Author(s): Marc Poinot, ONERA/DSNA
Contact: marc.poinot@onera.fr

Rationale for an Inter Base Reference extension to CGNS
CPEX 039

The rationale for requiring an extension to CGNS/SIDS for inter base references explain base merge concerns in the first part, a modification proposal in the second part and then an analysis on the impact of CGNS/MLL in the last part.
The extension is not a large modification in terms of CGNS/SIDS rewriting or in CGNS/MLL changes.

1. MERGE CONCERNS

The generation of meshes can be an automated process where some parts of the final configuration are build or reuse from formerly existing CGNS trees. In that case, names (or nodes) collision do require a complete rewrite of the CGNS tree. We show there three examples of such issues: the zone, the family, the reference state.

1.1. Merging Zones for separate Bases

Zone merging appears when you perform meshing from different processes. The each tool or each tool output may have hard coded patterns. For example, some mesh tools may always produce the same zone name pattern: `domain.0001, domain.0002, ...`
So that any attempt to merge two meshes without change is impossible. Once you have changed these names in any way (renumbering, adding prefix...) you have to propagate this change to the nodes referencing these names. There are at least three ways to reference such a node name, in this proposal only case 1 is taken into account:

1. as a CGNS/SIDS value of another node
2. as a user defined value of another node
3. as key for another node

Case 1. is defined by the CGNS/SIDS when you refer to a zone in a node value. Two CGNS/SIDS nodes have a zone name as value: the `ZoneDonor` identifier in the connectivity node value[1], the `ZonePointers` in the `BaseIterativeData` node[2].
Case 2. is when you store a name of a node into a user defined data. For example you may embbed a python script in your CGNS tree, this script may contain hard-coded names refering to these zones. We think this is not a portable a good practice, this is not addressed by this proposal.
Case 3. is sometimes used by generators, but it should be considered as user defined. For example you have two zones '`A`' and '`B`' connecting one to the other and you name the connectivity between them using the zone names as key: '`A-B-01`', '`1to1 A B`'... The renaming concern is about tracability but not about the CGNS/SIDS compliance. The name is user defined and thus is not 'used' by CGNS/SIDS. However, some of your tools may parse the name for some internal needs. The proposal would not take this case into account, we consider that using a node name as an information storage mean is not CGNS/SIDS.

1.2. Family nodes renaming

The families have a single definition at the base level of a CGNS tree and a lot of references everywhere else in this tree (see *Annex* 4.2.). The collision with a family name can have a large impact in the CGNS tree itself, but the mst important impact is outside the CGNS tree. The families are used to identify a part of the configuration at the user level. In the CFD workflow, the simulation tools may use a family name as a reference for its own identification purpose: a CAD surface for a BC setting, a region for a 3D field code-coupling exchange, a set of cells for a post-processing display, and so on.
Then renaming a family in a CGNS tree is mandatory in the case of collisions, but has to be done at the workflow level and changing the name in a local pre/post preparation is a bit dangerous.

[1]http://www.grc.nasa.gov/WWW/cgns/CGNS_docs_current/sids/cnct.html#GridConnectivity
[2]http://www.grc.nasa.gov/WWW/cgns/CGNS_docs_current/sids/timedep.html#IterativeData

1.3. Other base level nodes renaming

The `ReferenceState`, `GoverningEquations` and `ConvergenceHistory` node are examples of base level nodes that cannot be renamed. The node name is set by CGNS/SIDS as a reserved name. Then a merge on multiple base would lead to either 'last found is set' or 'first found is set' depending on your tree parsing algorithm and implementation (see Annex 4.3. about parallel merging issues).

2. EXTENDING CGNS/SIDS FOR INTER BASE REFERENCES

We propose a change to CGNS/SIDS and to the CGNS file mapping to allow the reference to another base in a CGNS tree. The goal is to avoid to merge two bases for mesh assembly purpose. The merge collisions we illustrate in the first section are the actual problems we have to solve with his extension.

Extension proposal 1:

A multi-base CGNS tree is a CGNS/SIDS compliant tree with more than one base. These bases can have references to each other only:

1. from any node to any other node using links
2. to another base's zone name (that is a C1 value in a node data)
3. to another base's family name (that is a C1 value in a node data)

2.1. Node names and prefix

The CGNS/SIDS doesn't have any constraint about node names. It defines reserved or recommanded names, but there is no implementation requirement in particular about the authorized set of characters. However section 3.1, in the `Identifier` function paragraph[3], recalls the use of the '/' as path separator, but there is real requirement about the name. The CGNS filemapping document[4] defines some general ideas about the node names, but the syntaxic and implementation requirements are in the CGNS/CGIO document[5]:

We read there that the `'/'` character is not allowed in a node name. The reason is that most tree-oriented data systems (such as HDF5) are using the `'/'` as the historical separator of tokens in a tree path.

A base prefix can be added using the `'/'` as separator. We know there cannot exist any CGNS tree with a node name containing a `'/'` today.

Extension proposal 2:

A parser that would find a node name or a node value referencing a name containing the `'/'` character is a reference to another base's name[6]. An inter-base zone name would be `'basename/zonename'`, a family name `'basename/familyname'` and any other node referenced by another base would use such a prefix pattern as well: `'basename/`**`GoverningEquations`**`'`, `'basename/`**`ReferenceState`**`'` and so on.

2.2. CGNS/SIDS constraint to inter base reference

The base dimensions is the only mandatory node at base level, all other nodes have a base scope and can be defferent from one base to another[7]. The list of nodes that could lead to problems depending on their values w.r.t. other bases is in Annex 4.4..

Extension proposal 3:

We state that :

1. bases having different `CellDimension` and `PhysicalDimension` can reference each other anyway;
2. a reference to a base level node from one base to another is performed by links[8] (see next section);

[3] http://www.grc.nasa.gov/WWW/cgns/CGNS_docs_current/sids/conv.html#notation

[4] http://www.grc.nasa.gov/WWW/cgns/CGNS_docs_current/filemap/general.html#nodename

[5] http://www.grc.nasa.gov/WWW/cgns/CGNS_docs_current/cgio/library.html#attributes

[6] We can even suggest that ./family is a reference to a local family name. The `'.'` and `'..'` names are also reserved by the CGNS/HDF5 implementation and thus could not be used in an existing CGNS tree.

[7] http://www.grc.nasa.gov/WWW/cgns/CGNS_docs_current/sids/cgnsbase.html#CGNSBase

[8] Quite practical for reuse but quite dangerous for some nodes such as `DimensionalUnits`, if the defaut is used ond only one base defines its own specific parameters, it crashes all other bases...

The CGNS tree parsers have to be multi-base-aware or not. If they are, they can manage per-base contextes to store `ReferenceState`, `GoverningEquations`...

2.3. Inter-base links

The links are an important reuse feature of CGNS. There is no extension to add for using links as another inter-base mechanism. A link to another node into another file is always possible. And as a matter of fact, this destination node actually is into another base. It is up to the application to check if the destination base, zone or whichever other node is relevant in the source base context.
For example if you want to use the `ReferenceState` of another base in a zone, you have to use links because you cannot change the name of the `ReferenceState` node. The link would have the other base's `ReferenceState` as destination node.

2.4. CGNS/SIDS modification

The modifications of the document are bound to definition of/using zone names, family names and base level nodes. There is no CGNS/SIDS requirement about links, as the node name is kept unchanged one ***only*** has to check semantic correctness of the link.
In the CGNS/SIDS document, there is not explicit indication of the required scope of the `FamilyName` or `AdditionalFamilyName` value. No modification for families.
The function `Identifier`() takes arguments with the zone name in the connectivity definition. In the `BaseIterativeData` definition the function is not used, there is an explicit reference to a zone name so called `ZonePointers`. None of these two do require that the zone name has to be in the current base. However the CGNS/SIDS 8.2 note 3 states that the zone should be in the current base. This has to be changed. Now there is a requirement about the size of the value. We have to allow a size of 32+1+32 in order to store a base-scope identifier.
Extension proposal 4

The `ZoneDonorName` can be a base-prefixed zone name with the pattern `basename/zonedonorname`

Extension proposal 5

The size of the following node **values** are increased from 32 characters to 65 characters, this is required to store two 32 characters names and a `'/'` separator (CGNS/SIDS section indicated):

1. `BaseIterativeData_t/ZonePointers (11.1.1)`
2. `BaseIterativeData_t/FamilyPointers (11.1.1)`

3. MODIFICATIONS TO CGNS/MLL

The CGNS/MLL is the most used implementation f the CGNS/HDF5 mapping and the unique implementation of the CGNS/ADF mapping. We describe a non-exhaustive impact of the inter base reference to the current v3.2 source. ***This is an incomplete first pass on the sources.***

3.1. Interface impact

Unfortunately the CGNS/MLLtakes the base id as mandatory arguments. But we may not need any base to base function. If we stay with the current function set, only the values passed as arguments may refer to another base. For example `family_read` returns a string from a current base index and familyname index. The returned value may be another family reference, but when you are performing a look up on a family name the loop is an application loop, not a CNGS/MLL function. Then maybe we can stay with a zero add to the interface.

3.2. Source impact

We do not take into account impact on version < 3.2 (old structs as 1.1 or 1.2 not modified).
Node lookup is performed on a global data structure, the `cgns_file` struct in `cgns_header.h` file. It can contain more than one base but some name lookup, such as zone name for connectivity or family name, is always done with the current base as the search scope. This needs to be detailled, because out of `cgnscheck` it doesn't seem to have a lookup for actual zone reference in `cgnslib.c` or files.
The zonedonor name size is 32 in the `cgns_internal.c` this is only a constant change but there is a lot of pointer arithmetics that may lead to core dump. So heavy test is required to insure a good coverage.
The familyname is just a label and there is lookup elsewhere than in the application itself (see `tests/test_multifam.c`). Most functions are goto-oriented and the goto take the base id as first arg, so that it would be possible to set a goto pointer out of the current base. Then the modification impact is low.
The `BaseIterativeData_t` modification includes the extension of the value size to 65 (`cgnlib.c`, `cgns_internals.c`) and to the zone lookup. There is no check about the size value but only on the number and the types of the arrays, there is no zone lookup. The `cgnscheck` tools makes more verifications on the data array, the size and the actual target node is checked in the current base.

3.3. Modification cost

I would say a rough charge of 2 weeks of development for the libs and the tools, 1 week of local test, 1 week on multi-platform test (cluster, workstation, windows) and one more week for doc updates and probably some how-to docs on parsing and finding inter base names/ nodes. That makes a large month for a ready-to-use release if the development is restricted to zonedonorname, familynames and baseiterativedata.

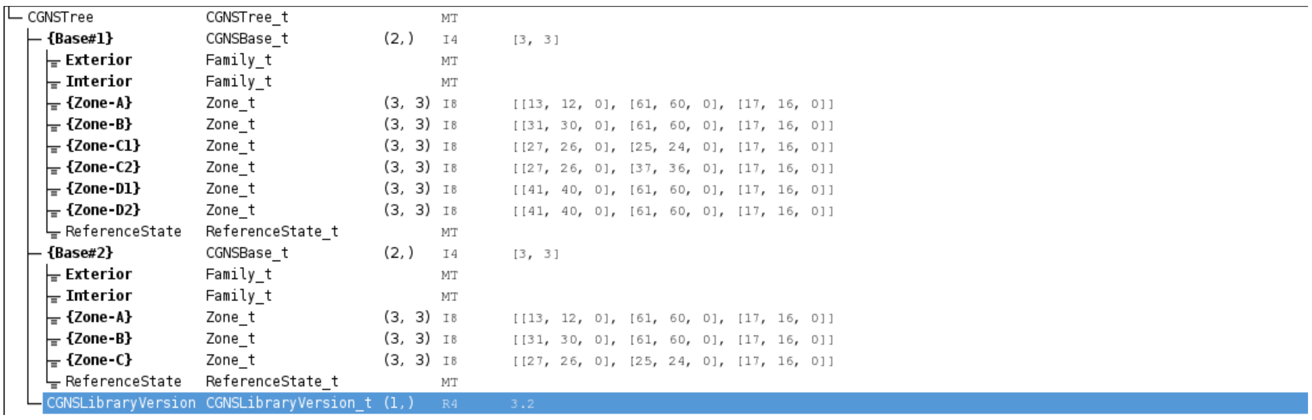
3.4. Compatibility concerns

A CGNS tree without inter-base names should have no change on the file contents. Same for the use of the CGNS/MLL interface.
A CGNS tree using inter-base references would automatically force the CGNS version to an update (upgrade to something like v3.3 or v4.0 depending on impact classification from CGNS steering committee). This would forbid lower version to read the file.
An application using the inter-base version of the CGNS/MLL but without any reference to inter-base in the CGNS tree would keep its version to v3.2 (binary compatibility insured).

4. ANNEXES

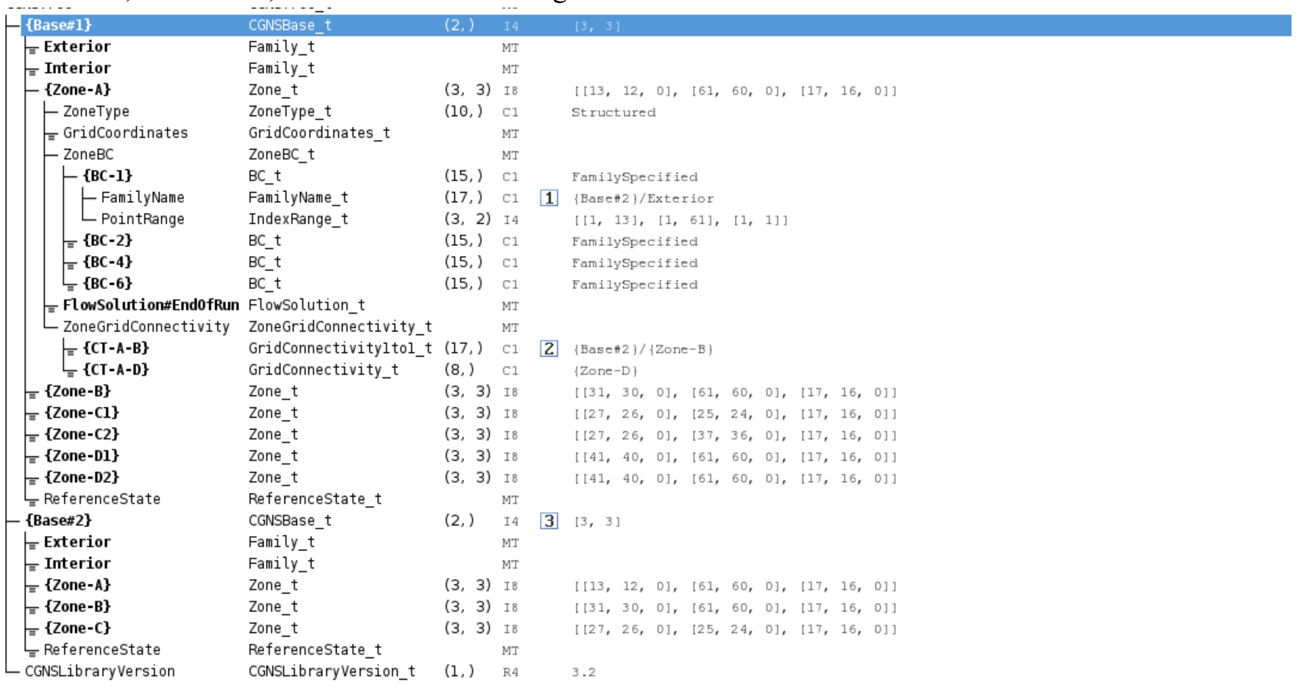
4.1. Examples

The screenshot below show a tree with two bases. Today no relationship can be set between the two bases, it is just a storage/definition but it has no actual use from the CGNS/SIDS semantic point of view.



```
└ CGNSTree          CGNSTree_t                    MT
  ├ {Base#1}        CGNSBase_t            (2,)    I4   [3, 3]
  │ ├ Exterior      Family_t                      MT
  │ ├ Interior      Family_t                      MT
  │ ├ {Zone-A}      Zone_t                (3, 3)  I8   [[13, 12, 0], [61, 60, 0], [17, 16, 0]]
  │ ├ {Zone-B}      Zone_t                (3, 3)  I8   [[31, 30, 0], [61, 60, 0], [17, 16, 0]]
  │ ├ {Zone-C1}     Zone_t                (3, 3)  I8   [[27, 26, 0], [25, 24, 0], [17, 16, 0]]
  │ ├ {Zone-C2}     Zone_t                (3, 3)  I8   [[27, 26, 0], [37, 36, 0], [17, 16, 0]]
  │ ├ {Zone-D1}     Zone_t                (3, 3)  I8   [[41, 40, 0], [61, 60, 0], [17, 16, 0]]
  │ ├ {Zone-D2}     Zone_t                (3, 3)  I8   [[41, 40, 0], [61, 60, 0], [17, 16, 0]]
  │ └ ReferenceState ReferenceState_t             MT
  ├ {Base#2}        CGNSBase_t            (2,)    I4   [3, 3]
  │ ├ Exterior      Family_t                      MT
  │ ├ Interior      Family_t                      MT
  │ ├ {Zone-A}      Zone_t                (3, 3)  I8   [[13, 12, 0], [61, 60, 0], [17, 16, 0]]
  │ ├ {Zone-B}      Zone_t                (3, 3)  I8   [[31, 30, 0], [61, 60, 0], [17, 16, 0]]
  │ ├ {Zone-C}      Zone_t                (3, 3)  I8   [[27, 26, 0], [25, 24, 0], [17, 16, 0]]
  │ └ ReferenceState ReferenceState_t             MT
  └ CGNSLibraryVersion CGNSLibraryVersion_t (1,)  R4   3.2
```

Now we propose that some reference in a base can include nodes from another base. In the example below, a BC family name (mark 1) refers to the second base (mark 3) and a connectivity (mark 2) refers to a second base zone as donor. From an implementation point of view, this is only a name convention in the node values. From a CGNS/SIDS semantic point of view, we have to state what is allowed, what is not, and what is the meaning of these inter-base references.



```
├ {Base#1}                  CGNSBase_t               (2,)    I4       [3, 3]
│ ├ Exterior                Family_t                         MT
│ ├ Interior                Family_t                         MT
│ ├ {Zone-A}                Zone_t                   (3, 3)  I8       [[13, 12, 0], [61, 60, 0], [17, 16, 0]]
│ │ ├ ZoneType              ZoneType_t               (10,)   C1       Structured
│ │ ├ GridCoordinates       GridCoordinates_t                MT
│ │ ├ ZoneBC                ZoneBC_t                         MT
│ │ │ ├ {BC-1}              BC_t                     (15,)   C1       FamilySpecified
│ │ │ │ ├ FamilyName        FamilyName_t             (17,)   C1   1   {Base#2}/Exterior
│ │ │ │ └ PointRange        IndexRange_t             (3, 2)  I4       [[1, 13], [1, 61], [1, 1]]
│ │ │ ├ {BC-2}              BC_t                     (15,)   C1       FamilySpecified
│ │ │ ├ {BC-4}              BC_t                     (15,)   C1       FamilySpecified
│ │ │ └ {BC-6}              BC_t                     (15,)   C1       FamilySpecified
│ │ ├ FlowSolution#EndOfRun FlowSolution_t                   MT
│ │ └ ZoneGridConnectivity  ZoneGridConnectivity_t           MT
│ │   ├ {CT-A-B}            GridConnectivity1to1_t   (17,)   C1   2   {Base#2}/{Zone-B}
│ │   └ {CT-A-D}            GridConnectivity_t       (8,)    C1       {Zone-D}
│ ├ {Zone-B}                Zone_t                   (3, 3)  I8       [[31, 30, 0], [61, 60, 0], [17, 16, 0]]
│ ├ {Zone-C1}               Zone_t                   (3, 3)  I8       [[27, 26, 0], [25, 24, 0], [17, 16, 0]]
│ ├ {Zone-C2}               Zone_t                   (3, 3)  I8       [[27, 26, 0], [37, 36, 0], [17, 16, 0]]
│ ├ {Zone-D1}               Zone_t                   (3, 3)  I8       [[41, 40, 0], [61, 60, 0], [17, 16, 0]]
│ ├ {Zone-D2}               Zone_t                   (3, 3)  I8       [[41, 40, 0], [61, 60, 0], [17, 16, 0]]
│ └ ReferenceState          ReferenceState_t                 MT
├ {Base#2}                  CGNSBase_t               (2,)    I4   3   [3, 3]
│ ├ Exterior                Family_t                         MT
│ ├ Interior                Family_t                         MT
│ ├ {Zone-A}                Zone_t                   (3, 3)  I8       [[13, 12, 0], [61, 60, 0], [17, 16, 0]]
│ ├ {Zone-B}                Zone_t                   (3, 3)  I8       [[31, 30, 0], [61, 60, 0], [17, 16, 0]]
│ ├ {Zone-C}                Zone_t                   (3, 3)  I8       [[27, 26, 0], [25, 24, 0], [17, 16, 0]]
│ └ ReferenceState          ReferenceState_t                 MT
└ CGNSLibraryVersion        CGNSLibraryVersion_t     (1,)    R4       3.2
```

4.2. FamilyName_t paths

All type paths for a `FamilyName_t` node, generated using pyCGNS (`CGNS.PAT.cgnsutils.getAllParentTypePaths`). A lot of paths due to the `UserDefinedData` node.

```
 /CGNSBase_t/Zone_t/ZoneBC_t/BC_t/FamilyName_t
 /CGNSBase_t/Zone_t/ArbitraryGridMotion_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneBC_t/BC_t/BCProperty_t/Area_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneGridConnectivity_t/GridConnectivity1to1_t/GridConnectivityProperty_t/AverageInterface_t/UserDefinedData_t/Fami
lyName_t
 /CGNSBase_t/Zone_t/ZoneGridConnectivity_t/GridConnectivity_t/GridConnectivityProperty_t/AverageInterface_t/UserDefinedData_t/FamilyNa
me_t
 /CGNSBase_t/Axisymmetry_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneBC_t/BC_t/BCDataSet_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Family_t/FamilyBC_t/BCDataSet_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneBC_t/BC_t/BCDataSet_t/BCData_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Family_t/FamilyBC_t/BCDataSet_t/BCData_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneBC_t/BC_t/BCProperty_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneBC_t/BC_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/BaseIterativeData_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/FlowEquationSet_t/ChemicalKineticsModel_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/FlowEquationSet_t/ChemicalKineticsModel_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ConvergenceHistory_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/DiscreteData_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/FlowEquationSet_t/EMConductivityModel_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/FlowEquationSet_t/EMConductivityModel_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/FlowEquationSet_t/EMElectricFieldModel_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/FlowEquationSet_t/EMElectricFieldModel_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/FlowEquationSet_t/EMMagneticFieldModel_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/FlowEquationSet_t/EMMagneticFieldModel_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/Elements_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Family_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/FlowEquationSet_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/FlowEquationSet_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/FlowSolution_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/FlowEquationSet_t/GasModel_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/FlowEquationSet_t/GasModel_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Family_t/GeometryReference_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/FlowEquationSet_t/GoverningEquations_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/FlowEquationSet_t/GoverningEquations_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Gravity_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneGridConnectivity_t/GridConnectivity1to1_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneGridConnectivity_t/GridConnectivity1to1_t/GridConnectivityProperty_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneGridConnectivity_t/GridConnectivity_t/GridConnectivityProperty_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneGridConnectivity_t/GridConnectivity_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/GridCoordinates_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/IntegralData_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/IntegralData_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneGridConnectivity_t/OversetHoles_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneGridConnectivity_t/GridConnectivity1to1_t/GridConnectivityProperty_t/Periodic_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneGridConnectivity_t/GridConnectivity_t/GridConnectivityProperty_t/Periodic_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneBC_t/BC_t/BCDataSet_t/ReferenceState_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Family_t/FamilyBC_t/BCDataSet_t/ReferenceState_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneBC_t/BC_t/ReferenceState_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/ReferenceState_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneBC_t/ReferenceState_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ReferenceState_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/RigidGridMotion_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/RotatingCoordinates_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Family_t/RotatingCoordinates_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/RotatingCoordinates_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/FlowEquationSet_t/ThermalConductivityModel_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/FlowEquationSet_t/ThermalConductivityModel_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/FlowEquationSet_t/ThermalRelaxationModel_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/FlowEquationSet_t/ThermalRelaxationModel_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/FlowEquationSet_t/TurbulenceClosure_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/FlowEquationSet_t/TurbulenceClosure_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/FlowEquationSet_t/TurbulenceModel_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/FlowEquationSet_t/TurbulenceModel_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/FlowEquationSet_t/ViscosityModel_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/FlowEquationSet_t/ViscosityModel_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneBC_t/BC_t/BCProperty_t/WallFunction_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneBC_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneGridConnectivity_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneIterativeData_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneSubRegion_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/UserDefinedData_t/FamilyName_t
 /CGNSBase_t/Zone_t/ZoneSubRegion_t/FamilyName_t
 /CGNSBase_t/Zone_t/FamilyName_t
```

4.3. Parallel Merging issues

The share of a ressource in a parallel system leads to the so called 'race condition'. That may lead to several behaviour depending on the implementation of the code and of the shared ressource. If we have a look at and HDF5 file shared by several CGNS/MLL processes, the order of the children of a node is an example of problems due to the implementation. In the case you use the alphabetical order to get the index of the child in the children list, the insertion a new child by another process would break the match between the index and the actual child. If you use the creation date as the ordering index, as the creation of an HDF5 node insures a mutual exclusion and a serialization so that only one child at a time can be created for all processes, your index is always correct.

4.4. Singleton base level nodes

The attributes that could appear only once or that should have the same value are listed below[9]. As a first statement, we say the list below are singleton nodes in a multi-base simulation.
Well more question that answers in there... remarks in italic are for discussion, but obviously for a first pass we should restrict the possibilities if we want this CPEX to be implemented one day.

`CellDimension, PhysicalDimension`
First answer is to say no: you cannot change it from base to base in the same configuration. However, how to have the reference to another base's family for code-coupling purpose?

`ReferenceState`
Do we allow multiple ReferenceStates, for example if the SpecificHeatRatio changes from one part of the configuration to another one? Then a ReferenceState is always relevant for its base an only its base?

`AxiSymmetry`
Oups... this is a 2D case but can we imagine to have relationship with 3D base for code-couping purpose? Another example with 3D/1D would be a transition line on a wing surface, how to reference it?

`SimulationType`
How to reuse a mesh defined in a steady CGNS tree into an unsteady base?

`DataClass, DimensionalUnits`
As one base may be reuse from a previous computation, dimensional data may change from one base to another.

`FlowEquationSet`
That would be nice to have different set of equations...

`GlobalConvergenceHistory`
No way to reuse this, the node is dedicated to a solution archival...

[9] This is an important list which have an strong impact on the reuse and versatility capability of the standard. Can we allow two different equation systems in two separate bases and still allow them to reference to each other, for example for code-coupling ?