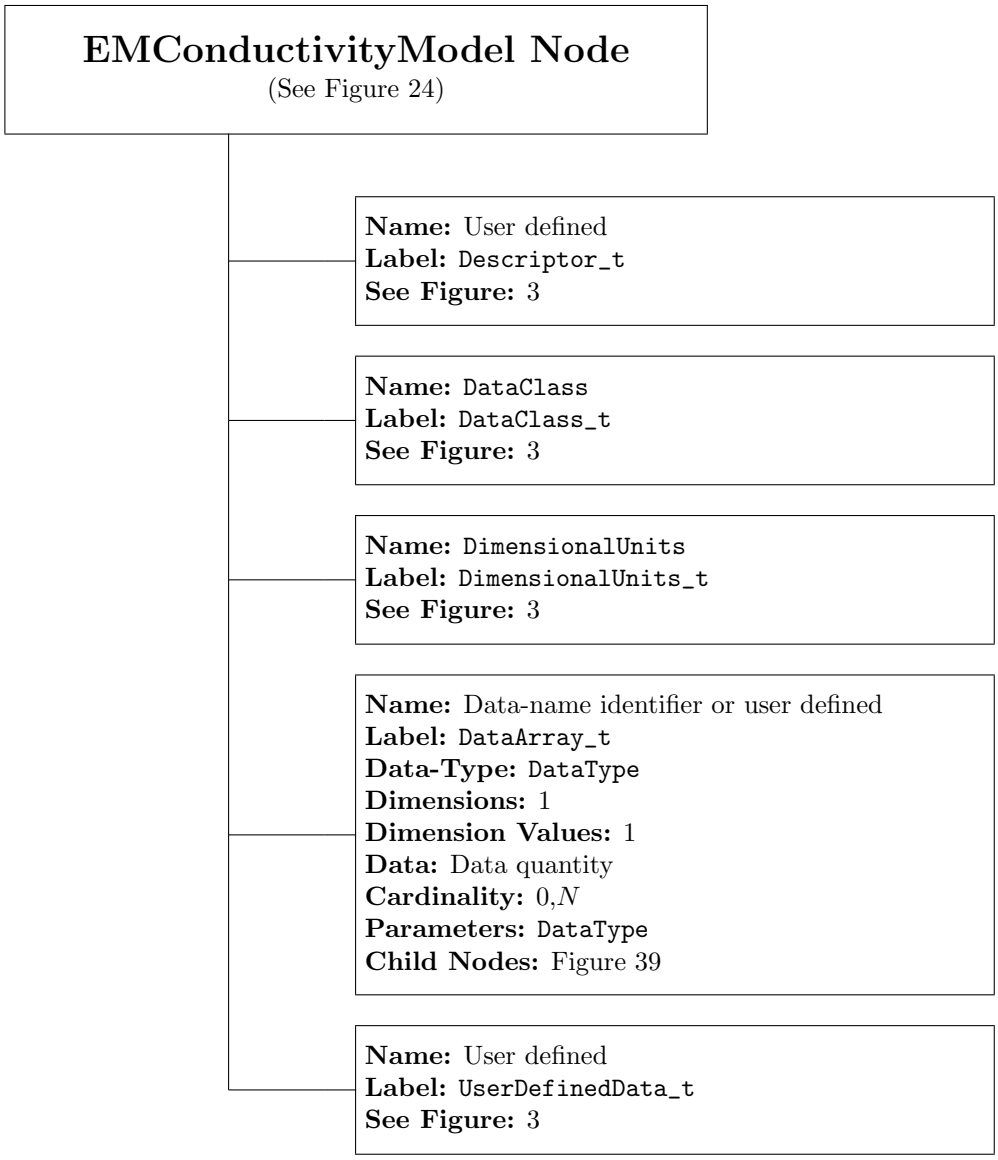


Figure 35: `EMConductivityModel_t` Data Structure

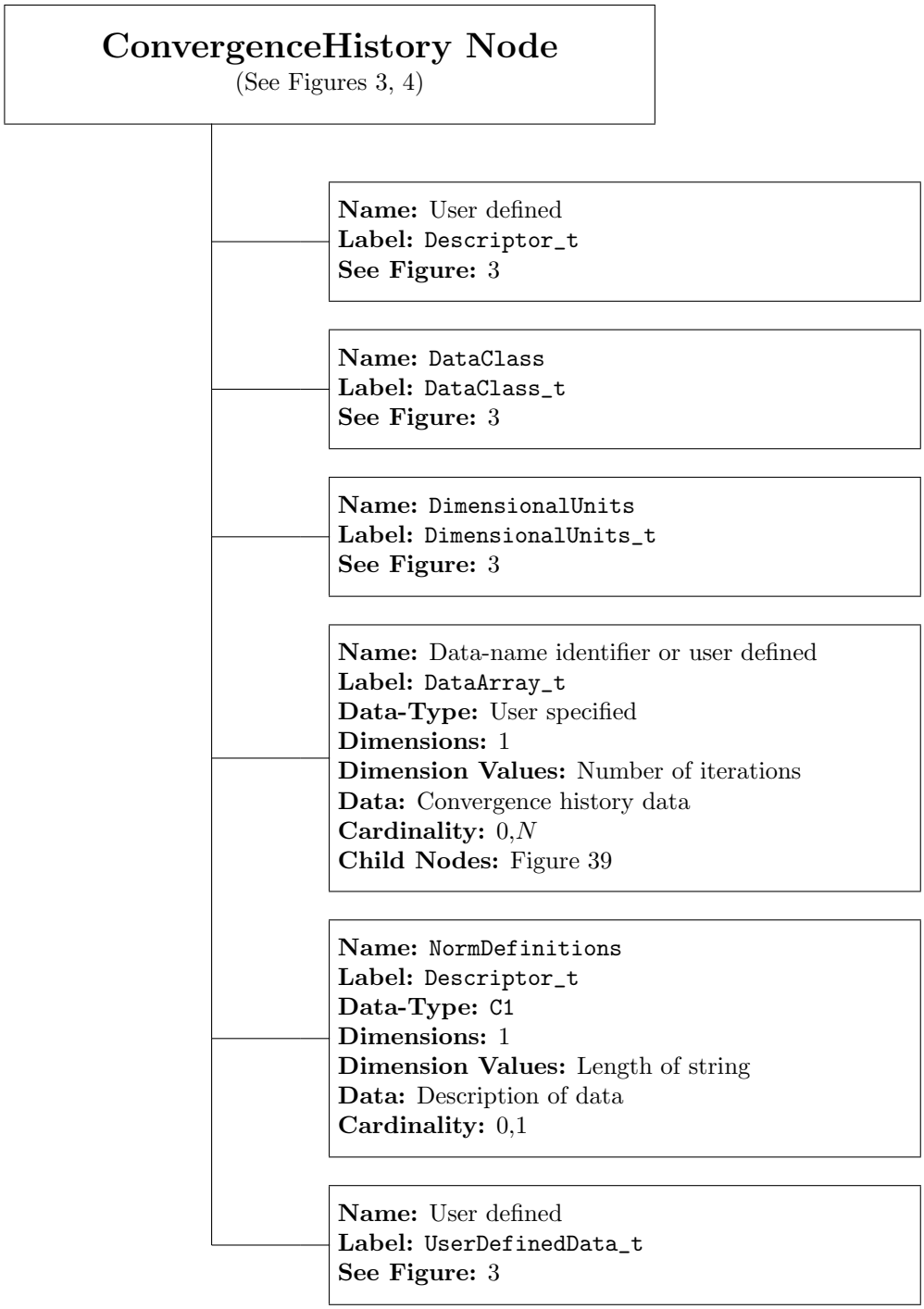


Figure 36: `ConvergenceHistory_t` Data Structure

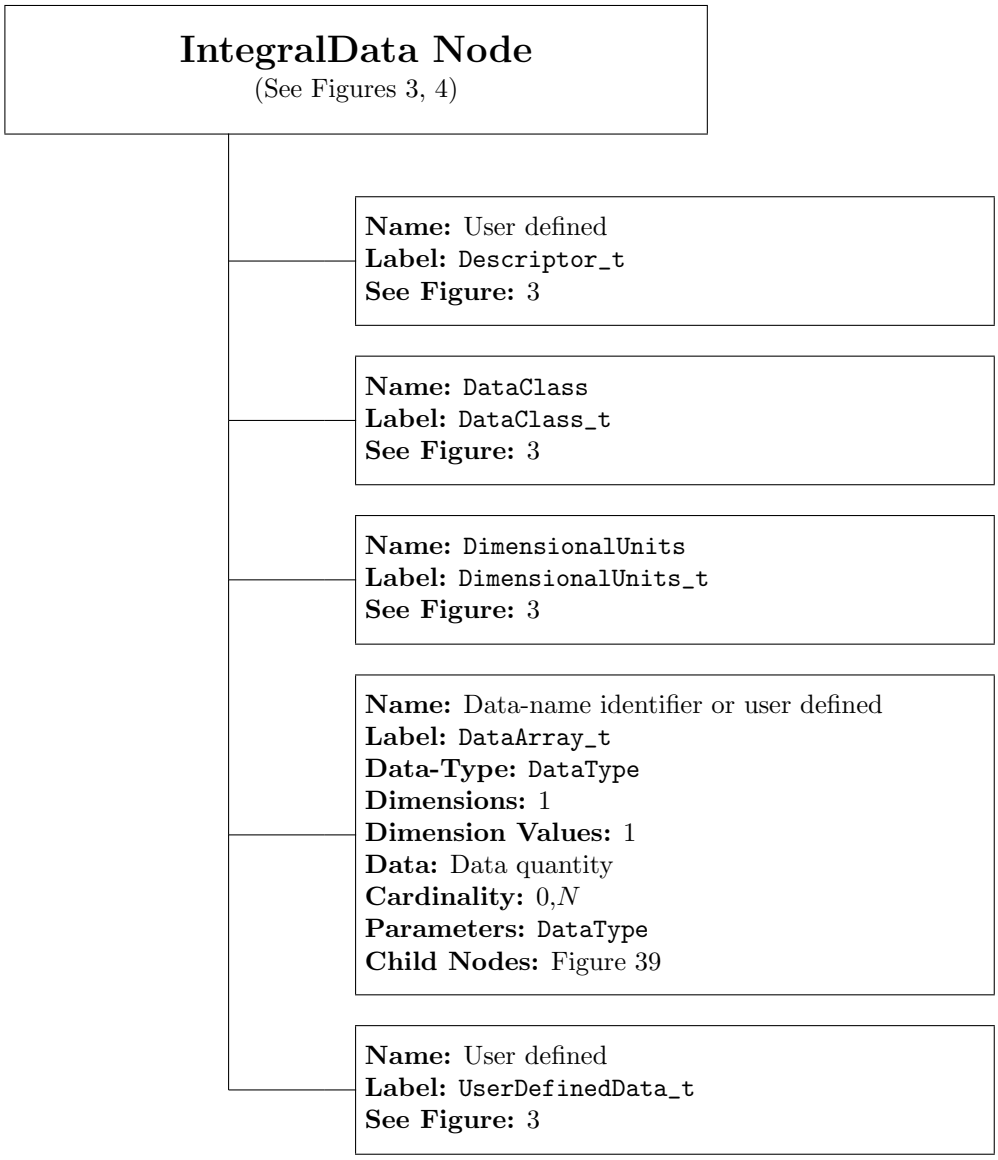


Figure 37: `IntegralData_t` Data Structure

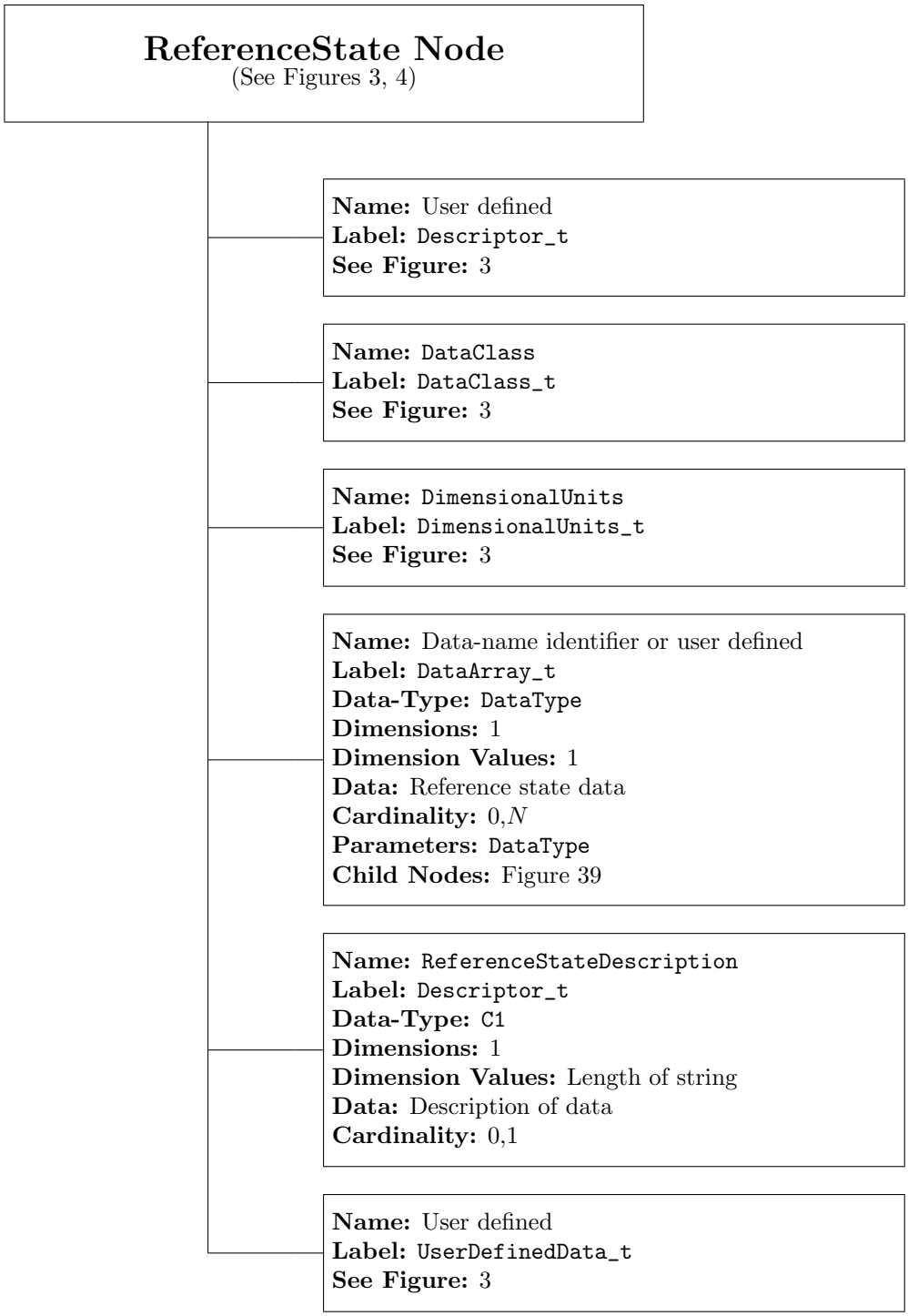


Figure 38: `ReferenceState_t` Data Structure

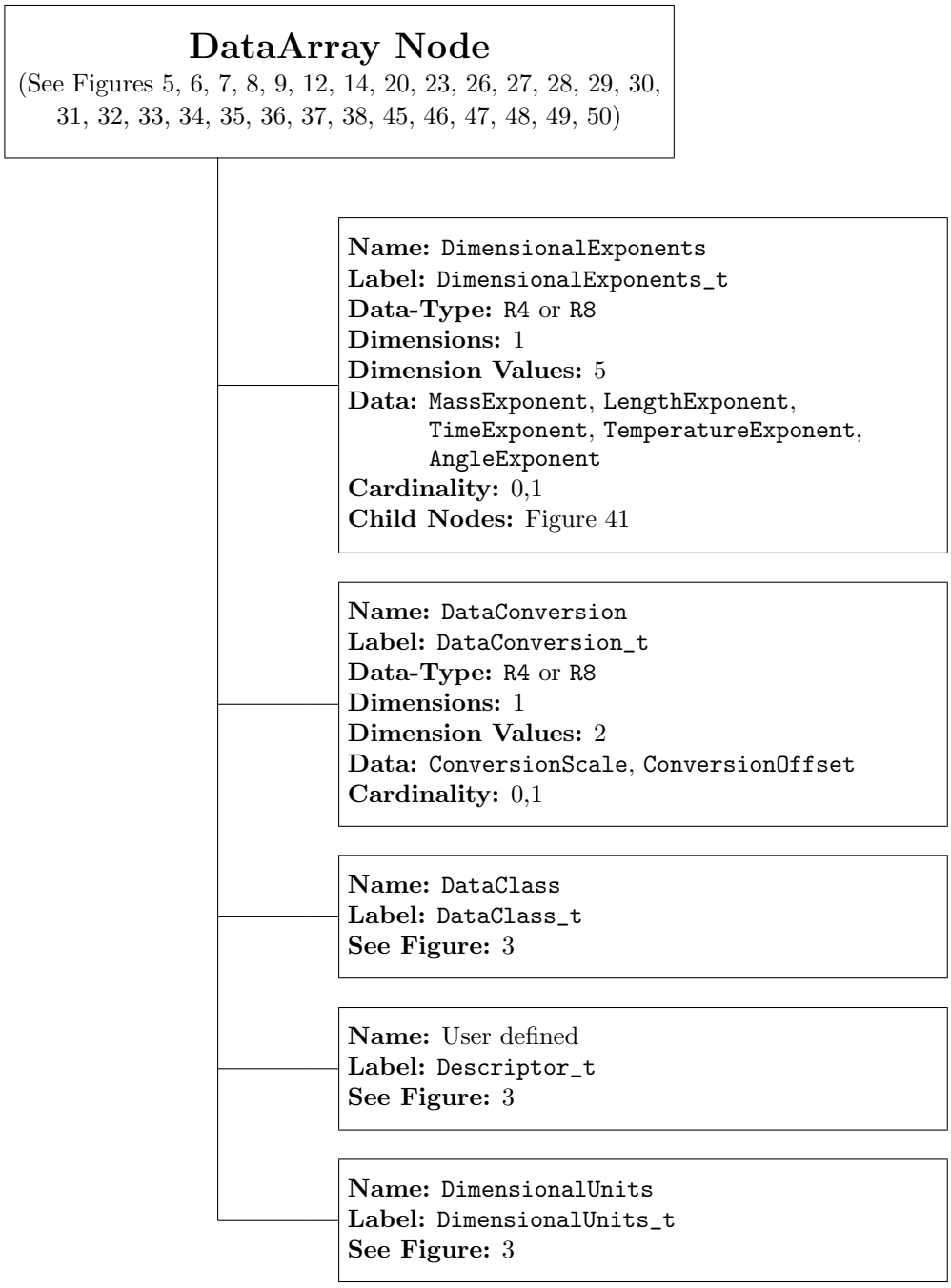


Figure 39: `DataArray_t` Data Structure

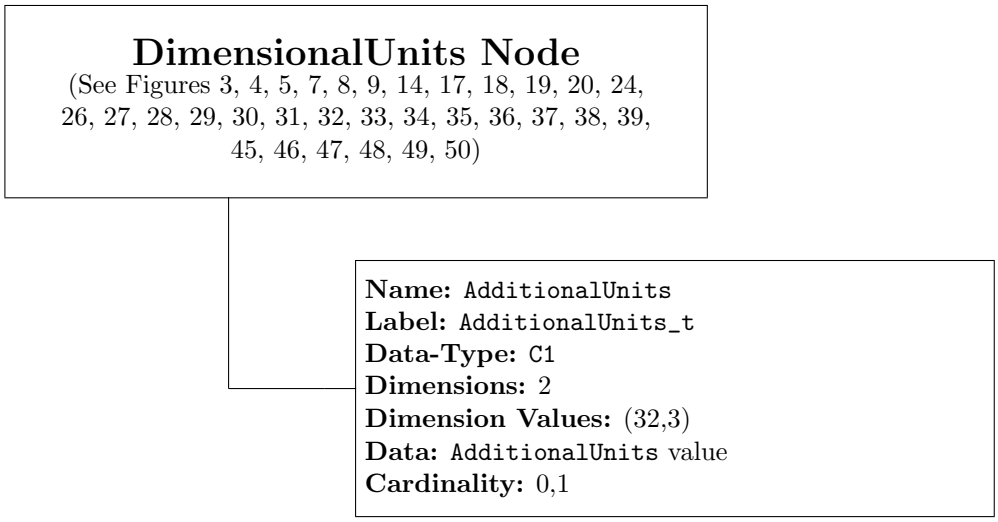


Figure 40: `DimensionalUnits_t` Data Structure

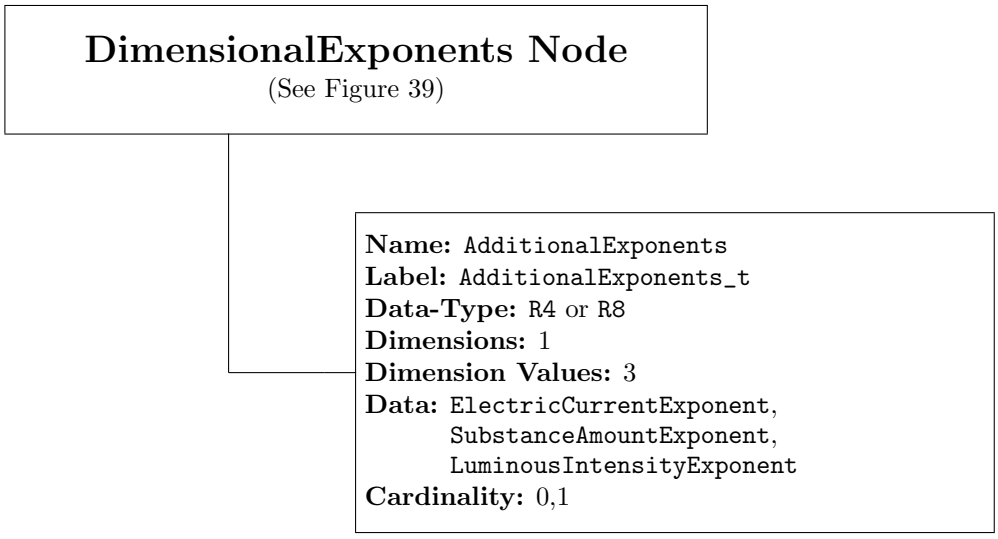


Figure 41: `DimensionalExponents_t` Data Structure

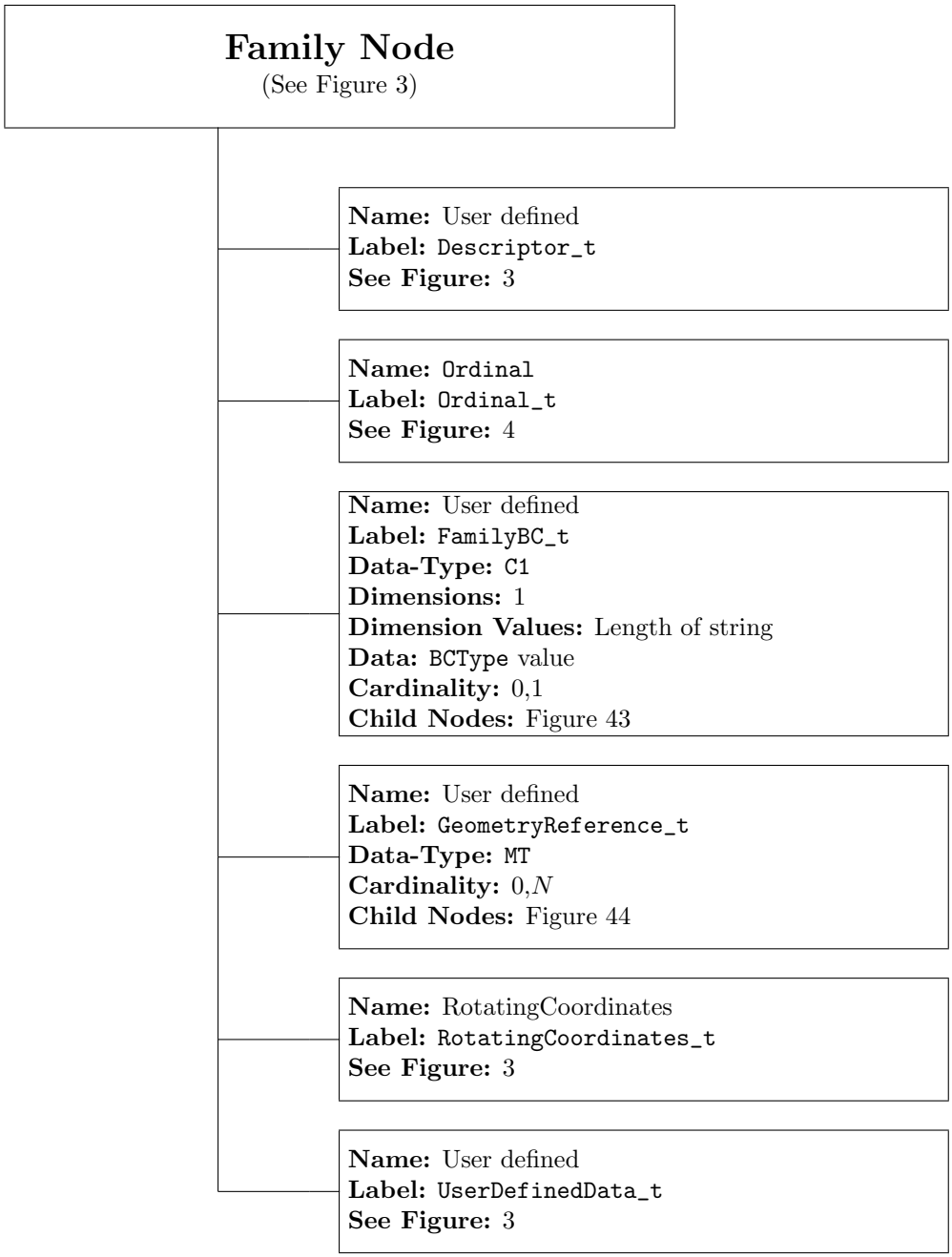


Figure 42: `Family_t` Data Structure

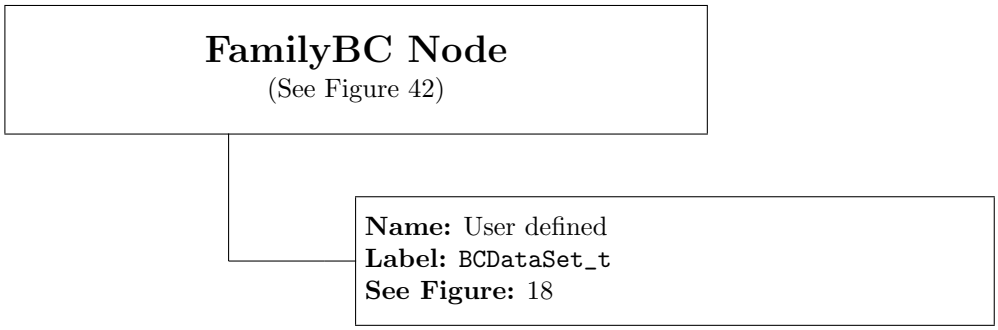


Figure 43: `FamilyBC_t` Data Structure

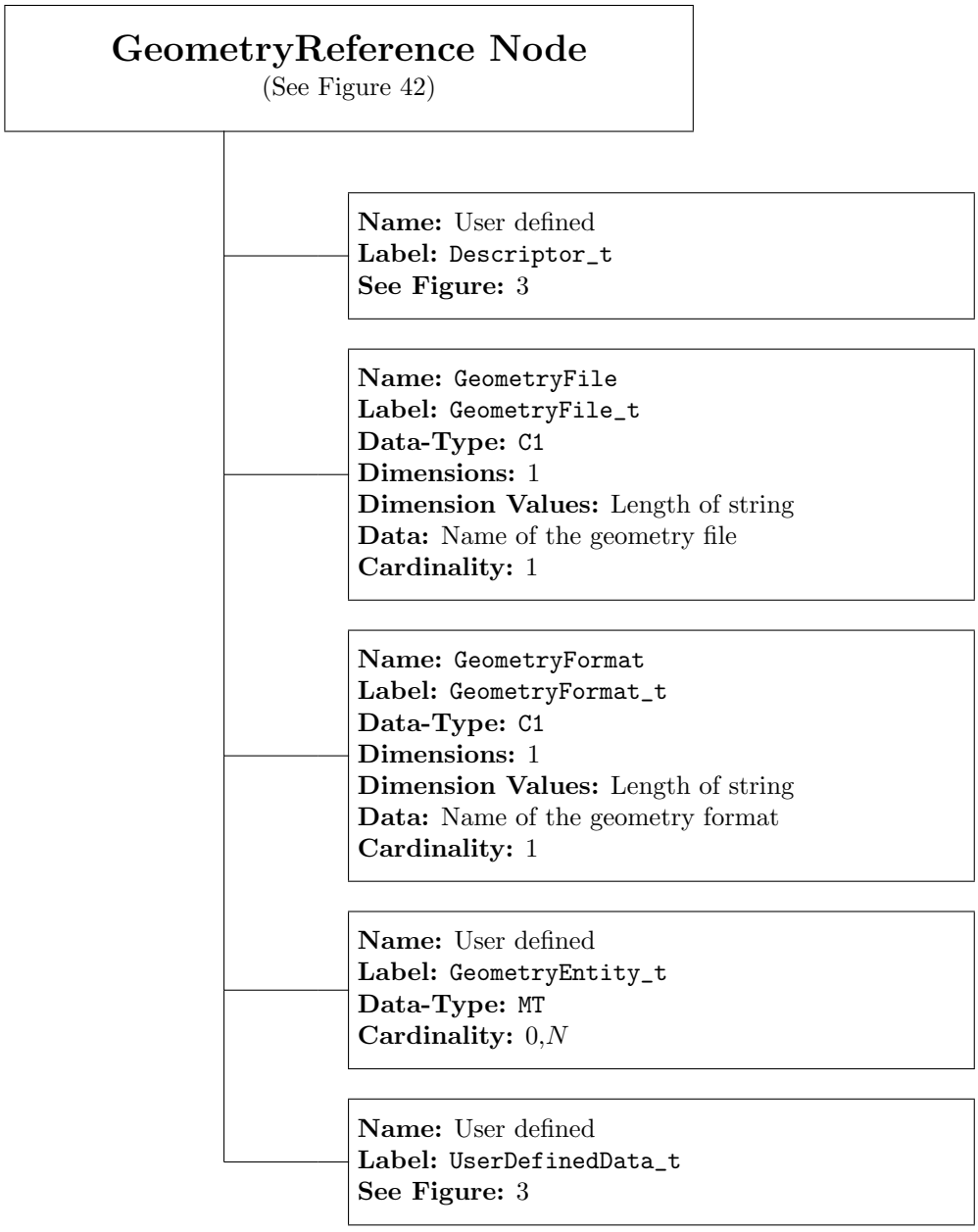


Figure 44: `GeometryReference_t` Data Structure

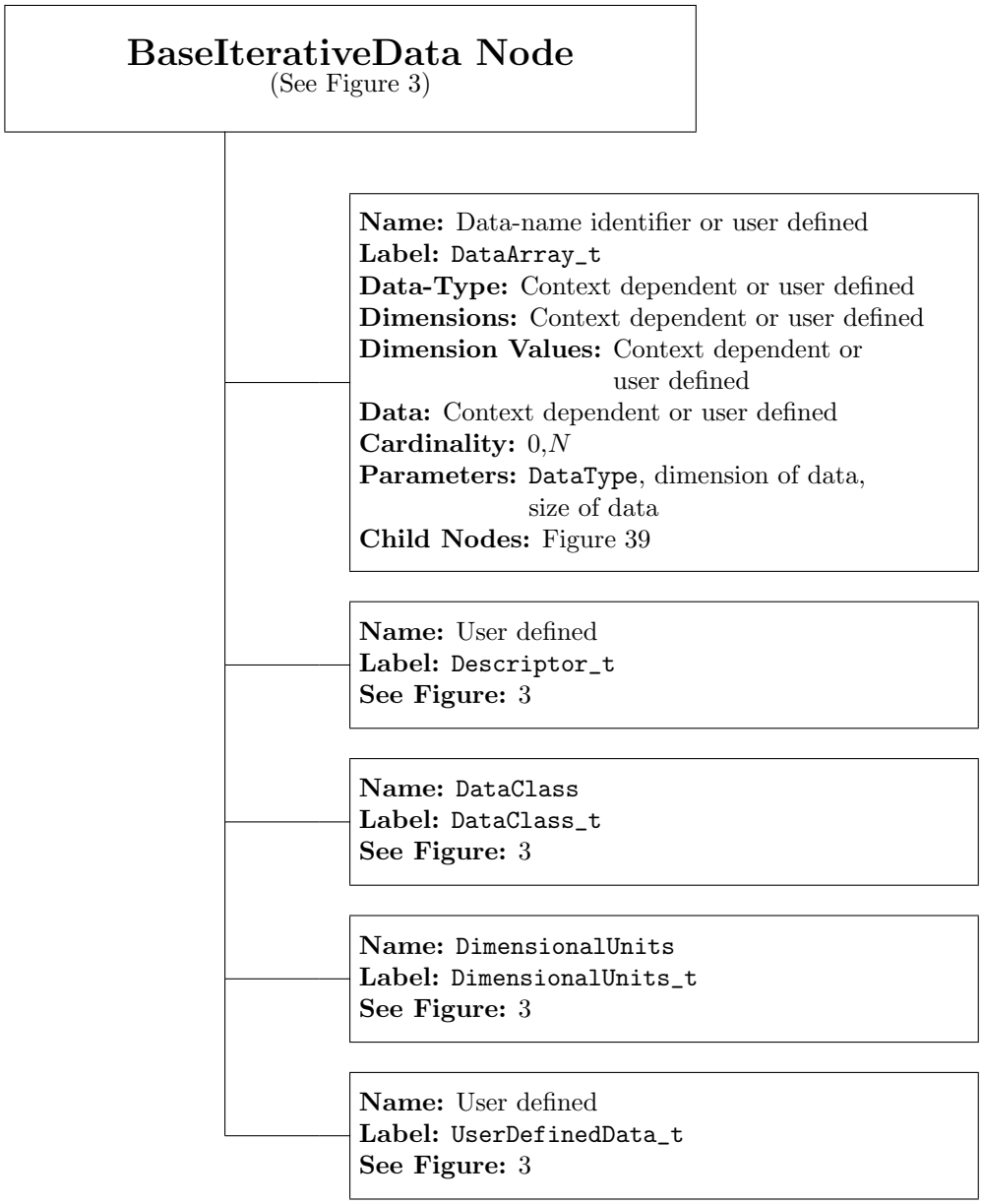


Figure 45: `BaseIterativeData_t` Data Structure

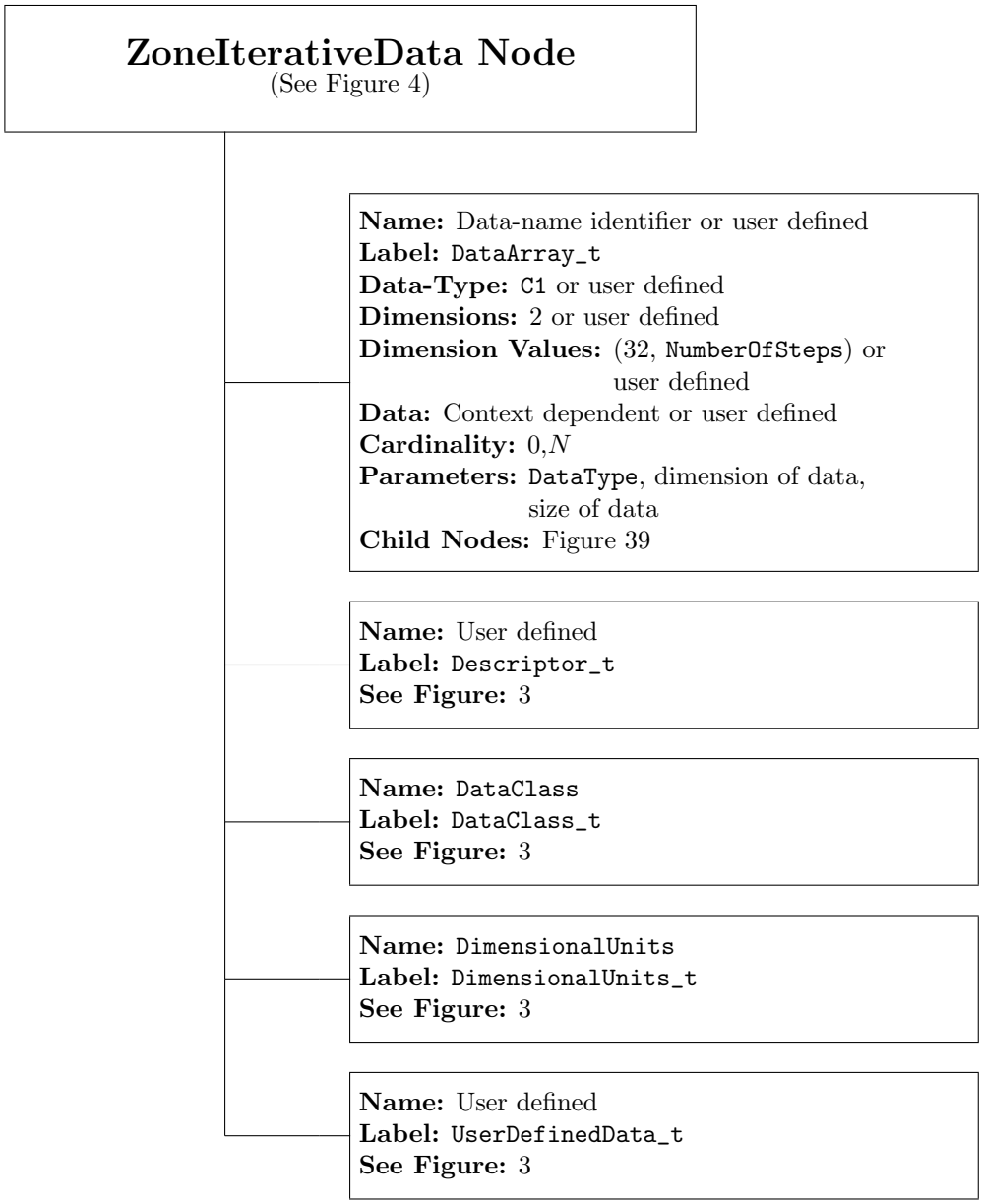


Figure 46: `ZoneIterativeData_t` Data Structure

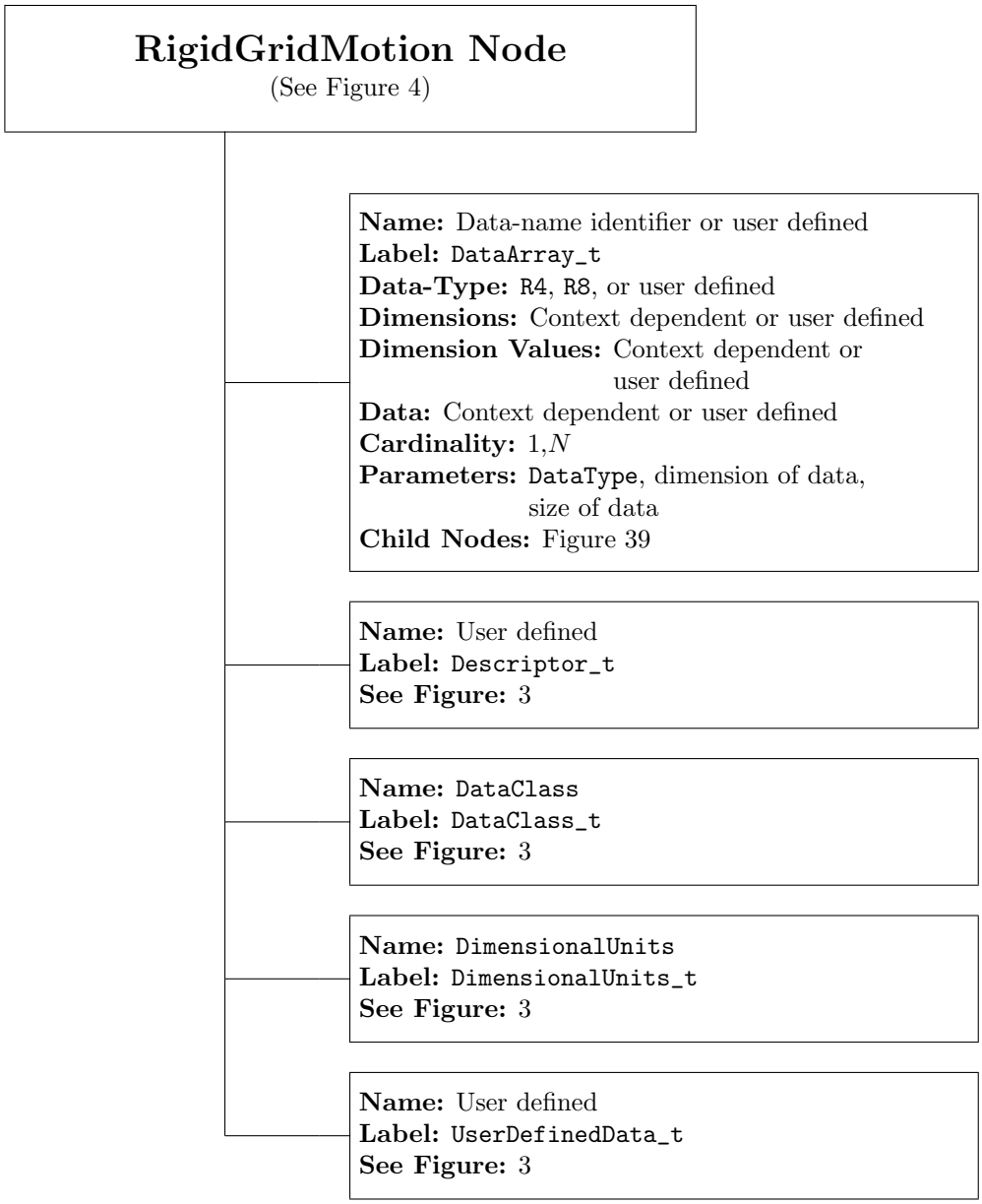


Figure 47: `RigidGridMotion_t` Data Structure

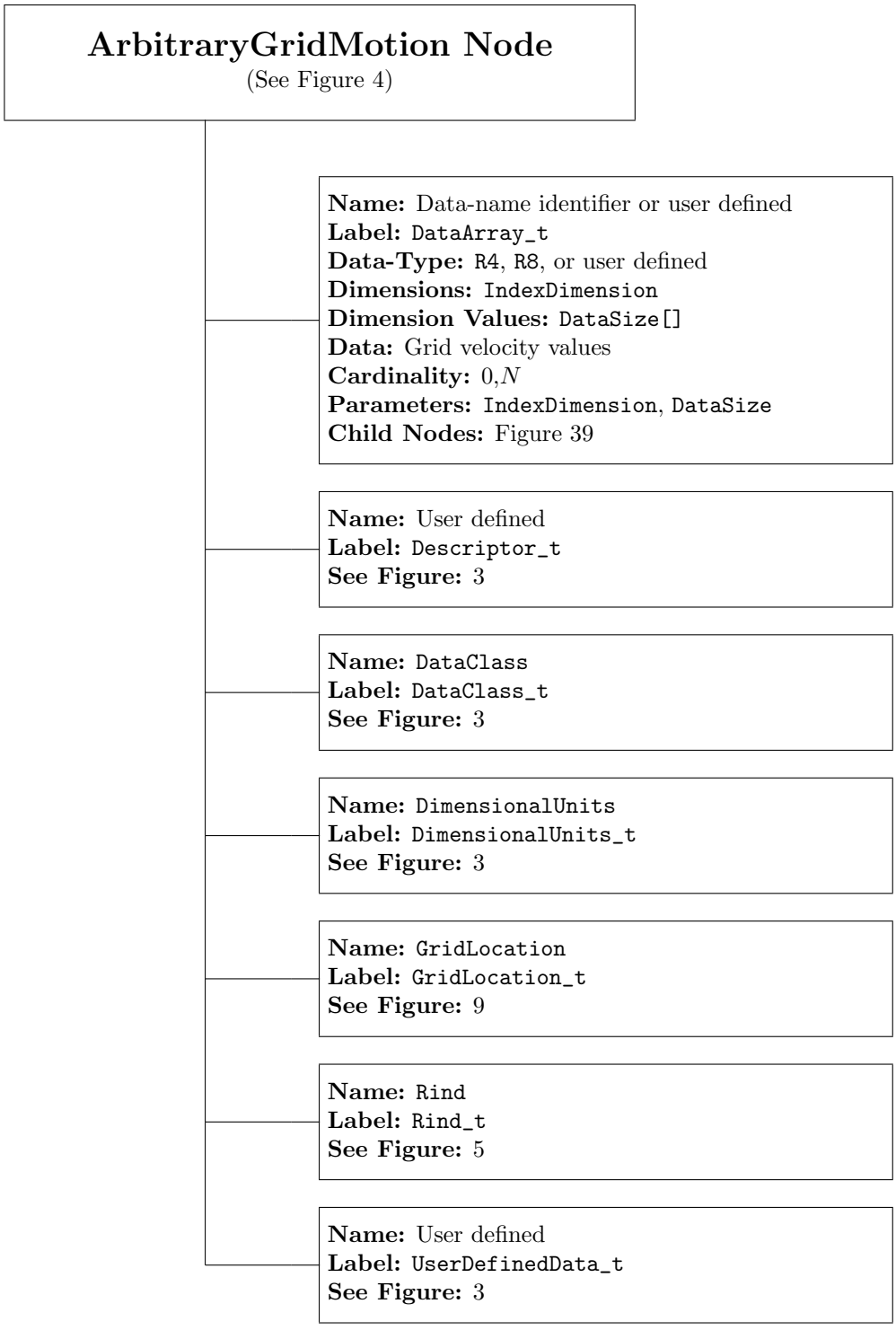


Figure 48: `ArbitraryGridMotion_t` Data Structure

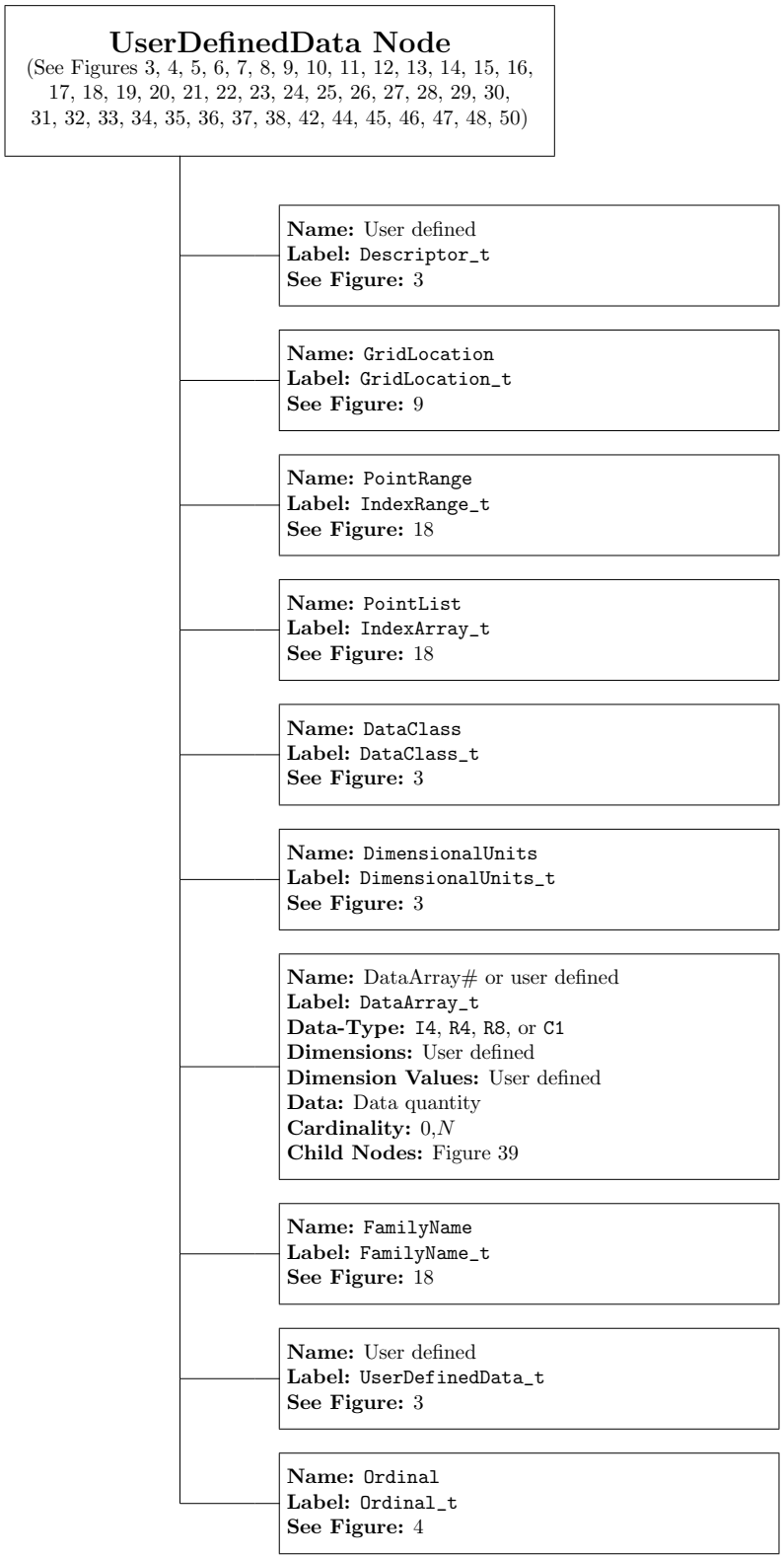


Figure 49: `UserDefinedData_t` Data Structure

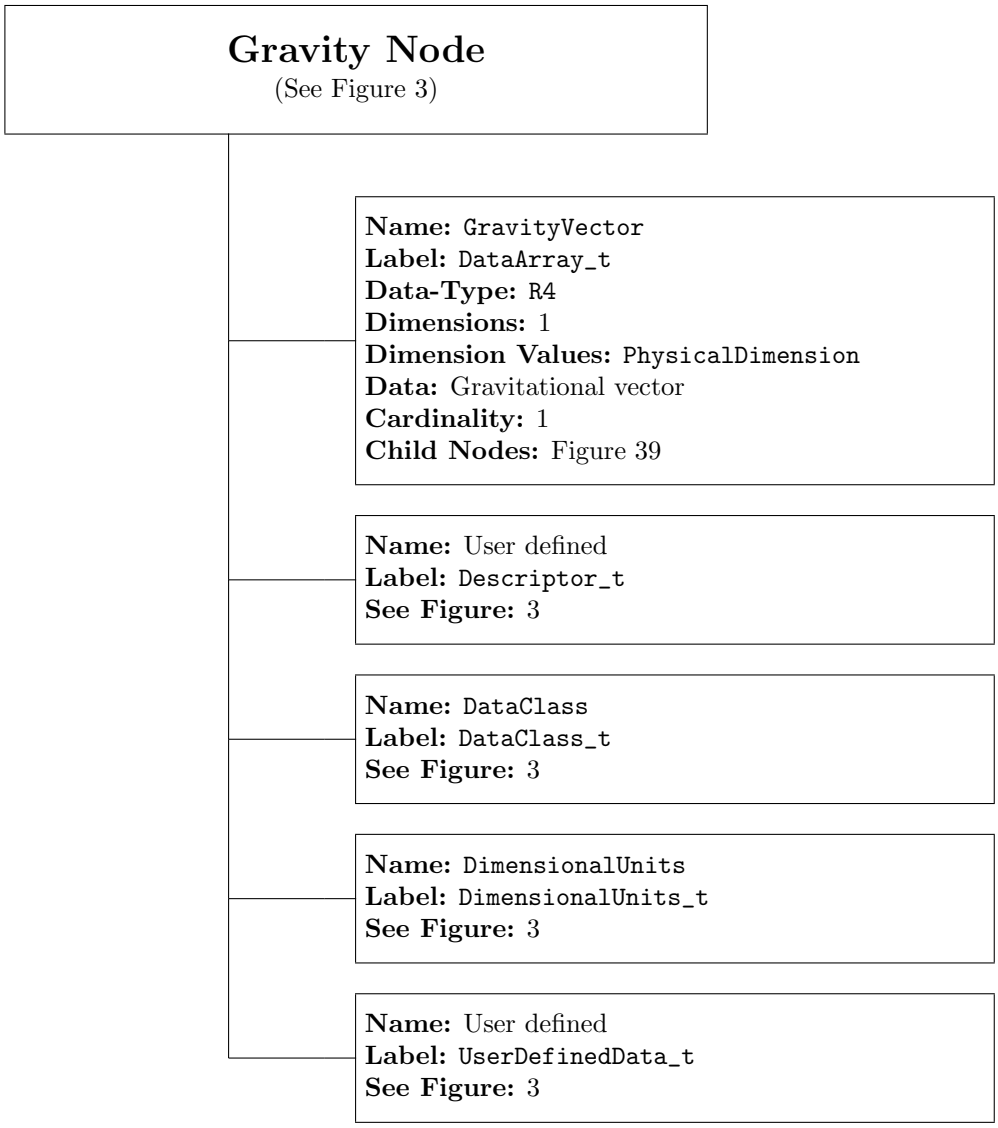


Figure 50: `Gravity_t` Data Structure