A more general format, using XML

XML, "Extensible Markup Language", is a language which can be used to transport and store data. It can be used to create markup languages, such as HTML (a language defining how text and images should be displayed). Note that HTML was designed to display data defining the size and position of text for example. XML does not know about layout.

In this XML-example we define our own tags to structure some data.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!-- This is a comment. -->
<!-- The first line is an XML declaration, defining
     version and encoding -->
<course>
<student>
Thomas Ericsson
</student>
<student>
Karin Andersson
</student>
</course>
```

XML is case sensitive, `<student>Thomas Ericsson</Student>` is illegal.
In the following example we use our own attributes as well:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<course>
<student sex="male">   <!-- " " or ' ' -->
Thomas Ericsson
</student>
<student sex="female">
Karin Andersson
</student>
</course>
```


This is all we need to know about XML (but one can learn more). Here comes a simple data file, `xml_ex.vtr` (note `vtr`), in XML-format. The line numbers are not part of the file.
The file describes a `RectilinearGrid` (line 2), like the following Matlab example:

```
x = [-1 2]; y = [2 4]; z = [0 1 4];
[X, Y, Z] = ndgrid(x, y, z);
```

```
1  <?xml version="1.0" encoding="iso-8859-1" ?>
2  <VTKFile type="RectilinearGrid" version="0.1">
3
4   <RectilinearGrid WholeExtent="0 1 0 1 0 2">
5    <Piece Extent="0 1 0 1 0 2">
6
7     <PointData>
8      <DataArray type="Float32" Name="temperature">
9        1 2 3 4 5 6 7 8 9 10 11 12
10     </DataArray>
11    </PointData>
12
13    <Coordinates>
14     <DataArray type="Float32" NumberOfComponents="1">
15       -1 2
16     </DataArray>
17     <DataArray type="Float32" NumberOfComponents="1">
18        2 4
19     </DataArray>
20     <DataArray type="Float32" NumberOfComponents="1">
21       0 1 4
22     </DataArray>
23    </Coordinates>
24
25   </Piece>
26  </RectilinearGrid>
27 </VTKFile>
```

Here is an [Image].

`WholeExtent`, on line 4, gives indices (corresponds to indices in the `x`, `y` and `z`-arrays in the Matlab example). It is possible to have more than one piece, so one particular `Piece Extent` can be a subset of `WholeExtent`.

Lines 7-11 define a temperature, say. Lines 13-23 define what corresponds to the Matlab arrays. It is possible to use additional keywords, and `NumberOfComponets` is not necessary (default one). The indentation is not necessary.

In the following example comes an input file for a structured grid. Think of producing a grid using Matlab's `ndgrid`, and then perturbing the points, but not so much that cells overlap or intersect. In this file I have reproduced the rectilinear grid using a structured grid (which is a waste).

```
1  <?xml version="1.0" encoding="iso-8859-1" ?>
2
3  <VTKFile type="StructuredGrid" version="0.1">
4   <StructuredGrid WholeExtent="0 1 0 1 0 2">
5    <Piece Extent="0 1 0 1 0 2">
6     <PointData>
7      <DataArray type="Float32" Name="temp">
8       1 2 3 4 5 6 7 8 9 10 11 12
9      </DataArray>
10    </PointData>
11
12    <!-- x varies fastest, then y and last z -->
13    <Points>
14     <DataArray type="Float32" NumberOfComponents="3">
15      -1     2     0
16       2     2     0
17      -1     4     0
18       2     4     0
19      -1     2     1
20       2     2     1
21      -1     4     1
22       2     4     1
```



```
23      -1     2     4
24       2     2     4
25      -1     4     4
26       2     4     4
27     </DataArray>
28    </Points>
29   </Piece>
30  </StructuredGrid>
31 </VTKFile>
```

Note that the keyword is `StructuredGrid`, that `NumberOfComponents="3"` and that filename ends in `.vts`.
One can perturb the coefficients and still have a structured grid.

If we perturb the point sufficiently we do not get a structured grid, the points can be in arbitrary positions and we get an unstructured grid. Here comes the house-example again, but this time in XML-format.

```
1  <?xml version="1.0" encoding="iso-8859-1" ?>
2
3  <VTKFile type="UnstructuredGrid" version="0.1">
4   <UnstructuredGrid>
5    <Piece NumberOfPoints="10" NumberOfCells="9">
6
7     <PointData>
8      <DataArray type="Float32" Name="temperature">
9          11 12 13 14 15 16 17 18 19 20
10     </DataArray>
11    </PointData>
12
13    <Cells>
14     <DataArray type="Int32" Name="connectivity">
15         6  2  1  0   6  1  4  0   6  2  3  1
16         6  3  7  1   6  5  4  1   6  7  5  1
17         9  7  5  4   9  6  4  8   9  6  7  4
18     </DataArray>
19
```

```
      <DataArray type="Int32" Name="offsets">
       4   8  12  16  20  24  28  32  36
      </DataArray>

      <DataArray type="UInt32" Name="types">
         10 10 10 10 10 10 10 10 10
      </DataArray>
     </Cells>

     <Points>
      <DataArray type="Float32" NumberOfComponents="3">
         0 0 0   2 0 0   0 1 0    2 1 0
         0 0 1   2 0 1   0 1 1    2 1 1
         0 0.5 1.5  2 0.5 1.5
      </DataArray>
     </Points>

    </Piece>
   </UnstructuredGrid>
  </VTKFile>
```

The `offsets`-array contains the indices into the `connectivity`-array for the *end* of each cell. For some reason the offsets start at one (and not zero).

Finally an animation example, a cube that moves to the right changing colour at the same time. We store a sequence of frames, using a rectilinear format, one frame in each file. The file `animation.pvd` is a main file referring to the frame-files. In a real application we would probably have more frames (50-100 say).


```
<?xml version="1.0"?>
<VTKFile type="Collection" version="0.1">
 <Collection>
  <DataSet timestep="1" file="1.vtr"/>
  <DataSet timestep="2" file="2.vtr"/>
  <DataSet timestep="3" file="3.vtr"/>
  <DataSet timestep="4" file="4.vtr"/>
  <DataSet timestep="5" file="5.vtr"/>
 </Collection>
</VTKFile>
```

Here is the first frame-file, `1.vtr`:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<VTKFile type="RectilinearGrid" version="0.1">

 <RectilinearGrid WholeExtent="0 1 0 1 0 1">
  <Piece Extent="0 1 0 1 0 1">
   <PointData>
    <DataArray type="Float32" Name="temp">
     1 1 1 1 1 1 1 1
    </DataArray>
   </PointData>
   <Coordinates>
    <DataArray type="Float32"> 0 1 </DataArray>
    <DataArray type="Float32"> 0 1 </DataArray>
    <DataArray type="Float32"> 0 1 </DataArray>
   </Coordinates>
  </Piece>
 </RectilinearGrid>
</VTKFile>
```

in the next frame frame, I change `temp` and the x-coordinates.

More about OpenDX, in case you are interested

OpenDX, `www.opendx.org`, is an open version of IBM's "Visualization Data Explorer". A similar, but not open, system is AVS (Advanced Visual Systems, `www.avs.com`). ParaView, see the assignments, share many of the ideas, as well.
Not quite: "If you have seen one, you have seen them all".

OpenDX is exclusively for visualization (no computational part as in Matlab. Very simple calculations are OK.).

Some, but not all, important points:

- Advanced tools for visualization of 3D-data.
- Takes longer to learn than Matlab, but you can do more. Often faster.
- Modules are connected using a GUI, graphical programming. Visual Program Editor, VPE.
- Input from files (not variables as in Matlab).
- The modules transform the input and sends it to the next module.
- Supports several input formats. Using the "Data Prompter"-program simple inputs can be handled (e.g. uniform, gridded input).
- Lots of documentation. Many demo programs. Few simple examples. Should read a book (or take this course :-)
  David Thompson, Jeff Braun, Ray Ford,
  OpenDX: Paths to Visualization. Consists of a sequence of solved visualization problems.
  `http://www.vizsolutions.com`.

Here comes a brief presentation of OpenDX. For more details see the "QuickStart Guide" (see the course page).