CGNS/SIDS proposal for extensions – 2011/11/16 – v0.6 – Family Hierarchy
Author: Marc Poinot, ONERA/DSNA
Contact: marc.poinot@onera.fr

# Definition of a Hierarchy of families

This is version 0.5, taking into comments and modifications received from:
Chris Rumsey, NASA
Xiangmin Jiao, University of Stony Brook
Leigh Lapworth, Rolls Royce
Robert Bush, Pratt & Whitney
Richard Hann, ANSYS
Will Stoffers, Boeing

Specific Q&A can be found at the end of the proposal.

We want to allow a hierarchy of *Families*. For example we want to have a *Plane* label[1] identifying all zones that belong to the airplane, a *Wing* label identifying the wings and so on. All *Wing* zones would have both the *Plane* and the *Wing* label and we would like to declare that *Wing* is child label of *Plane*, or that *Plane* is the parent of *Wing*. When an application is parsing a zone, it can find more than one label and we have to insure backward compatibility with existing CGNS versions. We give details in the following sections.

We use the Family concept of CGNS/SIDS for that purpose because it is close to our needs. The proposal is given in Sections 2 and 3 below. First, we need to go back to this *Family* concept in CGNS/SIDS to be clear on the actual effect of the extension to existing CGNS/SIDS.

[1] In this document, label does NOT refer to the CGNS/SIDS label or node type except in the section 3 example. A label is a string.

# 1. CGNS/SIDS Family concept

A CGNS/SIDS *Family* is no more than a label. The main goal is to factorize in a single node (the *Family*) some data that can be referred to by other nodes without copying this data. This reduces update, maintenance and as minor effect the disk or memory footprint of a CGNS tree. The implementation in a CGNS/SIDS tree of nodes has two aspects:

**The declaration:**

A declaration of a family consists in a new `Family_t` node as a child of a `CGNSBase_t` node. This new node has a CGNS/SIDS type `Family_t`, a name (so-called the family name) but no value (all information is declared in the children of this `Family_t` node)[2]. As a `CGNSBase_t` node may have more than one `Family_t`, its CGNS/SIDS definition in `CGNSBase_t` is:

```
List( Family_t Family1... FamilyN ) ;                                (o)
```

Which does not define a *list* as a type, but rather defines an **unordered** set of children with the same type, such a declaration is equivalent to:

```
Family_t Family1;                                  (o)
Family_t Family2;                                  (o)
...
Family_t FamilyN;                                  (o)
```

With *Family1* to *FamilyN* user defined names of the `Family_t` nodes. There is no `Family_t` node reserved name. The `Family_t` nodes can only be defined as children of a `CGNSBase_t` node. One can currently add children nodes to the `Family_t` node, as specified in CGNS/SIDS section 12.6.

**The reference:**

The actual use or reference to a family consists in the addition of the name of a family as child of a `Zone_t`, a `ZoneSubRegion_t` or a `BC_t` node. The reserved name for this node is *`FamilyName`*, its value (or contents) is the actual name of the referred-to family. An example of a CGNS/SIDS definition, in the `Zone_t` here, is: `FamilyName_t FamilyName;`

This reference is then no more than a label, the application using such a CGNS tree has to find the corresponding `Family_t` node and parse its children to get the actual information. As a matter of fact, only one `FamilyName_t` node is allowed as child of a `Zone_t`, a `ZoneSubRegion_t` or a `BC_t` node, due to the reserved and mandatory name *`FamilyName`*.

**Remarks on implementation into CGNS/MLL**

The `CGNSBase_t` level functions always define the familyname as argument. It is possible to have more than one `Family_t` node at `CGNSBase_t` level. A new `FamilyName_t` node is added using first a `cg_goto` to set the current cursor as a `Zone_t`, `ZoneSubRegion_t` or `BC_t` node. Then a c`g_famname_read` or `cg_famname_write` is used to get/set the value of the `FamilyName_t`. The implementation looks for `FamilyName_t` nodes, if the number is more than one there is an error. The write forces the `FamilyName` name for the node. The actual storage of the family name in zone, zonesubregion and bc nodes is a `char_33` (again only one value can be stored).

2 A CGNS/SIDS node has a type, a name and a value. `FamilyName_t` is a type, `FamilyName` is a name, `LeftWing` is a value. In this case the value refers to another node **name**: `Family_t` is the type, `LeftWing` is the name, no value.

## 2. CGNS/SIDS extension proposal

We propose to allow a new list of `FamilyName_t` nodes as *children* of the *`Family_t`*. These names would refer to an existing `Family_t` node. This creates a relationship between two or more `Family_t` nodes.

A hierarchy of families is then possible, because we can decide that a specific `FamilyName_t` node value (not the name) refers to its parent. (It is also allowable but more difficult to define children, because one would have to create an extensible table or list with all children, and would have to update this table each time a child is added or removed.) By definition, the `FamilyName_`t with the name *`FamilyParent`* would have the current `Family_t` node parent as its value. The *`FamilyParent`* name is reserved for this purpose. For example, one could define *Wings* as parent of *LeftWing*, and *LeftWing* as parent of *LeftFlap*. In other words, a child has knowledge about his parent, but a parent does not know how many children it has, unless the user application makes such a computation.

A family is also allowed to be a member of multiple other auxiliary families, if desired. For example, one could have *LeftFlap* not only be a child of *LeftWing*, but also be a subset of some secondary family called *Output#1* for postprocessing purposes. Such a simple extension could be used to mimic some CAD related hierarchy.

```
Family_t :=
   {
   List( Descriptor_t Descriptor1 ... DescriptorN ) ;                (o)
   FamilyBC_t FamilyBC ;                                             (o)
   List( GeometryReference_t
         GeometryReference1 ... GeometryReferenceN ) ;               (o)
   RotatingCoordinates_t RotatingCoordinates ;                       (o)

   List(  FamilyName_t FamilyName1 … FamilyNameN ) ;                  (o)

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

***Remarks***:

1- Depending on the user’s particular use of families, there can be overlapping or even inconsistent definitions when constructing a tree of families. It is strongly recommended that the Family hierarchy should be a directed acyclic graph. However, it is the responsibility of the application to manage the hierarchy parse and to check possible loops.

2- The `FamilyName_t` node name *FamilyParent* is reserved and refers to the primary parent family of the current *`Family_t`* node.

3- This proposal is tied closely to CPEX 0034, which defines `Zone_t`, `ZoneSubRegion_t` and `BC_t` nodes modifications regarding families.

## 3. Example

The following example shows the construction of a tree of family information using `FamilyName_t='FamilyParent'`, with the data in the `FamilyName_t` node giving the name of the parent family. Also shown is the use of other (auxiliary) `FamilyName_t` nodes, to associate the current family under a different useful grouping (in this case for postprocessing and for defining a deformable structure).

```
Label=CGNSBase_t;
       Label=Family_t; name='Wings'
       Label=Family_t; name='LeftWing'
              Label=FamilyName_t; name='FamilyParent'; data='Wings'
       Label=Family_t; name='RightWing'
              Label=FamilyName_t; name='FamilyParent'; data='Wings'
       Label=Family_t; name='LeftFlap'
              Label=FamilyName_t; name='FamilyParent'; data='LeftWing'
              Label=FamilyName_t; name='PostProcessing'; data='Output#1'
       Label=Family_t; name='RightFlap'
              Label=FamilyName_t; name='FamilyParent'; data='RightWing'
              Label=FamilyName_t; name='PostProcessing'; data='Output#1'
              Label=FamilyName_t; name='StructureFamily'; data='Deformable'
       Label=Family_t; name=Output#1
              DataArray_t SomeCoefficient='0.2'
       Label=Family_t; name=Deformable'
              DataArray_t Values='ForceX ForceY ForceZ'
```

## 4. Questions and Answers

**How can I identify whether a family is a parent or a child?**
You can identify a family (`Family_t`) as being a child because it has a parent (`FamilyName_t Parent`). You cannot say if a family is the parent of another one without parsing all the families of a `CGNSBase_t` and finding at least one `Family_t` node defining it as its parent.

**How about consistency?**
There is no consistency check. You can create cyclic parent/child relationship, you can define a parent which doesn't exist.

**Is there other consistency issues in CGNS/SIDS that require a management by the application?**
You can find some, for example:

A `FamilyName_t` can refer to a non existing `Family_t`.
You can have surface points not defined as BC nor zone to zone connectivity.
A time dependant structure can refer to non existing solutions, grids, connectivities...
A zone connectivity can refer to a non existing zone.
Two incompatible different boundary conditions can be defined on the same patch.
An element connectivity can define non-allowed parent elements.
A non-commutative rotation can be defined.
...

**How about self-contained declaration of a Zone_t for example?**
The FamilySpecified BCs of a zone are defined outside of the Zone. Same for the ReferenceState in case of global referencestate use.

**I understand the Parent FamilyName_t in the Family_t, what about the other FamilyName_t?**
We define a default hierarchy with the Parent node. The user can define its own hierarchy using his own node name or define a subset of families for a specific purpose. In the example, the subset 'Deformable' would refer to a set of families that would allow their zones to be deformable.

**Why can't we have a single FamilyName_t node in a Zone_t which refers to the deepest child family in the hierarchy, so we could find the ancestors by parsing the Family_t nodes?**
We could but some `FamilyName_t` are not related to the hierarchy but to a subset. For example 'Deformable'.