CGNS Proposal Extension #0047:

Quadrature rules definition and data storage

Main authors: Mickael Philit (mickael.philit@safrangroup.com), Fabien Huvelin (fabien.huvelin@safrangroup.com)

Table of contents

[1. Motivation 1](#_Toc55315024)

[2. Proposal to add a concept of Integration/Quadrature 1](#_Toc55315025)

[3. Conflict and compatibility concern 3](#_Toc55315026)

[4. Conclusion 3](#_Toc55315027)

[5. Document modification list 3](#_Toc55315028)

[A. Appendix - Extension to the CGNS/SIDS 4](#_Toc55315029)

[B. Appendix - Extension to the CGNS/Filemap 17](#_Toc55315030)

[C. Appendix - Extension to the CGNS/MLL 22](#_Toc55315031)

# Motivation

Finite element methods and high order methods (like ones used by the Center for Efficient Exascale Discretizations, CEED, <https://www.ceed.exascalproject.org>) require the concept of integration and even use quadrature vectors. In order to visualize, to allow accurate initializing and debugging those methods, CGNS SIDS need to have the capability to store data at integration points like VMAP (<https://www.vmap.eu.com>) or MED (<https://www.salome-platform.org/user-section/about/med>). This proposal is here to fill this gap.

# Proposal to add a concept of Integration/Quadrature

## Quadrature rule definition

To define a numerical integration rules on all the elements or a collection of elements, on each element the integration formula can be written as:

and are the variables to describe and to store. is an array of scalars of size number of Integration points. There is two ways in order to define the integration points. The first one use the notion of parametric coordinates (SIDS: <http://cgns.github.io/CGNS_docs_current/sids/cnct.html>)., where integrations points coordinates are defined in a reference frame. In the figure 1 (left), each integration point is defined thanks to two parametric coordinates. The second one use the notion of barycentric coordinates. Integration points are defined from the element vertices thanks to a tuple. In the figure 1 (right), for each integration point, three constant are neeeded in order to define their location.

|  |  |
| --- | --- |
| *Parametric description (frame r,s)* | *Barycentric description (tuple c1, c2, c3)* |
| *Figure 1 – Integration points location* | |

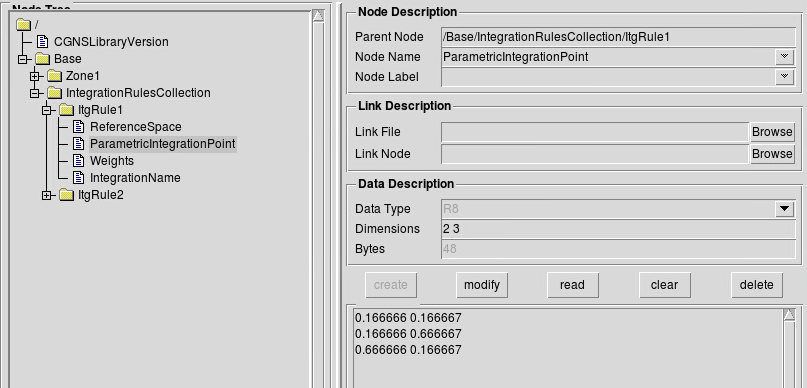
## Quadrature rule description

A type **IntegrationRule\_t** is introduced in order to store the integration rule description and should have some basic properties:

* **NumberOfIntegrationPoint** and **NumberOfParametricDimension**: The total number of integration points and the number of parametric dimension (from 1 to 3 for parametric coordinates and number of vertices per element for barycentric coordinates) are needed to size the array.
* **ElementType**: the element type for which this integration rule is defined and valid. This ElementType exclude the CGNS “MIXED” type.
* **ReferenceSpace**: The reference space definition, used to locate the integration points, is optional. It can be either Parametric or Barycentric. If the ElementType is polygonal or polyhedral it can only be set to Barycentric.
* Either one of the two following arrays is needed depending on the *ReferenceSpace* value:
* **ParametricIntegrationPoint**<NumberOfIntegrationPoint, NumberOfParametricDimension >: Real array storing the parametric coordinates. The Integration Points are stored following the principle of growing *r* then growing *s* and ending by growing *t*.
* **BarycentricIntegrationPoint**<NumberOfIntegrationPoint, NumberOfParametricDimension >: Real array storing the barycentric coordinates
* ***Weights*** : a real array of size NumberOfIntegrationPoint storing weigths
* ***IntegrationName*** : For parametric definition, the name of the quadrature can be provided, as optional parameter, and can be chosen among CGNS standard names (GaussLegendre, GaussLobatto,, …) or be application specific. The full rule is defined with an array of size NumberOfParametricDimension+1. The first cell allows to know the direction combination (see table 1), the following cells give the rule to use for each direction:

|  |  |
| --- | --- |
| CombineNo | No combination between the directions |
| Combine12 | Direction r and s combined with the same rule |
| Combine23 | Direction 2 and 3 combined with the same rule |
| Combine31 | Direction 3 and 1 combined with the same rule |
| *Table 1 : keyword for rule combination* | |

We suggest gathering the individual IntegrationRule\_t nodes in a parent IntegrationRulesCollection\_t node. This latter node is located under a Base\_tnode. It contains a list of IntegrationRule\_t nodes and an “IdToQualifier” information. This “IdToQualifier” information store an array of tuple (“id”, “nodename”) where id is an integer and nodename is a 32 characters string. It is used to map an id to an IntegrationRule\_t node name located under the current IntegrationRules\_t node. Thus, an integer array, instead of a string array, allow defining the integration rules for each cell (in a FlowSolution\_t node for example, see thereafter). The IntegrationRulesCollection\_t try to do efficient storage for definition of how to get Weights and IntegrationPoint for all the elements.



## Defining Variable values at a new “IntegrationPoint” location

Some modification have to be added under a FlowSolution\_t, ZoneSubRegion\_t, BC\_t and BCDataSet\_t node in order to use integration point such as vertex or cell center grid location:

* GridLocation will be allowed as IntegrationPoint
* An ItgPointsStartOffset array, re-using the same concept from NGON and NFACE, is present. For each element, it allows to know where in a solution array starts the data corresponding to integration points of each element and it allows get easily the number of integration points inside a specific element. Thus, one can either select data based on global integration point number (as it is done for vertex data) or by element. The same sorting is expected between IntegrationRule\_t points and data in the solution, subregion, bc and bcdataset. Since this offset notion is a bit different from the DataArray\_t type located under the FlowSolution\_t, ZoneSubRegion\_t, BC\_t, BCDataSet\_t nodes, it would be nice to create a new type “Offset\_t".

To associated IntegrationPoint to the data stored in the FlowSolution, ZoneSubRegion, BC, BCDataSet nodes, two elements are needed and stored inside an **ItgRules** node of type ElementAssociation\_t):

* A “***Path***”: path to an IntegrationRulesCollection\_t node (a simple character string, ex: “/Base/IntegrationGauss” or “/Base/Zone1/IntegrationRules”…)
* An “***Ids”:*** integer array, with size the number of cell elements that will store values of the corresponding **IntegrationRule** id associated to each cell element. Thus for an element, it is possible to downgrade its Integration Order as long as the linked IntegrationRule\_t is compatible with the element type. If the “Ids” array has only one value, it means that all the elements are of the same type and use the same integration rule.

If **ItgRules**is not defined under the **FlowSolution\_t, ZoneSubregion\_t, BC\_t or BCDataSet\_t** node, it can be searched as an alternative under the **Elements\_t** node defining each geometric element. In this case, under the **Element\_t** node will be added a node named “***ItgRules***” of type “***ElementAssociation\_t***” as described above.

This mechanism is generic and efficient as one can even do partial read of element associated information. This allow to not duplicate information in the CGNS tree***.***

## 

# Conflict and compatibility concern

No conflict are expected since only extension of existing data structures is done.

# Conclusion

This extension proposal of Integration and Quadrature storage completes the existing interpolation functionalities. It is meant to be parallel efficient and have low impact on existing CGNS SIDS structure.

# Document modification list

None

# Appendix - Extension to the CGNS/SIDS

The previous section presented the different features needed to have a proper definition of quadrature in CGNS. This section presents the modification applied to the CGNS SIDS.

## Extension of section 4 “Building-Block Structure Definition”

## Extension of section 4.5 “GridLocation\_t”

GridLocation\_t identifies locations with respect to the grid; it is an enumeration type.

|  |  |  |  |
| --- | --- | --- | --- |
| GridLocation\_t := Enumeration( | | | |
|  | GridLocationNull, |  |
|  | GridLocationUserDefined, |  |
|  | Vertex, |  |
|  | CellCenter, |  |
|  | FaceCenter, |  |
|  | IFaceCenter, |  |
|  | JFaceCenter, |  |
|  | KFaceCenter, |  |
|  | EdgeCenter, |  |
|  | IntegrationPoint); |  |

## New section 4.9 “MapName\_t”

The MapName\_t structure provides a way to associate an identification number with a node name.

|  |  |  |
| --- | --- | --- |
| MapName\_t<int Length> := | | |
| { | | |
|  | Data(int, 1, Length) Ids ; | (r) |
|  | Data(char, 2, , [32, Length]) Names ; | (r) |
| } ; | | |

## New section 4.10 “ElementSpace\_t”

|  |  |  |
| --- | --- | --- |
| *ElementSpace\_t := Enumeration(* | | |
|  | *Null,* |  |
|  | *UserDefined,* |  |
|  | *Parametric,* |  |
|  | *Barycentric) ;* |  |

## Extension of section 6 “Hierarchical Structures”

## Extension of section 6.2 “CGNS Entry Level Structure Definition: CGNSBase\_t”

|  |  |  |
| --- | --- | --- |
| CGNSBase\_t := | | |
| { | | |
|  | List( Descriptor\_t Descriptor1 ... DescriptorN ) ; | (o) |
|  | int CellDimension ; | (r) |
|  | int PhysicalDimension ; | (r) |
|  | BaseIterativeData\_t BaseIterativeData ; | (o) |
|  | List( Zone\_t<CellDimension, PhysicalDimension> Zone1 ... ZoneN ) ; | (o) |
|  | ReferenceState\_t ReferenceState ; | (o) |
|  | Axisymmetry\_t Axisymmetry ; | (o) |
|  | RotatingCoordinates\_t RotatingCoordinates ; | (o) |
|  | Gravity\_t Gravity ; | (o) |
|  | SimulationType\_t SimulationType ; | (o) |
|  | DataClass\_t DataClass ; | (o) |
|  | DimensionalUnits\_t DimensionalUnits ; | (o) |
|  | FlowEquationSet\_t<CellDimension> FlowEquationSet ; | (o) |
|  | ConvergenceHistory\_t GlobalConvergenceHistory ; | (o) |
|  | List( IntegrationRulesCollection\_t ItgRules1... ItgRulesN ) ; | (o) |
|  | List( IntegralData\_t IntegralData1... IntegralDataN ) ; | (o) |
|  | List( Family\_t Family1... FamilyN ) ; | (o) |
|  | List( UserDefinedData\_t UserDefinedData1 ... UserDefinedDataN ) ; | (o) |
| } ; | | |

## Extension of section 7 “Grid Coordinates, Elements, and Flow Solutions”

## Extension of section 7.3 “Elements Structure Definition: Elements\_t”

|  |  |  |
| --- | --- | --- |
| Elements\_t := | | |
| { | | |
|  | List( Descriptor\_t Descriptor1 ... DescriptorN ) ; | (o) |
|  | Rind\_t<IndexDimension> Rind ; | (o/d) |
|  | IndexRange\_t ElementRange ; | (r) |
|  | int ElementSizeBoundary ; | (o/d) |
|  | ElementType\_t ElementType ; | (r) |
|  | DataArray\_t<int, 1, ElementDataSize> ElementConnectivity ; | (r) |
|  | DataArray\_t<int, 1, ElementSize + 1> ElementStartOffset ; | (r) |
|  | DataArray\_t<int, 2, [ElementSize, 2]> ParentElements ; | (o) |
|  | DataArray\_t<int, 2, [ElementSize, 2]> ParentElementsPosition ; | (o) |
|  | List(ElementAssociation\_t<ElementSize> Property1 ...PropertyN ) ; | (o) |
|  | List( UserDefinedData\_t UserDefinedData1 ... UserDefinedDataN ) ; | (o) |
| } ; | | |

**Following text is added:**

*The ElementAssociation\_t data structure allows arbitrary mapping of properties on each individual element of the Elements\_t. This mechanism is describre in section 12 as a miscellaneous data structures that create a link to a collection of property nodes.*

## Extension of section 7.7 “Flow Solution Structure Definition FlowSolution\_t”

|  |  |  |
| --- | --- | --- |
| FlowSolution\_t< int CellDimension, int IndexDimension,  int VertexSize[IndexDimension],  int CellSize[IndexDimension],  int IntegrationPointSize[IndexDimension]> := | | |
| { | | |
|  | List( Descriptor\_t Descriptor1 ... DescriptorN ) ; | (o) |
|  | GridLocation\_t GridLocation ; | (o/d) |
|  | ElementAssociation\_t<CellSize> ItgRules ; | (o) |
|  | Offset\_t<CellSize+1> ItgPointStartOffset ; | (o) |
|  | Rind\_t<IndexDimension> Rind ; | (o/d) |
|  | IndexRange<IndexDimension> PointRange ; | (o) |
|  | IndexArray<IndexDimension, ListLength[], int> PointList ; | (o) |
|  | List( DataArray\_t<DataType, IndexDimension, DataSize[]>  DataArray1 ... DataArrayN ) ; | (o) |
|  | DataClass\_t DataClass ; | (o) |
|  | DimensionalUnits\_t DimensionalUnits ; | (o) |
|  | List( UserDefinedData\_t UserDefinedData1 ... UserDefinedDataN ) ; | (o) |
| } ; | | |

**Proposal for modification in the notes:**

*Notes:*

*…*

*5. For unstructured zones* [*GridLocation*](https://cgns.github.io/CGNS_docs_current/sids/build.html#GridLocation) *options are limited to Vertex, CellCenter or IntegrationPoint unless one of PointList or PointRange is present.*

*…*

*For unstructured grids, the value of GridLocation alone specifies location and indexing of flow solution data only for vertex and cell-centered data. The reason for this is that element-based grid connectivity provided in the* [*Elements\_t*](https://cgns.github.io/CGNS_docs_current/sids/gridflow.html#Elements) *data structures explicitly indexes only vertices and cells. For data stored at alternate grid locations (e.g., edges), additional connectivity information is needed. This is provided by the optional fields PointRange and PointList; these refer to vertices, edges, faces or cell centers, depending on the values of CellDimension and GridLocation. The following table shows these relations. The NODE element type should not be used in place of the vertex. A vertex GridLocation should use the GridLocation = Vertex pattern, which implies an indexing on the grid coordinates arrays and not a NODE Elements\_t array.*

*For data stored at an IntegrationPoint GridLocation, the indexes follow the cell indexing and the GridLocation node should provide information for sub-indexing of element integration point. In this case two data are required. They are store under the nodes named “ItgRules” and”ItgPointStartOffset”. The former node is of type ElementAssociation\_t and define how to build the integration points. If it is absent, the integration points should be deduced from ElementAssociation\_t nodes named similarly ItgRules located under the Elements\_t structures. The latter node is typed as an Offset\_t and is similar to ElementStartOffset, it gives the location in a Solution field of the start of an element’s integration point’s data. This allows quick retrieval by element indices besides the standard Solution field retrieval by integration point index.*

*If GridLocation is set to IntegrationPoint, ItgPointsStartOffset is required. It contains the starting positions of each element in the a solution data array and its last value corresponds to the IntegrationPointSize :*

*ItgPointsOffset = 0, NItgPE\_1, NItgPE\_1+ NItgPE\_2, ... ItgPointsOffset[n-1] + NItgPE\_n, ..., ItgPointsOffset[M-1] + NItgPE\_M = IntegrationPointSize*

*where NItgPE\_n is the number of integration point in element n.*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CellDimension** | **GridLocation** | | | | |
| **Vertex** | **EdgeCenter** | **\*FaceCenter** | **CellCenter** | **IntegrationPoint** |
| 1 | vertices | - | - | cells (line elements) | Integration Points |
| 2 | vertices | edges | - | cells (area elements) | Integration Points |
| 3 | vertices | edges | faces | cells (volume elements) | Integration Points |

*……*

#### **FUNCTION DataSize[]**:

return value: one-dimensional int array of length IndexDimension

dependencies: IndexDimension, VertexSize[], CellSize[], IntegrationPointSize[],GridLocation, Rind, ListLength[]

if (GridLocation = IntegrationPoint) then

{

DataSize[] = IntegrationPointSize[] ;

}

else if (PointRange/PointList is present) then

{

DataSize[] = ListLength[] ;

}

else if (Rind is absent) then

{

if (GridLocation = Vertex) or (GridLocation is absent)

{

DataSize[] = VertexSize[] ;

}

else if (GridLocation = CellCenter) then

{

DataSize[] = CellSize[] ;

}

}

else if (Rind is present) then

{

if (GridLocation = Vertex) or (GridLocation is absent) then

{

DataSize[] = VertexSize[] + [a + b,...] ;

}

else if (GridLocation = CellCenter)

{

DataSize[] = CellSize[] + [a + b,...] ;

}

}

## Extension of section 7.9 “Zone Subregion Structure Definition ZoneSubRegion\_t”

|  |  |  |
| --- | --- | --- |
| ZoneSubRegion\_t< int IndexDimension,  int CellDimension> := | | |
| { | | |
|  | List( Descriptor\_t Descriptor1 ... DescriptorN ) ; | (o) |
|  | int RegionCellDimension ; | (o/d) |
|  | GridLocation\_t GridLocation ; | (o/d) |
|  | ElementAssociation\_t< ListLength[]> ItgRules ; | (o) |
|  | Offset\_t<ListLength[]+1> ItgPointStartOffset ; | (o) |
|  | IndexRange\_t<IndexDimension> PointRange ; | (r:o:o:o) |
|  | IndexArray\_t<IndexDimension, ListLength, int> PointList ; | (o:r:o:o) |
|  | Descriptor\_t BCRegionName ; | (o:o:r:o) |
|  | Descriptor\_t GridConnectivityRegionName ; | (o:o:o:r) |
|  | Rind\_t<IndexDimension> Rind; | (o/d) |
|  | List( DataArray\_t<DataType, 1, DataSize[]> DataArray1...DataArrayN ) ; | (o) |
|  | FamilyName\_t FamilyName ; | (o) |
|  | List( AdditionalFamilyName\_t AddFamilyName1 ... AddFamilyNameN ) ; | (o) |
|  | DataClass\_t DataClass ; | (o) |
|  | DimensionalUnits\_t DimensionalUnits ; | (o) |
|  | List( UserDefinedData\_t UserDefinedData1 ... UserDefinedDataN ) ; | (o) |
| } ; | | |

**Proposal for modification in the notes:**

*Notes:*

**…**

The extent of the subregion and the distribution of data within that subregion is determined by RegionCellDimension, GridLocation, one of PointRange/List, BCRegionName, or GridConnectivityRegionName, and ItgRules (for IntegrationPoint\_t grid location). For a 3-D subregion (RegionCellDimension = 3), data can be located at vertices, edges, face centers, cell centers or integration points. For a 2-D subregion (RegionCellDimension = 2), data can be located at vertices, edges, cell centers (i.e. area elements) or integration points.

**…**

PointRange/List refer to vertices, edges, faces or cell centers, depending on the values of RegionCellDimension and GridLocation. Note that it is both the dimensionality of the zone (CellDimension) as well as the dimensionality of the subregion (RegionCellDimension), that determines the types of elements permissible in PointRange/List. The following table shows these relations.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CellDimension** | **RegionCellDimension** | **GridLocation** | | | | |
| **Vertex** | **EdgeCenter** | **\*FaceCenter** | **CellCenter** | **IntegrationPoint** |
| 1 | 1 | vertices | - | - | Cells (line elements) | Integration Points |
| 2 | 1 | vertices | edges | - | - | Integration Points |
| 2 | 2 | vertices | edges | - | Cells (area elements) | Integration Points |
| 3 | 1 | vertices | edges | - | - | Integration Points |
| 3 | 2 | vertices | edges | faces | - | Integration Points |
| 3 | 3 | vertices | edges | faces | Cells (volumes elements) | Integration Points |

*Note*: In the table, \*FaceCenter stands for the possible types: IFaceCenter, JFaceCenter, KFaceCenter, or FaceCenter.

For both structured and unstructured grids, GridLocation = Vertex means that PointRange/List refers to vertex indices. For structured grids, edges, faces and cell centers are indexed using the minimum of the connecting vertex indices, as described in the section [Structured Grid Notation and Indexing Conventions](https://cgns.github.io/CGNS_docs_current/sids/conv.html#structgrid). For unstructured grids, edges, faces and cell centers are indexed using their element numbering, as defined in the [Elements\_t](https://cgns.github.io/CGNS_docs_current/sids/gridflow.html#Elements) data structures.

*For data stored at an IntegrationPoint GridLocation, the indexes follow the cell indexing and the GridLocation node should provide information for sub-indexing of element integration point. In this case two data are required. They are store under the nodes named “ItgRules” and”ItgPointStartOffset”. The former node is of type ElementAssociation\_t and define how to build the integration points. If it is absent, the integration points should be deduced from ElementAssociation\_t nodes named similarly ItgRules located under the Elements\_t structures. The latter node is typed as an Offset\_t and is similar to ElementStartOffset, it gives the location in a Solution field of the start of an element’s integration point’s data. This allows quick retrieval by element indices besides the standard Solution field retrieval by integration point index.*

*If GridLocation is set to IntegrationPoint, ItgPointsStartOffset is required. It contains the starting positions of each element in the a solution data array and its last value corresponds to the IntegrationPointSize :*

*ItgPointsOffset = 0, NItgPE\_1, NItgPE\_1+ NItgPE\_2, ... ItgPointsOffset[n-1] + NItgPE\_n, ..., ItgPointsOffset[M-1] + NItgPE\_M = IntegrationPointSize*

*where NItgPE\_n is the number of integration point in element n.*

**….**

ZoneSubRegion\_t requires the structure function [ListLength[]](https://cgns.github.io/CGNS_docs_current/sids/bc.html#ListLength), which is used to specify the number of data points (e.g. vertices, cell centers, face centers, edge centers) corresponding to the given PointRange/List. If PointRange is specified, then ListLength is obtained from the number of points (inclusive) between the beginning and ending indices of PointRange. If PointList is specified, then ListLength is the number of indices in the list of points. In this situation, ListLength becomes a user input along with the indices of the list PointList. By *user* we mean the application code that is generating the CGNS database. ZoneSubRegion\_t requires the structure function [DataSize](https://cgns.github.io/CGNS_docs_current/sids/bc.html#ListLength), which is used to specify the size of the data array. The function is the same than the one used in the FlowSolution\_t section.

[Rind](https://cgns.github.io/CGNS_docs_current/sids/build.html#Rind) is an optional field that indicates the number of rind planes (for structured grids) or rind points (for unstructured grids). If Rind is absent, then the DataArray\_t structure entities contain only core data of length [DataSize](https://cgns.github.io/CGNS_docs_current/sids/bc.html#ListLength), as defined for this region. If Rind is present, it will provide information on the number of rind elements, in addition to the [DataSize](https://cgns.github.io/CGNS_docs_current/sids/bc.html#ListLength), that are contained in the DataArray\_t structures. The bottom line is that Rind simply adds a specified number to [DataSize](https://cgns.github.io/CGNS_docs_current/sids/bc.html#ListLength), as used by the DataArray\_t structures.

## Extension of section 9 “Boundary Conditions”

## Extension of section 9.3 “Boundary Condition Structure Definition: BC\_t”

|  |  |  |
| --- | --- | --- |
| BC\_t< int CellDimension,  int IndexDimension,  int PhysicalDimension> := | | |
| { | | |
|  | List( Descriptor\_t Descriptor1 ... DescriptorN ) ; | (o) |
|  | BCType\_t BCType ; | (r) |
|  | GridLocation\_t GridLocation ; | (o/d) |
|  | ElementAssociation\_t<ListLength[]> ItgRules ; | (o) |
|  | Offset\_t<ListLength[]> ItgPointStartOffset ; | (o) |
|  | IndexRange\_t<IndexDimension> PointRange ; | (r:o) |
|  | IndexArray\_t<IndexDimension, ListLength[], int> PointList ; | (o:r) |
|  | int[IndexDimension] InwardNormalIndex ; | (o) |
|  | IndexArray\_t<PhysicalDimension, ListLength[], real> InwardNormalList ; | (o) |
|  | List( BCDataSet\_t<CellDimension, IndexDimension, DataSize[], GridLocation>  BCDataSet1 ... BCDataSetN ) ; | (o) |
|  | BCProperty\_t BCProperty ; | (o) |
|  | FamilyName\_t FamilyName ; | (o) |
|  | List( AdditionalFamilyName\_t AddFamilyName1 ... AddFamilyNameN ) ; | (o) |
|  | ReferenceState\_t ReferenceState ; | (o) |
|  | DataClass\_t DataClass ; | (o) |
|  | DimensionalUnits\_t DimensionalUnits ; | (o) |
|  | List( UserDefinedData\_t UserDefinedData1 ... UserDefinedDataN ) ; | (o) |
|  | int Ordinal , | (o) |
| } ; | | |

**Proposal for modification in the notes:**

*Notes:*

*…*

The BC patch may be specified by PointRange if it constitutes a logically rectangular region. In all other cases, PointList should be used to list the vertices, cell edges/faces or integration points making up the BC patch. When GridLocation is set to Vertex, then PointList or PointRange refer to vertex indices, for both structured and unstructured grids. When GridLocation is set to EdgeCenter, then PointRange/List refer to edge elements. For 3-D grids, when GridLocation is set to FaceCenter, IFaceCenter, etc., then PointRange/List refer to face elements.

*When GridLocation is set to IntegrationPoint, the indexes follow the cell indexing and the GridLocation node should provide information for sub-indexing of element integration point. In this case two data are required. They are store under the nodes named “ItgRules” and”ItgPointStartOffset”. The former node is of type ElementAssociation\_t and define how to build the integration points. If it is absent, the integration points should be deduced from ElementAssociation\_t nodes named similarly ItgRules located under the Elements\_t structures. The latter node is typed as an Offset\_t and is similar to ElementStartOffset, it gives the location in a Solution field of the start of an element’s integration point’s data. This allows quick retrieval by element indices besides the standard Solution field retrieval by integration point index.*

*If GridLocation is set to IntegrationPoint, ItgPointsStartOffset is required. It contains the starting positions of each element in the a solution data array and its last value corresponds to the IntegrationPointSize :*

*ItgPointsOffset = 0, NItgPE\_1, NItgPE\_1+ NItgPE\_2, ... ItgPointsOffset[n-1] + NItgPE\_n, ..., ItgPointsOffset[M-1] + NItgPE\_M = IntegrationPointSize*

*where NItgPE\_n is the number of integration point in element n*

The interpretation of PointRange/List is summarized in the table below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CellDimension** | **GridLocation** | | | | |
| **Vertex** | **EdgeCenter** | **\*FaceCenter** | **CellCenter** | **IntegrationPoint** |
| 1 | vertices | - | - | cells (line elements) | Integration Points |
| 2 | vertices | edges | - | cells (area elements) | Integration Points |
| 3 | vertices | edges | faces | cells (volume elements) | Integration Points |

…

#### **FUNCTION ListLength[]**:

return value: int

dependencies: PointRange, PointList, GridLocation, IntegrationPointSize[]

BC\_t requires the structure function ListLength, which is used to specify the number of vertices, edge/face elements or integration points making up the BC patch. If PointRange is specified, then ListLength is obtained from the number of points (inclusive) between the beginning and ending indices of PointRange. If PointList is specified, then ListLength is the number of indices in the list of points. In this situation, ListLength becomes a user input along with the indices of the list PointList. By *user* we mean the application code that is generating the CGNS database.

ListLength is also the number of elements in the list InwardNormalList. Note that syntactically PointList and InwardNormalList must have the same number of elements.

If neither PointRange or PointList is specified in a particular BCDataSet\_t substructure, ListLength must be passed into it to determine the length of BC data arrays.

#### **FUNCTION DataSize[]**:

return value: int

dependencies: IntegrationPointSize[],GridLocation, ListLength[]

if (GridLocation = IntegrationPoint) then

{

DataSize[] = IntegrationPointSize[] ;

}

else

{

DataSize[] = ListLength[] ;

}

## Extension of section 9.4 “Boundary Condition Data Structure Definition: BCDataSet\_t”

|  |  |  |
| --- | --- | --- |
| BCDataSet\_t< int CellDimension,  int IndexDimension,  int ListLengthParameter,  GridLocation\_t GridLocationParameter> := | | |
| { | | |
|  | List( Descriptor\_t Descriptor1 ... DescriptorN ) ; | (o) |
|  | BCTypeSimple\_t BCTypeSimple ; | (r) |
|  | BCData\_t<ListLengthBCData[]> DirichletData ; | (o) |
|  | BCData\_t<ListLengthBCData[]> NeumannData ; | (o) |
|  | GridLocation\_t GridLocation ; | (o/d) |
|  | ElementAssociation\_t< ListLength[]> ItgRules ; | (o) |
|  | Offset\_t<ListLength[]> ItgPointStartOffset ; | (o) |
|  | IndexRange\_t<IndexDimension> PointRange ; | (o) |
|  | IndexArray\_t<IndexDimension, ListLength, int> PointList ; | (o) |
|  | ReferenceState\_t ReferenceState ; | (o) |
|  | DataClass\_t DataClass ; | (o) |
|  | DimensionalUnits\_t DimensionalUnits ; | (o) |
|  | List( UserDefinedData\_t UserDefinedData1 ... UserDefinedDataN ) ; | (o) |
| } ; | | |

**Proposal for modification in the notes:**

*Notes :*

*…*

*3.* GridLocation is optional; if absent its default value is GridLocationParameter. For 2-D grids (CellDimension = 2), GridLocation may take the values of Vertex, EdgeCenter or IntegrationPoint. For 3-D grids (CellDimension = 3), GridLocation may take the values of Vertex, EdgeCenter, FaceCenter, IFaceCenter, JFaceCenter, KFaceCenter or IntegrationPoint.

*…*

#### **FUNCTION ListLengthBCData[]**:

return value: int  
dependencies: ListLengthParameter, ListLength, PointRange, PointList, GridLocation, IntegrationPointSize[]

BCDataSet\_t also requires the structure function ListLengthBCData

if (GridLocation = IntegrationPoint) then

{

ListLengthBCData  [] = IntegrationPointSize[] ;

}

else if (PointRange/PointList is present) then

{

ListLengthBCData  [] = ListLength[] ;

}

else

{

ListLengthBCData  [] = ListLengthParameter ;

}

## Extension of section 12 “Miscellaneous Data Structures”

## New section 12.12: Element Association Structure Definition ElementAssociation\_t

*The* ElementAssociation\_t *specifies an array of identification numbers. The array size is ElementSize or 1.*

|  |  |  |
| --- | --- | --- |
| ElementAssociation\_t< int ElementSize > := | | |
| { | | |
|  | List( Descriptor\_t Descriptor1 ... DescriptorN ) ; | (o) |
|  | Data(char, 1,string\_length) Path; | (r) |
|  | DataArray\_t<int, 1, 1> Ids; | (r:o) |
|  | DataArray\_t<int, 1, ElementSize> Ids; | (o:r) |
|  | List( UserDefinedData\_t UserDefinedData1 ... UserDefinedDataN ) ; | (o) |
| } ; | | |

**Following text is added:**

*The ElementAssociation\_t structure can be located under an Elements\_t node, or FlowSolution\_t, a ZoneSubRegion\_t, a BC\_t or a BCDataSet\_t node which GridLocation is set to InterpolationPoints. The path of the ElementAssociation\_t is a string which define a target node containing an IdToQualifier information. This latter information will translate the “Ids” stored in ElementAssociation\_t node into a node name located in the children of the target node. Then it allows to specify a collection of property nodes as children of the target and do an assignment by elements.*

*In the case of an ItgRules node of type ElementAssociation\_t, the path should points to a valid IntegrationRulesCollection\_t node (for instance located at /Base/GaussIntegration)*

*The array named “Ids” can be of size 1 if the information is global or else it should be of size ElementSize for local assignment.*

## New section 12.13 : Integration Rules Structure Definition IntegrationRuleCollection\_t

*The* IntegrationRulesCollection\_t *specifies a collection of indexed IntegrationRule\_t node.*

|  |  |  |
| --- | --- | --- |
| IntegrationRulesCollection\_t< int NumIndexedIntegrationRules > := | | |
| { | | |
|  | List( Descriptor\_t Descriptor1 ... DescriptorN ); | (o) |
|  | MapName\_t<NumIndexedIntegrationRules> IdToQualifier; | (r) |
|  | List( IntegrationRule\_t ItgRule1 ... ItgRuleN ): | (r) |
|  | List( UserDefinedData\_t UserDefinedData1 ... UserDefinedDataN ) ; | (o) |
| } ; | | |

**Following text is added:**

The number of stored IntegrationRule\_t node by the IntegrationRulesCollection\_t structure should be greater or equal to the number of indexed IntegrationRule\_t. The IdToQualifier node bind a number to a node name. Each node name of the IdToQualifier should be present in the list of IntegrationRule\_t. When given an id provided by an ElementAssociation\_t node, a comparison with Ids present in the IdToQualifier structure allows to get the corresponding IntegrationRule\_t node name where to read the element integration weights and point location. The IntegrationRulesCollection\_t node can be a child of a Base\_t.

## New section 12.14 : Integration Rule Structure Definition IntegrationRule\_t

*The* IntegrationRule\_t *specifies an elementary quadrature scheme for a specific type of element.*

|  |  |  |
| --- | --- | --- |
| IntegrationRule\_t <int NumberOfElementVertex,  int ParametricDim,  int NumberOFIntegrationPoint> := | | |
| { | | |
|  | List( Descriptor\_t Descriptor1 ... DescriptorN ) ; | (o) |
|  | ElementType\_t ElementType; | (r) |
|  | ElementSpace\_t ReferenceSpace ; | (o/d) |
|  | int NumberOfIntegrationPoint; | (r) |
|  | int ParametricDim; | (o) |
|  | DataArray\_t <char, 1, string\_length> IntegrationName ; | (o) |
|  | DataArray\_t <real, 2, [ParametricDim, NumberOfIntegrationPoint]>  ParametricIntegrationPoint; | (r/o) |
|  | DataArray\_t <real, 2, [NumberOfElementVertex, NumberOfIntegrationPoint]> BarycentricIntegrationPoint; | (o/r) |
|  | DataArray\_t <real, 1, NumberOfIntegrationPoint> Weights; | (r) |
|  | List(UserDefinedData\_t UserDefinedData1 ... UserDefinedDataN ); | (o) |
|  |  |  |
| } ; | | |

**Following text is added:**

The ElementType define the element type for which the integration rule is valid. In this context, the ElementType “MIXED” is excluded.

The ReferenceSpace is either Parametric or Barycentric. The default value is Parametric if ReferenceSpace is absent. If ElementType is NGON\_n or NFACE\_n, the ReferenceSpace can only be set to Barycentric.

If ReferenceSpace is set to Barycentric, Integration Points are defined through a weighted sum on Element Vertex Points.

If ReferenceSpace is set to Parametric, Integration Points are determined through interpolation function (see section General Interface Connectivity in https://cgns.github.io/CGNS\_docs\_current/sids/cnct.html for the interpolation definition)

The NumberOfIntegrationPoint is a value that provides information to size the different array and indicates the overall numerical formula integration order. The ParametricDim is also needed in case of Parametric definition of the IntegrationRule\_t.

IntegrationName can be a unique name or a combination of multiple names corresponding to each parametric index. In this case, the character ‘x’ is inserted between each formula name. The available standard names are: Gauss, GaussLobatto, GaussLegendre, GausChebychev.

The ParametricIntegrationPoint stores coefficient in the parametric space of the element to describe the Integration Points position.

Thus the physical position is evaluated through, the formula:

where corresponds to the ParametricIntegrationPoint data and is the weight associated to the element node at the j position according to interpolation functions:

(<https://cgns.github.io/CGNS_docs_current/sids/cnct.html>)

In case of a *Parametric* definition, the Integration Points are stored following the principle of growing *r* then growing *s* and ending by growing *t*.

Alternatively, if the ReferenceSpace is Barycentric the formula is similar:

And the directly corresponds to the BarycentricIntegrationPoint array data.

To complete the quadrature definition, the “Weights” array provides the weight to use in the IntegrationRule formula for a given solution variable:

## Extension of Appendix A “Convention for Data-Name Identifiers”

## New section A.8 “Quadrature rules”

Data-name identifiers related to the quadrature include those associated with the IntegrationName node described in a IntegrationRule\_t node.

|  |  |
| --- | --- |
| **Data name Identifier** | **Description** |
| GaussLegendre | Gauss quadrature rule using Legendre polynomials |
| GaussLaguerre | Gauss quadrature rule using Laguerre polynomials |
| GaussChebyshev | Gauss quadrature rule using Chebyshev polynomials |
| GaussHermite | Gauss quadrature rule using Hermite polynomials |
| GaussLobatto | Gauss-Lobatto quadrature rule (using Legendre polynomials) |
| Hammer | Hammer quadrature rule (for triangle and tetrahedron) |
| Simpsons | Simpsons quadrature rule |
| Newton-Cotes | Newton-Cotes quadrature rule |

# Appendix - Extension to the CGNS/Filemap

Two children node will be added to FlowSolution\_t :

|  |  |
| --- | --- |
| **FlowSolution\_t** | |
|  | **Child Nodes** |
|  | .... |
|  | **Name:** ItgRules  **Label:** ElementAssociation\_t  **Cardinality:** 0,1  **See:** ElementAssociation\_t figure  **Parameters :** CellSize |
|  | **Name:** ItgPointStartOffset  **Data-Type:** cgsize\_t  **Dimensions:** 1  **DimensionValues:** CellSize+1  **Label:** Offset\_t  **Cardinality:** 0,1  **Parameters:** CellSize |

One children node will be added to Elements\_t :

|  |  |
| --- | --- |
| **Elements\_t** | |
|  | **Child Nodes** |
|  | .... |
|  | **Name:** ItgRules  **Label:** ElementAssociation\_t  **Cardinality:** 0,1  **See:** ElementAssociation\_t figure  **Parameters :** ElementSize |

|  |  |
| --- | --- |
| **ElementAssociation\_t** | |
| **Name : User defined**  **Label : ElementAssociation\_t**  **Data-Type: MT**  **Parameters: DataSize** | |
|  | **Child Nodes** |
|  | **Name:** Path  **Label:** DataArray\_t  **Data-Type:** C1  **Dimensions:** 1  **Dimension Values:** Length of String  **Cardinality:** 1 |
|  | **Name:** Ids  **Data-Type:** I4  **Dimensions:** 1  **DimensionValues:** DataSize  **Cardinality :** 1  **Data :** Local Identification number for each element |
|  | **Name:** Ids  **Data-Type:** I4  **Dimensions:** 1  **DimensionValues:** 1  **Cardinality :** 1  **Data :** Global Identification number for all elements |

One child node will be added to Base\_t :

|  |  |
| --- | --- |
| **CGNSBase\_t** | |
|  | **Child Nodes** |
|  | .... |
|  | **Name:** User defined  **Label:** IntegrationRulesCollection\_t  **DataType:** MT  **Cardinality:** 0,N  **See:** IntegrationRulesCollection\_t figure  **Parameters :** CellSize |

|  |  |
| --- | --- |
| **IntegrationRulesCollection\_t** | |
|  | **Child nodes** |
|  | **Name:** IdToQualifier  **Label:** MapName\_t  **DataType:** I4  **Dimensions:** 1  **DimensionValues:** Number Of Indexed Node Names  **Data :** Identification numbers associated to children nodesof the parent node  **Cardinality:** 1  **See:** MapName\_t figure  **Parameters :** Number Of Integration Rule |
|  | **Name:** User defined  **Label:** IntegrationRule\_t  **DataType:** I4  **Dimensions:**1  **DimensionValues:**1  **Data:** ElementType  **See:** IntegrationRule\_t figure  **Cardinality:** 1,N |
|  | |  |  | | --- | --- | | **Name:** | User defined | | **Label:** | [Descriptor\_t](https://cgns.github.io/CGNS_docs_current/sids/build.html#Descriptor) | | **See:** | [CGNSBase\_t figure](https://cgns.github.io/CGNS_docs_current/filemap/figures/CGNSBase.html) | |
|  | |  |  | | --- | --- | | **Name:** | User defined | | **Label:** | UserDefinedData\_t | | **See:** | [CGNSBase\_t figure](https://cgns.github.io/CGNS_docs_current/filemap/figures/CGNSBase.html) | |

|  |  |
| --- | --- |
| **MapNames\_t** | |
| **Name:** User Defined  **Label:** MapName\_t  **DataType:** I4  **Dimensions:** 1  **DimensionValues:** Number Of Indexed Node Names  **Data :** Identification numbers associated to children nodesof the parent node  **Parameter :** Number Of Indexed names | |
|  | **Child Node** |
|  | **Name:** Names  **Label:** DataArray\_t  **DataType:** C1  **Dimensions:** 2  **DimensionValues:** (32, Number Of Indexed Node Names)  **Data :** List of node names  **Cardinality:** 1  **Parameters :** Number Of Indexed Node Names |

|  |  |
| --- | --- |
| **IntegrationRule\_t** | |
| **Name:** User defined  **Label:** IntegrationRule\_t  **DataType:** I4  **Dimensions:**1  **DimensionValues:** 3  **Data:** ElementType, NumberOfIntegrationPoint, ParametricDimension | |
|  | **Child node** |
|  | **Name:** ReferenceSpace  **Label:** ElementSpace\_t  **DataType:** C1  **Dimensions:** 1  **DimensionValues:** Length of string  **Data :** Barycentric, Parametric  **Cardinality:** 0,1 |
|  | **Name:** IntegationName  **Label:** DataArray\_t  **DataType:** C1  **Dimensions:** 1  **DimensionValues:** Length of string  **Data :** UserDefined Names, Gauss, GaussLobatto, GaussLegendre,  **Cardinality:** 1 |
|  | **Name:** ParametricIntegrationPoint  **Label:** DataArray\_t  **DataType:** R4 or R8  **Dimensions:** 2  **DimensionValues:** [NumberOfIntegrationPoint, ParametricDimension]  **Data :** parametric coefficient values to define the integration points location in the reference element.  **Cardinality:** 0,1 |
|  | **Name:** BarycentricIntegrationPoint  **Label:** DataArray\_t  **DataType:** R4 or R8  **Dimensions:** 2  **DimensionValues:** [NumberOfIntegrationPoint, NumberOfElementVertices]  **Data :** interpolation weights to define the integration points location in the reference element.  **Cardinality:** 0,1 |
|  | **Name:** Weights  **Label:** DataArray\_t  **DataType:** R4 or R8  **Dimensions:** 1  **DimensionValues:** NumberOfIntegrationPoint  **Data :** Quadrature weights use to compute integrals for ElementType  **Cardinality:** 1 |
|  | |  |  | | --- | --- | | **Name:** | User defined | | **Label:** | [Descriptor\_t](https://cgns.github.io/CGNS_docs_current/sids/build.html#Descriptor) | | **See:** | [CGNSBase\_t figure](https://cgns.github.io/CGNS_docs_current/filemap/figures/CGNSBase.html) | |
|  | |  |  | | --- | --- | | **Name:** | User defined | | **Label:** | UserDefinedData\_t | | **See:** | [CGNSBase\_t figure](https://cgns.github.io/CGNS_docs_current/filemap/figures/CGNSBase.html) | |

The Grid location mapping just add a new possibility:

GridLocation\_t

|  |  |  |  |
| --- | --- | --- | --- |
| Node Attributes | | | |
|  | **Name:** |  | GridLocation |
|  | **Label:** |  | [GridLocation\_t](https://cgns.github.io/CGNS_docs_current/sids/build.html#GridLocation) |
|  | **DataType:** |  | C1 |
|  | **Dimension:** |  | 1 |
|  | **Dimension Values:** |  | Length of the string value |
|  | **Data:** |  | Vertex, CellCenter, FaceCenter, IFaceCenter, JFaceCenter, KFaceCenter, EdgeCenter, IntegrationPoint |
|  | **Children:** |  | None |
|  | **Cardinality:** |  | 0,1 |

# Appendix - Extension to the CGNS/MLL

In progress