PXDMF : A File Format for Separated Variables Problems
Version 1.0

Felipe Bordeu Weldt
Materials, Processing and Composites Technology group
GeM Institute, Ecole Centrale de Nantes
felipe.bordeu@ec-nantes.fr

15/11/2011

1 Introduction

This file format is based on the popular XDMF file format (www.xdmf.org). The P in front came from the PGD method. PXDMF files are fully compliant of XDMF, but XDMF file are not compliant of PXDMF. So a PXDMF file can be open with a XDMF file reader without any problem.

The motivation is the need for a standardized method to exchange scientific data of Separated Variables Problems between High Performance Computing codes. XDMF already propose a solution for exchanging scientific data of non Separated Problems. This document is based from the documentation of the XDMF format.

PXDMF (as XDMF) categorizes data by two main attribute; size and function. Data can be *Light* (typically less than about thousand values) or *Heavy* (megabytes, terabytes, etc.). In adition to raw values, data can refer to *Format* (rank and dimensions of an array) or *Model* (how that data is to be used .i.e. XYZ coordinates vs. Vector components).

PXDMF (as XDMF) uses XML to store Light data and to describe the data Model. Heavy data can be stored in the XML file, in a HDF5 file, or in a plain binary file. The data Format is stored redundantly in both XML and HDF5. This Allows tools to parse XML to determine the resources that will be required to acces the Heavy data.

At www.xdmf.org a C++ API is provided to read and write XDMF data. This API is not necessary in order to produce or consume PXDMF data. Currently several HPC codes that already produce HDF5 data, use native text output to produce the XML necessary for valid XMDF. This routines can be easily modified to write or read PXDMF files.

XML The eXtensible Markup Language (XML) format is widely used for many purposes and is well documented at many sites. There are numerous open source parsers avialable for XML.

In PXDMF (as XDMF) is case sensitive and is made of three major components : elements, entities, and processing information. In PXDMF we're primarily concerned with the elements. Theses elements follow the basic form :

```
<ElementTag
   AttributeName="AttributeValue"
   AttributeName="AttributeValue"
   ... >
   CData
</ElementTag>
```

Each element begins with a `<tag>` and ends with a `</tag>`. Optionally there can be several "Name=Value" pairs which convey additional information. Between the `<tag>` and the `</tag>` there can be other `<tag></tag>` pairs and/or character data (CData). CData is typically where the values are stored; like the actual text in an HTML document.

Comments in XML start with a "`<!--`" and end with a "`-->`". So `<!--This is a Comment -->`.

XML is said to be "well formed" if it is syntactically correct. That means all of the quotes match, all elements have end element, etc. XML is said to be "valid" if it conforms to the *Schema* or *DTD* defined at the head of the document. For example, the schema might specify that elements typ A can contain element B but not element C. Verifying that the provided XML is well formed and/or valid are functions typically performed by the XML parser. Additionally PXDMF (like XDMF) takes advantage of two extensions to XML:

- **Xinclude**

  As opposed to entity references in XML (which is a basic subtitution mechanism). XInclude allows for the inclusion of files that are not well formed XML. This means that with XInclude the included file could be well formed XML or perhaps a flat text file of values. The syntax looks like this :

  ```
  <Xdmf Version="2.0" xmlns:xi="[http://www.w3.org/2001/XInclude]">
  <xi:include href="Example3.xmf"/>
  </Xdmf>
  ```

  the xmlns:xi stablishes a namespace xi. Then anywhere within the Xdmf element, xi:include will pull in the URL.

- **XPath**

  This allows for elements in the XML document to reference specific element in a document. For example:

  The first Grid in the first Domain

  ```
  /Xdmf/Domain/Grid
  ```

  The tenth Grid .... XPath is one based.

  ```
  /Xdmf/Domain/Grid[10]
  ```

  The first grid with an attribute Name which has a value of "PGD1"

  ```
  /Xdmf/Domain/Grid[@Name="PGD1"]
  ```

  All valid PXDMF must appear between the `<Xdmf>` and the `</Xdmf>` tags. So a minimal (empty) PXDMF XML file would be :

  ```
  <?xml version="1.0" ?>
  <!DOCTYPE Xdmf SYSTEM "Xdmf.dtd" []>
  <Xdmf Version="2.0">
  </Xdmf>
  ```

While there exists an Pxdmf DTD and a Schema they are only necessary for validating parsers. For performance reasons, validation is typically disabled.

2 PXDMF Elements

The organization of PXDMF begins with the Xdmf element (Because PXDMF is base in Xdmf and we want to keep copativility). So that parsers can distinguish from previous versions of XDMF, there exists a Version attribute (currently at 2.0). Any element in XDMF can have a Name attribute or have a Reference attribute. The Name attribute becomes important for grids while the Reference attribute is used to take advantage of the XPath facility (more detail on this later). Xdmf elements contain one or more Domain elements (computational domain). There is seldom motivation to have more than one Domain. Until now all the application using the PXDMF format use only one Domain element.

A Domain can have one or more Grid elements. Each Grid element correspond to a dimension in the separated representation. Each Grid contains two or more Information elements, a Topology, Geometry, and zero or more Attribute elements. The Information elements specifies the dimensionality of the space and the names of each dimension. Topology specifies the connectivity of the grid while Geometry specifies the location of the grid nodes. Attribute elements are used to specify values such as scalars and vectors that are located at the node or cell center.

To specify actual values for connectivity, geometry, or attributes, XDMF defines a DataItem element. A DataItem can provide the actual values or provide the physical storage (which is typically an HDF5 file).

2.1 DataItem

Even if Xdmf suports six different types of DataItems, PXDMF only use uniform DataItems:

Uniform is the simples type to specifies a sigle array. As with all XDMF elements, there are reasonable defaults wherever possible. So the simples DataItem would be :

```
<DataItem Dimensions="3">
    1.0 2.0 3.0
</DataItem>
```

Since no *ItemType* has been specified, Uniform has been assumed. The default *Format* is XML and the default *NumberType* is a 32 but floating point value. So the fully qualified DataItem for the same data would be:

```
<DataItem ItemType="Uniform"
  Format="XML"
  NumberType="Float" Precision="4"
  Rank="1" Dimensions="3">
  1.0 2.0 3.0
</DataItem>
```

Since it is only practical to store a small amount of data values in the XML (ASCII), production codes typically write their data to HDF5 or binary file and specify the location in XML. HDF5 is a hierarchical, self describing data format. So an application can open an HDF5 file without any prior knowledge of the data and determine the dimensions and number type of all the arrays stored in the file. PXDMF requires that this information also be stored redundantly in the XML so that applications need not have access to the actual heavy data in order to determine storage requirements. Binary is just a plain binary file without format.

For example, suppose an application stored a three dimensional array of pressure values at each iteration into an HDF5 file. The XML might be :

```
<DataItem ItemType="Uniform"
  Format="HDF"
  NumberType="Float" Precision="8"
  Dimensions="64 128 256">
  OutputData.h5:/Results/Iteration 100/Part 2/Pressure
</DataItem>
```

Dimensions are specified with the slowest varying dimension first (i.e. KJI order). The HDF filename can be fully qualified, if it is not it is assumed to be located in the current directory or the same directory as the XML file.

And this is the verion of a DataItem for a binary file:

```
<DataItem ItemType="Uniform"
  Format="Binary"
  Endian="Big"
  Seek="48"
  NumberType="Float" Precision="8"
  Dimensions="64 128 256"
  >
OutputData.bin
</DataItem>
```

The `Endian` keyword is use to set the endiannes of the binary file (Little or Big) and the `Seek` is used to tell the starting point of the data (unit is byte).

2.2 Grid

The DataItem element is used to define the data format portion of PXDMF. The data model portion of PXDMF begins with the Grid element. A Grid is a container for information related to 2D or 3D points, structured or unstructured connectivity, and assigned velues.

Even if the Grid of Xdmf suports 4 different types of GridType, PXDMF only use uniform GridTypes.

Uniform Grid elements are the simples type and must contain two or more *Information* elements and a *Topology* and a *Geometry* element.

A basic Grid elemente would be:

```
<Grid Name="PGD1" >
<Information Name="Dims" Value="1" />
<Information Name="Dim0" Value="x" />
<Topology ...
<Geometry ...
</Grid>
```

The value of the Name attribute of the Grid element (`"PGD1"`) correspond to the name and number of the PGD dimension (starting form 1).

The first Information elements gives the dimentionality of the Grid (PGD dimention) 1, 2 or 3, for 1D 2D or 3D spaces. The others Information elements give the name of each dimension.

2.3 Topology

The Topology element is the same as the XDMF. The Topology element describes the general organization of the data. This is the part of the computational grid that is invariant with rotation, translation, and scale. For structured grids, the connectivity is implicit. For unstructured grids, if the connectivity differs from the standard, an Order may be specified. Currently, the following Topology cell types are defined :

- **Linear**
  - Polyvertex - a group of unconnected points
  - Polyline - a group of line segments
  - Polygon
  - Triangle
  - Quadrilateral
  - Tetrahedron
  - Pyramid
  - Wedge
  - Hexahedron
- **Quadratic**
  - Edge_3 - Quadratic line with 3 nodes
  - Tri_6
  - Quad_8
  - Tet_10
  - Pyramid_13
  - Wedge_15
  - Hex_20
- **Arbitrary**
  - Mixed - a mixture of unstructured cells
- **Structured**
  - 2DSMesh - Curvilinear
  - 2DRectMesh - Axis are perpendicular
  - 2DCoRectMesh - Axis are perpendicular and spacing is constant
  - 3DSMesh
  - 3DRectMesh
  - 3DCoRectMesh

There is a NodesPerElement attribute for the cell types where it is not implicit. For example, to define a group of Octagons, set TopologyType="Polygon" and NodesPerElement="8". This type of elements is not very comun, but Polylines with only 2 nodes correspond to 1D elements. For structured grid topologies, the connectivity is implicit. For unstructured topologies the Topology element must contain a DataItem that defines the connectivity :

```
<Topology TopologyType="Quadrilateral" NumberOfElements="2" >
  <DataItem Format="XML" DataType="Int" Dimensions="2 4">
    0 1 2 3
    1 6 7 2
  </DataItem>
</Topology>
```

The connectivity defines the indexes into the XYZ geometry that define the cell. In this example, the two quads share an edge defined by the line from node 1 to node 2. A Topology element can define Dimensions or NumberOfElements; this is just added for clarity.

Mixed topologies must define the cell type of every element. If that cell type does not have an implicit number of nodes, that must also be specified. In this example, we define a topology of three cells consisting of a Tet (cell type 6) a Polygon (cell type 3) and a Hex (cell type 9) :

```
<Topology TopologyType="Mixed" NumberOfElements="3" >
  <DataItem Format="XML" DataType="Int" Dimensions="20">
    6        0 1 2 7
    3   4   4 5 6 7
    9        8 9 10 11 12 13 14 15
  </DataItem>
</Topology>
```

Notice that the Polygon must define the number of nodes (4) before its connectivity. The cell type numbers are defined in the API documentation.

- 1 POLYVERTEX
- 2 POLYLINE
- 3 POLYGON
- 4 TRI
- 5 QUAD
- 6 TET
- 4 TRI
- 7 PYRAMID
- 8 WEDGE
- 9 HEX
- 22 EDGE_3
- 24 TRI_6
- 25 QUAD_8
- 26 TET_10
- 27 PYRAMID_13
- 28 WEDGE_15
- 29 HEX_20

2.4 Geometry

The Geometry element describes the XYZ values of the mesh. The important attribute here is the organization of the points. The default is XYZ; an X,Y, and Z for each point starting at parametric index 0. Possible organizations are :

- XYZ - Interlaced locations
- XY - Z is set to 0.0
- X_Y_Z - X,Y, and Z are separate arrays
- VXVYVZ - Three arrays, one for each axis
- ORIGIN_DXDYDZ - Six Values : Ox,Oy,Oz + Dx,Dy,Dz

The following Geometry element defines 8 points :

```
<Geometry GeometryType="XYZ">
  <DataItem Format="XML" Dimensions="2 4 3">
    0.0    0.0    0.0
    1.0    0.0    0.0
    1.0    1.0    0.0
    0.0    1.0    0.0
    0.0    0.0    2.0
    1.0    0.0    2.0
    1.0    1.0    2.0
    0.0    1.0    2.0
  </DataItem>
</Geometry>
```

In the case that the Grid has a dimensionality of 1 the second and the third columns are ignored.

Together with the Grid and Topology element we now have enough to define a full PXDMF XML file that defines two quadrilaterals that share an edge (notice not all points are used), and only one PGD dimension:

```
<?xml version="1.0" ?>
<!DOCTYPE Xdmf SYSTEM "Xdmf.dtd" []>
<Xdmf Version="2.0" xmlns:xi="[http://www.w3.org/2001/XInclude]">
<Domain>
<Grid Name="PGD1">
<Information Name="Dims" Value="3">
<Information Name="Dim0" Value="X">
<Information Name="Dim1" Value="Y">
<Information Name="Dim2" Value="Z">
<Topology TopologyType="Quadrilateral" NumberOfElements="2" >
<DataItem Format="XML" DataType="Int" Dimensions="2 4">
0 1 2 3
1 6 7 2
</DataItem>
</Topology>
<Geometry GeometryType="XYZ">
<DataItem Format="XML" Dimensions="2 4 3">
0.0    0.0    0.0
1.0    0.0    0.0
1.0    1.0    0.0
0.0    1.0    0.0
0.0    0.0    2.0
1.0    0.0    2.0
```

```
1.0    1.0    2.0
0.0    1.0    2.0
</DataItem>
</Geometry>
</Grid>
</Domain>
</Xdmf>
```

It is valid to have DataItem elements to be direct children of the Xdmf or Domain elements. This could be useful if several Grids share the same Geometry but have separate Topology :

```
<?xml version="1.0" ?>
<!DOCTYPE Xdmf SYSTEM "Xdmf.dtd" []>

<Xdmf Version="2.0" xmlns:xi="[http://www.w3.org/2001/XInclude]">
<Domain>
<DataItem Name="Point Data" Format="XML" Dimensions="2 4 3">
0.0    0.0    0.0
1.0    0.0    0.0
1.0    1.0    0.0
0.0    1.0    0.0
0.0    0.0    2.0
1.0    0.0    2.0
1.0    1.0    2.0
0.0    1.0    2.0
</DataItem>
<Grid Name="PGD1">
<Information Name="Dims" Value="3">
<Information Name="Dim0" Value="X">
<Information Name="Dim1" Value="Y">
<Information Name="Dim2" Value="Z">
<Topology Type="Quadrilateral" NumberOfElements="2" >
<DataItem Format="XML"
DataType="Int"
Dimensions="2 4">
0 1 2 3
1 6 7 2
</DataItem>
</Topology>
<Geometry Type="XYZ">
<DataItem Reference="XML">
/Xdmf/Domain/DataItem[@Name="Point Data"]
</DataItem>
</Geometry>
</Grid>
</Domain>
</Xdmf>
```

2.5 Attribute

The Attribute element defines values associated with the mesh. Currently supported types of values are :

- Scalar
- Vector
- Tensor - 9 values expected
- Tensor6 - a symmetrical tensor

These values can be centered on :

- Node
- Cell

Typically Attributes are assigned on the Node :

```
<Attribute Name="Node Values_0" Center="Node">
<DataItem Format="XML" Dimensions="6 4">
100 200 300 400
500 600 600 700
800 900 1000 1100
1200 1300 1400 1500
1600 1700 1800 1900
2000 2100 2200 2300
</DataItem>
</Attribute>
```

Or assigned to the cell centers :

```
<Attribute Name="Cell Values_0" Center="Cell">
<DataItem Format="XML" Dimensions="3">
 3000 2000 1000
 </DataItem>
</Attribute>
```

The number after the name of the attribute (`... Name="Cell Values_0"` ..) correspond to the number of the mode in the separeted representation.

2.6 Time

The **Time** element is valid for Xdmf file but not for PXDMF file.

2.7 Information

The Information elements are used for to store information about the dimensionality of the PGD dimension and the names of each axis inside it. But also can be used to store code or system specific information that needs to be stored with the data that does not map to the current data model. This is intended for application specific information that can be ignore. A good example might be the bounds of a grid for a use in visualization . Information elements have a Name and Value Attribute. If Value is nonexistent the value is in the CDATA of the element:

```
<Information Name="XBounds" Value="0.0 10.0"/>
<Information Name="Bounds"> 0.0 10.0 100.0 110.0 200.0 210.0 </Information>
```

Several items can be addressed using the Information element like time, units, descriptions, etc. without polluting the XDMF schema. If some of these get used extensively they may be promoted to XDMF elements in the future.

3 XML Element and Default(*) XML Attributes :

- Attribute

```
Name              (no default, composed by :
                   name + "_" + modenumber i.e. "Depx_0")
AttributeType     *Scalar | Vector | Tensor | Tensor6
Center            *Node | Cell
```

- DataItem

```
Name              (no default)
ItemType          *Uniform
Dimensions        (no default) in KJI Order
NumberType        *Float | Int | UInt | Char | UChar
Precision         1 | *4 | 8
Format            *XML | HDF
```

- Domain (Only one)

```
Name              (no default)
```

- Geometry

```
GeometryType       *XYZ | XY | X_Y_Z | VxVyVz | Origin_DxDyDz
```

- Grid (

```
Name               (no default) composed by : "PGD" + thedimensionnumber
                    (starting at 1 i.e. "PGD1")
GridType           *Uniform
```

- Information (

```
Name                (no default)
Value               (no default)
```

- Xdmf (XdmfRoot)

```
Version              *Current Version | x
```

- Topology

```
Name                (no default)
TopologyType        Polyvertex | Polyline | Polygon |
                    Triangle | Quadrilateral | Tetrahedron | Pyramid |
                    Wedge | Hexahedron | Edge_3 | Triagle_6 |
                    Quadrilateral_8 | Tetrahedron_10 | Pyramid_13 |
                    Wedge_15 | Hexahedron_20 |
                    Mixed |
                    2DSMesh | 2DRectMesh | 2DCoRectMesh |
                    3DSMesh | 3DRectMesh | 3DCoRectMesh
NodesPerElement     (no default) Only Important for Polyvertex, Polygon
                    and Polyline
NumberOfElement     (no default)
    OR
Dimensions          (no default)
Order               each cell type has its own default
BaseOffset          *0 | #
```

4 Example of a PXDMF File

This example correspont to a 4D problem (x, y, z, w), the proplem is separated in $(x, y) \times (z) \times (w)$. The field "dep_x" has only one mode (so only one attribute named "dep_x_0").

Then the solution can be reconstructed using:

$$dep\_x = \sum_{i}^{1} (dep\_x_i(x, y) \cdot dep\_x_i(z) \cdot dep\_x_i(w)) \tag{1}$$

This file is pure XML (ASCII).

```
<?xml version="1.0" ?>
<!DOCTYPE Xdmf SYSTEM "Xdmf.dtd" []>
<Xdmf Version="2.0" xmlns:xi="http://www.w3.org/2001/XInclude" >
  <Domain Name="test_separe3.pxdmf">
    <Grid Name="PGD1"  >
      <Information Name="Dims" Value="2"  />
      <Information Name="Dim0" Value="X"  />
      <Information Name="Dim1" Value="Y"  />
      <Topology TopologyType="Quadrilateral" NumberOfElements="2" >
        <DataItem  Format="XML" NumberType="int" Dimensions="2 4">
0 1 4 3
1 2 5 4
        </DataItem>
      </Topology>
      <Geometry GeometryType="XYZ">
        <DataItem  Format="XML" NumberType="float" Dimensions="6 3">
0   0   0
1   0   0
2   0   0
```

```
0   1   0
1.2 0.8 0
2   1   0
        </DataItem>
      </Geometry>
      <Attribute Name="dep_x_0" Center="Node"  AttributeType="Scalar"  >
        <DataItem  Format="XML" NumberType="float" Dimensions="1 6">
0 1 2 0.1 1.1 2.1
        </DataItem>
      </Attribute>
    </Grid>
    <Grid Name="PGD2"  >
      <Information Name="Dims" Value="1"  />
      <Information Name="Dim0" Value="Z"  />
      <Topology TopologyType="Polyline" NumberOfElements="3"
        NodesPerElement="2"  >
        <DataItem  Format="XML" NumberType="int" Dimensions="3 2">
0 1
1 2
2 3
        </DataItem>
      </Topology>
      <Geometry GeometryType="XYZ">
        <DataItem  Format="XML" NumberType="float" Dimensions="4 3">
0.00000000  0.00000000  0.00000000
0.00694444  0.00000000  0.00000000
0.01388889  0.00000000  0.00000000
0.02083333  0.00000000  0.00000000
        </DataItem>
      </Geometry>
      <Attribute Name="dep_x_0" Center="Node"  AttributeType="Scalar"  >
        <DataItem  Format="XML" NumberType="float" Dimensions="1 4">
0.0 1.0 1.2 1.22
        </DataItem>
        </Attribute>
    </Grid>
    <Grid Name="PGD3"  >
      <Information Name="Dims" Value="1"  />
      <Information Name="Dim0" Value="W"  />
      <Topology TopologyType="Polyline" NumberOfElements="4"
         NodesPerElement="2" >
        <DataItem  Format="XML" NumberType="int" Dimensions="4 2">
0 1
1 2
2 3
3 4
        </DataItem>
      </Topology>
      <Geometry GeometryType="XYZ">
```

```
        <DataItem  Format="XML" NumberType="float" Dimensions="5 3">
0.0 0.0 0.0
0.1 0.0 0.0
0.2 0.0 0.0
0.3 0.0 0.0
0.5 0.0 0.0
        </DataItem>
      </Geometry>
      <Attribute Name="dep_x_0" Center="Node"  AttributeType="Scalar"  >
        <DataItem  Format="XML" NumberType="float" Dimensions="1 5">
0.0 1.0 1.2 1.22 1.222
        </DataItem>
      </Attribute>
    </Grid>
  </Domain>
</Xdmf>
```