Main authors: Marc Poinot (Safran Tech), Yoan Collet (Numeca)
Reviewers:
Contacts: marc.poinot@safrangroup.com; yoan.collet@numeca.com

Family Hierachy as a tree

Motivation

The intensive use of families and hierarchy of families (SIDS 6.2.1) results in:

- a large amount of nodes of type `Family_t` as direct children of `CGNSBase_t`,
- an inability to use a `Family_t` name twice
- the required use of a tool to parse a `Family_t` hierarchy

As a side-effect of these points, we see now complex family hierarchies mimicking a tree-like structure by means of their names:

BASE#1		CGNSBase_t
	LPC_ROW1_STATOR_SHROUD	Family_t
	LPC_ROW1_ROTOR_SHROUD	Family_t
	LPC_ROW2_STATOR_SHROUD	Family_t
	LPC_ROW2_ROTOR_SHROUD	Family_t

And a reference to such one of these family uses the straighforward family name with or without the base prefix. For example a `/BASE#1/Zone/FamilyName` we would have `LPC_ROW1_STATOR_SHROUD` as value.

In this flat representation, we do not take benefits of any actual family hierarchy (SIDS 12.6 note 7) and we may have unreadable generated names in the case the name reaches the maximum size of 32 chars. Yet the family names are end-user names and should be readable.

Proposal

The example above would have a better structure using a true tree structure, such as :

BASE#1	CGNSBase_t				
	LPC	Family_t			
		ROW1	Family_t		
			STATOR	Family_t	
				SHROUD	Family_t
			ROTOR	Family_t	
				SHROUD	Family_t
		ROW2	Family_t		
			STATOR	Family_t	
				SHROUD	Family_t
			ROTOR	Family_t	
				SHROUD	Family_t

The CGNS modifications we propose are:

- Allow `Family_t` tree structure by adding `Family_t` as optional child of `Family_t`
- Change `FamilyName_t` or `FamilyAdditionalName_t` value to a path to a `Family_t` node

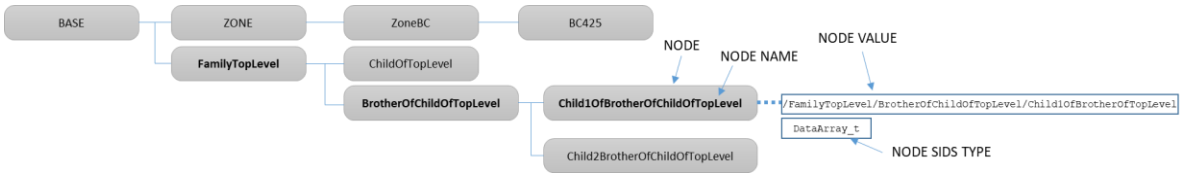
This CPEX insures a 100% compatibility with existing CGNS/HDF5 files, CGNS/Python uses and CGNS/MLL applications.

Main authors: Marc Poinot (Safran Tech), Yoan Collet (Numeca)
Reviewers:
Contacts: marc.poinot@safrangroup.com; yoan.collet@numeca.com

About 32 chars limitation

The CGNS/SIDS requires a node name limitation to 32 chars (CGNS/SIDS 3.1, CGNS/FMM, CGNS/MLL). This historical requirement only applies to the node names, not their values. A node can have a value of type `DataArray_t` or `Descriptor_t` with a string value with much more than 32 chars.

Then a path composed of a suite of node names limited to 32 chars can be stored into a string as a value. This path is also another way to identify the node and in that case the hierarchy has both a structuration and naming role :



The families are the identifier the CGNS user should use, all other identifiers such as zone names, BC names, solution names, are generated and manipulated by tools. These generated node names, in some `cases`, are not meaningful (e.g. *Domain.0001*, *HEXA#AC56*, *BC_TRI_892*) and they should NOT be meaningful. Thus a limitation to 32 for these generated node is acceptable. Moreover, all these generated node names are able to store a `FamilyName_t` or `AdditionalFamilyName_t` node which can finally hold a more explicit name as a value, as a path like in the example above. A path can even refer to another `CGNSBase_t` node than the current you holding the value.

A node of `AdditionalFamilyName_t` type can have a user defined name, limited to 32 chars, such as `TopologicalPart` or `SimulationStep` or `InputField` or whatever you want. For example:

`TopologicalPart` (node name) = `/TopologyBase/CFM56/Fan/Blade/Tip` (node value)

Then if you need to give a detailed name to nodes, use a value in a `FamilyName_t` node together with a Family Hierarchy.

Main authors: Marc Poinot (Safran Tech), Yoan Collet (Numeca)
Reviewers:
Contacts: marc.poinot@safrangroup.com; yoan.collet@numeca.com

CGNS/SIDS modifications

Modifications in the CGNS/SIDS 12.6

The `Family_t` structure contains all information pertinent to a CFD family. This information includes the name attribute or family name, the boundary conditions applicable to these mesh regions, and the referencing to the CAD databases.

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

  FamilyBC_t FamilyBC ;                                               (o)

  List( GeometryReference_t
        GeometryReference1 ... GeometryReferenceN ) ;                 (o)

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

  RotatingCoordinates_t RotatingCoordinates ;                         (o)

  List( FamilyName_t FamilyName1 ... FamilyNameN ) ;                  (o)

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

  int Ordinal ;                                                       (o)
  } ;
```

Notes
…

7. A hierarchy of families is possible through the list of `FamilyName_t` nodes. These nodes contain both a user defined node name and a family name. The node name `FamilyParent` may be used to specify the family name for the parent of the current `Family_t` node.
8. `Ordinal` is defined in the SIDS as a user-defined integer with no restrictions on the values that it can contain. It may be used here to attribute a number to the family.
9. A `Family_t` tree structure can be specified using the list of `Family_t` children nodes. Into each of these children nodes the note #7 can be used to have a back tracking of the node parent.

Modifications in the CGNS/SIDS 6.2.1

The green part is a correction of the existing SIDS without relationship with this CPEX.

6.2.1 Base Level Families

The `Family_t` data structure is used to record geometry reference data. It may also include boundary conditions linked to geometry patches. For the purpose of defining material properties, families may also be defined for groups of elements. The family-mesh association is defined under the `Zone_t`, the `ZoneSubRegion_t` and `BC_t` data structures by specifying the family name corresponding to a zone or a boundary patch. The family name can refer to a `Family_t` defined in a `CGNSBase_t` other than the referring `Zone_t`, the `ZoneSubRegion_t` or `BC_t`. This `Family_t` node can be a direct child of the `CGNSBase_t` or a child of another `Family_t`. The actual family name has the pattern `<CGNSBase>/<FamilyName1>/<FamilyName2>/…/<FamilyNameN>`. ~~In this case, the actual name of the `Family_t` has to be prefixed by the `CGNSBase_t` name. The pattern is then `basename/familyname`, only one single / character is allowed, and neither of `basename` nor `familyname` should be empty.~~ The family-mesh association is defined under the `Zone_t`, ZoneSubRegion_t and `BC_t` data structures by specifying the family name corresponding to a zone, zone sub-region or a boundary patch in a FamilyName_t node. If the value of the FamilyName node does not have a / character in it, then the name refers to a family being a direct child of the ancestor CGNS Base of this FamilyName node. Otherwise, if this value has at least one / in it, the pattern `<CGNSBase>/<FamilyName1>/<FamilyName2>/…/<FamilyNameN>` is mandatory.

The `UserDefinedData_t` data structure allows arbitrary user-defined data to be stored in `Descriptor_t` and `DataArray_t` children without the restrictions or implicit meanings imposed on these node types at other node locations.

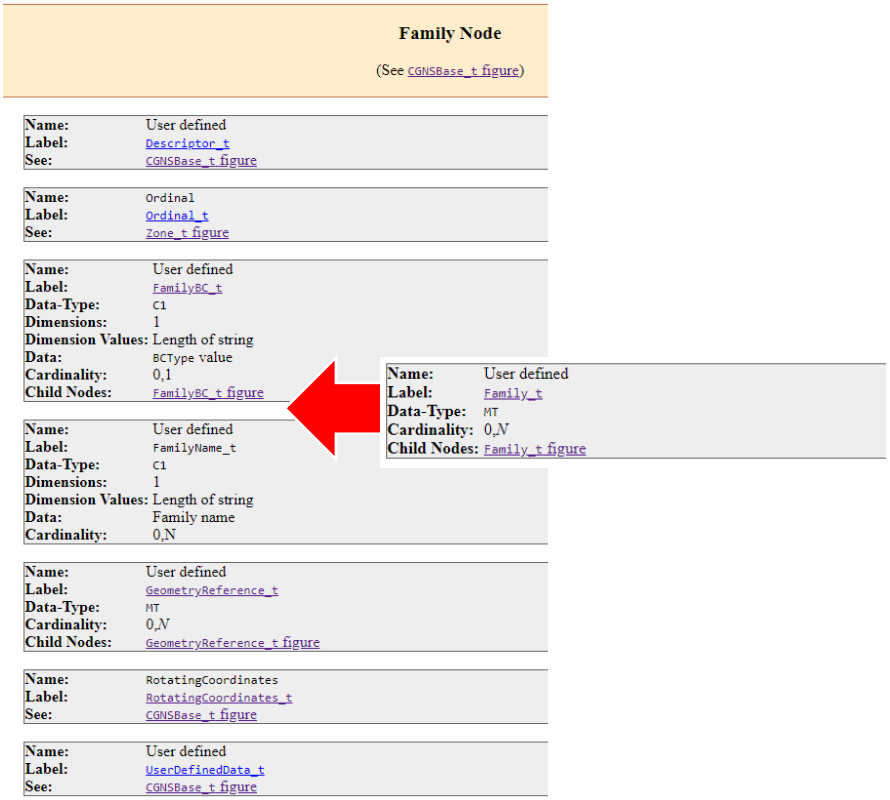
There is no impact to already existing `Family_t` nodes, the CPEX adds a new optional child node, existing applications would ignore it.

Main authors: Marc Poinot (Safran Tech), Yoan Collet (Numeca)
Reviewers:
Contacts: marc.poinot@safrangroup.com; yoan.collet@numeca.com

There is no impact to already existing `FamilyName_t` of `AdditionalFamilyName_t` nodes : if the name has no `/` the behavior is the same as before the CPEX, if the name has a `/` then we have the correct `<CGNSBase>/<FamilyName>` pattern as before the CPEX.

CGNS/FMM modifications

The `Family_t` 'CGNS File Mapping Figures' of the CGNS/FMM document has to be updated, an optional node of type `Family_t` is added to existing description:



CGNS/MLL modifications

The addition of a `Family_t` node inside a `Family_t` node would change this positional SIDS type (a node which position is fixed) into a non-positional SIDS type. This requires a new set of functions to be used following a `cg_goto` call. Of course, all `Family_t` existing functions should operate the same way they do today, though we add some feature that would be ignored by today's applications. They refer to a `Family_t` node using the `int Fam` index, which is kept unchanged as the first level index of the family as returned by `cg_nfamilies`.

Existing functions

Family Definition	Remark (note)
`cg_family_write` - Create a Family_t node	Family path accepted (1)
`cg_nfamilies` - Get number of families	unchanged
`cg_family_read` - Read family info	unchanged
`cg_family_name_write` - Write multiple family names under Family_t	unchanged
`cg_nfamily_names` - Get number of family names under Family_t	unchanged

cg_family_name_read - Read multiple family names under Family_t	unchanged
Geometry Reference	
cg_geo_write - Create a GeometryReference_t node	unchanged
cg_geo_read - Read geometry reference info	unchanged
cg_part_write - Write geometry entity name	unchanged
cg_part_read - Get geometry entity name	unchanged
Family Boundary Condition	
cg_fambc_write - Write boundary condition type for a family	unchanged
cg_fambc_read - Read boundary condition type for a family	unchanged
Family Name	
cg_famname_write - Write family name	Family path accepted (2)
cg_famname_read - Read family name	Family path accepted (2)
cg_multifam_write - Write multiple family names	Family path accepted (2)
cg_nmultifam - Get number of family names	Family path accepted (2)
cg_multifam_read - Read multiple family names	Family path accepted (2)

Note (1):

A modification is proposed to:

```
ier = cg_family_write(int fn, int B, char *FamilyName, int *Fam);                    - w m
```

We have to accept a path as `FamilyName` and create the correct tree of `Family_t` nodes. Today's application would ignore this feature as a Family path is not allowed yet.

Note (2):

`FamilyName` functions, such as writing or reading `FamilyName` or `AdditionalFamilyName` nodes are kept unchanged for existing applications. We extend the `FamilyName` value to a path.

New functions

A new set of functions is proposed, it has to be used after a `cg_goto` call (or similar). We use a function naming close to the existing set for the positional nodes, but the identification pattern `int fn, int B, int Fam` is useless after a `cg_goto`, this pattern is removed.

Functions	Modes
`ier = cg_node_family_write(char *FamilyName, int *Fam);`	`- w m`
`ier = cg_node_nfamilies(int *nfamilies);`	`r - m`
`ier = cg_node_family_read(char *FamilyName, int *nFamBC, int *nGeo);`	`r - m`
`ier = cg_node_family_name_write(char *NodeName, char *FamilyName);`	`- w m`
`ier = cg_node_nfamily_names(int *nNames);`	`r - m`
`ier = cg_node_family_name_read(int N, char *NodeName, char *FamilyName);`	`r - m`
`call cg_node_family_write_f(FamilyName, Fam, ier)`	`- w m`
`call cg_node_nfamilies_f(nfamilies, ier)`	`r - m`
`call cg_node_family_read_f(FamilyName, nFamBC, nGeo, ier)`	`r - m`

There is no new function for the `FamilyName` read/write features which already are positional.

Main authors: Marc Poinot (Safran Tech), Yoan Collet (Numeca)
Reviewers:
Contacts: marc.poinot@safrangroup.com; yoan.collet@numeca.com

```
ier = cg_geo_write(char *GeoName, char *FileName, char *CADSystem, int *G);  - w m
ier = cg_geo_read(int G, char *GeoName,                                      r - m
                  char **FileName, char *CADSystem, int *nparts);
ier = cg_part_write(int G, char *PartName, int *P);                          - w m
ier = cg_part_read(int G, int P, char *PartName);                            r - m
call cg_geo_write_f(GeoName, FileName, CADSystem, G, ier)                    - w m
call cg_geo_read_f(G, GeoName, FileName, CADSystem, nparts, ier)             r - m
call cg_part_write_f(G, PartName, P, ier)                                    - w m
call cg_part_read_f(G, P, PartName, ier)                                     r - m
```

Note:

There is no `cg_ngeometry (cg_npart)` which would return the number `of GeometryReference_t (GeometryEntity_t)` in a `Family_t (GeometryReference_t)`. The actual numbers (indexes) of *geometries* and *parts* are returned values of `cg_family_read and cg_geo_read.` This geometry/part index is unchanged, only the `Family_t` hierarchy is a new feature.

```
ier = cg_fambc_write(char *FamBCName, BCType_t BCType, int *BC);     - w m
ier = cg_fambc_read(int BC, char *FamBCName, BCType_t *BCType);      r - m
call cg_fambc_write_f(FamBCName, BCType, BC, ier)                    - w m
call cg_fambc_read_f(BC, FamBCName, BCType, ier)                     r - m
```

Same remark as geometry/part index note above, the `FamilyBC_t` is unchanged.

Main authors: Marc Poinot (Safran Tech), Yoan Collet (Numeca)
Reviewers:
Contacts: marc.poinot@safrangroup.com; yoan.collet@numeca.com

Example 1 – writing a Family hierarchy in `BASE#1`:

1- Create a new family `LCP` as `CGNSBase_t` child (assume file index is 1 and base index is 1)
2- Set the current CGNS/MLL cursor on this new family (goto with absolute path)
3- Add `ROW1` child family of current cursor (child of `LPC`)
4- Set the cursor on this new family `ROW1`
5- Add `STATOR` as child of `ROW1`
6- Set the cursor on this new family `STATOR` (goto with relative path)
7- Add a `GeometryReference_t` node as child of `/BASE#1/LPC/ROW1/STATOR`

```
int fam1, fam2, fam3; /* new families indexes */

int geo1; /* new geometry index */

cg_family_write(1, 1, "LPC",  &fam1);

cg_goto(1, 1, "Family_t", fam1, NULL);

cg_node_family_write("ROW1", &fam2);

cg_goto(1, 1, "Family_t", fam1, "Family_t", fam2, NULL);

cg_node_family_write("STATOR", &fam3);

cg_gorel(1, "Family_t", fam3, NULL);

cg_node_geo_write("CAD", "user-file-somewhere.stp", "STEP", &geo1);
```

Notes:

- A goto + `cg_node_family_write` can replace the step 1
- CGNS/MLL has NO check on CAD format (STEP in this example)

Example 2 – reading a Family hierarchy from `BASE#1` (the easy way), we assume the char* variables already had a memory allocation:

```
cg_gopath(1, "/BASE#1/LPC/ROW1/STATOR");

cg_node_geo_read(1, &name, &filename, &CAD);
```

Known issues

A today's application would be unable to read a Family hierarchy and thus could not find any reference to a `FamilyBC_t` or any other critical data for the CFD solver use.