



# **CFD General Notation System SIDS File Mapping Manual**

Document Version 3.1.3

CGNS Version 3.1.3



# Contents

<b>1</b>	<b>Brief Description of CGNS</b>	<b>1</b>
<b>2</b>	<b>CGNS Documentation</b>	<b>3</b>
2.1	Description of the Documents . . . . .	3
2.1.1	CGNS Overview and Entry-Level Document . . . . .	3
2.1.2	A User's Guide to CGNS . . . . .	3
2.1.3	The CGNS Mid-Level Library . . . . .	3
2.1.4	Standard Interface Data Structures . . . . .	4
2.1.5	The SIDS File Mapping Manual . . . . .	4
2.1.6	CGIO User's Guide . . . . .	4
2.1.7	ADF Implementation . . . . .	4
2.1.8	HDF Implementation . . . . .	5
2.2	Which Documents Do You Need? . . . . .	5
2.2.1	Prospective Users . . . . .	5
2.2.2	End Users . . . . .	5
2.2.3	Applications Code Developers . . . . .	5
2.2.4	CGNS System Developers . . . . .	6
2.3	How to Use This Document . . . . .	6
<b>3</b>	<b>CGNS Background</b>	<b>7</b>
3.1	Purpose . . . . .	7
3.2	Participation and Brief History . . . . .	7
3.3	Scope . . . . .	8
<b>4</b>	<b>General CGNS File Mapping Concepts</b>	<b>9</b>
4.1	The Conceptual Structure of a CGNS File . . . . .	9
4.2	The Structure of a CGNS Node . . . . .	10
4.2.1	The Node Name . . . . .	10
4.2.2	The Label . . . . .	10
4.2.3	The Data Type . . . . .	11
4.2.4	The Number of Dimensions . . . . .	12
4.2.5	The Dimension Values . . . . .	12
4.2.6	The Data . . . . .	12
4.2.7	The Node ID . . . . .	12
4.2.8	The Child Table . . . . .	13
4.3	Use of Nodes in CGNS . . . . .	13
4.3.1	Cardinality . . . . .	13
4.3.2	Parameters . . . . .	13
4.3.3	Functions . . . . .	14
4.4	CGNS Databases . . . . .	14
4.4.1	Definition of a CGNS Database . . . . .	14
4.4.2	Location of CGNS Databases within a File . . . . .	14
4.4.3	File Management . . . . .	15
4.5	Internal Organization of a CGNS Database . . . . .	15
4.5.1	The <code>CGNSBase_t</code> Node . . . . .	15
4.5.2	The <code>CGNSLibraryVersion_t</code> Node . . . . .	15
4.5.3	Topological Basis of CGNS Database Organization . . . . .	15
4.5.4	Topics Not Currently Covered . . . . .	16
<b>5</b>	<b>Detailed CGNS Node Descriptions</b>	<b>17</b>

5.1	Basic CGNS Nodes . . . . .	17
5.1.1	Descriptor Group . . . . .	17
5.1.1.1	Descriptor_t . . . . .	17
5.1.1.2	Ordinal_t . . . . .	18
5.1.2	Physical Data Group . . . . .	19
5.1.2.1	DataClass_t . . . . .	19
5.1.2.2	DimensionalUnits_t . . . . .	19
5.1.2.3	AdditionalUnits_t . . . . .	20
5.1.2.4	DataConversion_t . . . . .	20
5.1.2.5	DimensionalExponents_t . . . . .	21
5.1.2.6	AdditionalExponents_t . . . . .	21
5.1.2.7	DataArray_t . . . . .	21
5.1.2.8	Integer Arrays . . . . .	22
5.1.3	Location and Position Group . . . . .	22
5.1.3.1	GridLocation_t . . . . .	22
5.1.3.2	Rind_t . . . . .	24
5.1.3.3	IndexRange_t . . . . .	24
5.1.3.4	IndexArray_t . . . . .	25
5.1.4	Auxiliary Data Group . . . . .	25
5.1.4.1	ReferenceState_t . . . . .	25
5.1.4.2	ConvergenceHistory_t . . . . .	26
5.1.4.3	IntegralData_t . . . . .	26
5.1.4.4	UserDefinedData_t . . . . .	26
5.1.4.5	Gravity_t . . . . .	27
5.2	Specialized Nodes . . . . .	27
5.2.1	Grid Specification . . . . .	27
5.2.1.1	GridCoordinates_t . . . . .	27
5.2.1.2	Elements_t . . . . .	28
5.2.1.3	Axisymmetry_t . . . . .	28
5.2.1.4	RotatingCoordinates_t . . . . .	28
5.2.2	Field Specification . . . . .	29
5.2.2.1	FlowSolution_t . . . . .	29
5.2.2.2	DiscreteData_t . . . . .	29
5.2.2.3	ZoneSubRegion_t . . . . .	30
5.2.3	Connectivity Group . . . . .	30
5.2.3.1	Transform Node . . . . .	30
5.2.3.2	GridConnectivityType_t . . . . .	31
5.2.3.3	GridConnectivity1to1_t . . . . .	31
5.2.3.4	GridConnectivity_t . . . . .	32
5.2.3.5	GridConnectivityProperty_t . . . . .	32
5.2.3.6	Periodic_t . . . . .	33
5.2.3.7	AverageInterface_t . . . . .	33
5.2.3.8	OversetHoles_t . . . . .	33
5.2.3.9	ZoneGridConnectivity_t . . . . .	34
5.2.4	Boundary Condition Group . . . . .	34
5.2.4.1	InwardNormalIndex . . . . .	34
5.2.4.2	InwardNormalList . . . . .	35
5.2.4.3	BCData_t . . . . .	35
5.2.4.4	BCDataSet_t . . . . .	35
5.2.4.5	BC_t . . . . .	36
5.2.4.6	ZoneBC_t . . . . .	36

5.2.4.7	BCProperty_t . . . . .	37
5.2.4.8	WallFunction_t . . . . .	37
5.2.4.9	Area_t . . . . .	37
5.2.5	Equation Specification Group . . . . .	37
5.2.5.1	GoverningEquations_t . . . . .	37
5.2.5.2	GasModel_t . . . . .	38
5.2.5.3	ViscosityModel_t . . . . .	38
5.2.5.4	EquationDimension . . . . .	38
5.2.5.5	ThermalConductivityModel_t . . . . .	39
5.2.5.6	TurbulenceClosure_t . . . . .	39
5.2.5.7	TurbulenceModel_t . . . . .	39
5.2.5.8	ThermalRelaxationModel_t . . . . .	39
5.2.5.9	ChemicalKineticsModel_t . . . . .	40
5.2.5.10	EMElectricFieldModel_t . . . . .	40
5.2.5.11	EMMagneticFieldModel_t . . . . .	40
5.2.5.12	EMConductivityModel_t . . . . .	41
5.2.5.13	FlowEquationSet_t . . . . .	41
5.2.6	Family Group . . . . .	41
5.2.6.1	Family_t . . . . .	41
5.2.6.2	FamilyName_t . . . . .	42
5.2.6.3	FamilyBC_t . . . . .	42
5.2.6.4	FamilyBCDataSet_t . . . . .	42
5.2.6.5	GeometryReference_t . . . . .	43
5.2.6.6	GeometryFile_t . . . . .	43
5.2.6.7	GeometryFormat_t . . . . .	43
5.2.6.8	GeometryEntity_t . . . . .	43
5.2.7	Time-Dependent Group . . . . .	44
5.2.7.1	BaseIterativeData_t . . . . .	44
5.2.7.2	ZoneIterativeData_t . . . . .	44
5.2.7.3	RigidGridMotion_t . . . . .	44
5.2.7.4	ArbitraryGridMotion_t . . . . .	45
5.2.8	Structural Nodes . . . . .	45
5.2.8.1	Zone_t . . . . .	45
5.2.8.2	CGNSBase_t . . . . .	45
5.2.8.3	SimulationType_t . . . . .	46
5.2.8.4	ZoneType_t . . . . .	46
5.2.8.5	CGNSLibraryVersion_t . . . . .	47

## A CGNS File Mapping Figures 49

### List of Figures

1	Example Hierarchy . . . . .	50
2	Example Node Structure . . . . .	51
3	ArbitraryGridMotion_t Data Structure . . . . .	52
4	Area_t Data Structure . . . . .	53
5	AverageInterface_t Data Structure . . . . .	54
6	Axisymmetry_t Data Structure . . . . .	55
7	BaseIterativeData_t Data Structure . . . . .	56
8	BC_t Data Structure . . . . .	57

9	BCData_t Data Structure . . . . .	59
10	BCDataSet_t Data Structure . . . . .	60
11	BCProperty_t Data Structure . . . . .	61
12	CGNSBase_t Data Structure . . . . .	62
13	ChemicalKineticsModel_t Data Structure . . . . .	64
14	ConvergenceHistory_t Data Structure . . . . .	65
15	DataArray_t Data Structure . . . . .	66
16	DimensionalExponents_t Data Structure . . . . .	67
17	DimensionalUnits_t Data Structure . . . . .	68
18	Elements_t Data Structure . . . . .	69
19	EMConductivityModel_t Data Structure . . . . .	70
20	EMElectricFieldModel_t Data Structure . . . . .	71
21	EMMagneticFieldModel_t Data Structure . . . . .	72
22	Family_t Data Structure . . . . .	73
23	FamilyBC_t Data Structure . . . . .	74
24	FamilyBCDataSet_t Data Structure . . . . .	75
25	FlowEquationSet_t Data Structure . . . . .	76
26	FlowSolution_t and DiscreteData_t Data Structures . . . . .	78
27	GasModel_t Data Structure . . . . .	79
28	GeometryReference_t Data Structure . . . . .	80
29	GoverningEquations_t Data Structure . . . . .	81
30	Gravity_t Data Structure . . . . .	82
31	GridConnectivity_t Data Structure . . . . .	83
32	GridConnectivityProperty_t Data Structure . . . . .	84
33	GridConnectivity1to1_t Data Structure . . . . .	85
34	GridCoordinates_t Data Structure . . . . .	86
35	IntegralData_t Data Structure . . . . .	87
36	OversetHoles_t Data Structure . . . . .	88
37	Periodic_t Data Structure . . . . .	89
38	ReferenceState_t Data Structure . . . . .	90
39	RigidGridMotion_t Data Structure . . . . .	91
40	RotatingCoordinates_t Data Structure . . . . .	92
41	ThermalConductivityModel_t Data Structure . . . . .	93
42	ThermalRelaxationModel_t Data Structure . . . . .	94
43	TurbulenceClosure_t Data Structure . . . . .	95
44	TurbulenceModel_t Data Structure . . . . .	96
45	UserDefinedData_t Data Structure . . . . .	97
46	ViscosityModel_t Data Structure . . . . .	98
47	WallFunction_t Data Structure . . . . .	99
48	Zone_t Data Structure . . . . .	100
49	ZoneBC_t Data Structure . . . . .	102
50	ZoneGridConnectivity_t Data Structure . . . . .	103
51	ZoneIterativeData_t Data Structure . . . . .	104
52	ZoneSubRegion_t Data Structure . . . . .	105

# 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). This document 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 a database manager. Adherence to the mapping conventions guarantees uniform meaning and location of CFD data within CGNS files, and thereby allows the construction of universal software to read and write the data. Physical data storage is accomplished via a general database manager, either ADF (Advanced Data Format) or HDF (Hierarchical Data Format).





## 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
- CGNS Mid-Level Library
- Standard Interface Data Structures
- SIDS File Mapping Manual (this document)
- CGIO User's Guide
- ADF Implementation
- HDF5 Implementation

### 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 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.

### 2.1.4 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 exactly parallels the File Mapping described by the current document. 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.

### 2.1.5 The SIDS File Mapping Manual

The *SIDS 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 a database manager. Adherence to the mapping conventions guarantees uniform meaning and location of CFD data within database files, and thereby allows the construction of universal software to read and write the data.

### 2.1.6 CGIO User’s Guide

The *CGIO User’s Guide* describes the interface between the SIDS File Mapping and an underlying database manager. It also documents the low-level routines that provide this functionality as used in the Mid-Level Library. This is the preferred way to access the database directly if additional requirements beyond those provided by the MLL are needed.

### 2.1.7 ADF Implementation

The *ADF Implementation* 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.

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*.

### 2.1.8 HDF Implementation

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 *HDF5 Implementation* document describes the CGNS conventions governing the placement of CFD-specific data within the HDF5 format.

## 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 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.

### 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 the *User's Guide to CGNS*, as well as 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 SIDS File Mapping documents will be useful. It will likely be necessary to consult the SIDS to determine the precise meaning of the nomenclature. In the unlikely case that the MLL does not provide all the functionality needed by the application, the *CGIO User's Guide* may be consulted for information on the low-level access to the CGNS database.

### 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 File Mapping* 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 General File Mapping Concepts in [Section 4](#). The detailed textual node descriptions in [Section 5](#) 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 General CGNS File Mapping Concepts

This document describes the general concepts behind and structure of a CGNS Database. The implementation details of reading and writing to a storage medium are left to an underlying database manager. The [CGIO User's Guide](#) provides the core routines to access the database manager at that level.

Since CGNS was originally developed based on the [ADF](#) (Advanced Data Format) as the database manager, much of this documentation parallels those concepts. With the introduction of the [HDF5](#) (Hierarchical Data Format) database manager, the concepts in this document have been "mapped" to HDF5.

We first describe the structure of the database, and the nodes from which it is constructed, followed by a description of the overall layout of the tree structure itself.

The section Detailed CGNS Node Descriptions ([Section 5](#)) describes the exact conventions for each type of data.

### 4.1 The Conceptual Structure of a CGNS File

Although the physical structure of an CGNS file in storage is (or should be) of little concern to users of CGNS, 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 low-level database access routines.

An CGNS 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. 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 a CGNS file has exactly the same internal structure. Each node contains identifying information, pointers to any children, and, optionally, data.

When a CGNS file is opened (by an appropriate database manager), 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 CGNS file, or in another part of the same CGNS 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 a CGNS file to span multiple physical files, and also allows a portion of one CGNS file to be referenced by several other CGNS files.

### 4.2 The Structure of a CGNS 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 recorded. 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 the underlying database manager itself. These are:

- The Node ID
- The Child Table

#### 4.2.1 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.

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 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.

#### 4.2.2 The Label

The Label is a 32-byte character field which is used 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.

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.

#### 4.2.3 The Data Type

The Data Type is a 2-byte character field which specifies the type and precision of any data which is stored in the data field. The data types allowed by CGNS are:

**Table 1: Data Types**

<b>Data Type</b>	<b>Notation</b>
No Data	MT
Link	LK
Integer 32	I4
Integer 64	I8
Real 32	R4
Real 64	R8
Character	C1

The MT node contains no data, thus the number of dimensions and dimension values are ignored. This node is typically used as a container for children nodes.

The LK node is a link to a node within the current or an external database. The actual data stored with the node is a file name and node path name to the link (as C1 data). This information is available only through the core link access routines. A normal read of a LK node will return the node properties (with the exception of the Name) from the linked-to node.

The specification of data types within the File Mapping allows for the probability that files

written under different circumstances, or on different platforms, may differ in precision or format. It is the responsibility of the database manager to provide any data conversions required, and to ensure precision of the data.

### 4.2.4 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. The limit for the Number of Dimensions is 12.

Whenever data is stored at a node, it is in the form of a single array of elements of a single data type. *Note* that Fortran-indexing conventions are used within CGNS. For multi-dimensional data, this means that the first index of the array varies the fastest, and indexing starts at 1. This is important to understand when reading the data into multi-dimensional “C” arrays, since the indexing order is opposite to that as used by CGNS.

Another thing to note, is that character data (C1) is not stored with a “C” terminating ‘0’ character, but rather is padded out with blanks (space character) as in Fortran character data. The MLL, however, will add the terminating-‘0’ to the strings when returning this data to a “C” application.

### 4.2.5 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. With the introduction of 64-bit capability with CGNS Version 3.1.3, these dimensions are stored internally as 64-bit integers, regardless of the compilation mode (32 or 64-bit mode). They are returned to the application, however, as *cgsizes\_t*.

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 newline characters; this means an entire document can be stored in the data field.

### 4.2.6 The Data

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

CGNS imposes no conventions on the data itself. *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.

### 4.2.7 The Node ID

The node ID is a unique identifier assigned to each existing node by a database manager when the file containing it is opened, and to new nodes as they are created. Core routine 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 a further search. The Node ID is a 64-bit real and is not under user control.

### 4.2.8 The Child Table

The database manager maintains a table recording the children of each node, which is adjusted when children are added or deleted.

Children may be identified by their names and labels, and, thence, by their node IDs once these have been determined. There is no provision for the notion of order among children. In particular, the order of a list of children returned by the database manager may or may not have anything 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 modified.

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.

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

In addition to the meaning of attributes of individual 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.

## 4.3 Use of Nodes in CGNS

There are 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 are often needed by CGNS to properly interpret the data; namely,

- Cardinality
- Parameters
- Functions

### 4.3.1 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 ( $N$ ).

### 4.3.2 Parameters

CGNS relies on the fact that 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*.

### 4.3.3 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.

## 4.4 CGNS Databases

### 4.4.1 Definition of a CGNS Database

A CGNS database is defined by the existence of a node or nodes 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 is not the actual root of the database file. However, current CGNS conventions require that it be located directly below the database root node which is identified by the name “/”.

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 *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 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 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 a 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 File Mapping.*

### 4.4.2 Location of CGNS Databases within a File

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

A CGNS database may link to CGNS nodes in the same or other 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 file into which the link is made is invisible except below the node to which the link is made.

### 4.4.3 File Management

Beyond *Open* and *Close* neither CGNS or the database manager provides any file management facilities. The user is responsible for ensuring that:

- The file containing the root of the required database is available and its permissions are properly set at runtime.
- If links are made to other 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”.

## 4.5 Internal Organization of a CGNS Database

### 4.5.1 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.

### 4.5.2 The CGNSLibraryVersion\_t Node

An 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 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.

### 4.5.3 Topological Basis of CGNS Database Organization

Below the root, the organization of a CGNS database reflects the problem topology. Omitting detail, [Figure 1](#) shows the overall structure of the CGNS file. Below the 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 48](#)). The CGNS convention is that such a node overrides (for the associated portion of the topology only) any data found at a higher level.

### 4.5.4 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.

## 5 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 5.1](#) defines a number of basic types which recur often in the structure, and [Section 5.2](#) describes higher level nodes of more specific function built from those in [Section 5.1](#).<sup>1</sup>

### 5.1 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.

#### 5.1.1 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.

##### 5.1.1.1 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.

---

<sup>1</sup>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.

## SIDS File Mapping Manual

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

<b>Name:</b>	User defined
<b>Label:</b>	<code>Descriptor_t</code>
<b>DataType:</b>	<code>C1</code>
<b>Dimension:</b>	1
<b>Dimension Values:</b>	Length of string to be stored, including any carriage control or null bytes.
<b>Data:</b>	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.
<b>Children:</b>	None
<b>Cardinality:</b>	0,N

### 5.1.1.2 `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

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



### 5.1.2 Physical Data Group

#### 5.1.2.1 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

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

#### 5.1.2.2 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

<b>Name:</b>	<code>DimensionalUnits</code>
<b>Label:</b>	<code>DimensionalUnits_t</code>
<b>DataType:</b>	<code>C1</code>
<b>Dimension:</b>	2
<b>Dimension Values:</b>	(32,5)
<b>Data:</b>	For mass, one of: <code>MassUnitsNull</code> , <code>MassUnitsUserDefined</code> , <code>Kilogram</code> , <code>Gram</code> , <code>Slug</code> , <code>Pound-Mass</code>
	For length, one of: <code>LengthUnitsNull</code> , <code>LengthUnitsUserDefined</code> , <code>Meter</code> , <code>Centimeter</code> , <code>Millimeter</code> , <code>Foot</code> , <code>Inch</code>
	For time, one of: <code>TimeUnitsNull</code> , <code>TimeUnitsUserDefined</code> , <code>Second</code>
	For temperature, one of: <code>TemperatureUnitsNull</code> , <code>TemperatureUnitsUserDefined</code> , <code>Kelvin</code> , <code>Celsius</code> , <code>Rankine</code> , <code>Fahrenheit</code>
	For angles, one of: <code>AngleUnitsNull</code> , <code>AngleUnitsUserDefined</code> , <code>Degree</code> , <code>Radian</code>
<b>Children:</b>	See <a href="#">Figure 17</a>
<b>Cardinality:</b>	0,1

### 5.1.2.3 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

<b>Name:</b>	AdditionalUnits	
<b>Label:</b>	AdditionalUnits_t	
<b>DataType:</b>	C1	
<b>Dimension:</b>	2	
<b>Dimension Values:</b>	(32,3)	
<b>Data:</b>	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
<b>Children:</b>	None	
<b>Cardinality:</b>	0,1	

### 5.1.2.4 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

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

## 5.1.2.5 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*

<b>Name:</b>	DimensionalExponents
<b>Label:</b>	DimensionalExponents_t
<b>DataType:</b>	R4 or R8
<b>Dimension:</b>	1
<b>Dimension Values:</b>	5
<b>Data:</b>	MassExponent, LengthExponent, TimeExponent, TemperatureExponent, AngleExponent
<b>Children:</b>	See <a href="#">Figure 16</a>
<b>Cardinality:</b>	0,1

## 5.1.2.6 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*

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

## 5.1.2.7 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 “**In**”, “**Rn**”, or “**Cn**”.

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

<b>Name:</b>	Context dependent
<b>Label:</b>	<code>DataArray_t</code>
<b>DataType:</b>	Context dependent
<b>Dimension:</b>	Context dependent
<b>Dimension Values:</b>	Context dependent
<b>Data:</b>	The array of data values
<b>Children:</b>	See <a href="#">Figure 15</a>
<b>Cardinality:</b>	0,N
<b>Parameters:</b>	<code>DataType</code> , dimension of the data, size of the data

#### 5.1.2.8 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[]` or `csize_t[]`, with the dimensions and values given either explicitly in the appropriate fields, or as parameters or functions. The `csize_t` data type is used wherever there is the potential of exceeding the size limit for a 32-bit integer (I4), such as `PointList`, `PointRange`, and `ElementRange`.

### Node Attributes

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

#### 5.1.3 Location and Position Group

##### 5.1.3.1 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

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

### 5.1.3.2 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 5.2.8.1](#)). 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 **DataSet** 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

<b>Name:</b>	Rind
<b>Label:</b>	Rind_t
<b>DataType:</b>	I4
<b>Dimension:</b>	1
<b>Dimension Values:</b>	2*IndexDimension
<b>Data:</b>	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)
<b>Children:</b>	None
<b>Cardinality:</b>	0,1
<b>Parameters:</b>	IndexDimension

### 5.1.3.3 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

<b>Name:</b>	PointRange, PointRangeDonor, ElementRange, or user defined
<b>Label:</b>	IndexRange_t
<b>DataType:</b>	<i>cgsizes_t</i>
<b>Dimension:</b>	2

<b>Dimension Values:</b>	IndexDimension, 2
<b>Data:</b>	First indices, last indices
<b>Children:</b>	None
<b>Cardinality:</b>	Context dependent
<b>Parameters:</b>	IndexDimension

#### 5.1.3.4 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 than only the first and last ones. Its use is similar to `IndexRange_t`.

##### Node Attributes

<b>Name:</b>	PointList, PointListDonor, CellListDonor, or InwardNormalList
<b>Label:</b>	IndexArray_t
<b>DataType:</b>	<i>csize_t</i> , or (for InwardNormalList) R4 or R8
<b>Dimension:</b>	2
<b>Dimension Values:</b>	IndexDimension, number of items in the list; or (for InwardNormalList) PhysicalDimension, number of items in the list
<b>Data:</b>	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
<b>Children:</b>	None
<b>Cardinality:</b>	0,1
<b>Parameters:</b>	IndexDimension, either PointListSize or ListLength, and DataType; or (for InwardNormalList) PhysicalDimension, ListLength, and DataType

#### 5.1.4 Auxiliary Data Group

##### 5.1.4.1 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

<b>Name:</b>	ReferenceState
<b>Label:</b>	ReferenceState_t
<b>DataType:</b>	MT
<b>Children:</b>	See <a href="#">Figure 38</a>
<b>Cardinality:</b>	0,1

### 5.1.4.2 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

<b>Name:</b>	GlobalConvergenceHistory if under a <b>CGNSBase_t</b> node; ZoneConvergenceHistory if under a <b>Zone_t</b> node
<b>Label:</b>	ConvergenceHistory_t
<b>DataType:</b>	I4
<b>Dimension:</b>	1
<b>Dimension Values:</b>	1
<b>Data:</b>	Number of iterations
<b>Children:</b>	See <a href="#">Figure 14</a>
<b>Cardinality:</b>	0,1

### 5.1.4.3 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

<b>Name:</b>	User defined
<b>Label:</b>	IntegralData_t
<b>DataType:</b>	MT
<b>Children:</b>	See <a href="#">Figure 35</a>
<b>Cardinality:</b>	0,N

### 5.1.4.4 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

<b>Name:</b>	User defined
<b>Label:</b>	UserDefinedData_t
<b>DataType:</b>	MT
<b>Children:</b>	See <a href="#">Figure 45</a>



**Cardinality:** 0, $N$

#### 5.1.4.5 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 30](#)  
**Cardinality:** 0,1

## 5.2 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.

### 5.2.1 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 5.2.2](#)).

#### 5.2.1.1 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 specify a node location. For a structured grid, `IndexDimension` varies 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 5.2.2](#)), `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 `DataSetSize[]`.

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

<b>Name:</b>	GridCoordinates or user defined
<b>Label:</b>	GridCoordinates_t
<b>DataType:</b>	MT
<b>Children:</b>	See <a href="#">Figure 34</a>
<b>Cardinality:</b>	0,N
<b>Parameters:</b>	IndexDimension, VertexSize
<b>Functions:</b>	DataSetSize

#### 5.2.1.2 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

<b>Name:</b>	User defined
<b>Label:</b>	Elements_t
<b>DataType:</b>	I4
<b>Dimension:</b>	1
<b>Dimension Values:</b>	2
<b>Data:</b>	ElementType value, ElementSizeBoundary
<b>Children:</b>	See <a href="#">Figure 18</a>
<b>Cardinality:</b>	0,N

#### 5.2.1.3 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

<b>Name:</b>	Axisymmetry
<b>Label:</b>	Axisymmetry_t
<b>DataType:</b>	MT
<b>Children:</b>	See <a href="#">Figure 6</a>
<b>Cardinality:</b>	0,1

#### 5.2.1.4 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

<b>Name:</b>	RotatingCoordinates
<b>Label:</b>	RotatingCoordinates_t
<b>DataType:</b>	MT
<b>Children:</b>	See <a href="#">Figure 40</a>
<b>Cardinality:</b>	0,1

**5.2.2 Field Specification**

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

**5.2.2.1 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

<b>Name:</b>	User defined
<b>Label:</b>	FlowSolution_t
<b>DataType:</b>	MT
<b>Children:</b>	See <a href="#">Figure 26</a>
<b>Cardinality:</b>	0,N
<b>Parameters:</b>	IndexDimension, VertexSize, CellSize
<b>Functions:</b>	DataSize

**5.2.2.2 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

<b>Name:</b>	User defined
<b>Label:</b>	DiscreteData_t
<b>DataType:</b>	MT
<b>Children:</b>	See <a href="#">Figure 26</a>
<b>Cardinality:</b>	0,N
<b>Parameters:</b>	IndexDimension, VertexSize, CellSize
<b>Functions:</b>	DataSize

#### 5.2.2.3 ZoneSubRegion\_t

The ZoneSubRegion\_t node allows for the ability to give flowfield or other information over a subset of the entire zone in a CGNS file. This subset may be over a portion of a boundary, or it may be over a portion of the entire field.

### Node Attributes

<b>Name:</b>	User defined
<b>Label:</b>	ZoneSubRegion_t
<b>DataType:</b>	I4
<b>Dimension:</b>	1
<b>Dimension Values:</b>	RegionCellDimension
<b>Children:</b>	See <a href="#">Figure 52</a>
<b>Cardinality:</b>	0,N
<b>Parameters:</b>	IndexDimension, VertexSize, CellSize
<b>Functions:</b>	DataSize

### 5.2.3 Connectivity Group

#### 5.2.3.1 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,  $\pm 1$ ,  $\pm 2$ , and  $\pm 3$ . 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

<b>Name:</b>	Transform
<b>Label:</b>	“int[IndexDimension]”
<b>DataType:</b>	I4
<b>Dimension:</b>	1
<b>Dimension Values:</b>	IndexDimension
<b>Data:</b>	Transformation matrix (shorthand)
<b>Children:</b>	None
<b>Cardinality:</b>	0,1

**Parameters:**                      `IndexDimension`

### 5.2.3.2 `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*

<b>Name:</b>	<code>GridConnectivityType</code>
<b>Label:</b>	<code>GridConnectivityType_t</code>
<b>DataType:</b>	<code>C1</code>
<b>Dimension:</b>	<code>1</code>
<b>Dimension Values:</b>	Length of string
<b>Data:</b>	<code>Abutting</code> , <code>Abutting1to1</code> , or <code>Overset</code>
<b>Children:</b>	<code>None</code>
<b>Cardinality:</b>	<code>0,1</code>

### 5.2.3.3 `GridConnectivity1to1_t`

This node is a shorthand format of `GridConnectivity_t` (see [Section 5.2.3.4](#)) 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

<b>Name:</b>	User defined
<b>Label:</b>	<code>GridConnectivity1to1_t</code>
<b>Data Type:</b>	C1
<b>Dimension:</b>	1
<b>Dimension Values:</b>	Length of string
<b>Data:</b>	<code>ZoneDonorName</code>
<b>Children:</b>	See <a href="#">Figure 33</a>
<b>Cardinality:</b>	0,N
<b>Parameters:</b>	<code>IndexDimension</code>

#### 5.2.3.4 `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

<b>Name:</b>	User defined
<b>Label:</b>	<code>GridConnectivity_t</code>
<b>Data Type:</b>	C1
<b>Dimension:</b>	1
<b>Dimension Values:</b>	Length of string
<b>Data:</b>	<code>ZoneDonorName</code>
<b>Children:</b>	See <a href="#">Figure 31</a>
<b>Cardinality:</b>	0,N
<b>Parameters:</b>	<code>IndexDimension</code> , <code>CellDimension</code>
<b>Functions:</b>	<code>PointListSize</code>

#### 5.2.3.5 `GridConnectivityProperty_t`

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

### Node Attributes

<b>Name:</b>	<code>GridConnectivityProperty</code>
<b>Label:</b>	<code>GridConnectivityProperty_t</code>

<b>DataType:</b>	MT
<b>Children:</b>	See <a href="#">Figure 32</a>
<b>Cardinality:</b>	0,1

#### 5.2.3.6 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

<b>Name:</b>	Periodic
<b>Label:</b>	Periodic_t
<b>DataType:</b>	MT
<b>Children:</b>	See <a href="#">Figure 37</a>
<b>Cardinality:</b>	0,1

#### 5.2.3.7 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

<b>Name:</b>	AverageInterface
<b>Label:</b>	AverageInterface_t
<b>DataType:</b>	MT
<b>Children:</b>	See <a href="#">Figure 5</a>
<b>Cardinality:</b>	0,1

#### 5.2.3.8 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

<b>Name:</b>	User defined
<b>Label:</b>	OversetHoles_t
<b>DataType:</b>	MT
<b>Children:</b>	See <a href="#">Figure 36</a>
<b>Cardinality:</b>	0,N
<b>Parameters:</b>	IndexDimension

## 5.2.3.9 ZoneGridConnectivity\_t

Each `Zone_t` node may have zero or more children of type `ZoneGridConnectivity_t`. It holds no data, but serves as the point below which all connectivity data associated with the zone can be found. Multiple `ZoneGridConnectivity_t` nodes may be used to specify time-dependent changes in the connectivity information.

### Node Attributes

<b>Name:</b>	ZoneGridConnectivity or user defined
<b>Label:</b>	ZoneGridConnectivity_t
<b>DataType:</b>	MT
<b>Children:</b>	See <a href="#">Figure 50</a>
<b>Cardinality:</b>	0,N
<b>Parameters:</b>	IndexDimension, CellDimension

## 5.2.4 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.

### 5.2.4.1 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

<b>Name:</b>	InwardNormalIndex
<b>Label:</b>	“int[IndexDimension]”



<b>DataType:</b>	I4
<b>Dimension:</b>	1
<b>Dimension Values:</b>	IndexDimension
<b>Data:</b>	Index of inward normal
<b>Children:</b>	None
<b>Cardinality:</b>	0,1
<b>Parameters:</b>	IndexDimension

#### 5.2.4.2 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

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

#### 5.2.4.3 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

<b>Name:</b>	DirichletData or NeumannData
<b>Label:</b>	BCData_t
<b>DataType:</b>	MT
<b>Children:</b>	See <a href="#">Figure 9</a>
<b>Cardinality:</b>	0,1
<b>Parameters:</b>	ListLength

#### 5.2.4.4 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

<b>Name:</b>	User defined
<b>Label:</b>	BCDataSet_t
<b>DataType:</b>	C1
<b>Dimension:</b>	1
<b>Dimension Values:</b>	Length of string
<b>Data:</b>	BTypeSimple value
<b>Children:</b>	See <a href="#">Figure 10</a>
<b>Cardinality:</b>	0,N
<b>Functions:</b>	ListLength
<b>Parameters:</b>	ListLength

### 5.2.4.5 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

<b>Name:</b>	User defined
<b>Label:</b>	BC_t
<b>DataType:</b>	C1
<b>Dimension:</b>	1
<b>Dimension Values:</b>	Length of string
<b>Data:</b>	BType value
<b>Children:</b>	See <a href="#">Figure 8</a>
<b>Cardinality:</b>	0,N
<b>Parameters:</b>	IndexDimension, PhysicalDimension

### 5.2.4.6 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

<b>Name:</b>	ZoneBC
<b>Label:</b>	ZoneBC_t
<b>DataType:</b>	MT
<b>Children:</b>	See <a href="#">Figure 49</a>
<b>Cardinality:</b>	0,1
<b>Parameters:</b>	IndexDimension, PhysicalDimension

## 5.2.4.7 BCProperty\_t

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

*Node Attributes*

<b>Name:</b>	BCProperty
<b>Label:</b>	BCProperty_t
<b>DataType:</b>	MT
<b>Children:</b>	See <a href="#">Figure 11</a>
<b>Cardinality:</b>	0,1

## 5.2.4.8 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*

<b>Name:</b>	WallFunction
<b>Label:</b>	WallFunction_t
<b>DataType:</b>	MT
<b>Children:</b>	See <a href="#">Figure 47</a>
<b>Cardinality:</b>	0,1

## 5.2.4.9 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*

<b>Name:</b>	Area
<b>Label:</b>	Area_t
<b>DataType:</b>	MT
<b>Children:</b>	See <a href="#">Figure 4</a>
<b>Cardinality:</b>	0,1

## 5.2.5 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`.

## 5.2.5.1 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*

<b>Name:</b>	GoverningEquations
<b>Label:</b>	GoverningEquations_t

<b>DataType:</b>	C1
<b>Dimension:</b>	1
<b>Dimension Values:</b>	Length of string
<b>Data:</b>	GoverningEquationsType value
<b>Children:</b>	See <a href="#">Figure 29</a>
<b>Cardinality:</b>	0,1
<b>Parameters:</b>	CellDimension

### 5.2.5.2 GasModel\_t

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

#### Node Attributes

<b>Name:</b>	GasModel
<b>Label:</b>	GasModel_t
<b>DataType:</b>	C1
<b>Dimension:</b>	1
<b>Dimension Values:</b>	Length of string
<b>Data:</b>	GasModelType value
<b>Children:</b>	See <a href="#">Figure 27</a>
<b>Cardinality:</b>	0,1

### 5.2.5.3 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

<b>Name:</b>	ViscosityModel
<b>Label:</b>	ViscosityModel_t
<b>DataType:</b>	C1
<b>Dimension:</b>	1
<b>Dimension Values:</b>	Length of string
<b>Data:</b>	ViscosityModelType value
<b>Children:</b>	See <a href="#">Figure 46</a>
<b>Cardinality:</b>	0,1

### 5.2.5.4 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  $k$ - $\epsilon$  turbulence model in three dimensions, it is 7.

#### Node Attributes

<b>Name:</b>	EquationDimension
<b>Label:</b>	"int"
<b>DataType:</b>	I4
<b>Dimension:</b>	1
<b>Dimension Values:</b>	1
<b>Data:</b>	EquationDimension value
<b>Children:</b>	None
<b>Cardinality:</b>	0,1

## 5.2.5.5 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*

<b>Name:</b>	ThermalConductivityModel
<b>Label:</b>	ThermalConductivityModel_t
<b>DataType:</b>	C1
<b>Dimension:</b>	1
<b>Dimension Values:</b>	Length of string
<b>Data:</b>	ThermalConductivityModelType value
<b>Children:</b>	See <a href="#">Figure 41</a>
<b>Cardinality:</b>	0,1

## 5.2.5.6 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*

<b>Name:</b>	TurbulenceClosure
<b>Label:</b>	TurbulenceClosure_t
<b>DataType:</b>	C1
<b>Dimension:</b>	1
<b>Dimension Values:</b>	Length of string
<b>Data:</b>	TurbulenceClosureType value
<b>Children:</b>	See <a href="#">Figure 43</a>
<b>Cardinality:</b>	0,1

## 5.2.5.7 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*

<b>Name:</b>	TurbulenceModel
<b>Label:</b>	TurbulenceModel_t
<b>DataType:</b>	C1
<b>Dimension:</b>	1
<b>Dimension Values:</b>	Length of string
<b>Data:</b>	TurbulenceModelType value
<b>Children:</b>	See <a href="#">Figure 44</a>
<b>Cardinality:</b>	0,1
<b>Parameters:</b>	CellDimension

## 5.2.5.8 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 42](#)  
**Cardinality:** 0,1

#### 5.2.5.9 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 13](#)  
**Cardinality:** 0,1

#### 5.2.5.10 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 20](#)  
**Cardinality:** 0,1

#### 5.2.5.11 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 21](#)

**Cardinality:** 0,1

#### 5.2.5.12 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 19](#)  
**Cardinality:** 0,1

#### 5.2.5.13 FlowEquationSet\_t

A node of type `FlowEquationSet_t` appears either at the highest level of the tree (under `CGNS-Base_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 25](#)  
**Cardinality:** 0,1  
**Parameters:** CellDimension

### 5.2.6 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 be 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.

#### 5.2.6.1 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

<b>Label:</b>	Family_t
<b>DataType:</b>	MT
<b>Children:</b>	See <a href="#">Figure 22</a>
<b>Cardinality:</b>	0,N

### 5.2.6.2 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

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

### 5.2.6.3 FamilyBC\_t

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

#### Node Attributes

<b>Name:</b>	FamilyBC
<b>Label:</b>	FamilyBC_t
<b>DataType:</b>	C1
<b>Dimension:</b>	1
<b>Dimension Values:</b>	Length of string
<b>Data:</b>	BCType value
<b>Children:</b>	See <a href="#">Figure 23</a>
<b>Cardinality:</b>	0,1

### 5.2.6.4 FamilyBCDataSet\_t

A **FamilyBCDataSet\_t** node contains Dirichlet and Neumann data for a single set of boundary condition equations to be applied to a Family. It’s intended use is for simple boundary condition types, where the equations imposed do not depend on local flow conditions.

#### Node Attributes

<b>Name:</b>	User defined
<b>Label:</b>	FamilyBCDataSet_t
<b>DataType:</b>	C1
<b>Dimension:</b>	1
<b>Dimension Values:</b>	Length of string
<b>Data:</b>	BCTypeSimple value
<b>Children:</b>	See <a href="#">Figure 24</a>
<b>Cardinality:</b>	0,N



## 5.2.6.5 GeometryReference\_t

GeometryReference\_t nodes are used to associate a CFD family with one or more CAD databases.

Node Attributes

<b>Name:</b>	User defined
<b>Label:</b>	GeometryReference_t
<b>DataType:</b>	MT
<b>Children:</b>	See <a href="#">Figure 28</a>
<b>Cardinality:</b>	0,N

## 5.2.6.6 GeometryFile\_t

This node contains the name of the CAD geometry file.

Node Attributes

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

## 5.2.6.7 GeometryFormat\_t

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

Node Attributes

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

## 5.2.6.8 GeometryEntity\_t

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

Node Attributes

<b>Name:</b>	User defined
<b>Label:</b>	GeometryEntity_t
<b>DataType:</b>	MT
<b>Children:</b>	None
<b>Cardinality:</b>	0,N

## 5.2.7 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.

### 5.2.7.1 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

<b>Name:</b>	User defined
<b>Label:</b>	<code>BaseIterativeData_t</code>
<b>DataType:</b>	I4
<b>Dimension:</b>	1
<b>Dimension Values:</b>	1
<b>Data:</b>	<code>NumberOfSteps</code>
<b>Children:</b>	See <a href="#">Figure 7</a>
<b>Cardinality:</b>	0,1

### 5.2.7.2 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

<b>Name:</b>	User defined
<b>Label:</b>	<code>ZoneIterativeData_t</code>
<b>DataType:</b>	MT
<b>Children:</b>	See <a href="#">Figure 51</a>
<b>Cardinality:</b>	0,1
<b>Parameters:</b>	<code>NumberOfSteps</code>

### 5.2.7.3 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

<b>Name:</b>	User defined
<b>Label:</b>	<code>RigidGridMotion_t</code>
<b>DataType:</b>	C1
<b>Dimension:</b>	1
<b>Dimension Values:</b>	Length of string
<b>Data:</b>	<code>RigidGridMotionType</code> value
<b>Children:</b>	See <a href="#">Figure 39</a>
<b>Cardinality:</b>	0,N

## 5.2.7.4 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*

<b>Name:</b>	User defined
<b>Label:</b>	ArbitraryGridMotion_t
<b>DataType:</b>	C1
<b>Dimension:</b>	1
<b>Dimension Values:</b>	Length of string
<b>Data:</b>	ArbitraryGridMotionType value
<b>Children:</b>	See <a href="#">Figure 3</a>
<b>Cardinality:</b>	0,N
<b>Parameters:</b>	IndexDimension, VertexSize, CellSize
<b>Functions:</b>	DataSize

## 5.2.8 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. Their primary function is to provide organization to the data below.

## 5.2.8.1 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*

<b>Name:</b>	User defined
<b>Label:</b>	Zone_t
<b>DataType:</b>	cgsizes_t
<b>Dimension:</b>	2
<b>Dimension Values:</b>	IndexDimension, 3
<b>Data:</b>	VertexSize[IndexDimension], CellSize[IndexDimension], VertexSizeBoundary[IndexDimension]
<b>Children:</b>	See <a href="#">Figure 48</a>
<b>Cardinality:</b>	0,N
<b>Parameters:</b>	CellDimension, PhysicalDimension

## 5.2.8.2 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

<b>Name:</b>	User defined
<b>Label:</b>	<code>CGNSBase_t</code>
<b>DataType:</b>	I4
<b>Dimension:</b>	1
<b>Dimension Values:</b>	2
<b>Data:</b>	<code>CellDimension</code> , <code>PhysicalDimension</code>
<b>Children:</b>	See <a href="#">Figure 12</a>
<b>Cardinality:</b>	0,N

#### 5.2.8.3 `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 5.2.7](#).

### Node Attributes

<b>Name:</b>	<code>SimulationType</code>
<b>Label:</b>	<code>SimulationType_t</code>
<b>DataType:</b>	C1
<b>Dimension:</b>	1
<b>Dimension Values:</b>	Length of string
<b>Data:</b>	<code>TimeAccurate</code> or <code>NonTimeAccurate</code>
<b>Children:</b>	None
<b>Cardinality:</b>	0,1

#### 5.2.8.4 `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

<b>Name:</b>	<code>ZoneType</code>
<b>Label:</b>	<code>ZoneType_t</code>
<b>DataType:</b>	C1
<b>Dimension:</b>	1
<b>Dimension Values:</b>	Length of string
<b>Data:</b>	<code>Structured</code> or <code>Unstructured</code>
<b>Children:</b>	None

**Cardinality:** 1

#### 5.2.8.5 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

<b>Name:</b>	CGNSLibraryVersion
<b>Label:</b>	CGNSLibraryVersion_t
<b>DataType:</b>	R4
<b>Dimension:</b>	1
<b>Dimension Values:</b>	1
<b>Data:</b>	CGNS version number
<b>Children:</b>	None
<b>Cardinality:</b>	1



## A CGNS File Mapping Figures

This Appendix contains figures illustrating the nodal structure of a CGNS database. An example file hierarchy is shown in [Figure 1](#), but without any internal detail of the individual nodes. Just below the `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.

**Note:** In some of the figures the data type is specified as *cgsizet*. This data type has been introduced with the advent of CGNS library version 3.1.3, and is the natural size of a word for a given compilation of the library. For a 32-bit compilation, *cgsizet* will be a 32-bit integer (`I4`), and for a 64-bit compilation it will be a 64-bit integer (`I8`). Regardless of how the file is written, the data is returned to an application as *cgsizet*. When reading `I8` values in a 32-bit application, an error will be generated if the size of the data exceeds the 32-bit integer limitation.

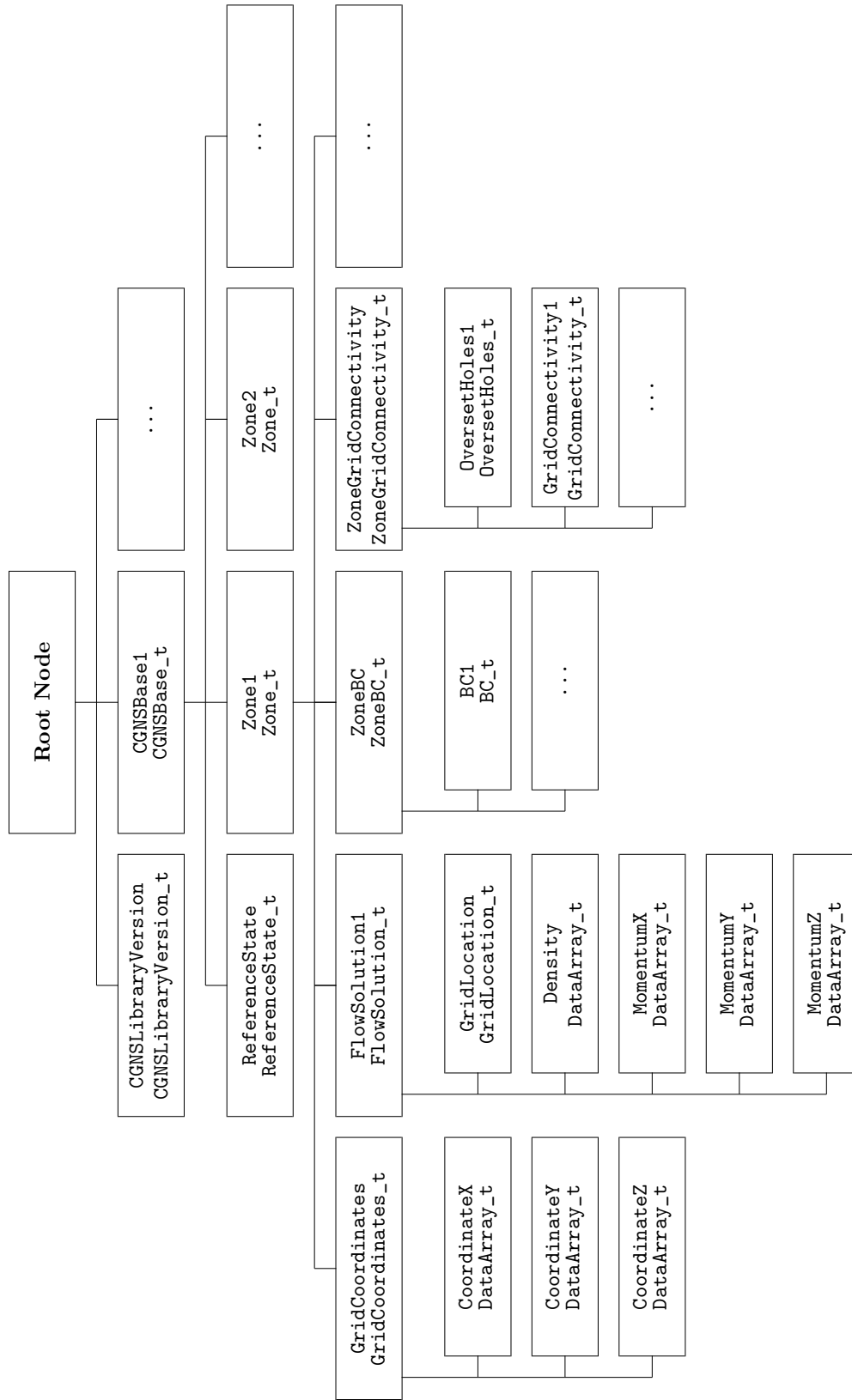


Figure 1: Example Hierarchy



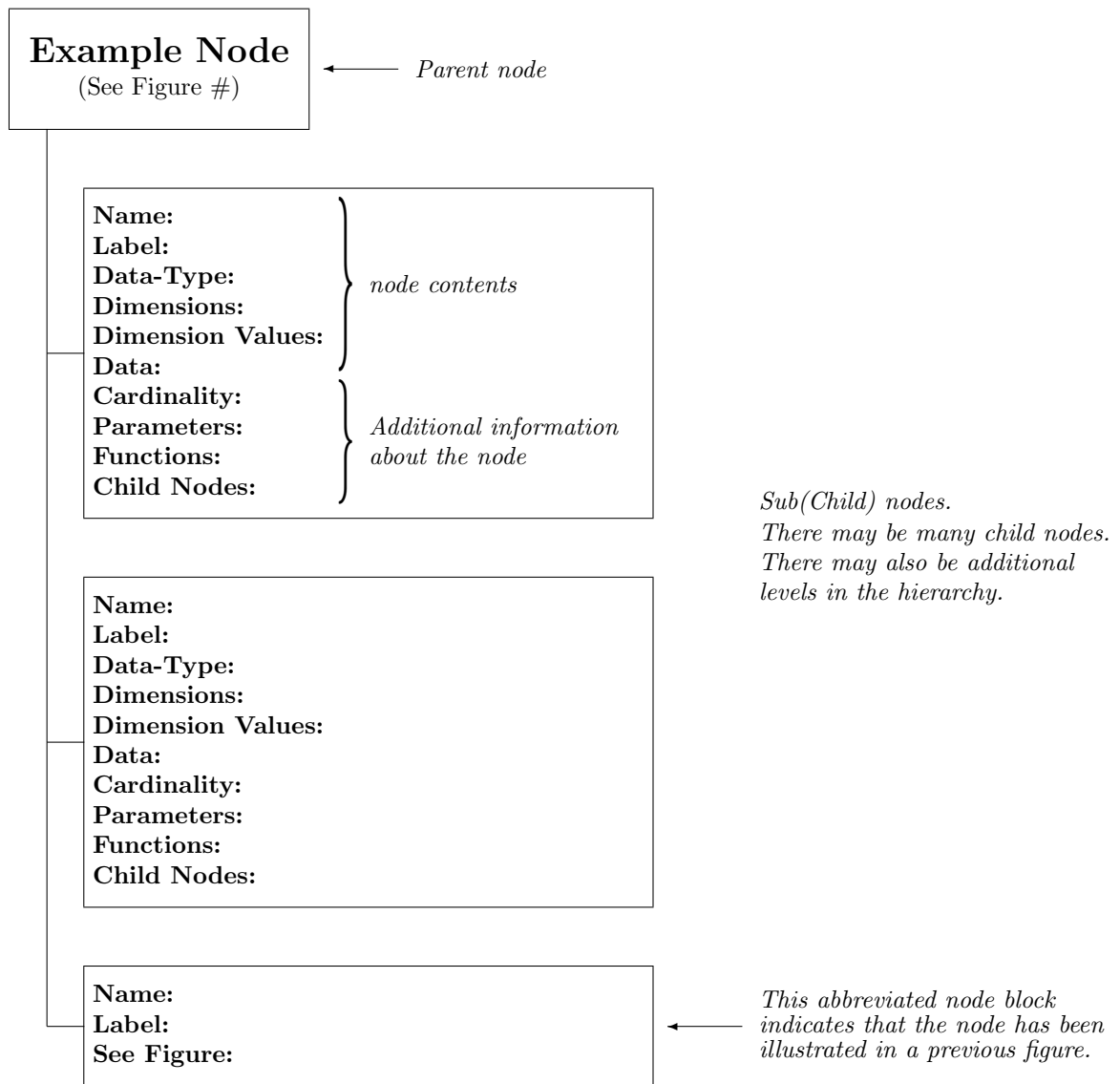
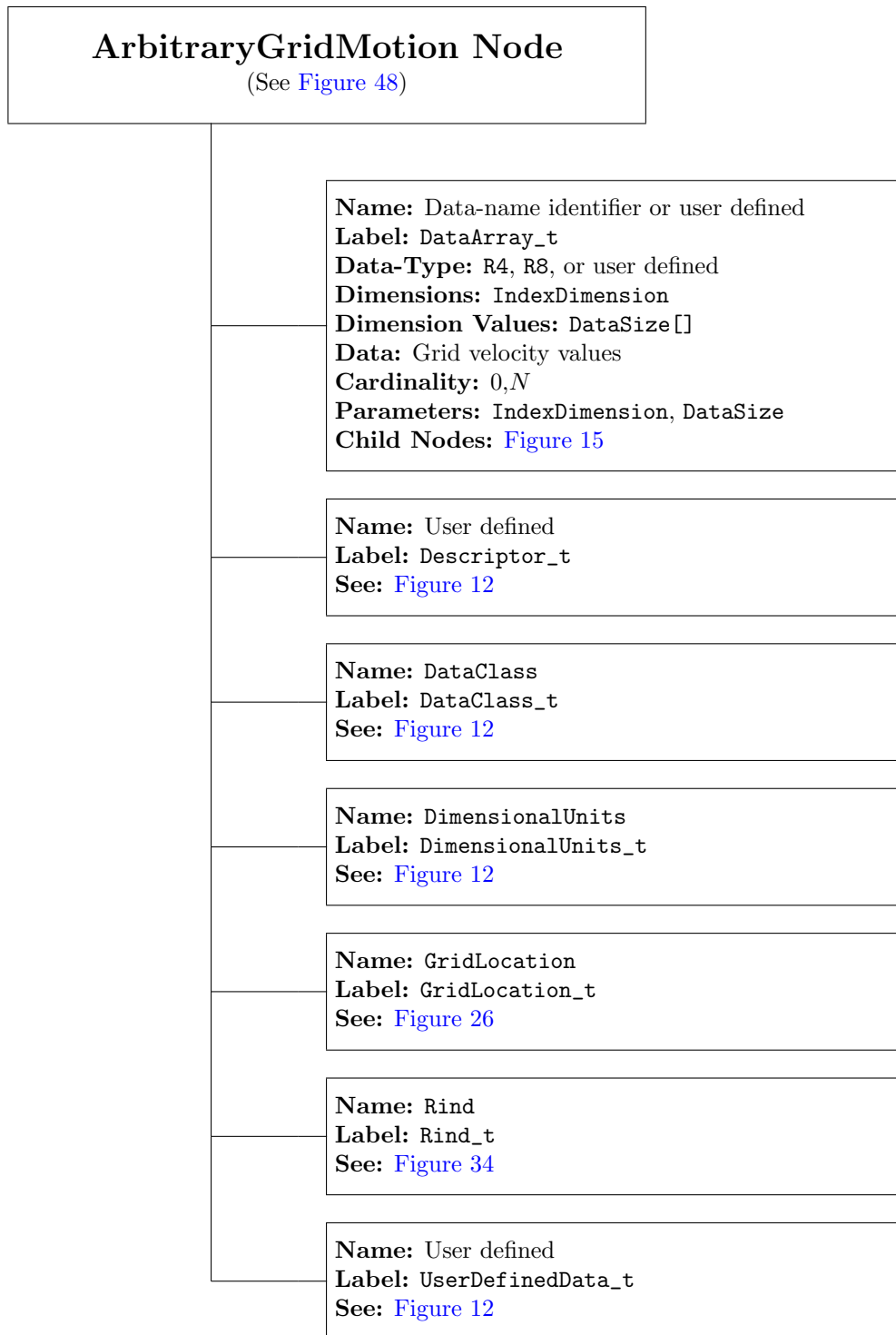


Figure 2: Example Node Structure



**Figure 3:** ArbitraryGridMotion\_t Data Structure

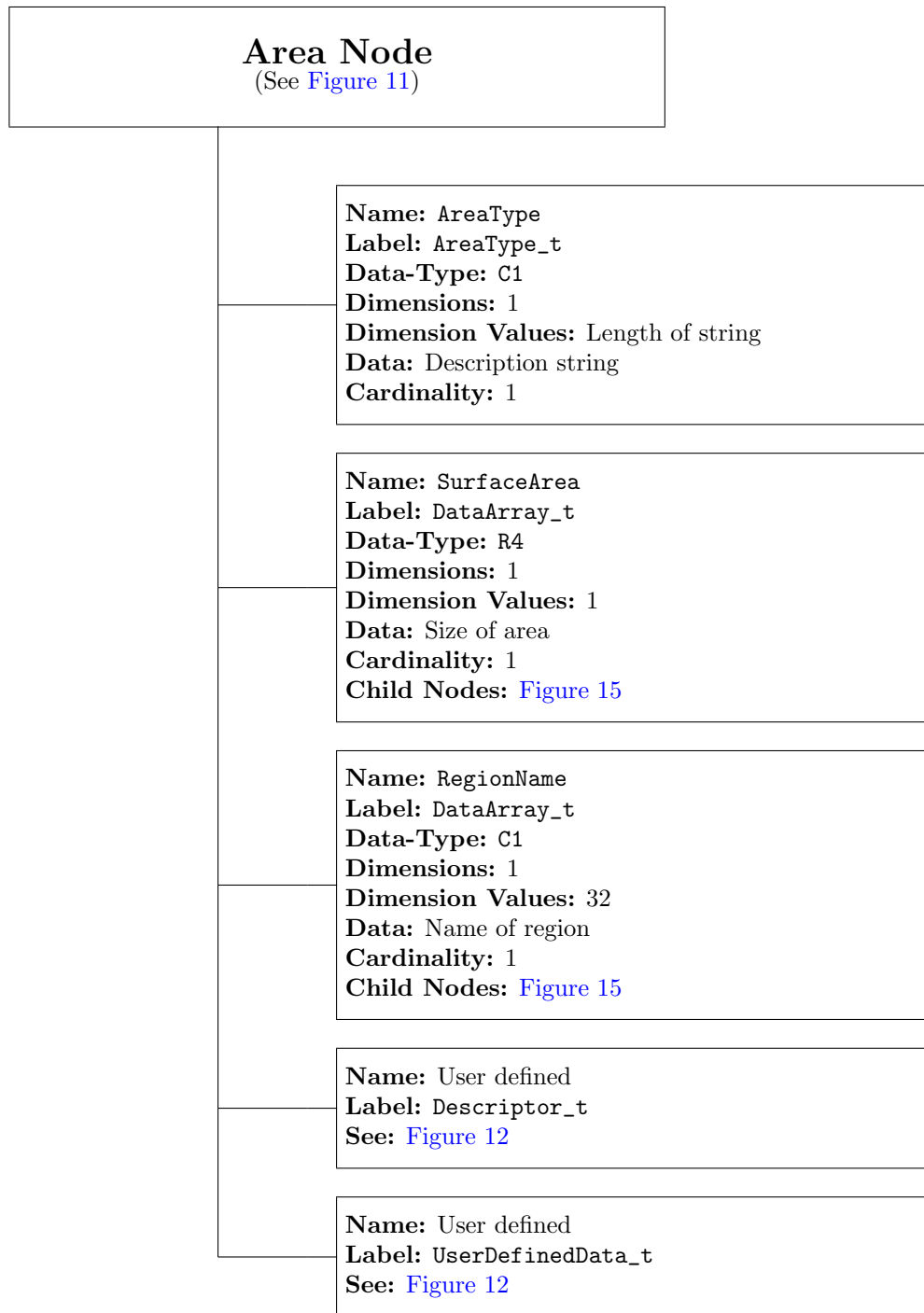
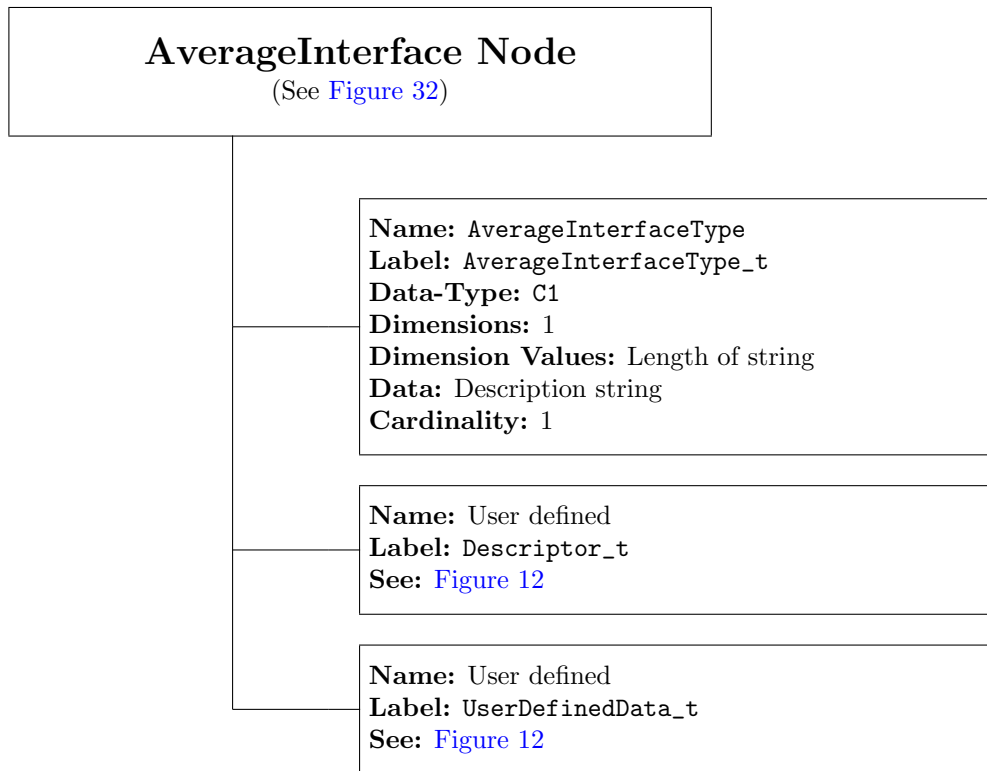


Figure 4: Area\_t Data Structure



**Figure 5:** AverageInterface\_t Data Structure

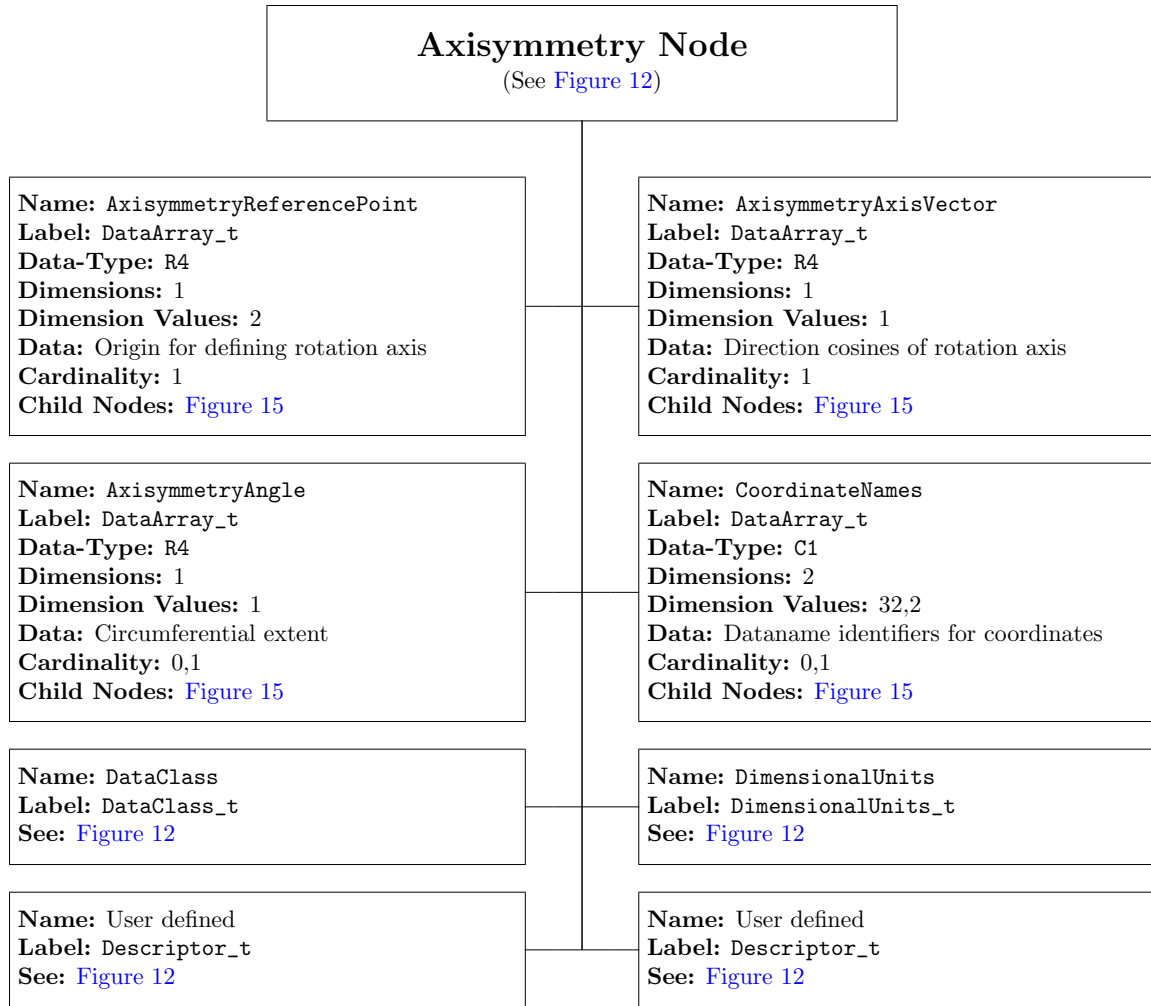
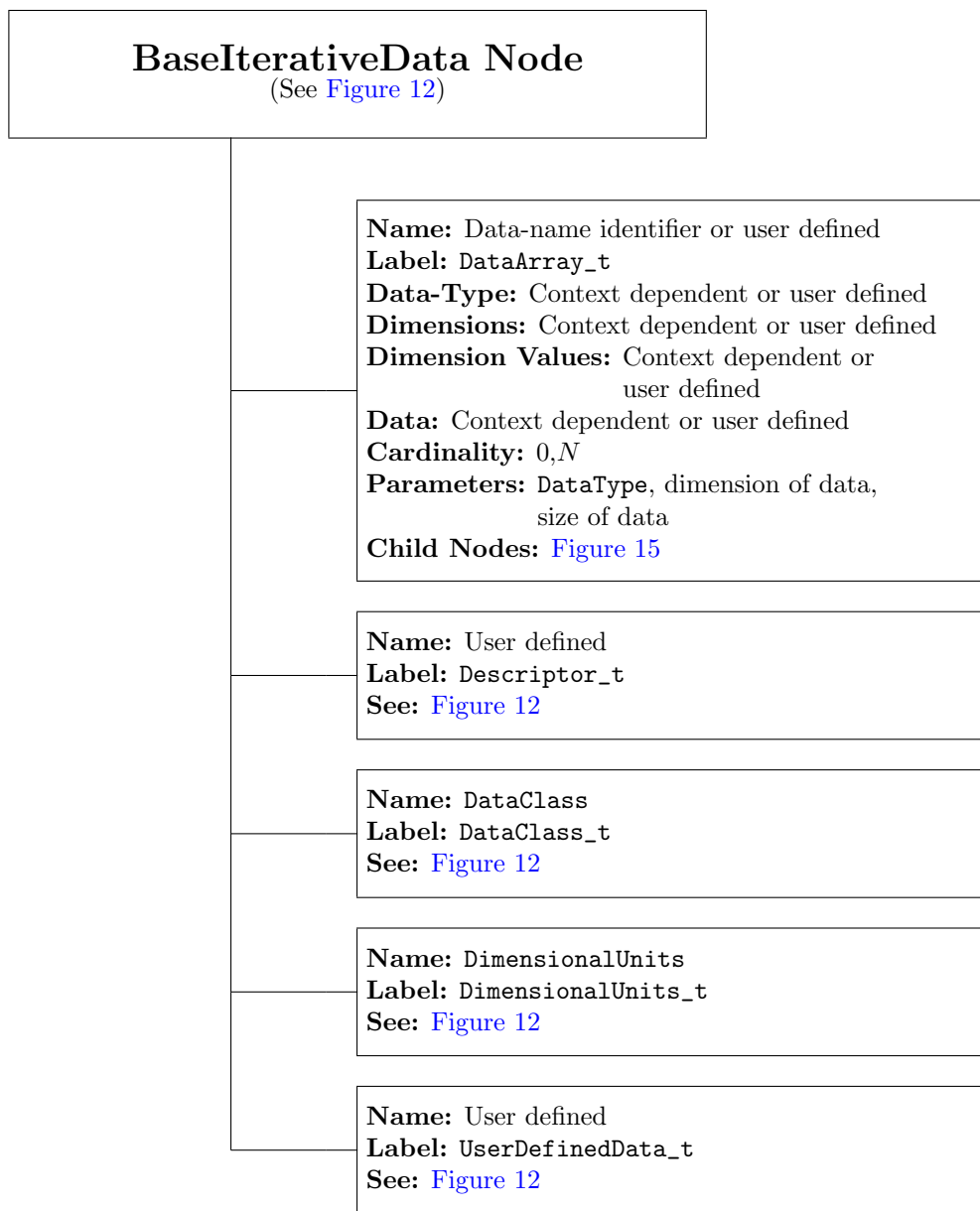
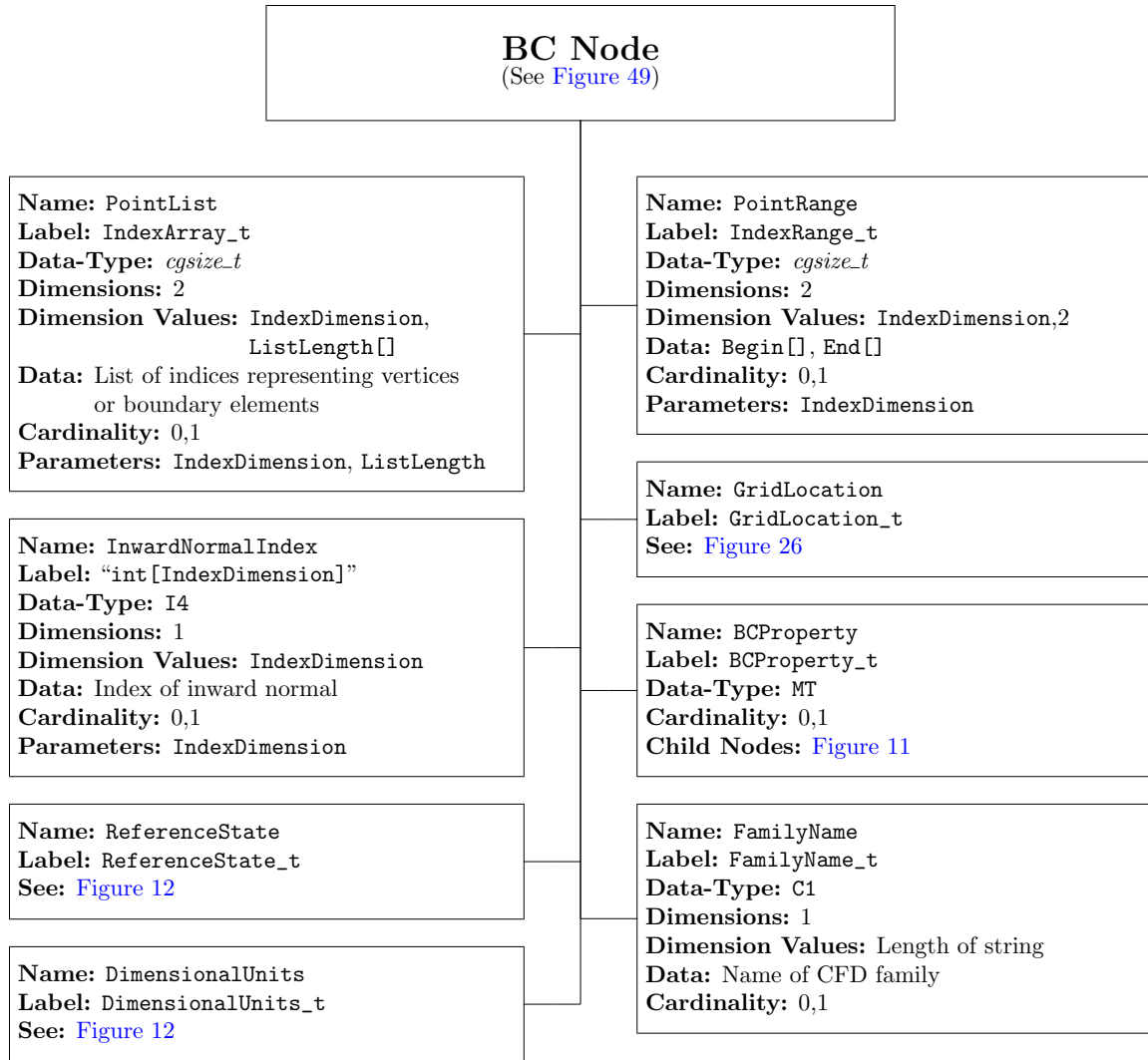


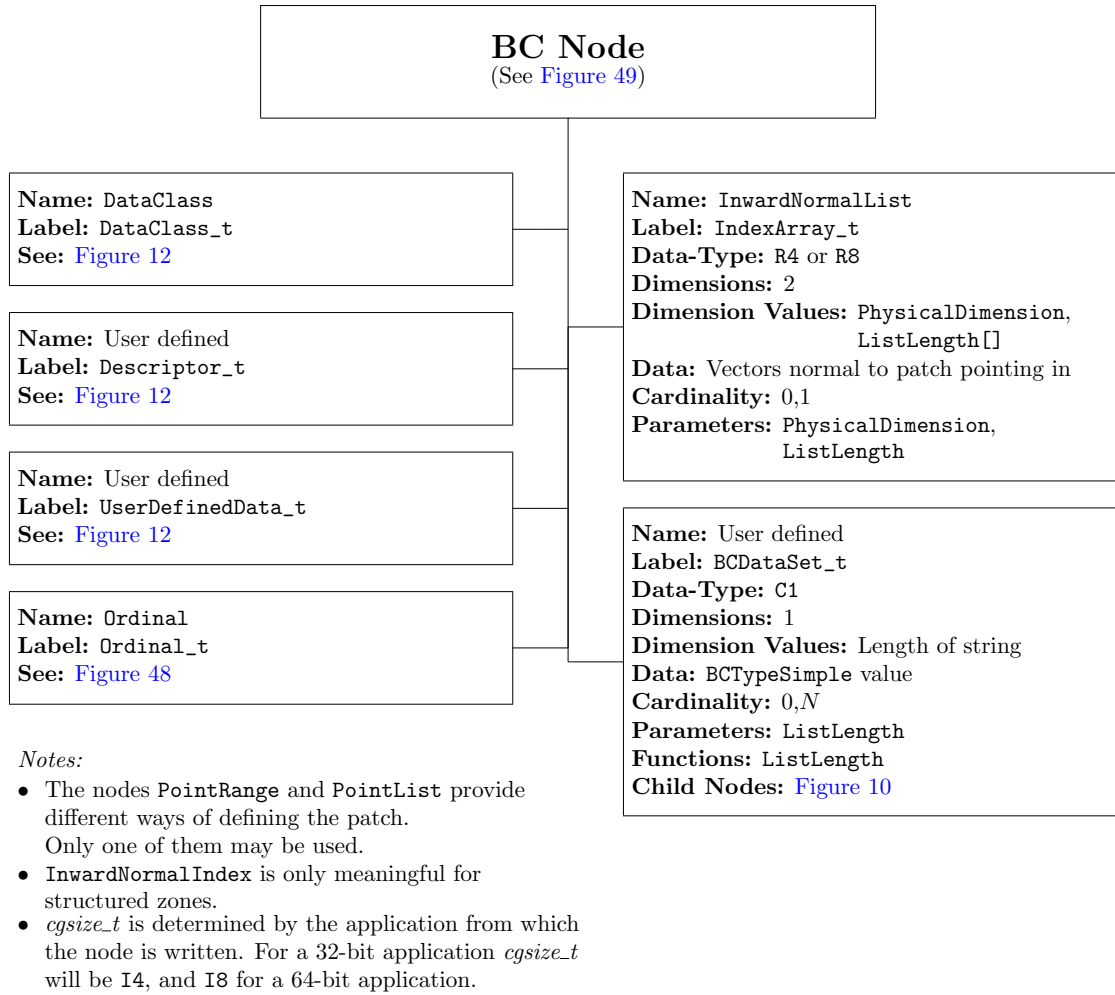
Figure 6: Axisymmetry\_t Data Structure



**Figure 7:** BaseIterativeData\_t Data Structure



**Figure 8:** BC\_t Data Structure (*Continued on next page*)



**Figure 8:** BC\_t Data Structure (*Continued from previous page*)



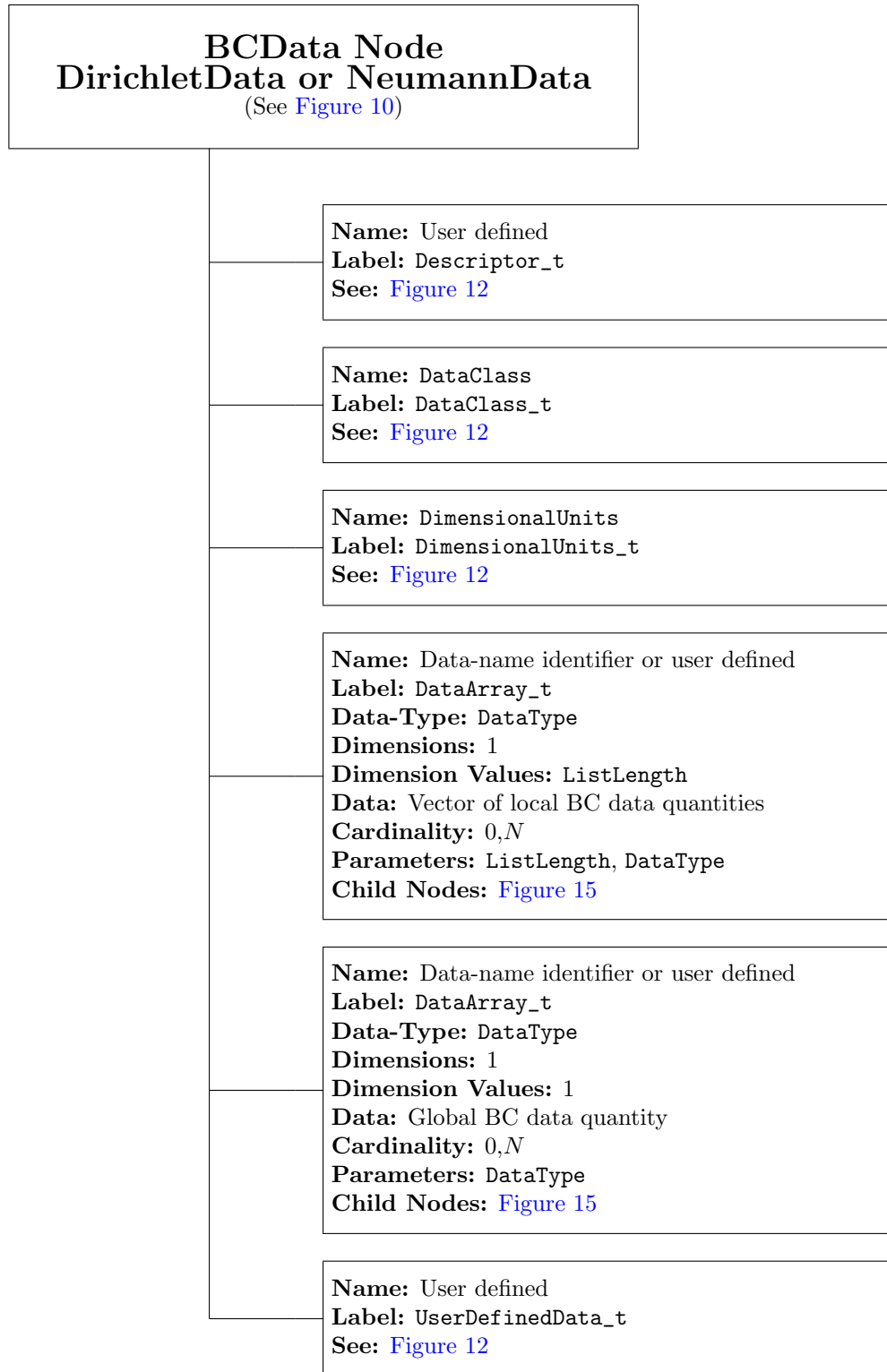


Figure 9: BCDData\_t Data Structure

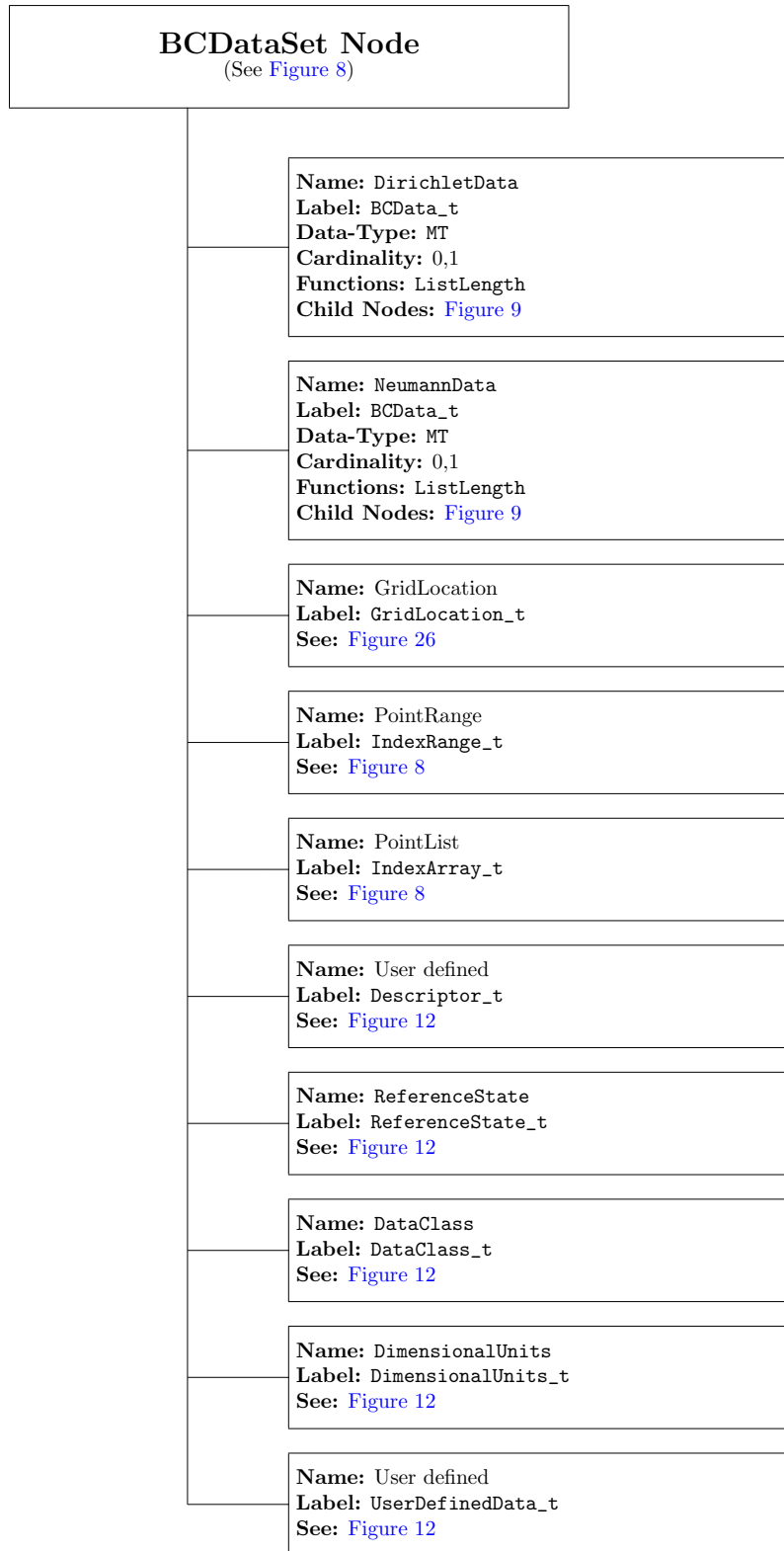


Figure 10: BCDataset\_t Data Structure

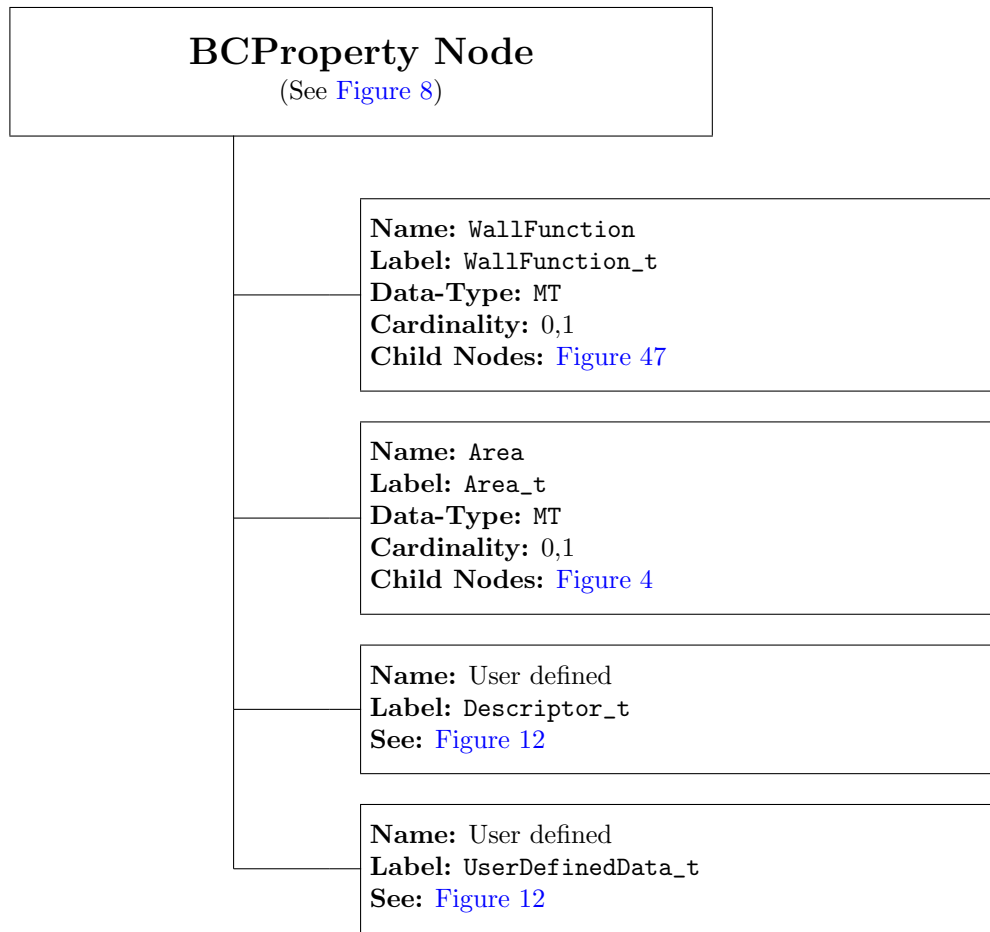


Figure 11: BCProperty\_t Data Structure

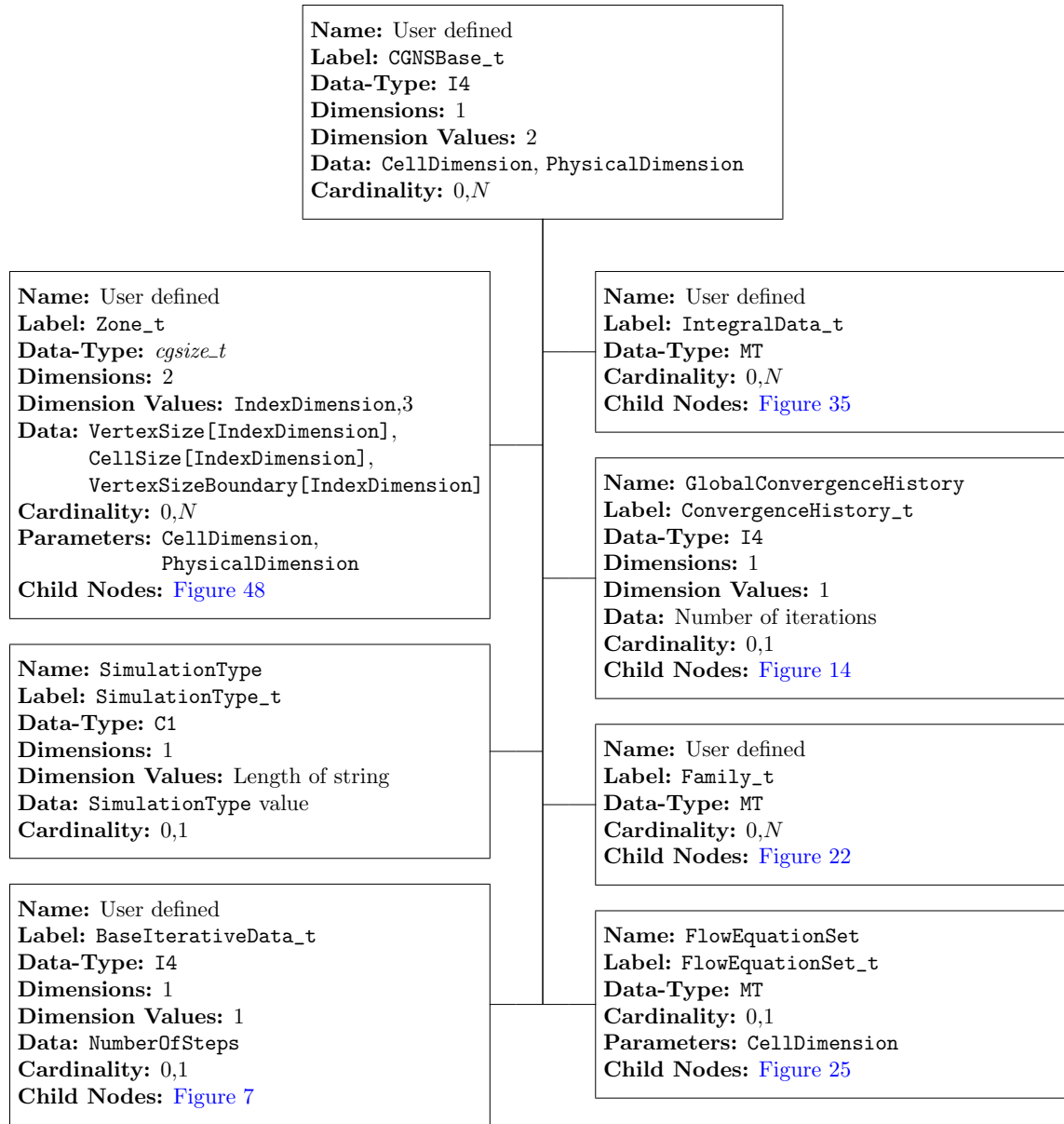


Figure 12: CGNSBase\_t Data Structure (Continued on next page)

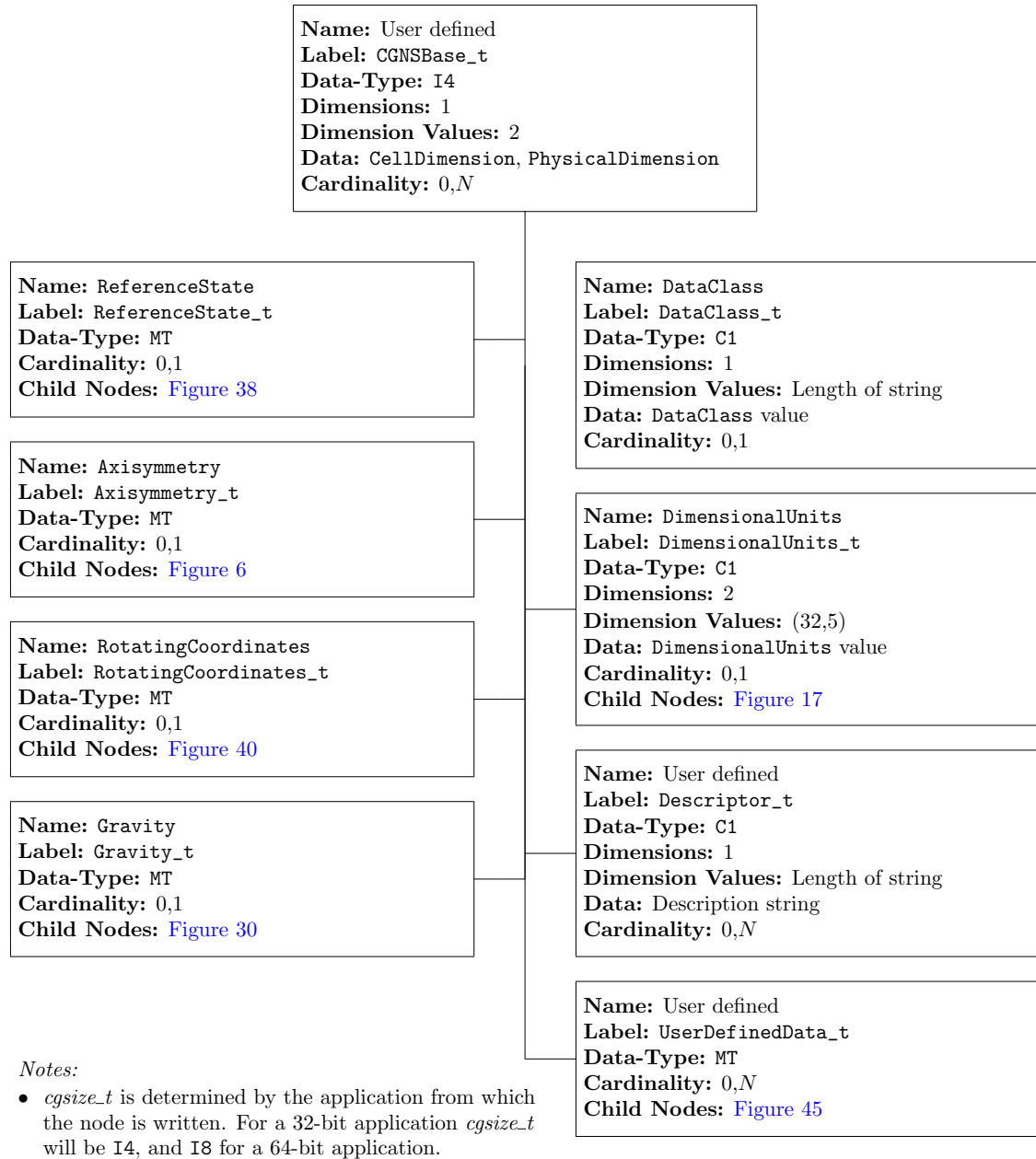
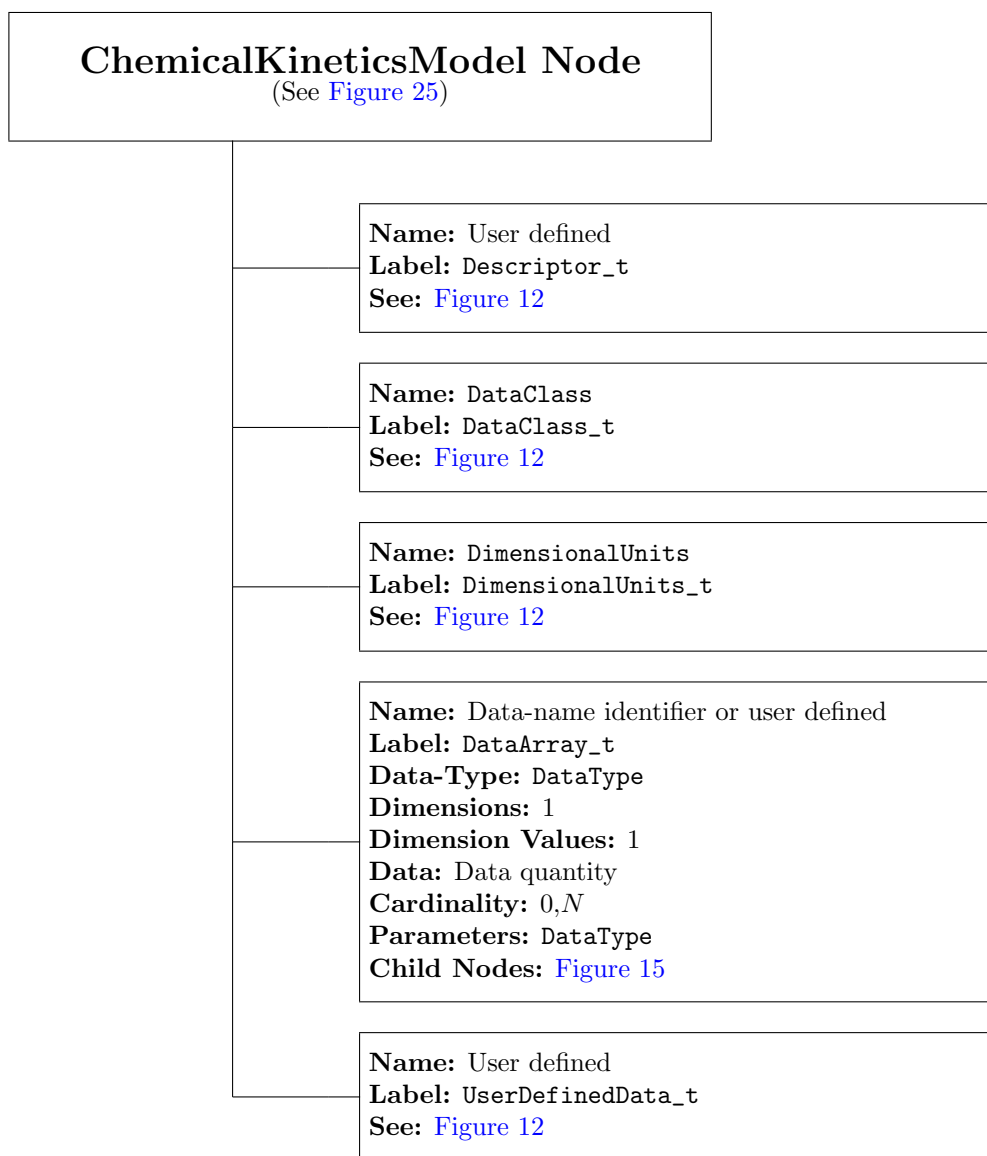


Figure 12: CGNSBase\_t Data Structure (Continued from previous page)



**Figure 13:** ChemicalKineticsModel\_t Data Structure

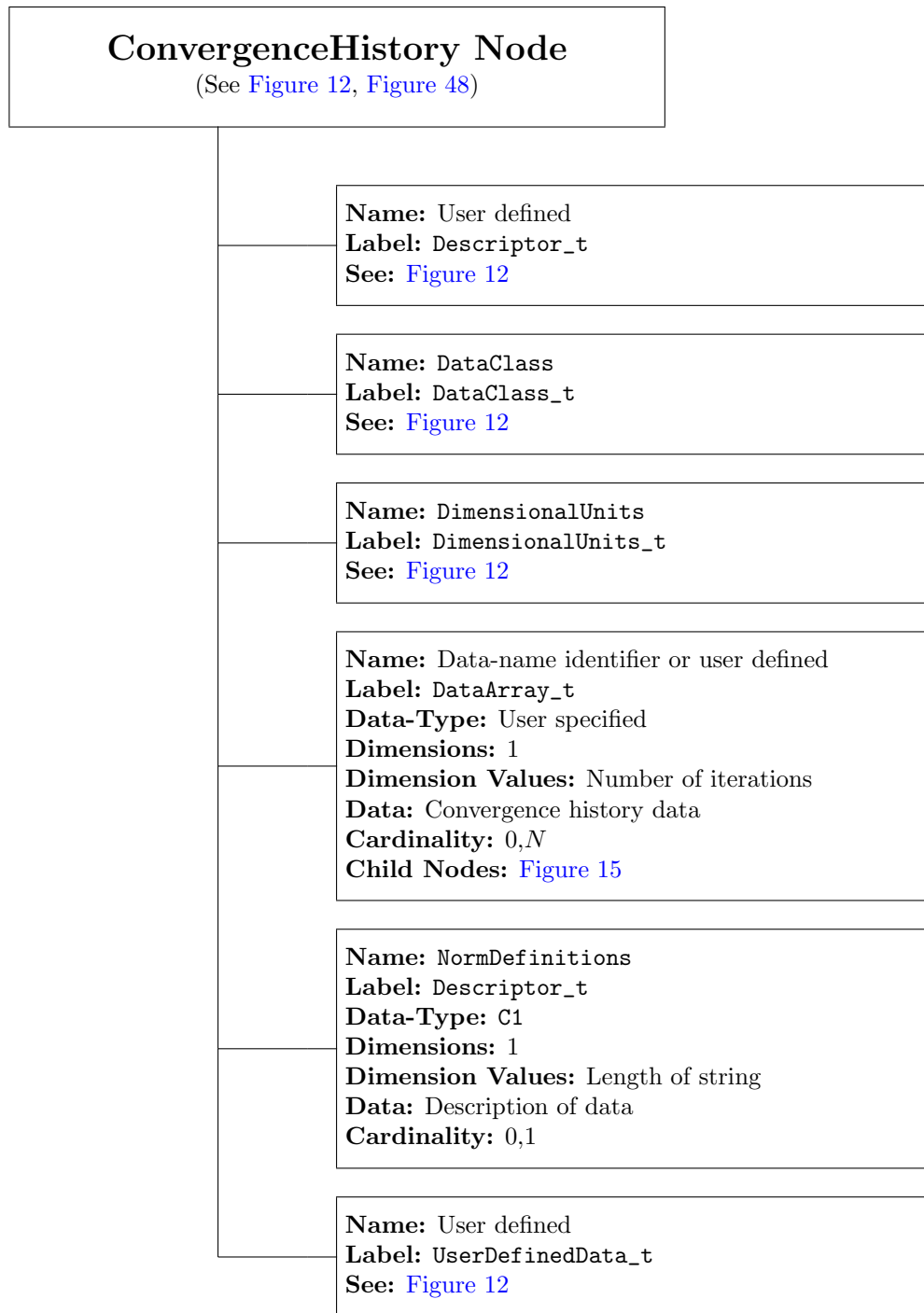
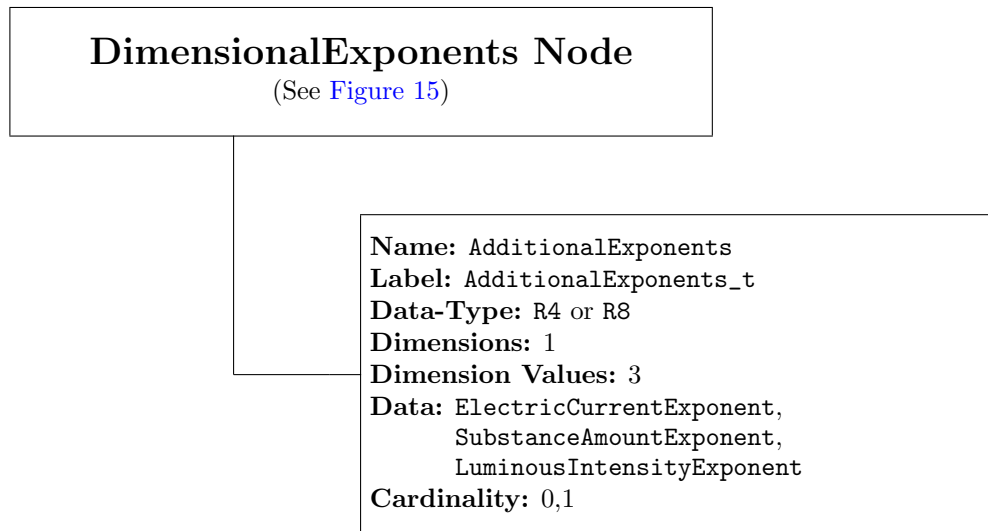


Figure 14: ConvergenceHistory\_t Data Structure

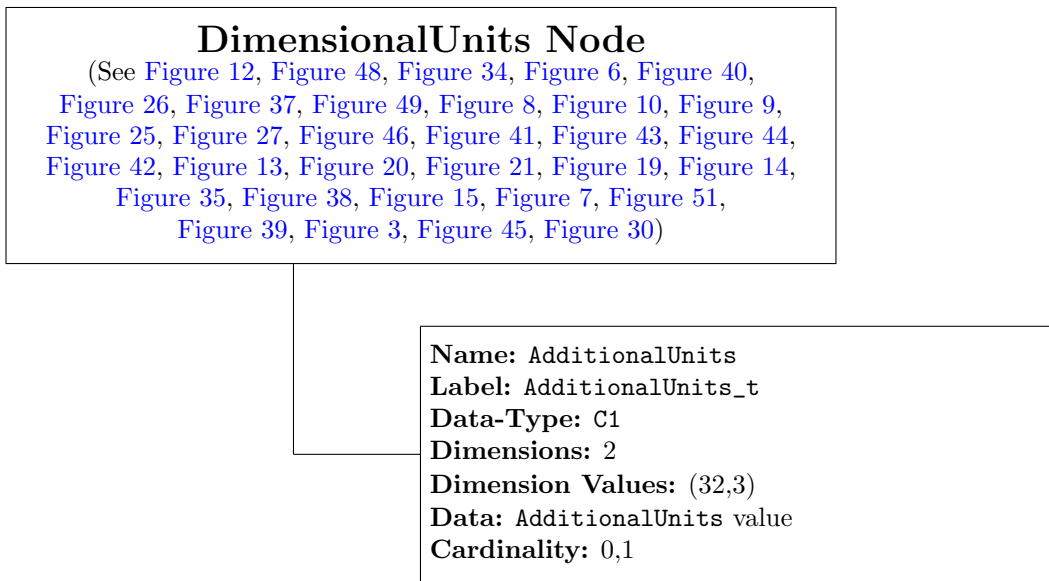


Figure 15: DataArray\_t Data Structure

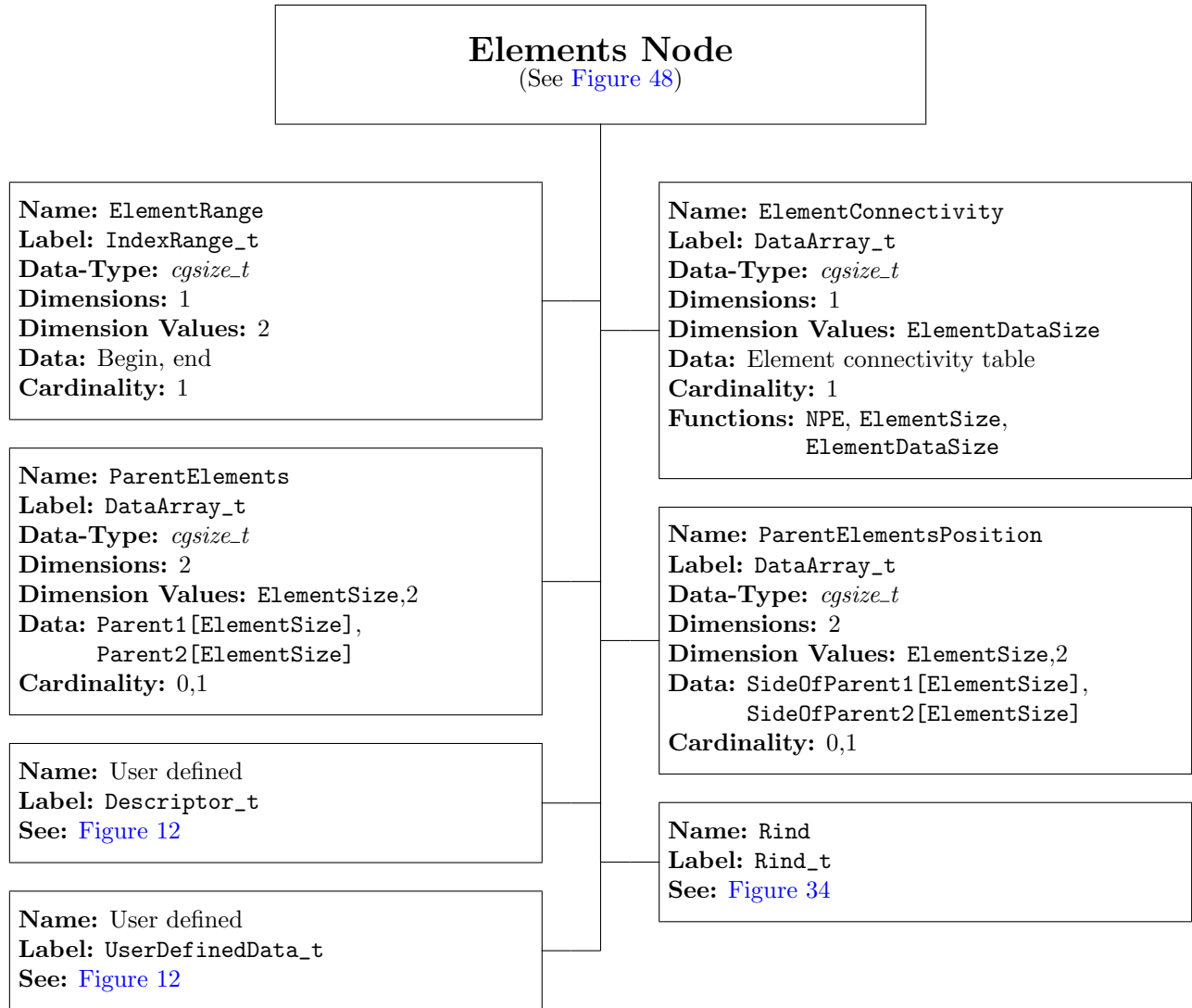




**Figure 16:** DimensionalExponents\_t Data Structure



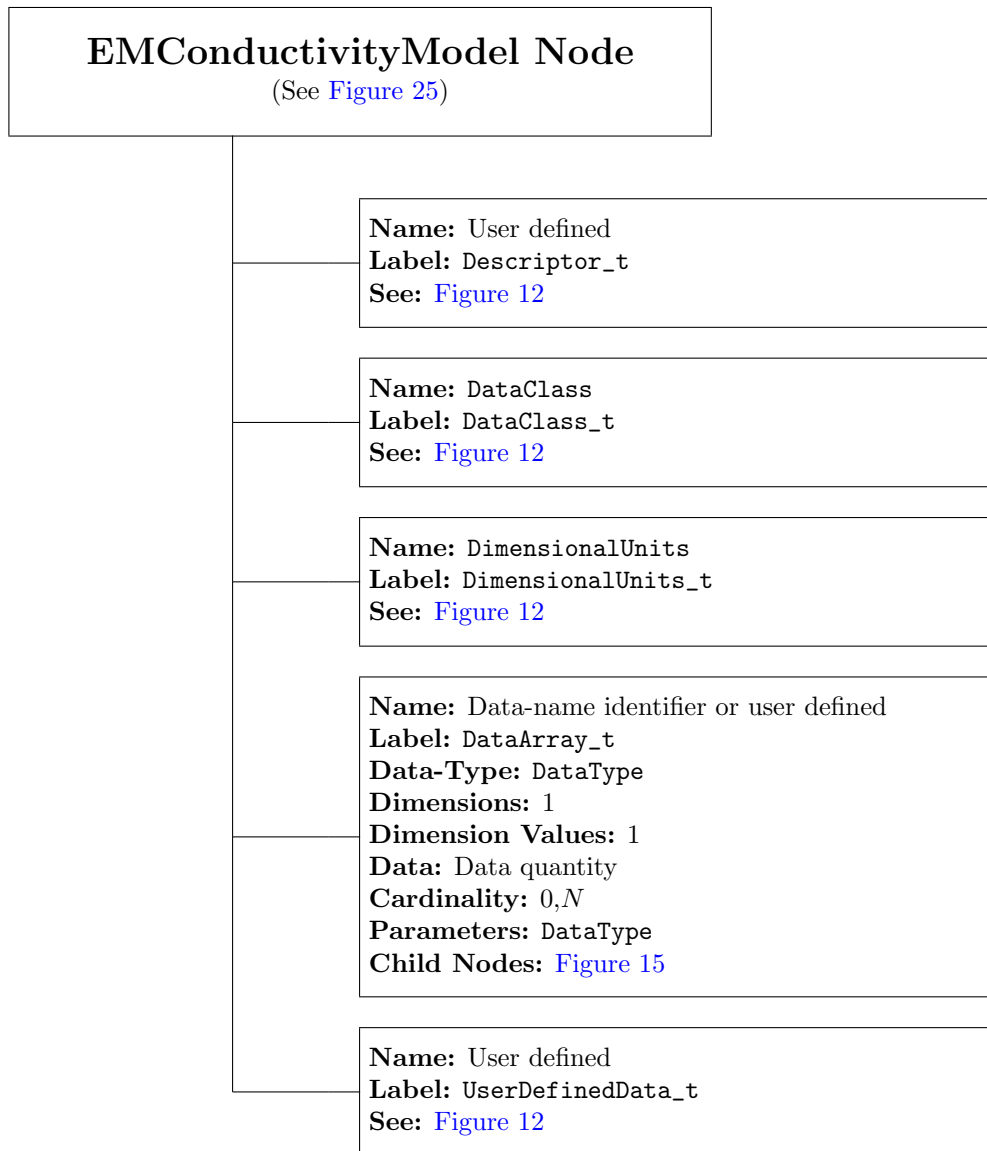
**Figure 17:** DimensionalUnits\_t Data Structure



*Notes:*

- *cgsizet* is determined by the application from which the node is written. For a 32-bit application *cgsizet* will be I4, and I8 for a 64-bit application.

**Figure 18:** Elements\_t Data Structure



**Figure 19:** EMConductivityModel\_t Data Structure

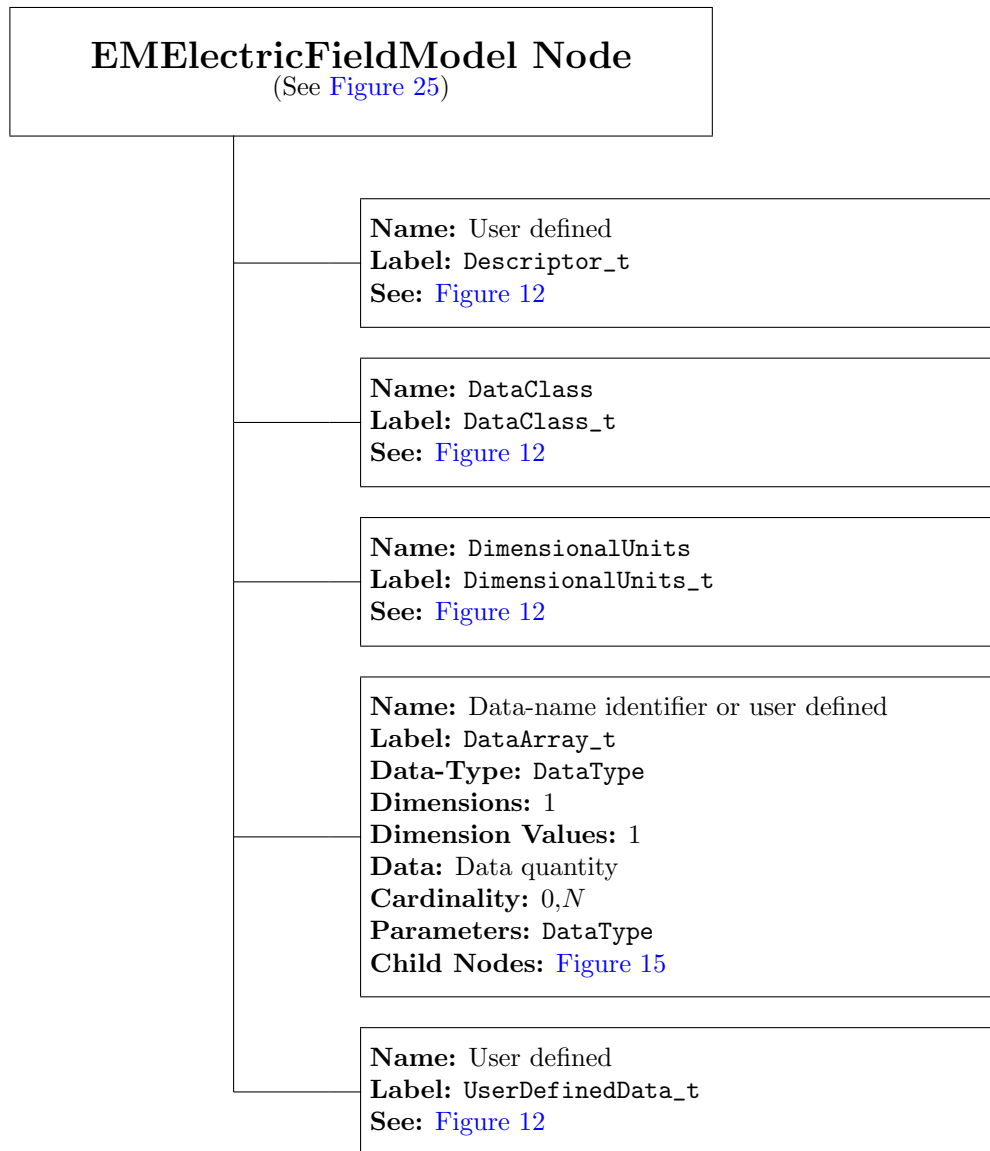
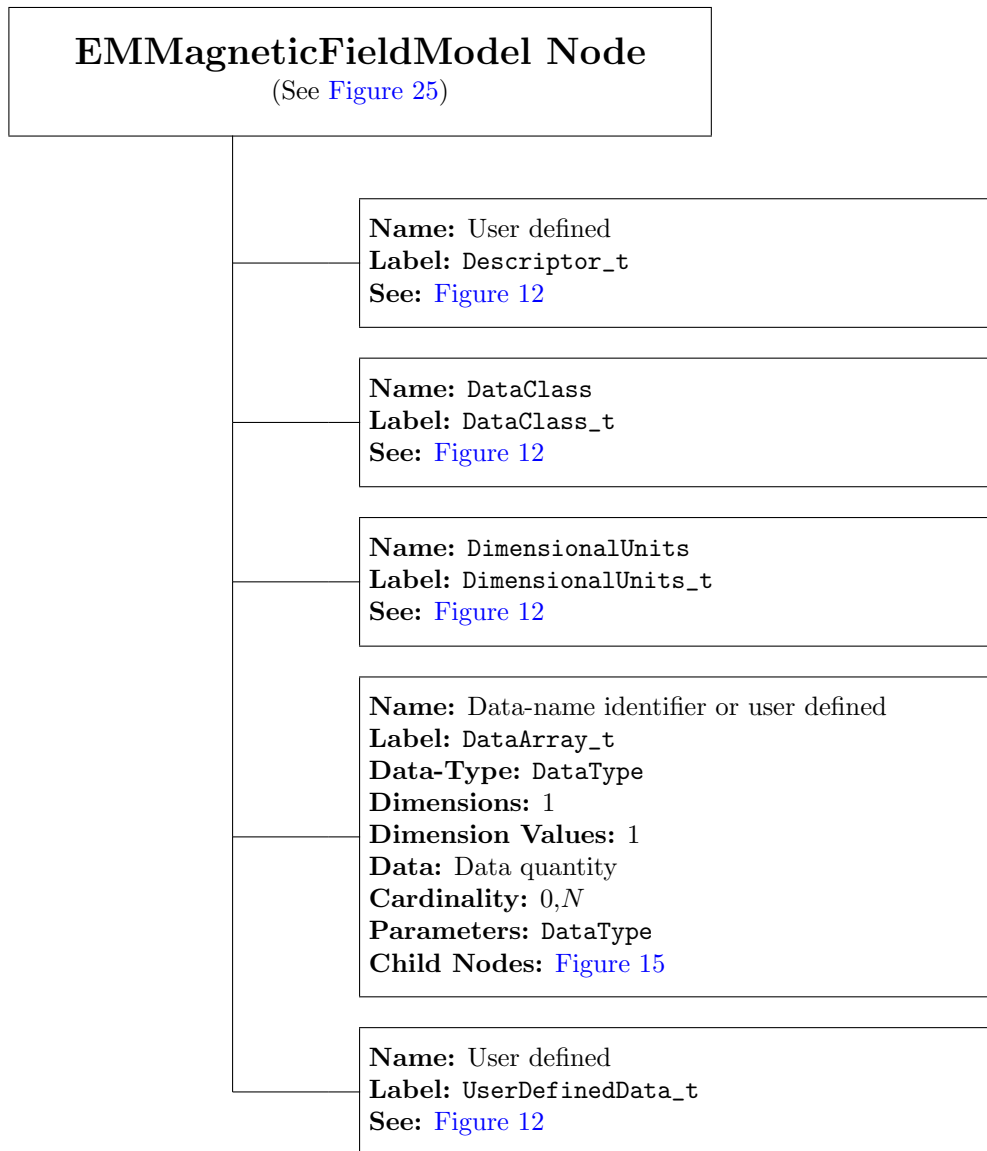


Figure 20: EMElectricFieldModel\_t Data Structure



**Figure 21:** EMMagneticFieldModel\_t Data Structure

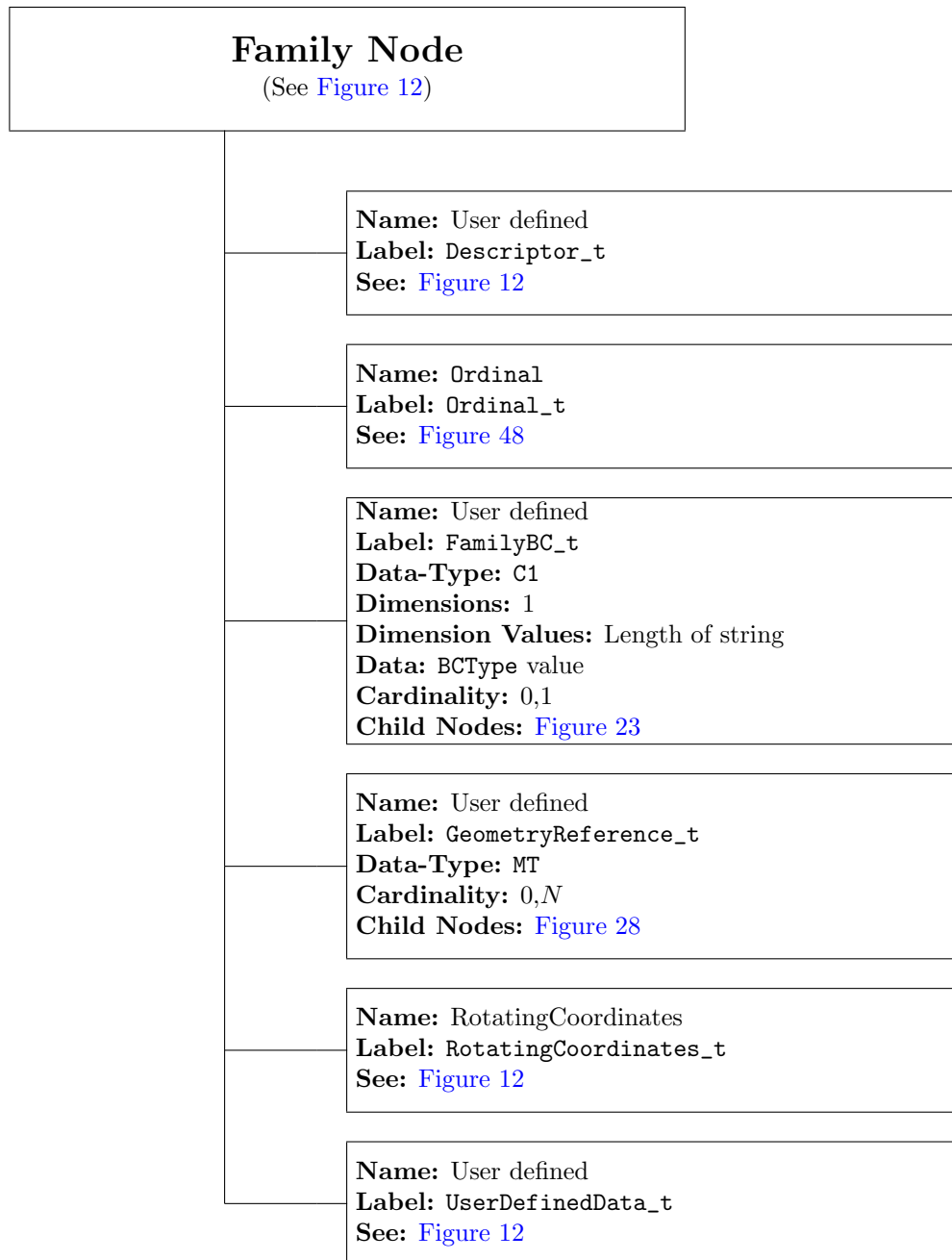
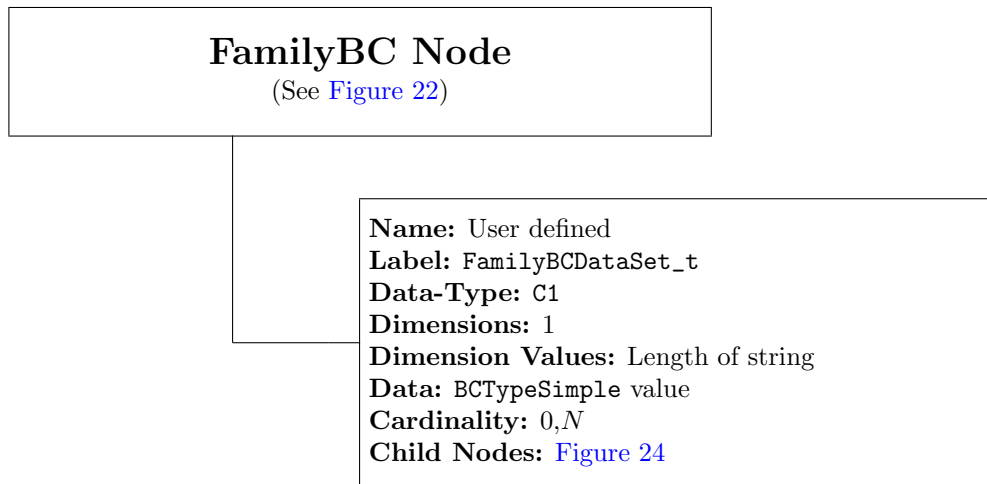


Figure 22: Family\_t Data Structure



**Figure 23:** FamilyBC\_t Data Structure



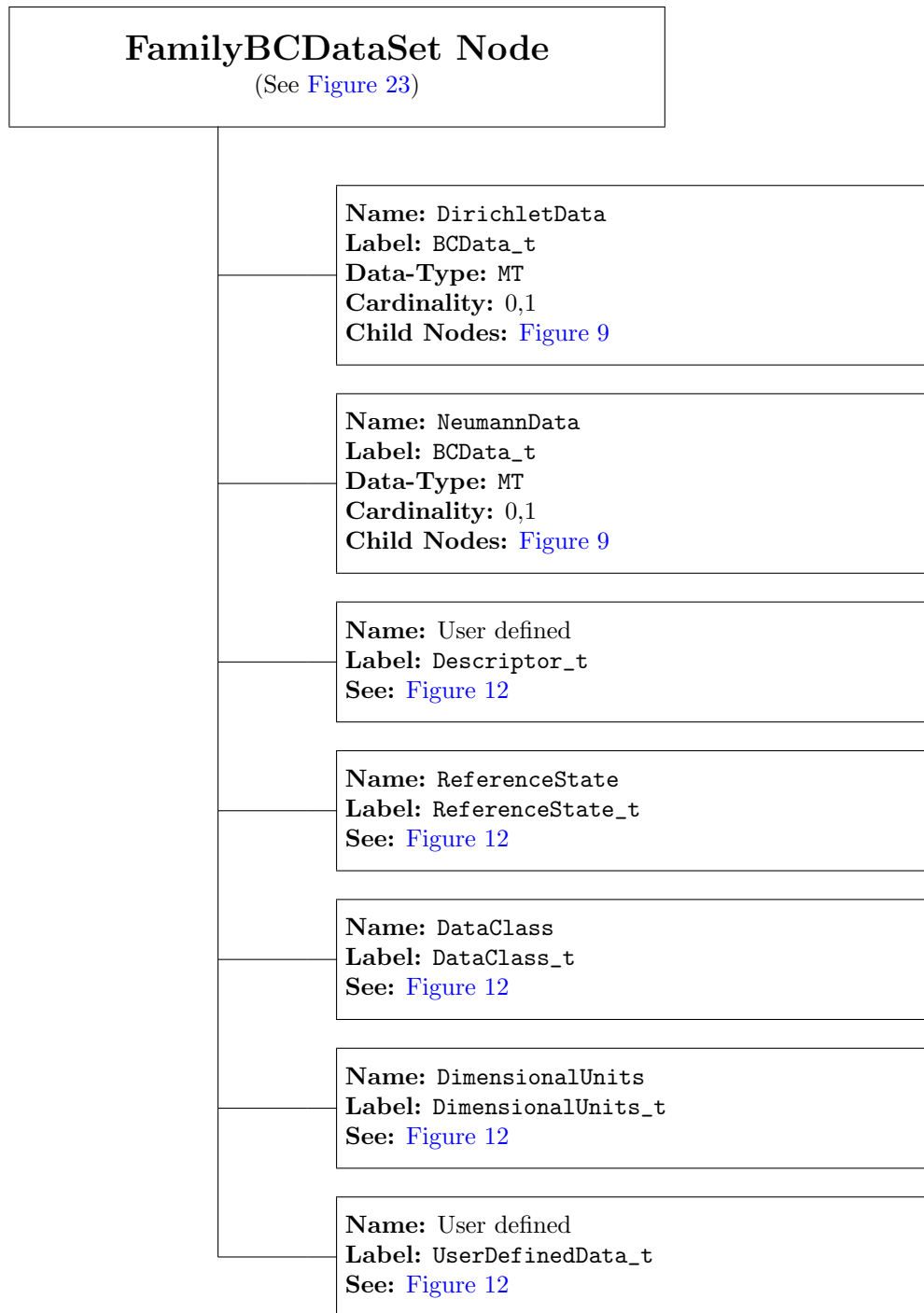
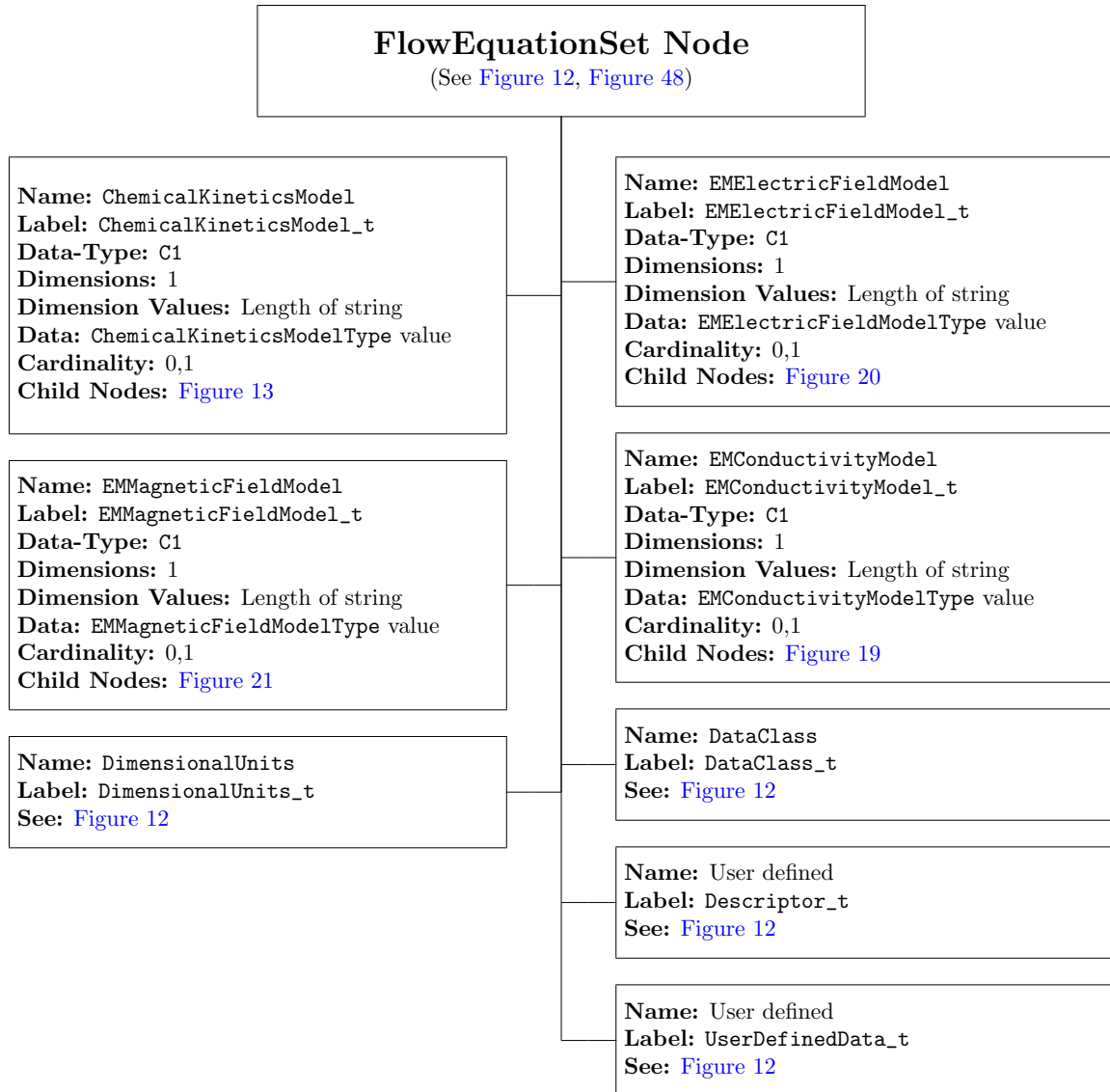
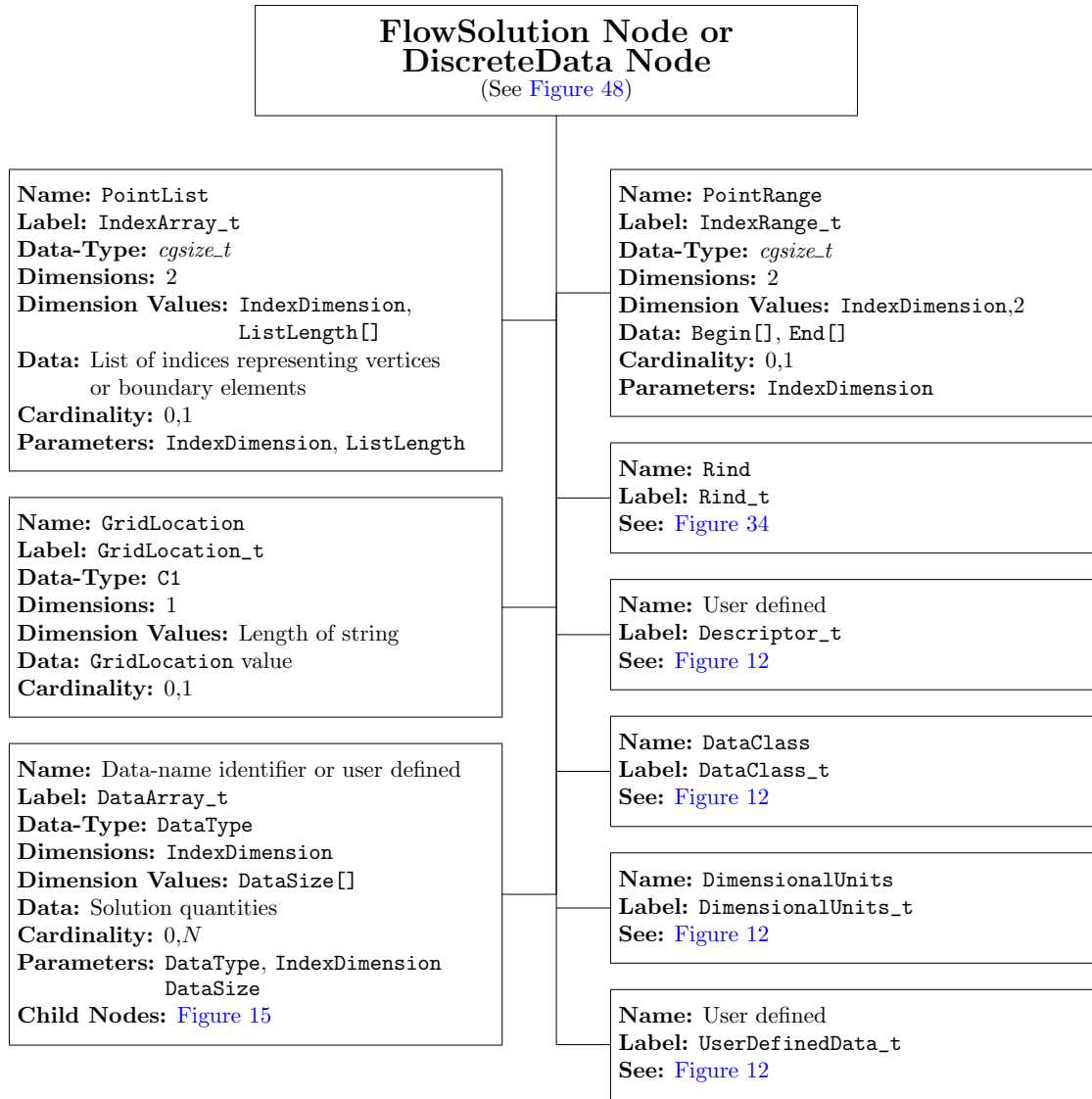


Figure 24: FamilyBCDataSet\_t Data Structure

Figure 25: FlowEquationSet\_t Data Structure (*Continued on next page*)

**Figure 25:** FlowEquationSet\_t Data Structure (*Continued from previous page*)



*Notes:*

- *cgsizet* is determined by the application from which the node is written. For a 32-bit application *cgsizet* will be I4, and I8 for a 64-bit application.

**Figure 26:** FlowSolution\_t and DiscreteData\_t Data Structures

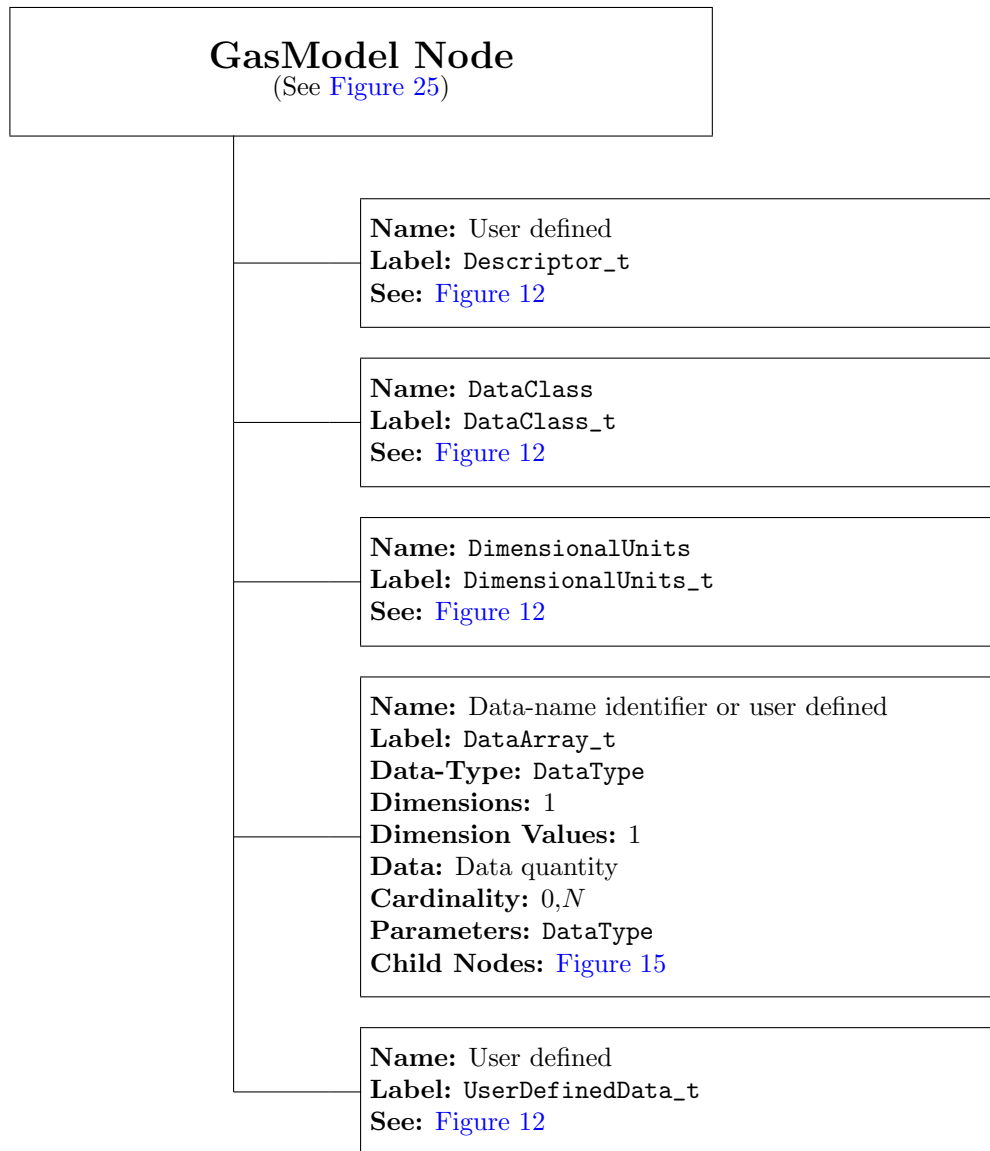
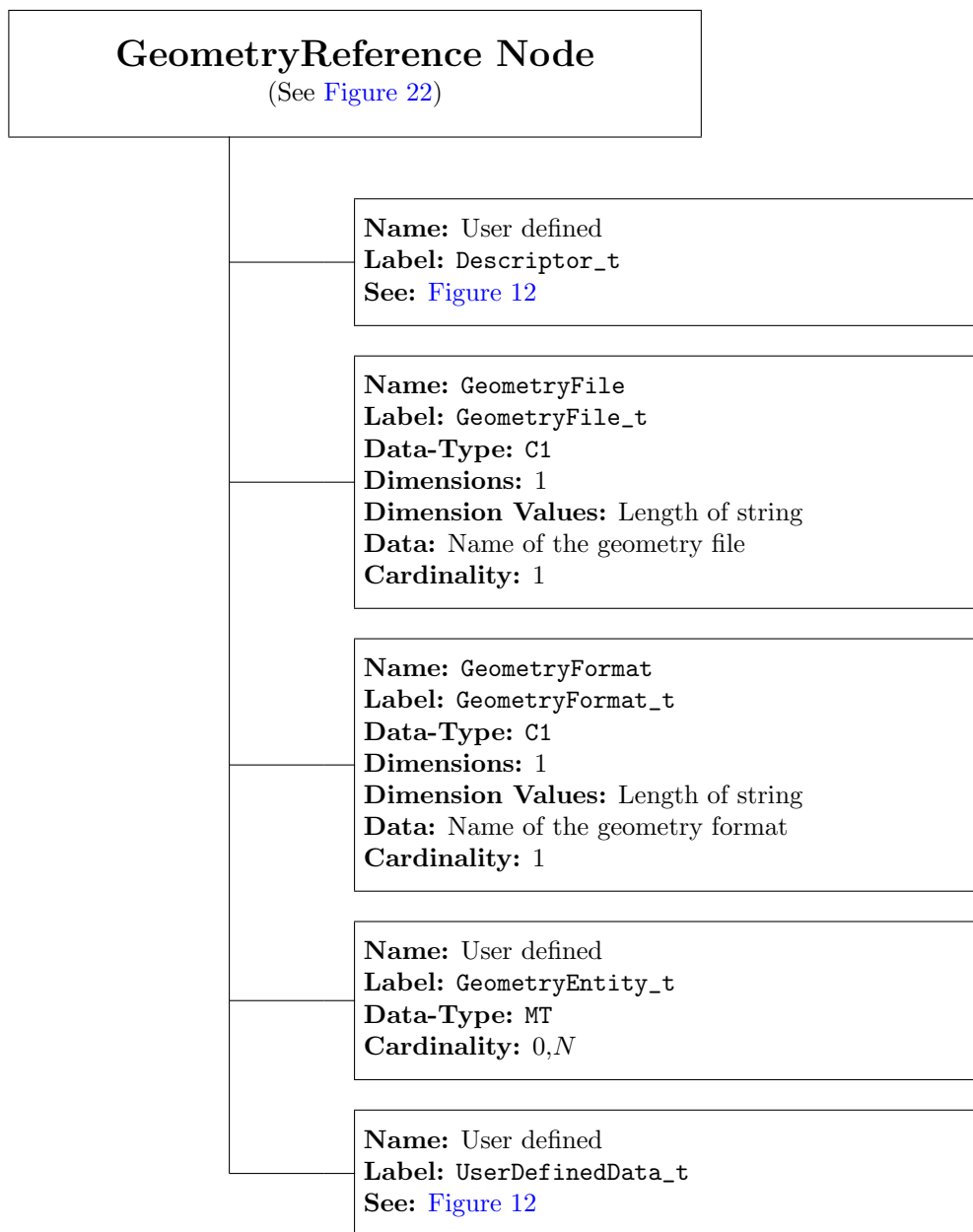
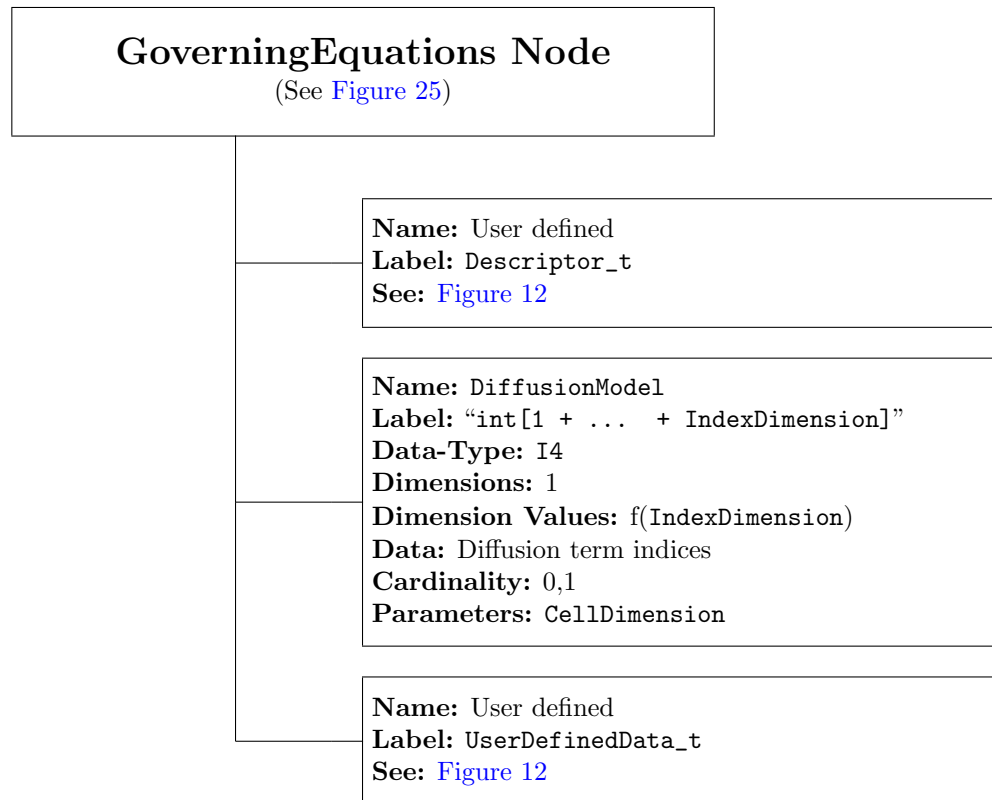


Figure 27: GasModel\_t Data Structure

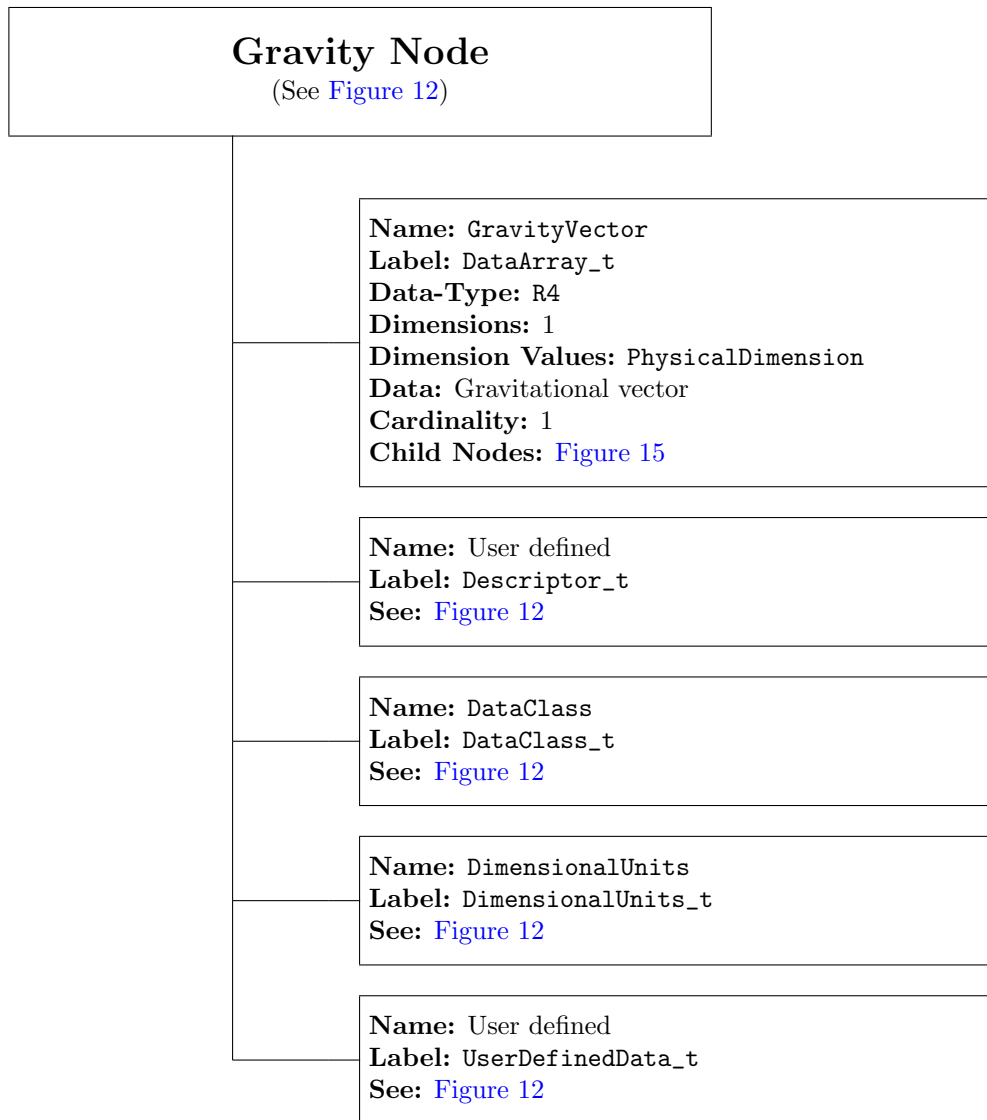
**Figure 28:** GeometryReference\_t Data Structure



*Note:*

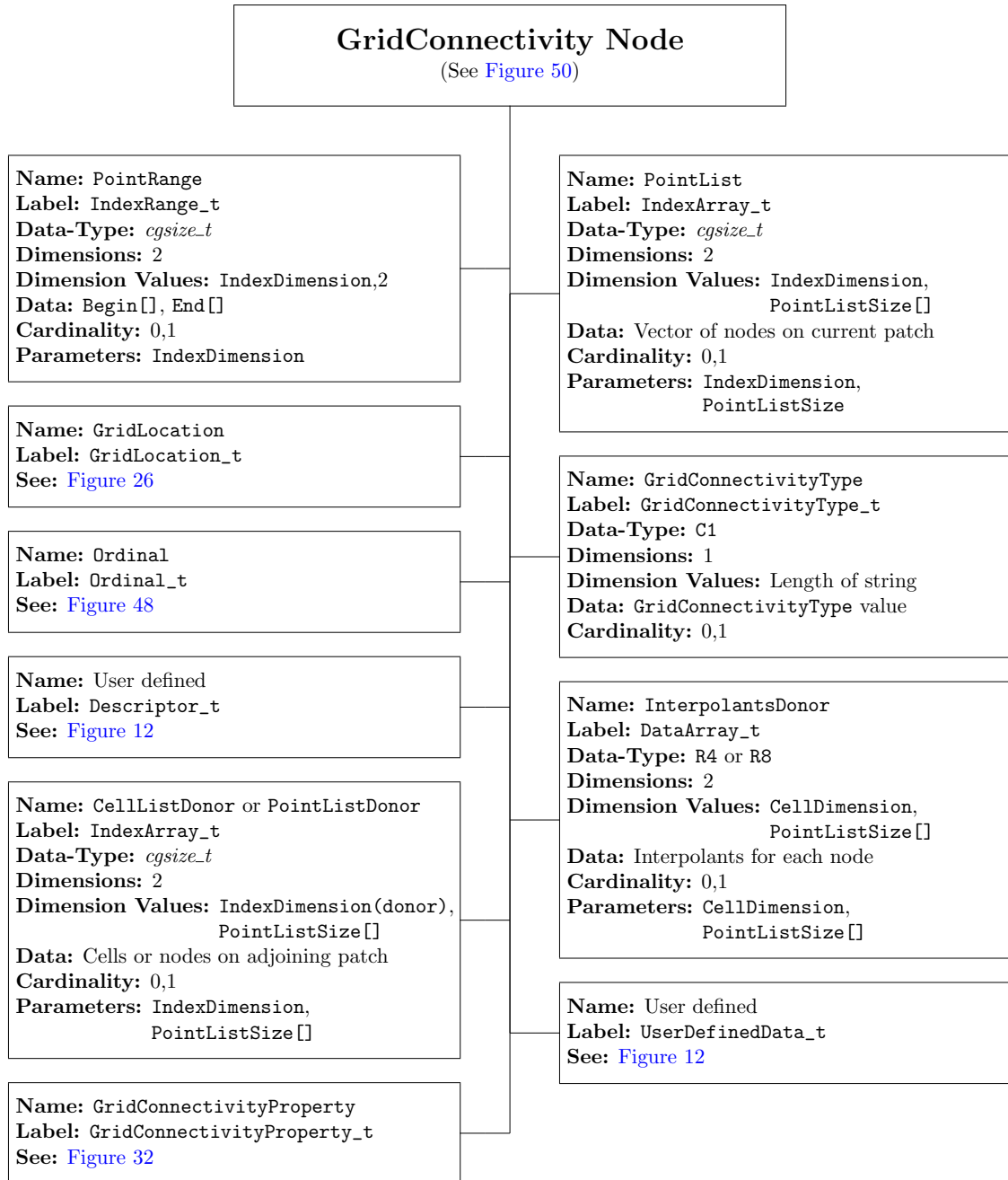
- The diffusion model defined here is only applicable to structured zones.
- For a structured zone, `IndexDimension` = `CellDimension`.

**Figure 29:** GoverningEquations\_t Data Structure



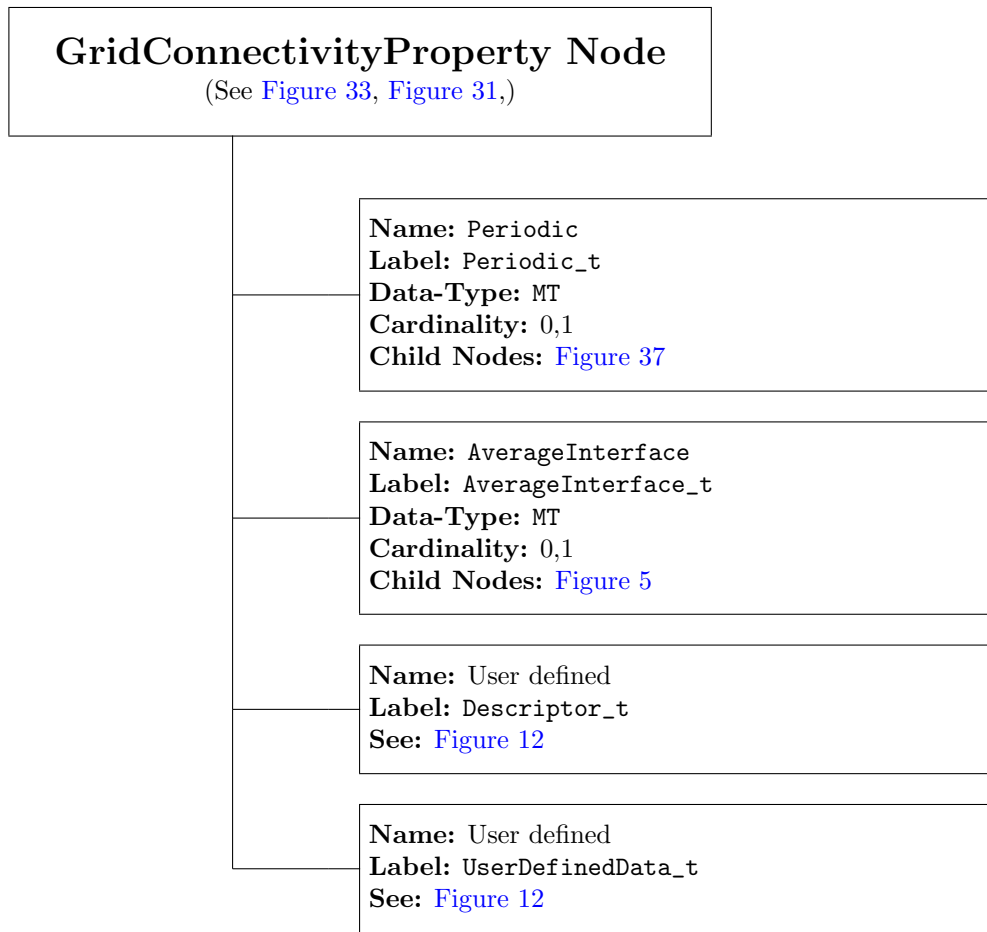
**Figure 30:** Gravity\_t Data Structure



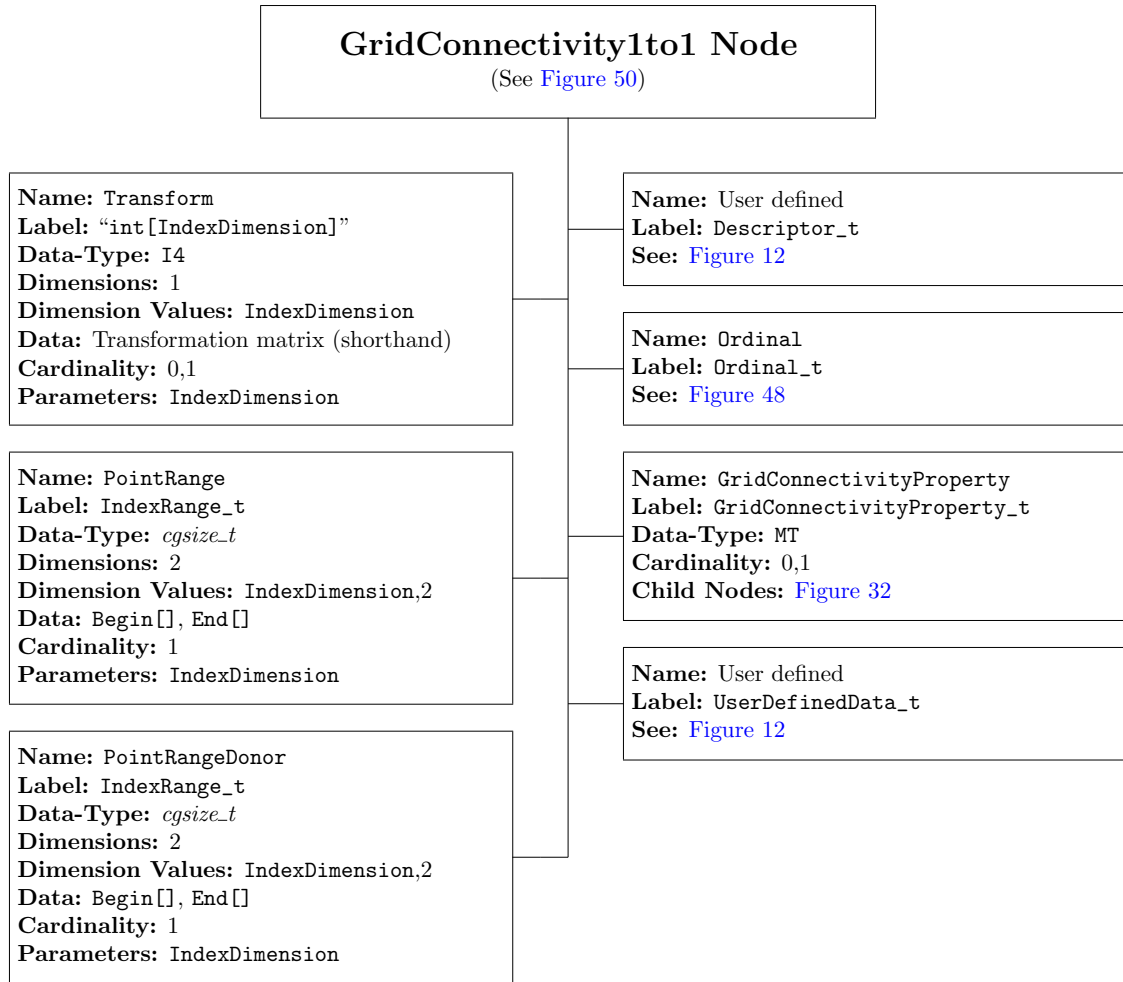
*Notes:*

- 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.
- *csize\_t* is determined by the application from which the node is written. For a 32-bit application *csize\_t* will be I4, and I8 for a 64-bit application.

**Figure 31:** GridConnectivity\_t Data Structure

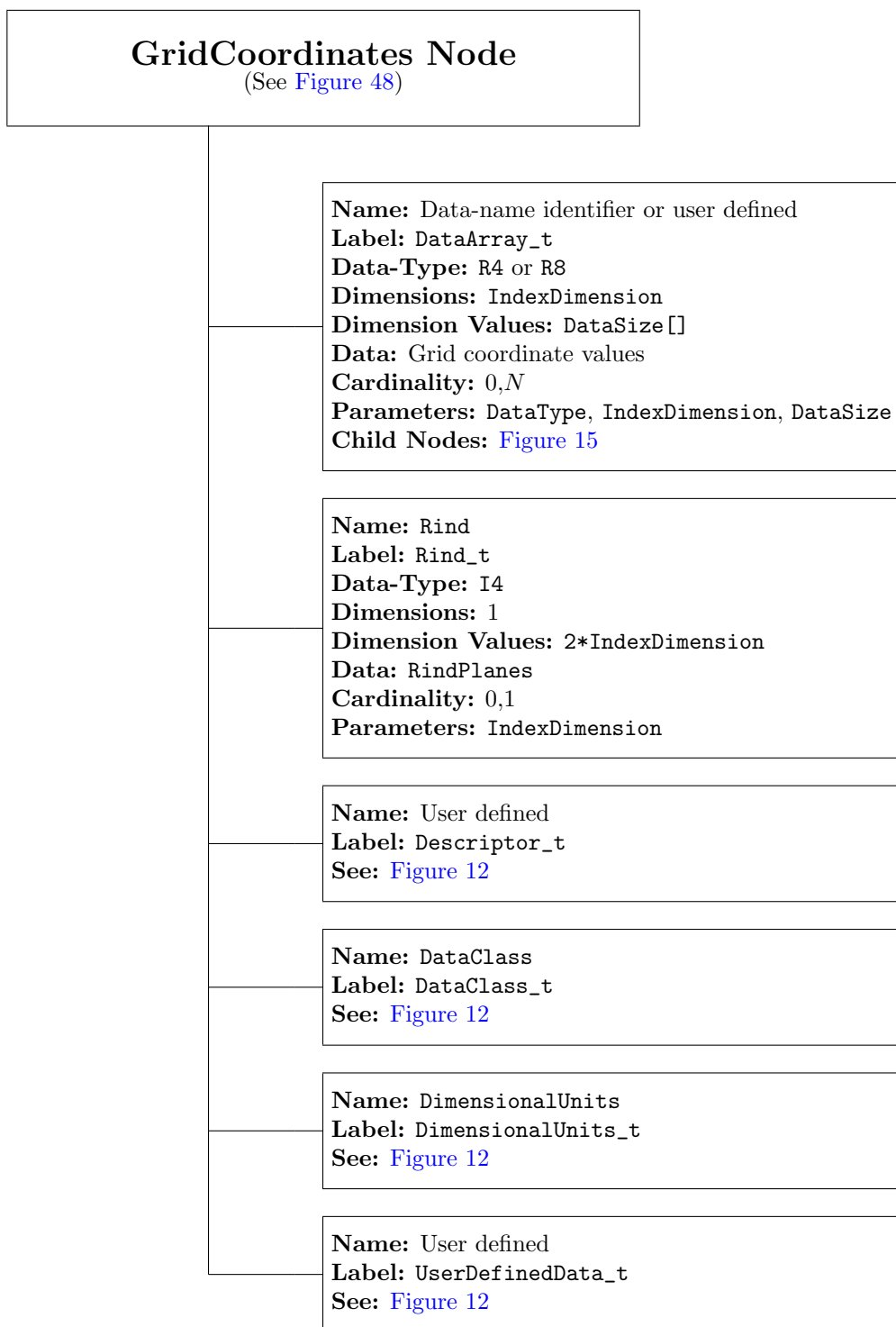


**Figure 32:** GridConnectivityProperty\_t Data Structure

*Notes:*

- GridConnectivity1to1 is only applicable to structured-to-structured 1-to-1 mesh connectivity.
- *cgsizet* is determined by the application from which the node is written. For a 32-bit application *cgsizet* will be I4, and I8 for a 64-bit application.

**Figure 33:** GridConnectivity1to1\_t Data Structure



*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 34:** GridCoordinates\_t Data Structure

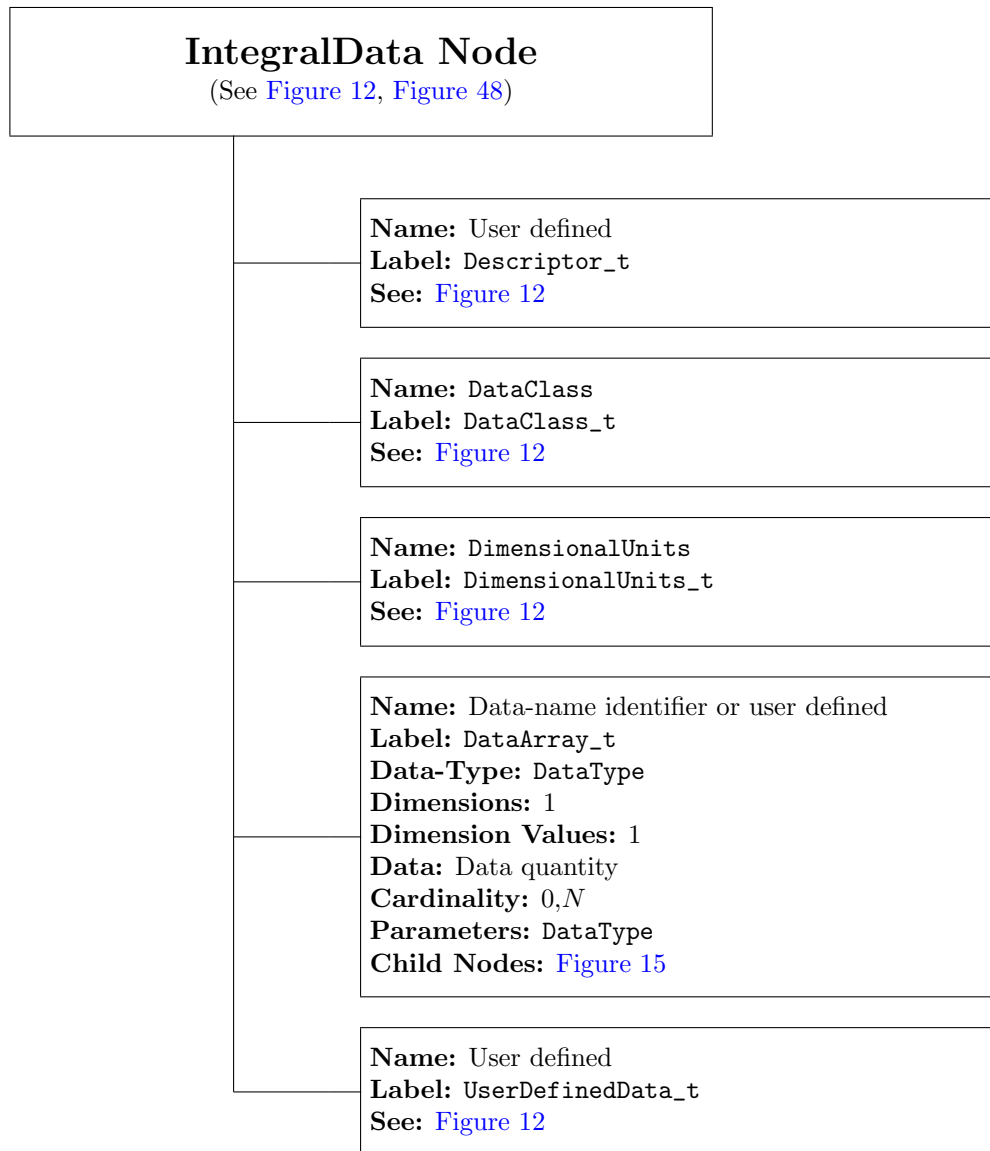
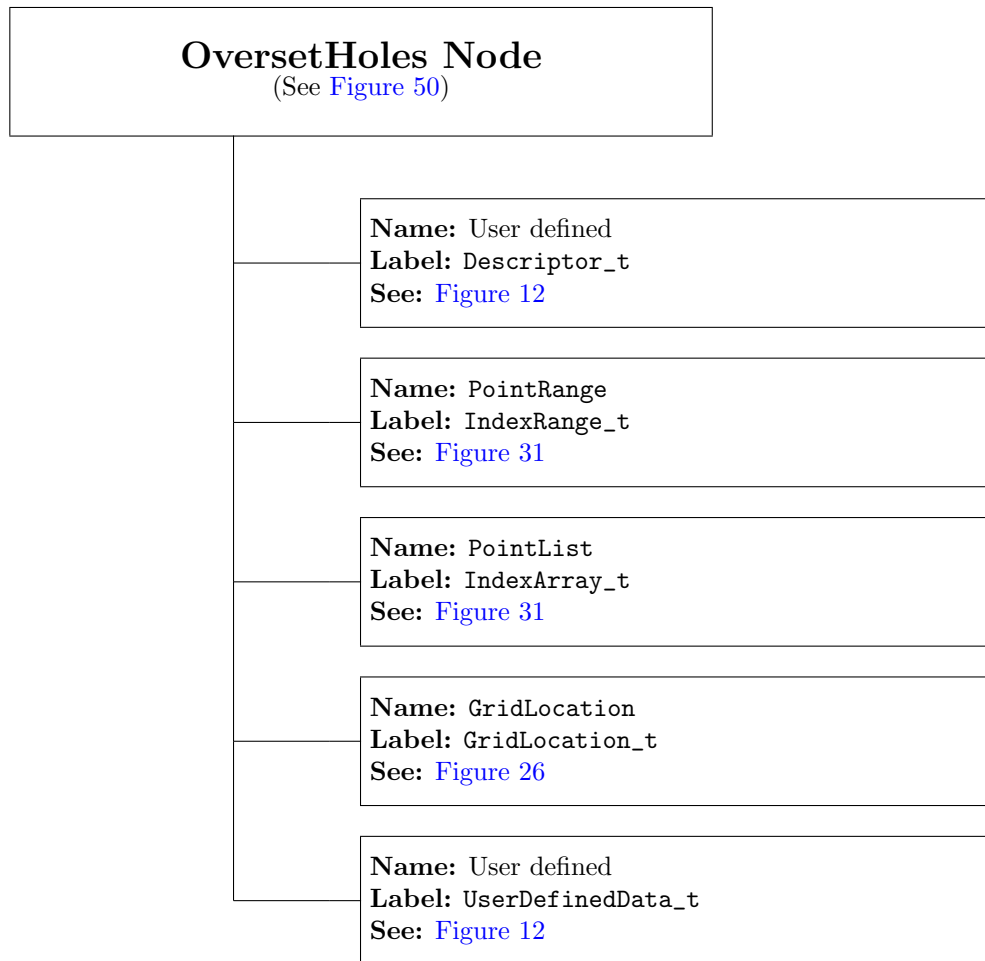


Figure 35: IntegralData\_t Data Structure



*Note:*

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

**Figure 36:** OversetHoles\_t Data Structure

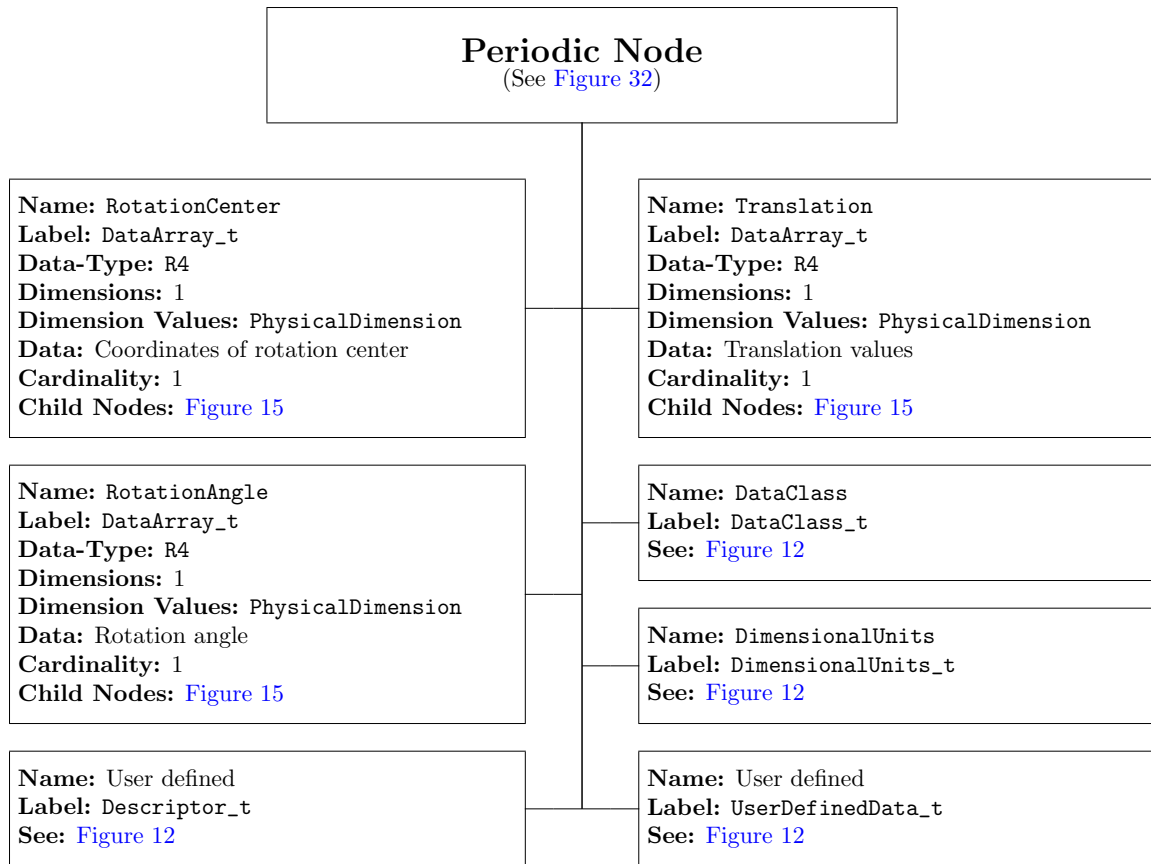


Figure 37: Periodic\_t Data Structure

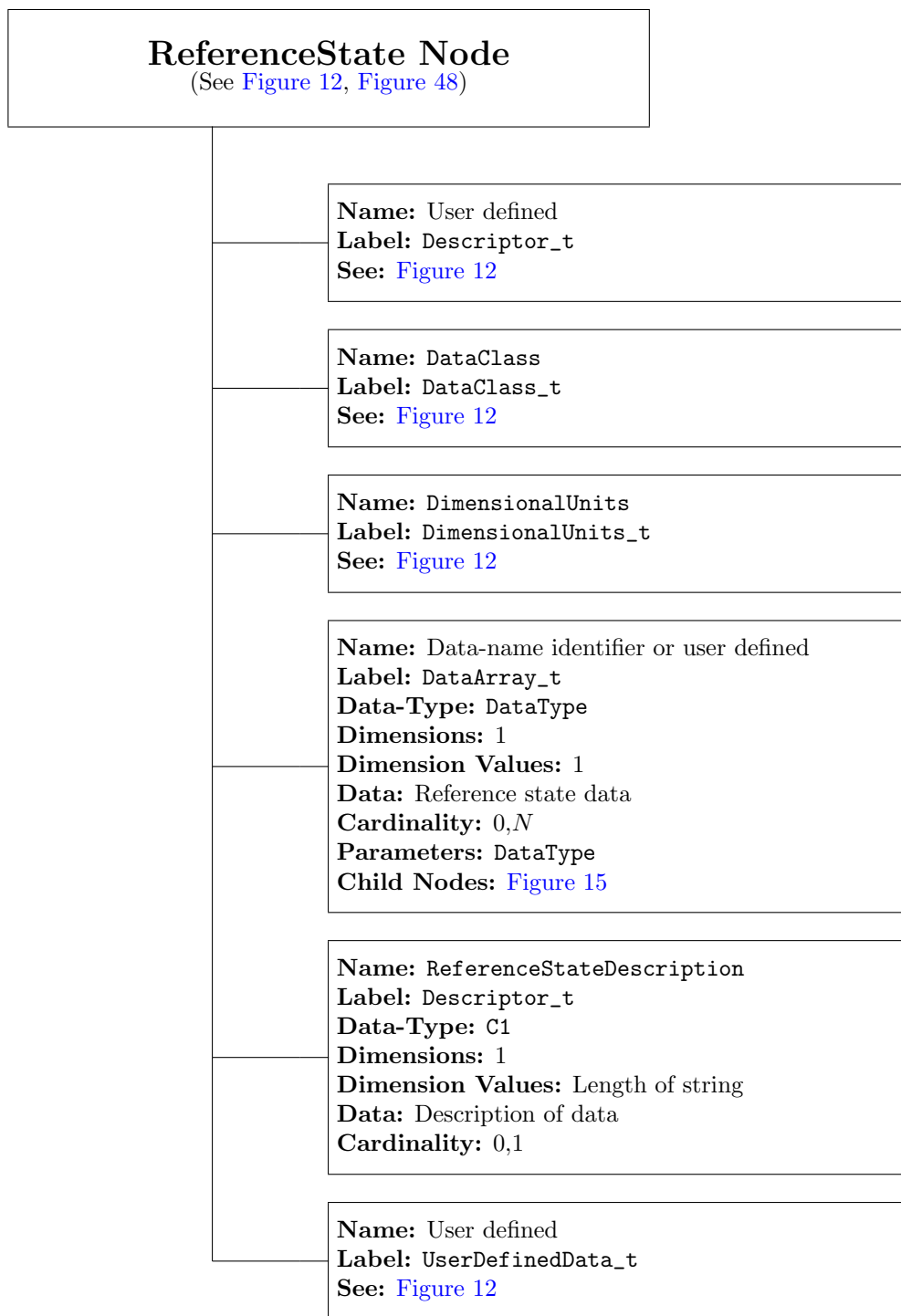


Figure 38: ReferenceState\_t Data Structure



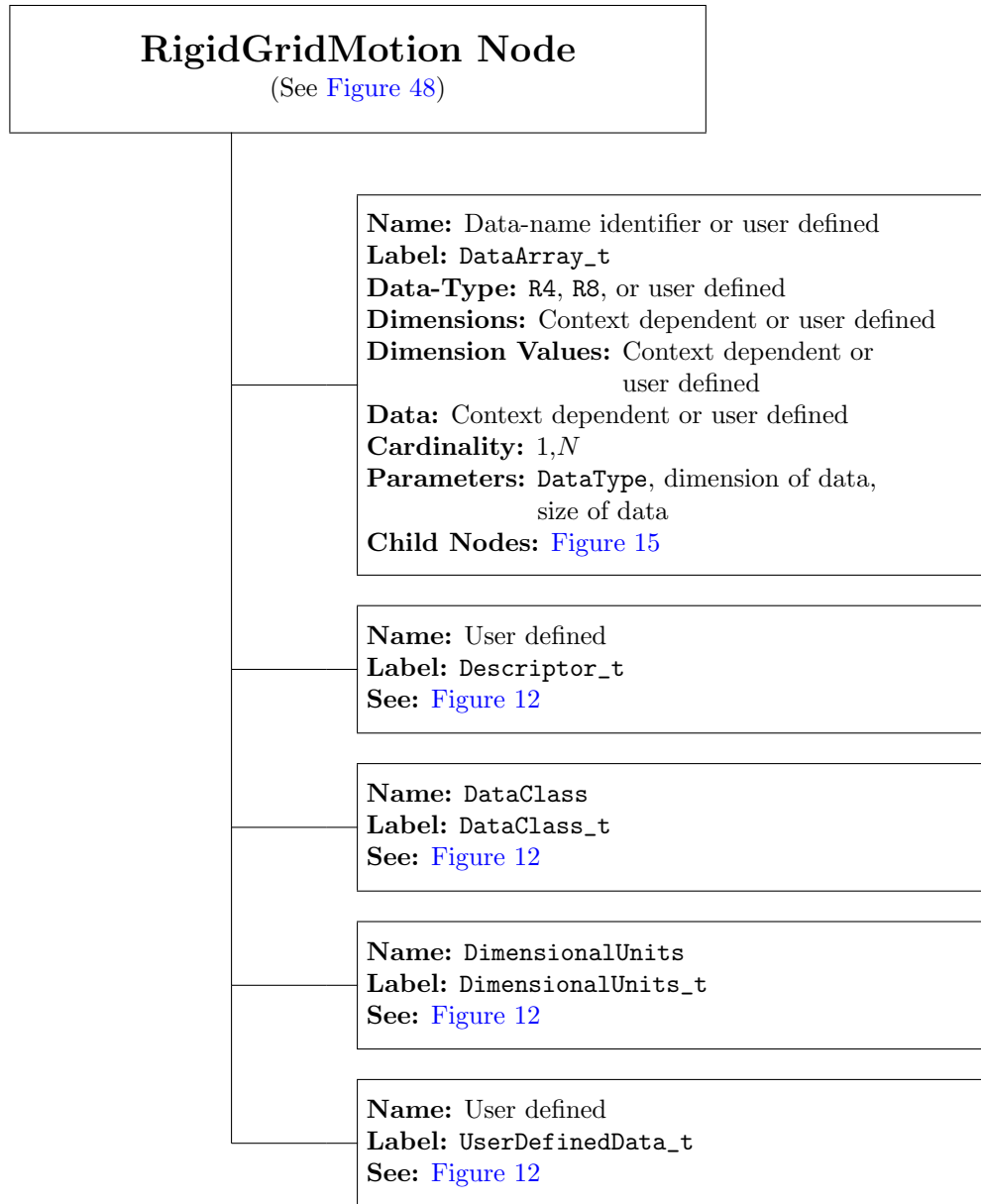


Figure 39: RigidGridMotion\_t Data Structure

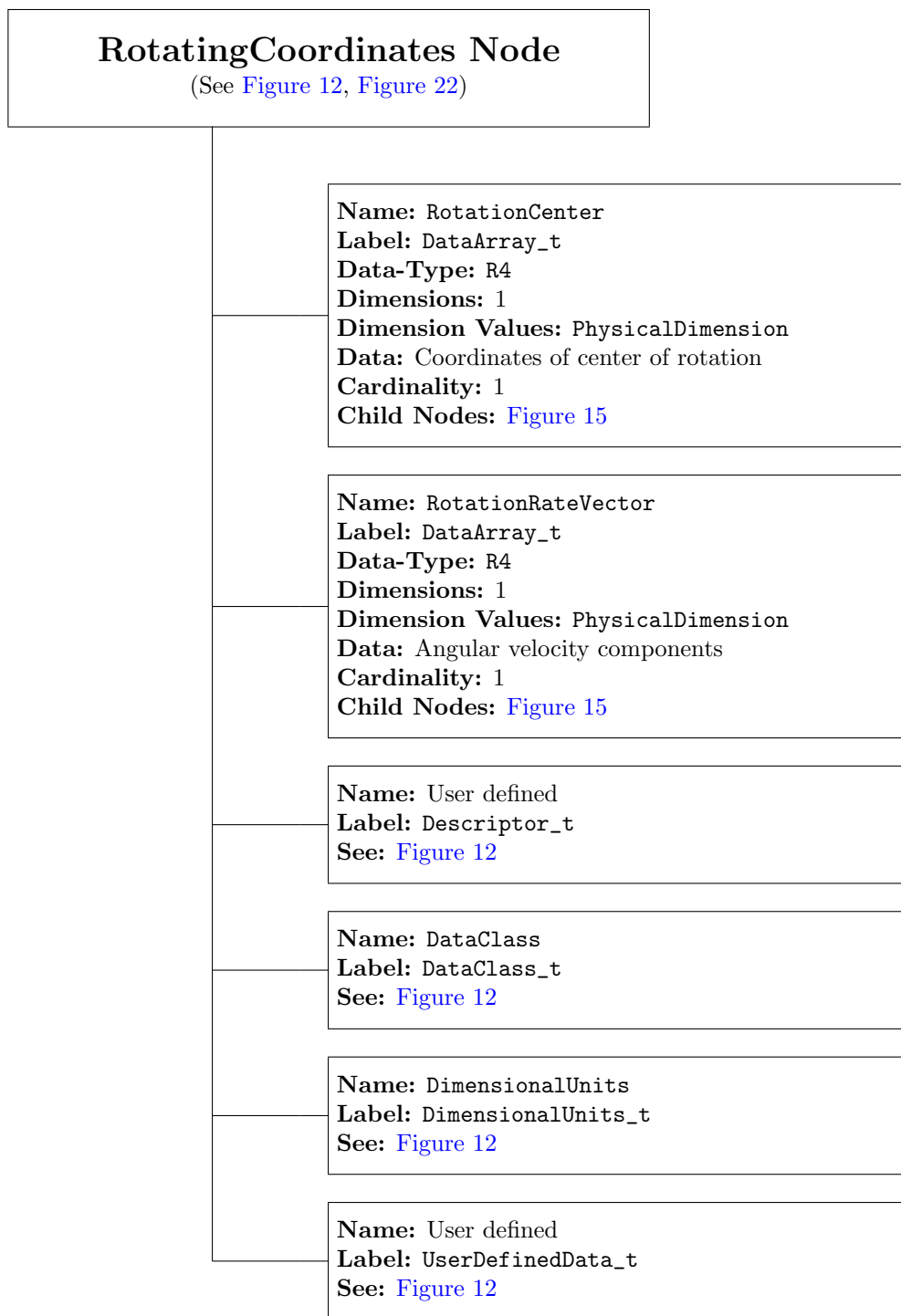
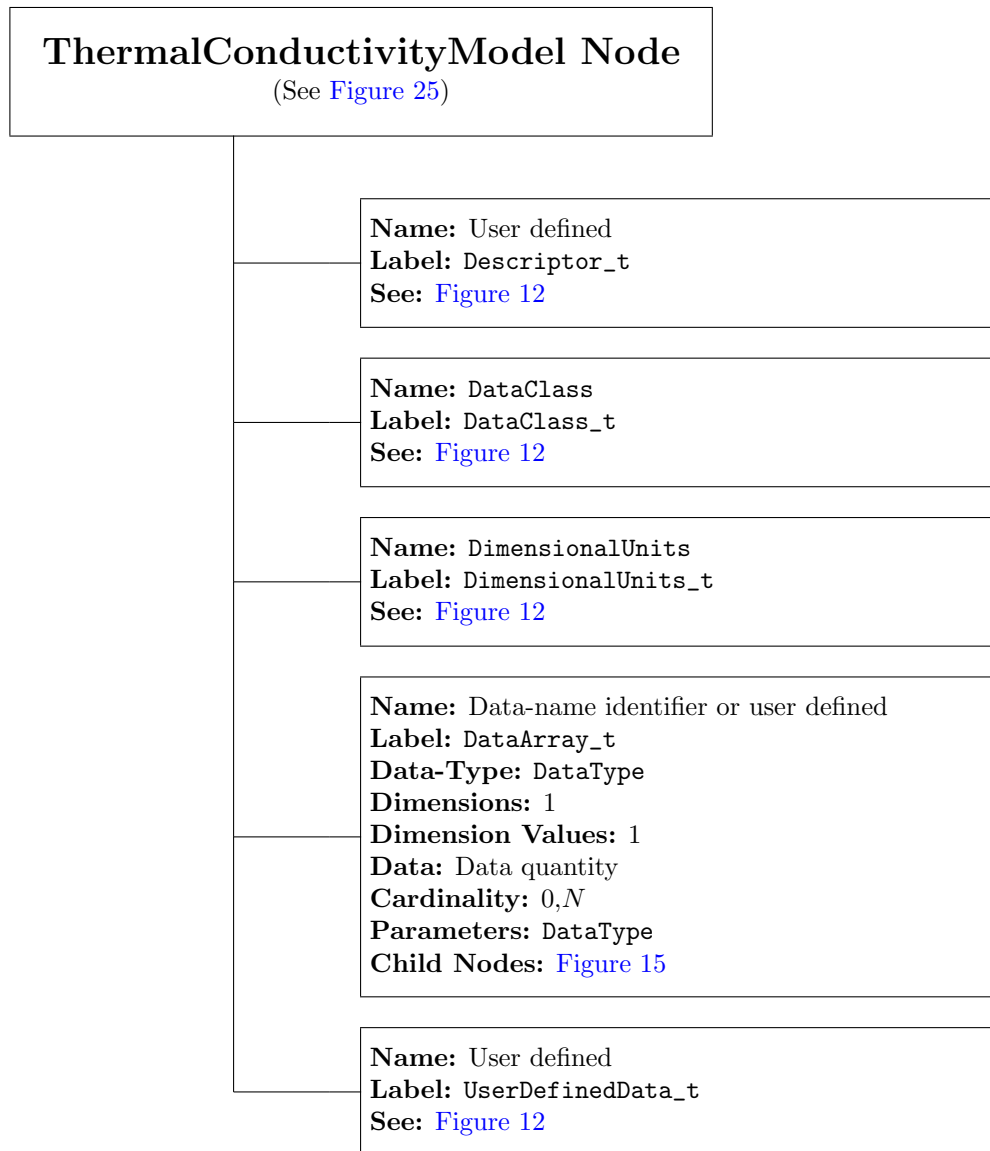
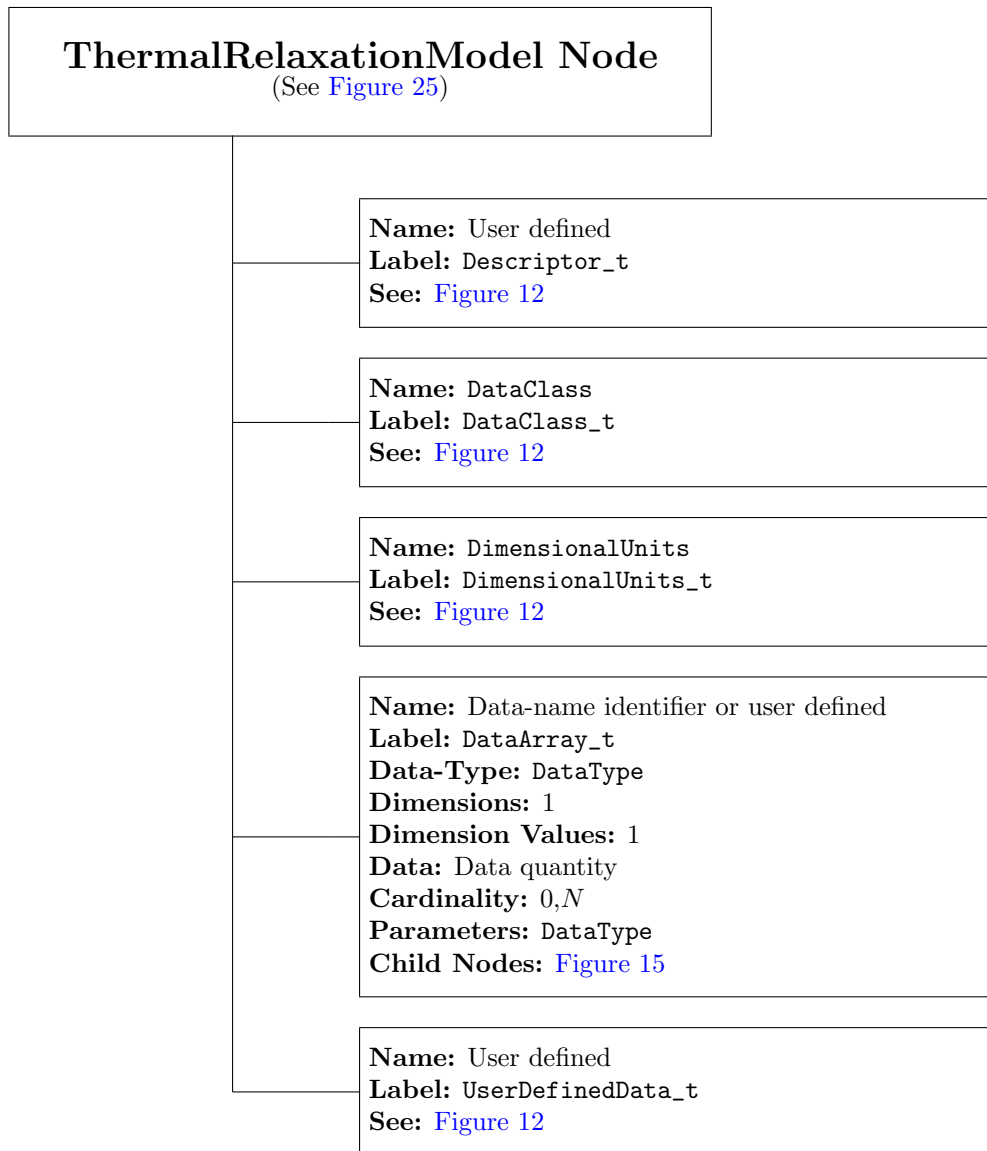


Figure 40: RotatingCoordinates\_t Data Structure

**Figure 41:** ThermalConductivityModel\_t Data Structure



**Figure 42:** ThermalRelaxationModel\_t Data Structure

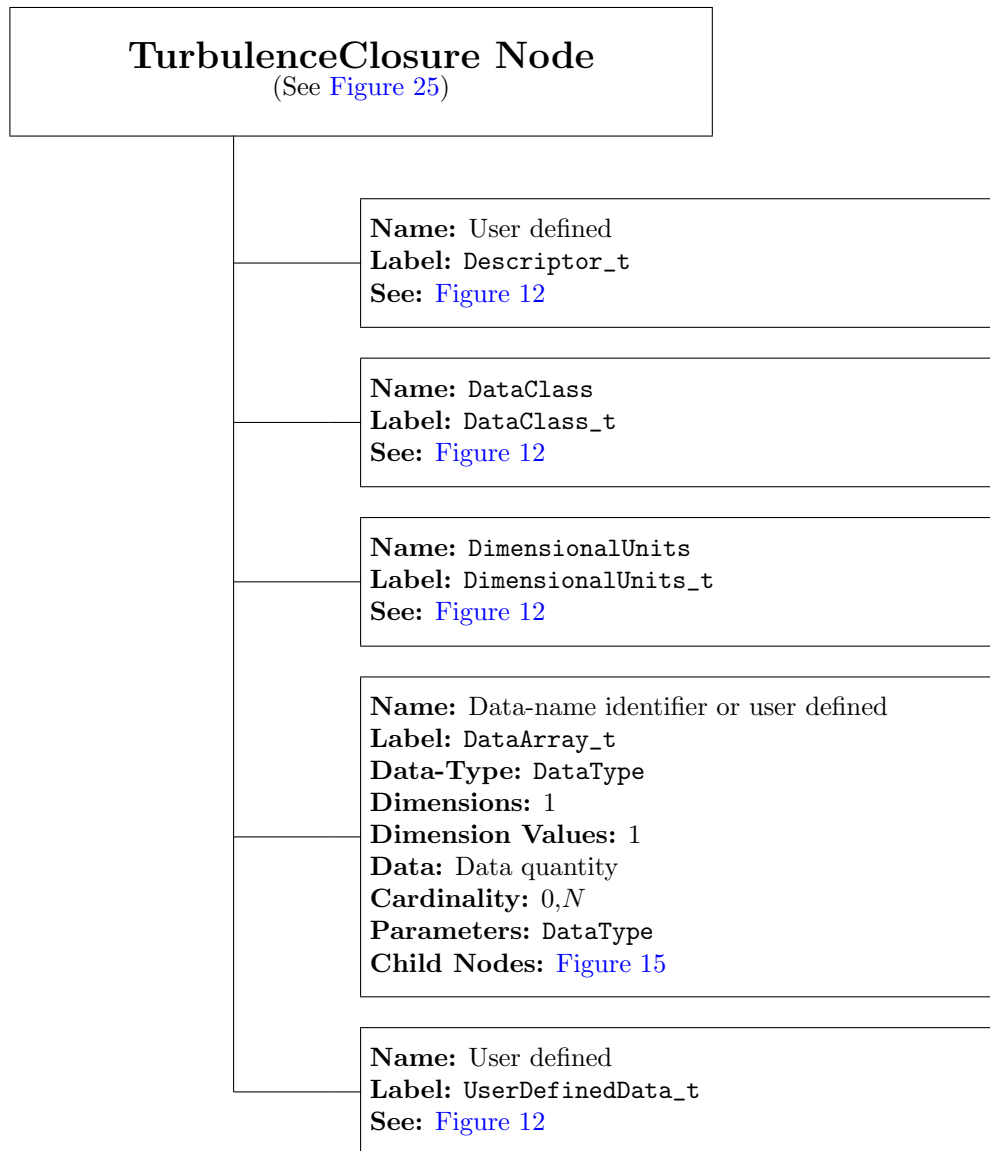


Figure 43: TurbulenceClosure\_t Data Structure



*Note:*

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

**Figure 44:** TurbulenceModel\_t Data Structure

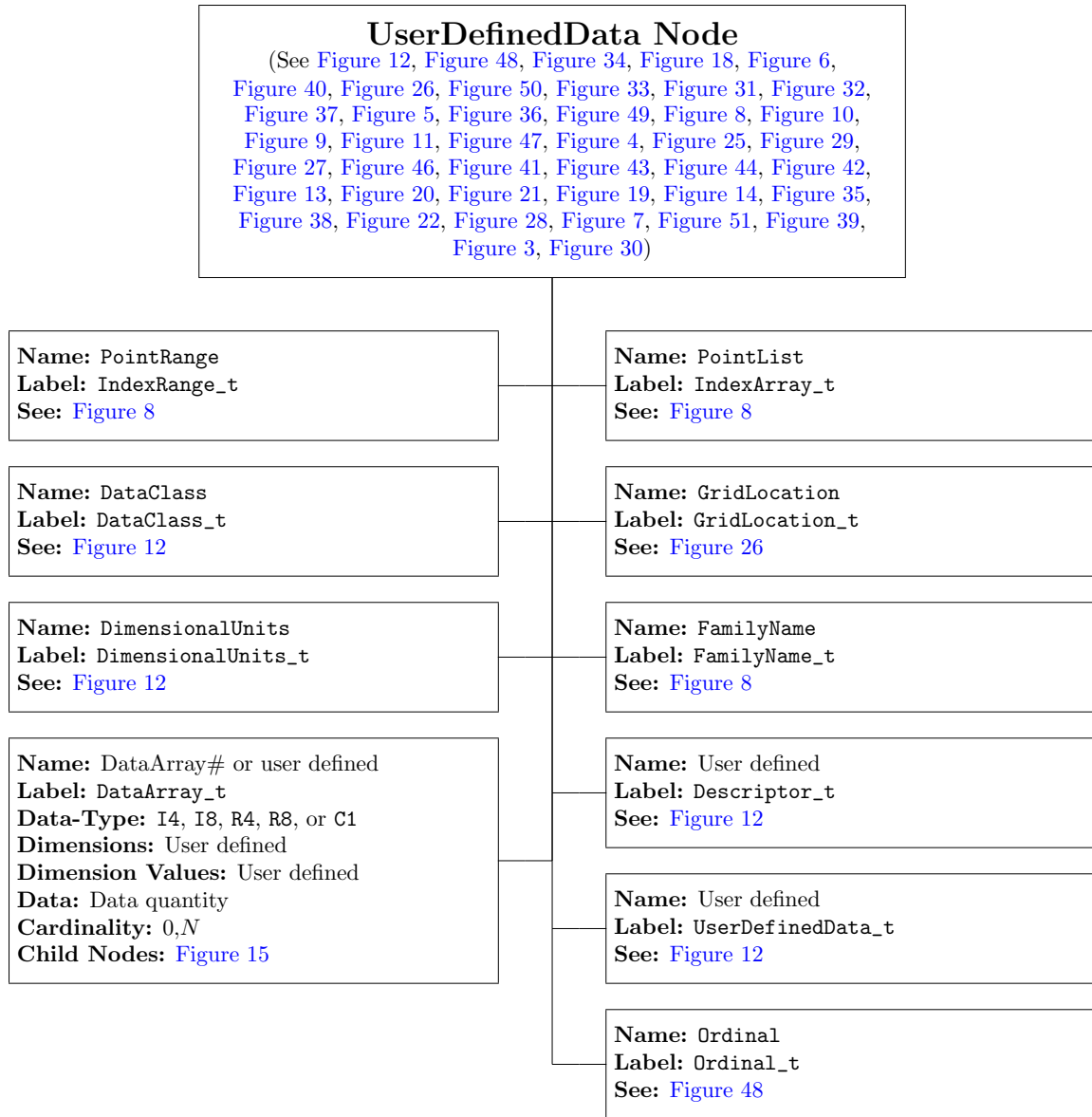
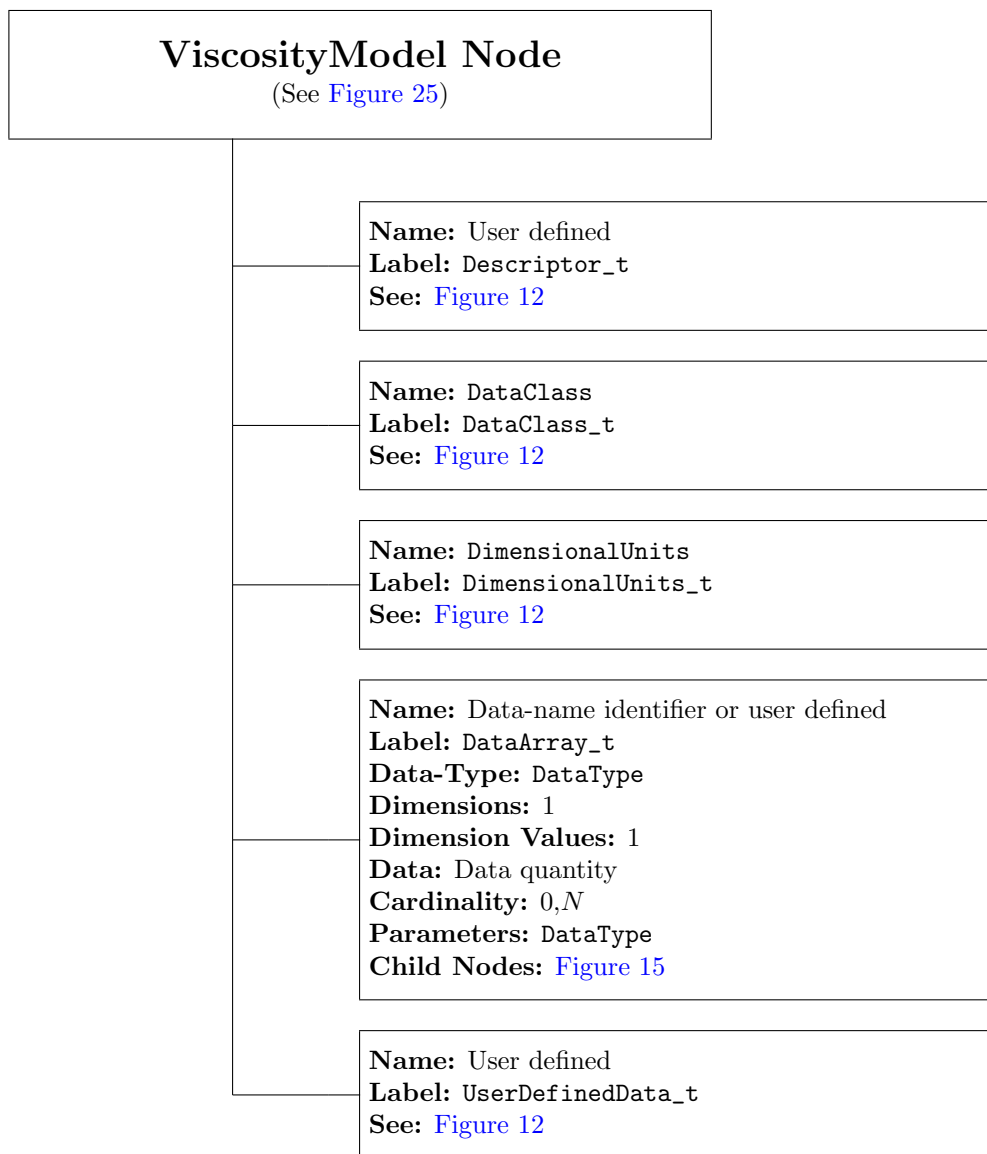
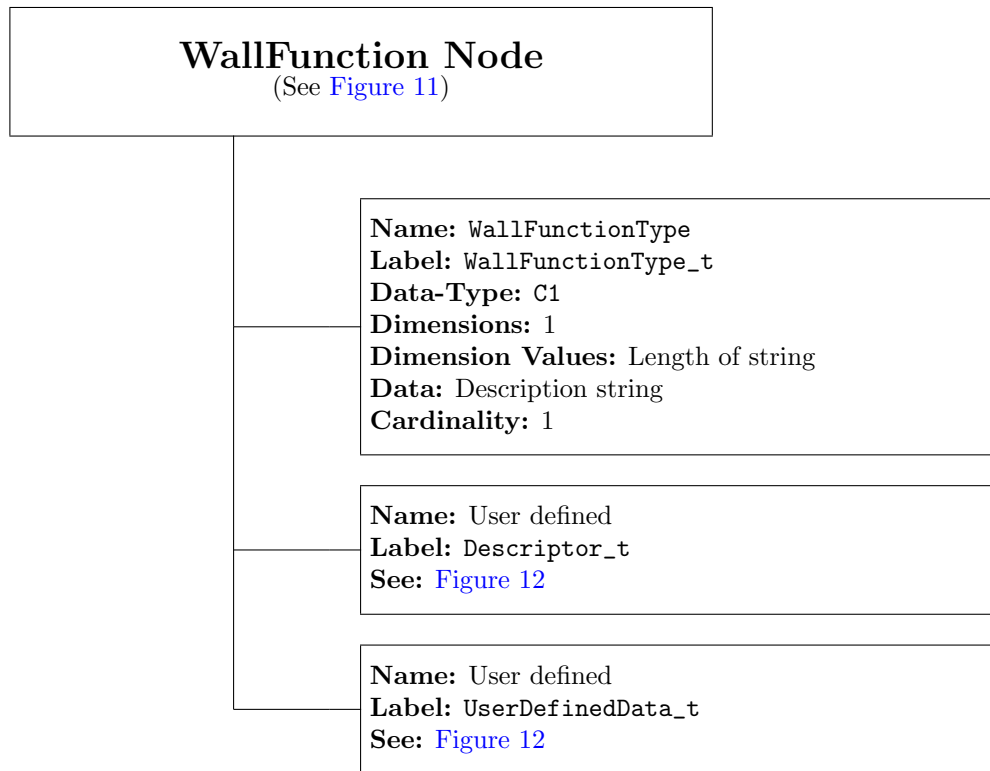


Figure 45: UserDefinedData\_t Data Structure



**Figure 46:** ViscosityModel\_t Data Structure



**Figure 47:** WallFunction\_t Data Structure

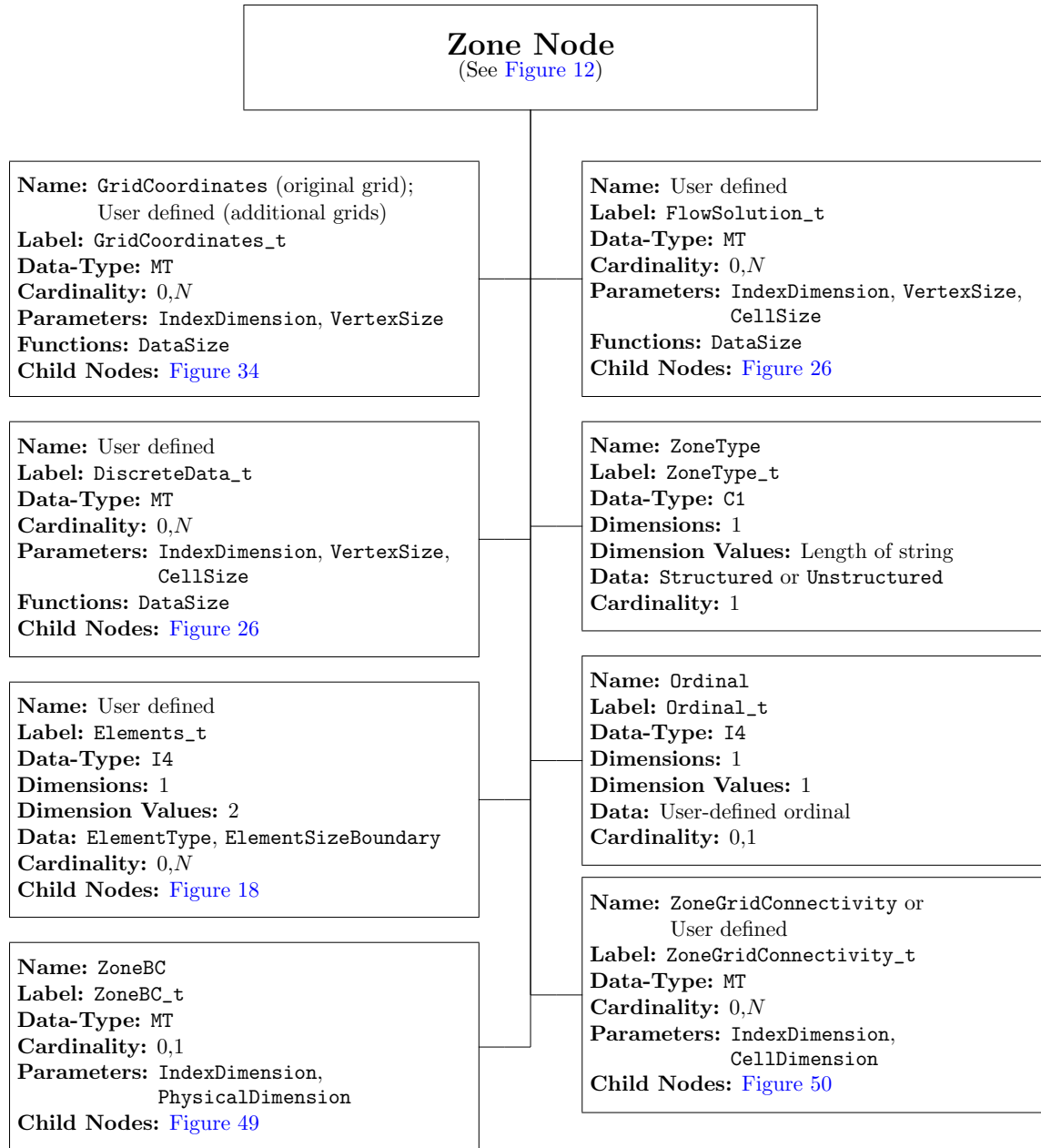


Figure 48: Zone\_t Data Structure (Continued on next page)

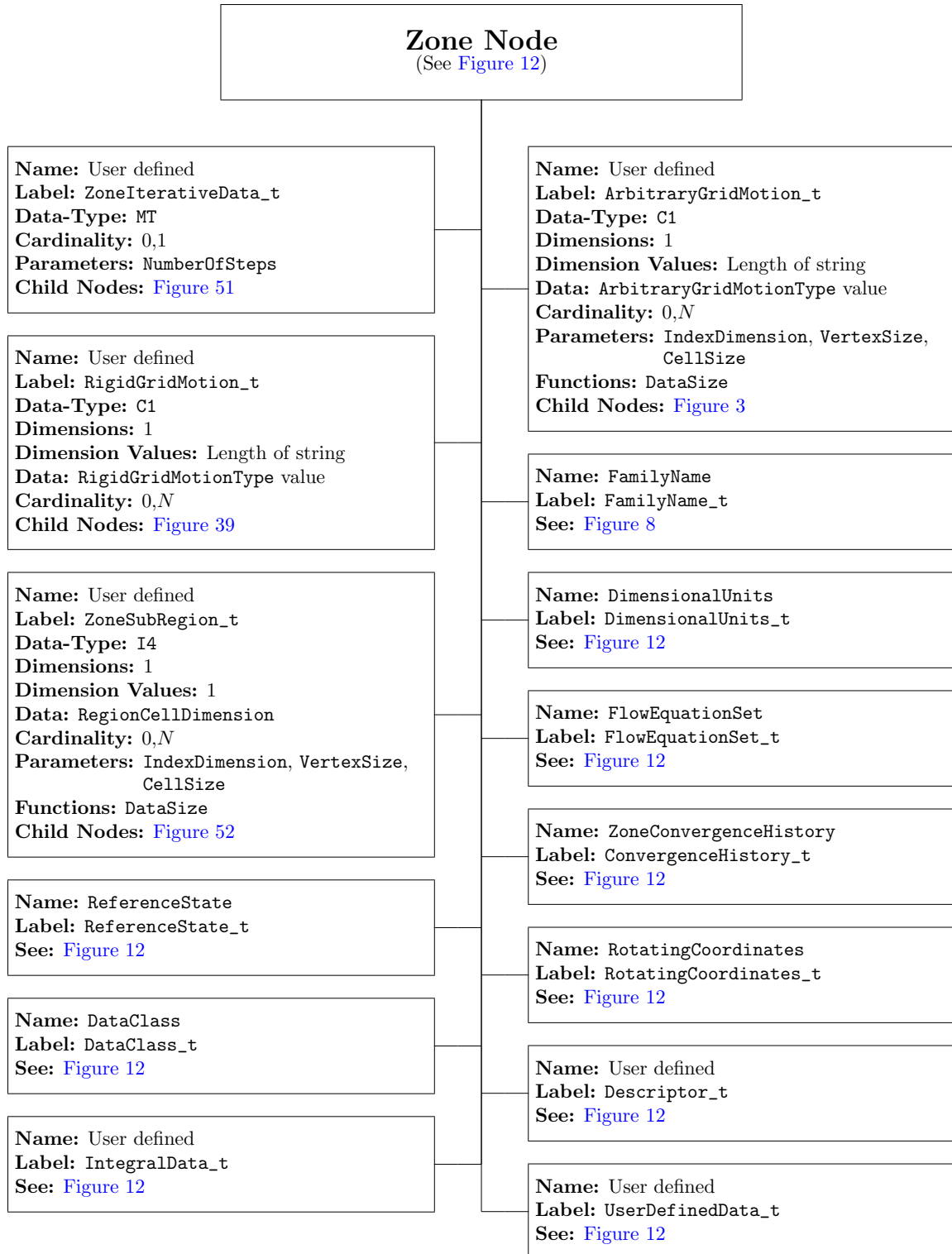


Figure 48: Zone\_t Data Structure (Continued from previous page)

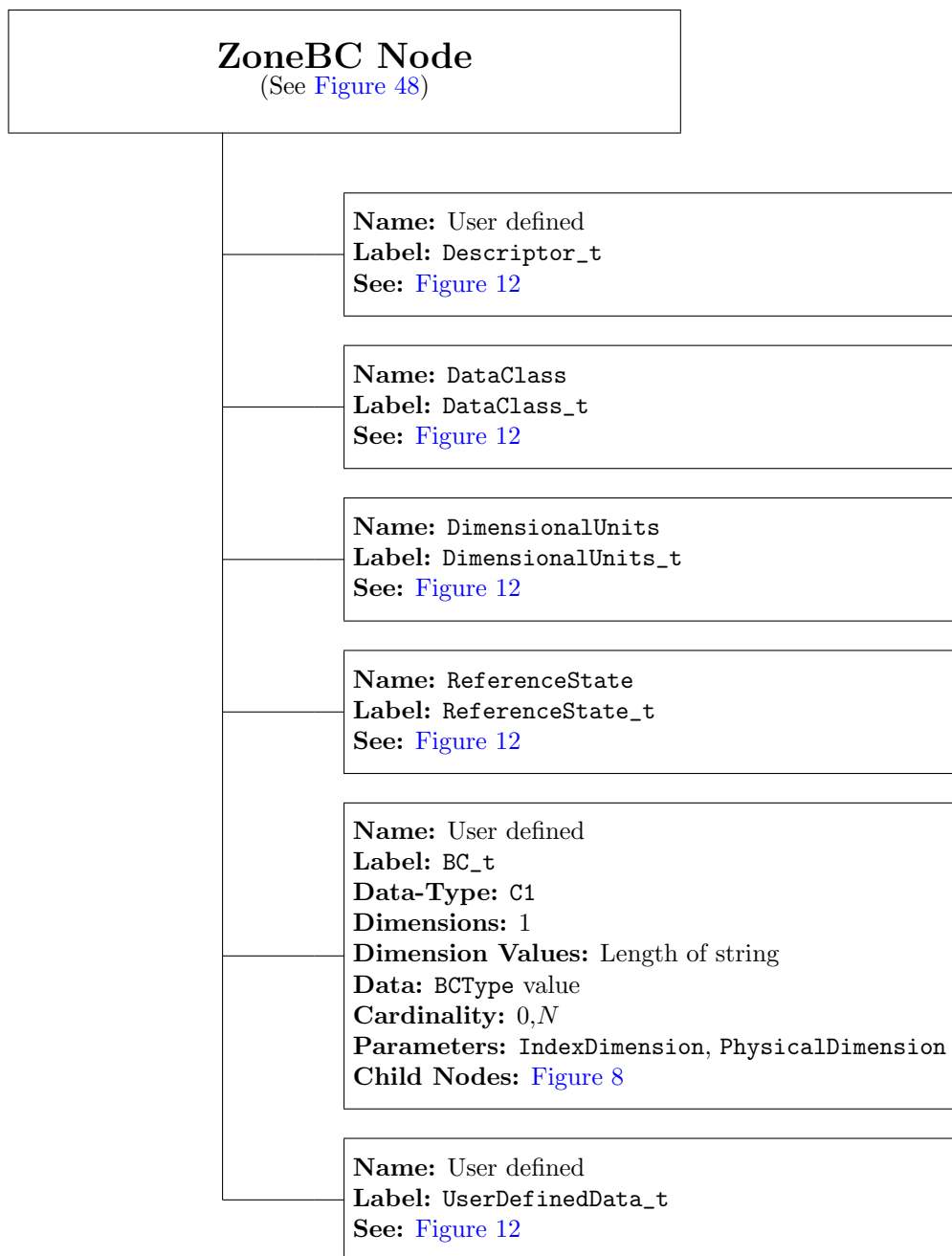
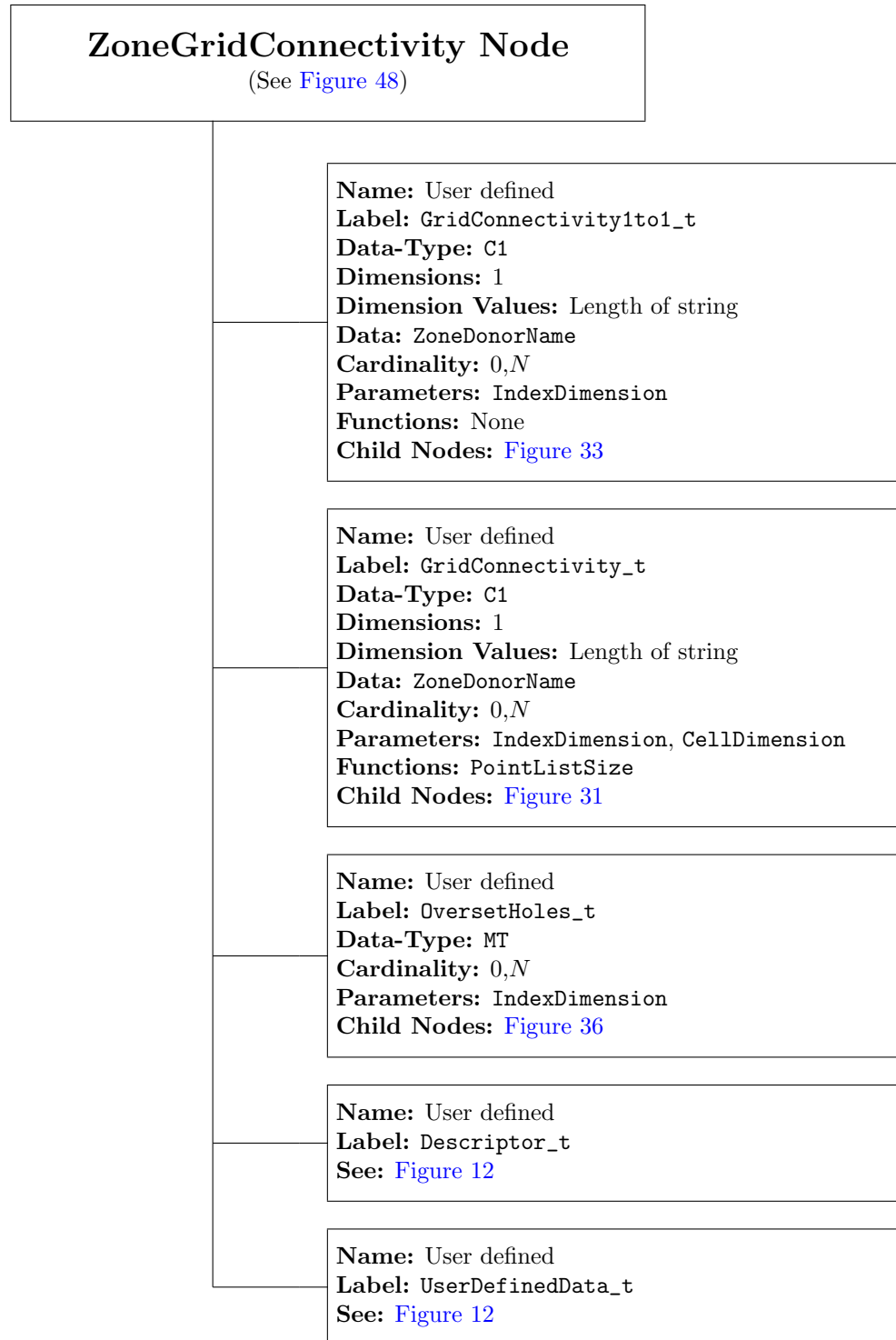


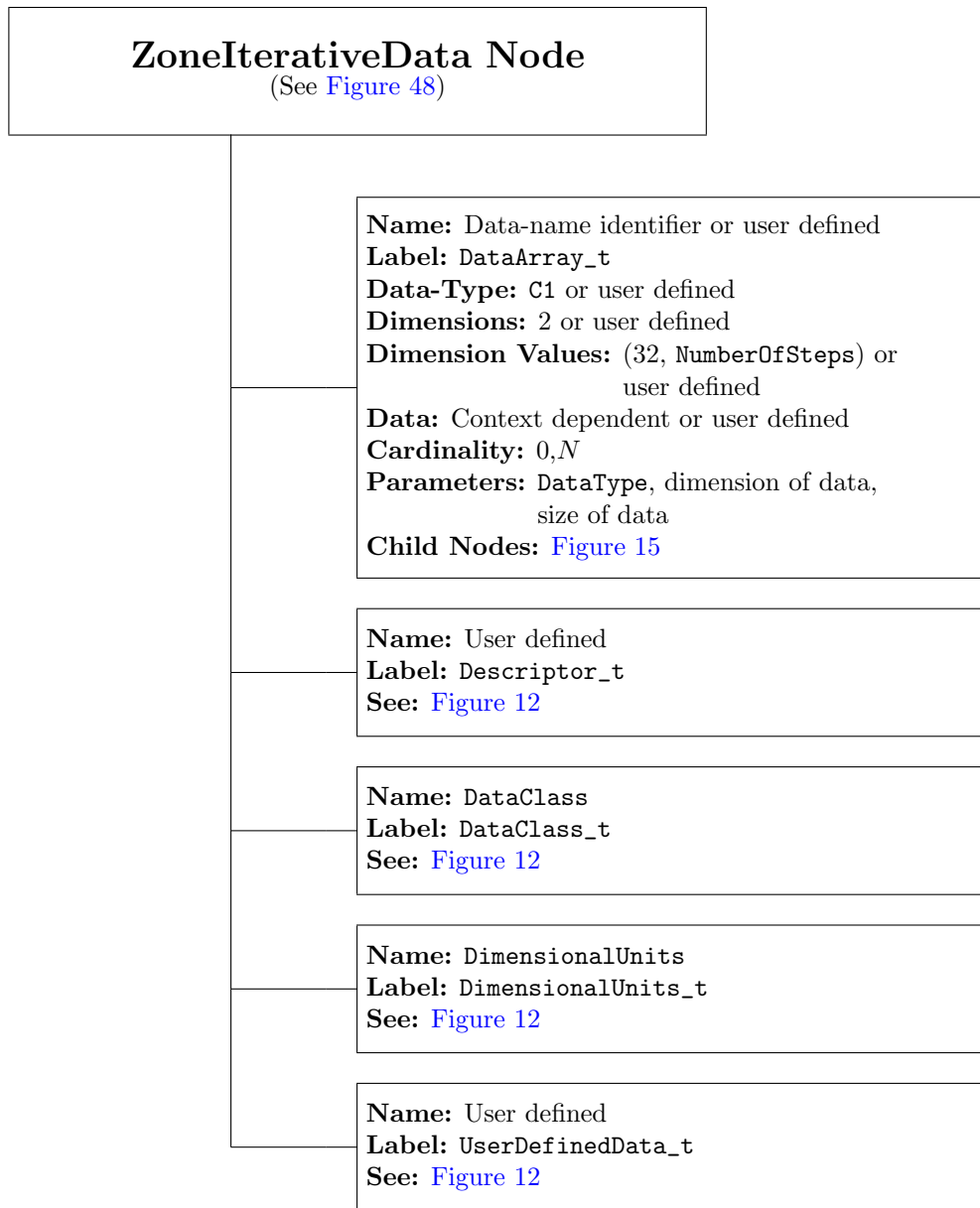
Figure 49: ZoneBC\_t Data Structure



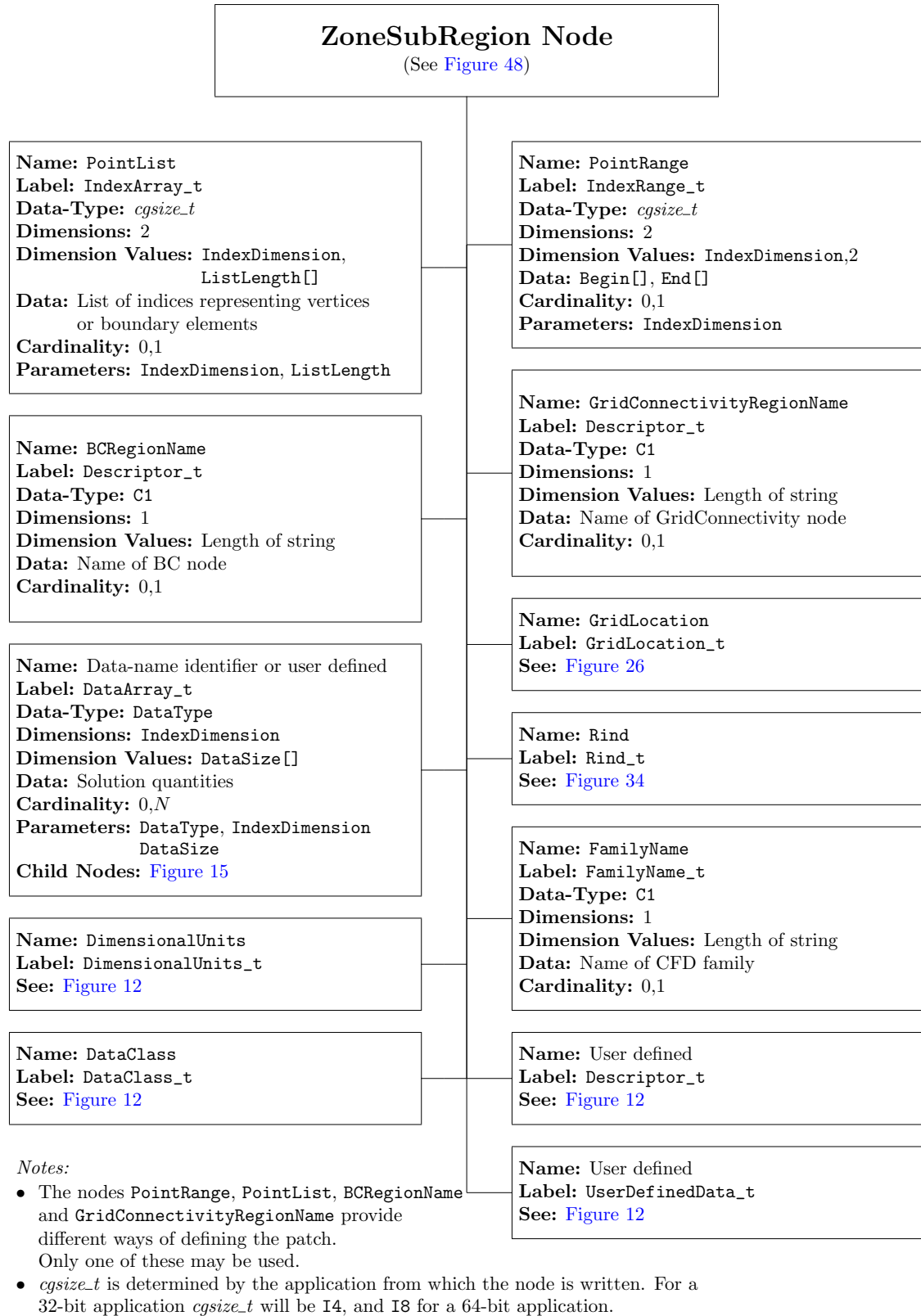
*Note:*

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

**Figure 50: ZoneGridConnectivity\_t Data Structure**



**Figure 51:** ZoneIterativeData\_t Data Structure

**Figure 52:** ZoneSubRegion\_t Data Structure