CFD General Notation System
HDF5 Implementation

Document Version 3.1.2

CGNS Version 3.1.3

Contents

List of Figures

1 HDF5 Implementation

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

2 SLL Mapping to HDF5

The purpose of the current document is to describe the way in which CFD data is to be represented in an HDF5 data tree. To do this, it is necessary to first describe the HDF5 data structure itself in some detail. Therefore, a conceptual summary of HDF5 is given here in order to make the current document relatively independent, and to allow the reader to focus on those aspects of HDF5 which are essential to understanding the file mapping.

Any HDF5 file with a conformant mapping is *CGNS/HDF5* compliant. The mapping has been made using a *per-node* basis. Instead of having a new mapping dedicated to HDF5, we have made a mapping from the ADF nodes to a set of HDF5 groups. While this is not an optimal mapping, the use of such an HDF5 node allows us to re-use the ADF API without change.

The HDF5 documentation should be used as the authoritative references to resolve any issues not covered by this summary.

2.1 General Description of HDF5

The HDF5 library provides CGNS with a low level storage system. This library is developed and maintained by the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign. It offers a free storage system, with support for a very large number of platforms used in the scientific world. This system is in charge of all physical features such as storage, compression, virtual access to data and many other services also useful for the CGNS users.

HDF5 allows a fine tuning of the physical devices, however, the SLL use of HDF5 forces all *property lists* to `DEFAULT` values.

Refer to the HDF5 web site at `http://hdf.ncsa.uiuc.edu/HDF5` for further information about HDF5.

2.2 Dedicated HDF5 Structures

The *CGNS node* is an HDF5 group. It contains attributes, an optional dataset and optional child groups. This structure is required and can be extended by other implementors as long as the following requirements are fullfilled.

In a CGNS tree, most of the nodes are *normal* nodes. These nodes can optionally contain data and have other nodes as children.

There are special nodes, such as the root node and the nodes managing the *links*. These special link structures use the HDF5 mount system. It is not necessary for either the MLL or SLL users to understand the HDF5 mount system, but it should be understood by SLL implementors.

2.2.1 The CGNS Node Mapping

The CGNS node uses *groups* and *attributes* elements of HDF5. A basic CGNS node is composed of elements listed in Table 1. The attribute names have been enclosed in square brackets, because the SLL uses private attributes with a leading *space* character. These brackets are not part of the names.

All the children groups of a node are understood as SLL children nodes of this node, unless the group name starts with a *space* character. The *dataset* contains the actual data of a node. In the

Table 1: Basic Node

GROUP		
REQUIRED ATTRIBUTE	[name]	U8LE[33]
REQUIRED ATTRIBUTE	[label]	U8LE[33]
REQUIRED ATTRIBUTE	[type]	U8LE[3]
REQUIRED ATTRIBUTE	[order]	H5T_NATIVE_INT
OPTIONAL DATASET	[data]	

case of a node without data (i.e. MT type), the dataset does not exist.

In addition to the attributes of a *normal* node, special attribute names are reserved for the root node (see Table 2). The name of the root node is [HDF5 MotherNode], its label is [Root Node of HDF5 File], type is MT.

Table 2: Root Node

GROUP	[/]	
REQUIRED ATTRIBUTE	[version]	U8LE[33]
REQUIRED ATTRIBUTE	[format]	U8LE[33]

The node hosting the link entries is a special node named [mount] without attributes nor data. This group should be a child of the root node, its absolute path is [/ mount]. The group [mount] contains one group for each mounted file refered to by a link. Each entry has the attributes described in Table 3. The link node itself is a group with the special name [link]. This implementation is discussed further in Section 2.2.10.

Table 3: Link Node

GROUP	[/ mount/ *]	
REQUIRED ATTRIBUTE	[refcnt]	H5T_NATIVE_INT
REQUIRED ATTRIBUTE	[file]	H5T_NATIVE_CHAR

2.2.2 The Node ID

The node ID is a unique identifier assigned to each existing node by HDF5 when the file containing it is opened, and to new nodes as they are created. SLL inquiries generally return node IDs as a result and accept node IDs as input. By building a table of IDs, calling software can subsequently

access specific nodes without further search. The Node ID is real and is not under user control, its lifetime is ended by the closing of the HDF5 file.

2.2.3 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. Constraints related to the node name are detailed in Section 3.1.2.

2.2.4 The Label

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

2.2.5 The Data Type

The Data Type is a 32-byte character field which specifies the type and precision of any data which is stored in the data field. Types provided by HDF5 are listed in Table 4.

Table 4: Data Types

Data Type	Notation	HDF5 Type
No Data	`MT`	-
Integer 32	`I4`	`H5T_NATIVE_INT32`
Integer 64	`I8`	`H5T_NATIVE_INT64`
Unsigned Integer 32	`U4`	`H5T_NATIVE_UINT32`
Unsigned Integer 64	`U8`	`H5T_NATIVE_UINT64`
Real 32	`R4`	`H5T_NATIVE_FLOAT`
Real 64	`R8`	`H5T_NATIVE_DOUBLE`
Complex 64	`X4`	-
Complex 128	`X8`	-
Character	`C1`	`H5T_NATIVE_CHAR`
Byte	`B1`	`H5T_NATIVE_UCHAR`
Link	`LK`	-

There is no mapping to HDF5 `MT` and `LK` types, because there is no actual *data space* associated with the nodes. The type information itself is stored in the node as the strings "`MT`" and "`LK`". The types `X4` and `X8` are also not mapped.

The data storage format is translated as described in Table 5.

Table 5: Native Formats

Native format	HDF5 Type
H5T_IEEE_F32BE	IEEE_BIG_32
H5T_IEEE_F32LE	IEEE_LITTLE_32
H5T_IEEE_F64BE	IEEE_BIG_64
H5T_IEEE_F64LE	IEEE_LITTLE_64

2.2.6 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.

2.2.7 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. These dimensions are stored by the *dataspace* associated with the *dataset*; no other attributes contains these values.

2.2.8 The Data

In an HDF5 node, the portion of the node holding the actual stored data array is a *dataset*.

2.2.9 The Child Table

All groups of the current group are said to be the children of this group, except the groups with a name starting with a space character. Each of these children groups is a *normal* node (i.e. group).

Children may be identified by their names and labels, and, thence, by their node IDs once these have been determined. HDF5 provides no notion of order among children, but the SLL layer adds a creation order stored in the `[ order]` attribute. This order is guaranteed to be the same from call to call, even after the file has been closed and re-opened.

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.

2.2.10 The link mechanism

A `LK` typed node is a *link*. Such a node refers to another node elsewhere. In other words, the *link* has no child or contents, but is a name of a node somewhere in the current file or in another file. The `LK` typed node is said to be the *emph* node, while the node elsewhere is said to be the *destination* node.

Is is the role of the SLL layer to insure consistency[1] and transparency of the link mechanism, so that any *normal* node request to the *link* node is performed as if it is performed on the destination node.

A *link* destination can be in the same file as the *link* source or in another file. In both cases, the *link* is made using an HDF5 soft link from source to destination. In the case of a destination in another file, the destination file is mounted in the `[/ mount]`[2] group and the soft link is made from the source to the destination now present in this mounted file.

Each time a file is mounted, the `[/ refcnt]`[3] attribute is incremented. The file is unmounted if there is no reference to itself.

[1] In particular in order to avoid an acyclic graph.
[2] Absolute path.
[3] Absolute path.

3 General CGNS SLL Mapping Concepts

This section describes the general philosophy underlying the use of the HDF5 tree structure by CGNS. **??** describes the exact conventions for each type of data.

In Section 3.1, we first describe the roles of the various HDF5 elements (i.e. *groups* or *attributes*) as they are specifically applied within CGNS. Section 3.3 describes the overall layout of the tree structure itself.

3.1 Use of HDF5 elements in CGNS

Section 2.2 described the general role of each of the HDF5 elements without reference to CFD. Here we note any additional information regarding their use within CGNS.

Attributes described in 3.1.1 through 3.1.8 are those recognized by both HDF5 and CGNS. In 3.1.9 through 3.1.11 we describe certain elements which are derived from context, i.e., which the node possesses by virtue of its location within a CGNS database. These notions, namely, Cardinality, Parameters and Functions, are unique to CGNS.

3.1.1 The Node ID

The Node ID is completely controlled by HDF5, and thus its role is exactly the same for CGNS as it is for HDF5. CGNS software accesses the Node ID only through calls to HDF5. HDF5 itself guarantees that Node IDs are unique and constant within any HDF5 file (or collection of files) while the file(s) are open.

3.1.2 The Node Name

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

Because every HDF5 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.

By extension, a node name is a series of names separated by a *slash* '`/`' (like the POSIX file system names), moreover '`/`' is the root name of the CGNS tree.

A CGNS Name can contain any printable ASCII character except the *slash* '/' and the *dot* '.' when this *dot* is the first character of the name.

3.1.3 The Label

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

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

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

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

3.1.4 The Data Type

Data Types are completely specified by the file mapping. Although HDF5 provides a number of types, in CGNS the only types used are MT (No Data), I4 (Integer), R4 and R8 (Real), C1 (Character), and LK (Link).

The specification of data types within the File Mapping allows for the probability that files written under different circumstances may differ in precision. The issue is complicated by the ability of HDF5 to sense the capabilities of the platform on which it is running and interpret or record data accordingly. The general rule is that, although the user of HDF5 can specify the precision in which it is desired to read or write the data, HDF5 knows both the precision available at the source and the precision acceptable to the destination and will mitigate accordingly. Thus to specify the precison of real data as R4, for example, has no meaning unless both R4 and R8 are available. Therefore, the generic specification "DataType" is used to allow for all possibilities.

For all integer data specified by the SIDS, I4 provides sufficient precision.

3.1.5 The Number of Dimensions

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

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

array in HDF5.

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

3.1.6 The Dimension Values

These are used exactly as specified by HDF5. In the case of rectangular arrays of physical data, the dimension values are set to the actual sizes in physical space. Note that these sizes often depend on whether the values are associated with grid nodes, cell centers or other physical locations with respect to the grid. In any event, they refer to the amount of data actually stored, not to any larger array from which it may have been extracted.

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

3.1.7 The Data

CGNS imposes no conventions on the data itself beyond those specified by HDF5. 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.

3.1.8 The Child Table

The Child Table is completely controlled by HDF5, and thus its role is exactly the same for CGNS as it is for HDF5. CGNS software accesses and modifies the child table only through calls to HDF5.

In addition to the meaning of attributes of individual HDF5 nodes, the File Mapping specifies the relations between nodes in a CGNS database. Consequently, the File Mapping determines what kinds of nodes will lie in the child table.

It is important to reemphasize that HDF5 provides no notion of order among children. This means children can be identified only by their names, labels and system-provided node IDs. In particular, the order of a list of children returned by HDF5 has nothing to do with the order in which they were inserted in the file. However, the order returned is consistent from call to call provided the file has not been closed and the node structure has not been modifed.

3.1.9 Cardinality

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

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

3.1.10 Parameters

CGNS relies on the fact that SLL 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*.

3.1.11 Functions

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

3.2 CGNS Databases

3.2.1 Definition of a CGNS Database

By definition, a CGNS database is created when, within an HDF5 file, a node is created which conforms to the specifications given below for a node of type "`CGNSBase_t`". This node is conceptually the root of the CGNS database. Because it is created and controlled by the user, it cannot be the root of the HDF5 file. Current CGNS conventions require that it be located directly below the HDF5 root node.

Further, by the mechanism of links, a CGNS database may span multiple files. Thus there is no notion of a CGNS *file*, only of a CGNS *database* implemented within one or more *HDF5 files*.

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

Conversely, there is no proscription against the inclusion, anywhere within an HDF5 file containing a CGNS database, of nodes of any form whatsoever, provided only that their naming and labeling does not mimic CGNS conventions. Such "non-CGNS" nodes, and those below them in the HDF5 tree, are not regarded as part of the CGNS database. CGNS software will not detect the existence of non-CGNS nodes.

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

> *A CGNS database is a subtree of an HDF5 file or files which is rooted at a node with label "*`CGNSBase_t`*" and which conforms to the SIDS data model as implemented by the SIDS-to-HDF File Mapping.*

3.2.2 Location of CGNS Databases within HDF5 Files

An HDF5 file may contain more than one `CGNSBase_t` node; i.e., there may be more than one CGNS database rooted within the same HDF5 file. CGNS software accepts the *name* of the desired

database as an argument, and will locate the correct `CGNSBase_t` node within the specified HDF5 file. Obviously, each `CGNSBase_t` node in a single HDF5 file must have a unique name.

A CGNS database may link to CGNS nodes in the same or other HDF5 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 HDF5 file into which the link is made is invisible except below the node to which the link is made.

3.2.3 File Management

Beyond *Open* and *Close* neither HDF5 nor CGNS provides any file management facilities. The user is responsible for ensuring that:

- The HDF5 file containing the root of the required database is available and its permissions are properly set at runtime.
- If links are made to other HDF5 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".

3.3 Internal Organization of a CGNS Database

3.3.1 The `CGNSBase_t` Node

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

3.3.2 The `CGNSLibraryVersion_t` Node

An HDF5 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 HDF5 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.

3.3.3 Topological Basis of CGNS Database Organization

Below the root, the organization of a CGNS database reflects the problem topology. Omitting detail, **??**, Appendix **??** shows the overall structure of the HDF5 file. Below the HDF5 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 ??, ??). The CGNS convention is that such a node overrides (for the associated portion of the topology only) any data found at a higher level.

3.3.4 Topics Not Currently Covered

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

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

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

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