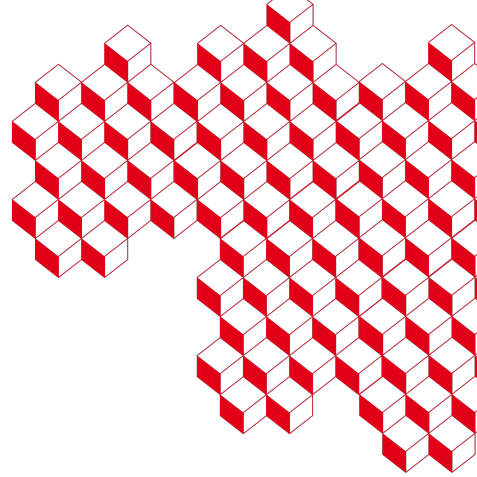


MEDLoader - Reading and writing MED Files

08-03-2024

Aymeric SONOLET, Guillaume BROOKING

Outline

Agenda

Agenda

- Overview of the MEDLoader, fundamental differences with MEDCoupling model
- Basic API
  - Quick export to VTU (visualization)
  - Reading and writing a MED file
- Advanced API
  - Some insights into the MED file format
  - Advanced concepts
    - Groups
    - Profiles

Outline

MEDLoader, from a memory model to a file

- The tool to save / load what you do in memory with MEDCoupling

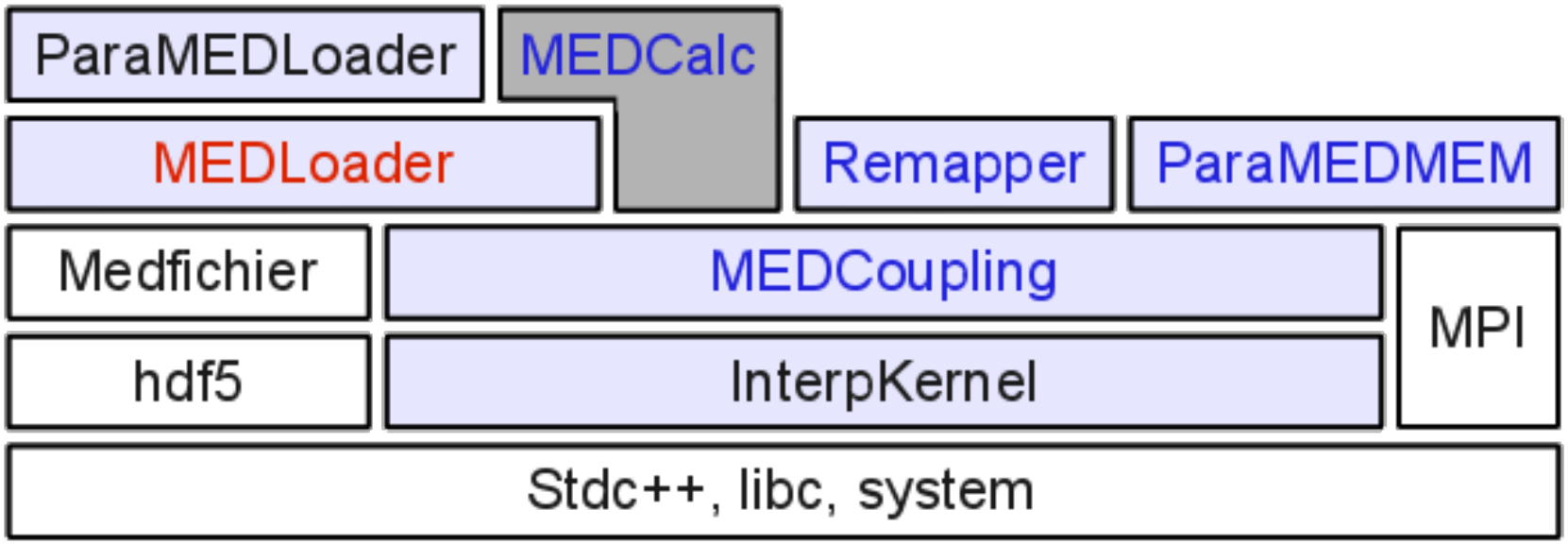


Figure 1: MEDLoader in the MEDCoupling distribution

MEDLoader, not only a “loader”

- Why do we need a loader/writer?
  - Communicate between processes / tools
  - Load / backup results
- Very unfortunate name
  - Can obviously read, but also write meshes and fields . . .
  - . . . into a file with the MED type (“.med” extension): “Modèle Echange de Données” (Data Exchange Model)
- Part of the MEDCoupling library, requires:
  - MED-file library (currently version 3.0.8) to compile and run
    - MED-file is the low level C/Fortran API to deal with MED files
  - HDF5 (prerequisite of MED-file library)
  - MPI if you’re working in parallel
- You can use the core structures of MEDCoupling without MEDLoader
  - The reverse does not work
- Like before, most of it in C++, but wrapped in Python
  - Most of the algorithms are actually pure MEDCoupling core

MEDLoader Services

- Read / write
- Groups manipulation
  - A collection of entities on the mesh (volumes, faces, points . . .)
- Geometric algorithms
  - `convertAllToPoly()`
  - `unPolyze()`
  - `zipCoords()`
  - `duplicateNodes()`
  - `...`
- Those services often have a MEDCoupling equivalent
- MED file conversion tools
  - Between MED versions, from SAUV, . . .

Differences with MEDCoupling, more flexibility

- **Warning:** `MEDFilexxx` $\neq$ `MEDCouplingxxx`
  - MEDCoupling tries to rationalize the large flexibility provided by the MED-file format
  - Loosing some advanced functionalities, but more *user-friendly*
- Capabilities of MED-file that are “restricted” in MEDCoupling:
  - A *mesh*:
    - can have multiple dimensions
    - can have groups
  - A *field*:
    - can be defined only on a part of the mesh: “profiles”
    - can have more than one spatial discretization (ON_CELLS, ON_NODES, . . .)
  - A MEDFile field can have multiple time-steps (`MEDCouplingFieldDouble` = single step)
    - Solution: use multiple `MEDCouplingFieldDouble`
    - Solution: use `MEDCouplingFieldOverTime`
- *Rule of thumb:* try to do it with the `MEDCoupling` data model only

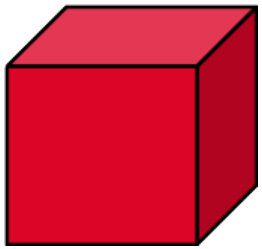
Differences with MEDCoupling, some constraints

- “Objects” need to have a name
  - *Mesh* and *fields* need to be named
    - `setName()` method
  - MED-file has a max length of 64 characters
- *Cells* need to be ordered by *geometric type*
  - E.g. if you mix TRIA3 and QUAD4 in your mesh (sometimes called a hybrid mesh)
  - Invoke `sortCellsInMEDFileFrmt()`
  - Try to keep cell types ordered if you can (will avoid save/load time)
  - Or convert everything to polyhedrons!
    - `MEDCouplingUMesh::convertAllToPoly()`
    - Only one cell type
    - Can be converted back with `unPolyze()`

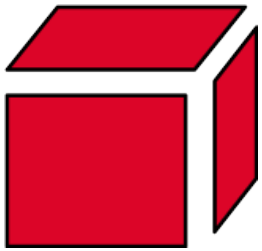
MEDFile - Mesh Dimension

Up to 4 dimensions for a single `MEDFileUMesh`

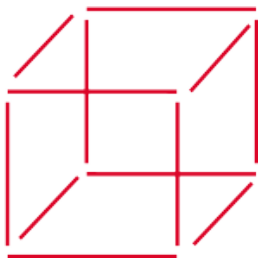
- Start point = most upper level – here 3D
- `levelRelativeToMax = 0` (volumes)



- `levelRelativeToMax = -1` (faces)



- `levelRelativeToMax = -2` (edges)



Outline

Basic API - Overview (1/2)

I need a quick look

- `VTK/VTU` export

```
m = <some mesh/field I just created>
m.writeVTK("/tmp/foo.vtu")
```

- Visualization with `ParaView`

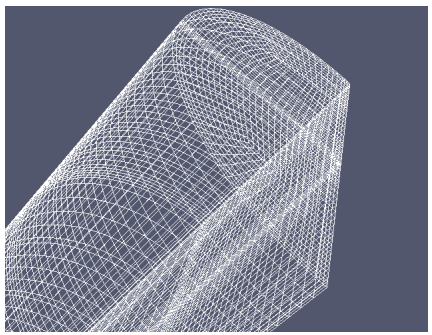


Figure 2: A vtu mesh

Basic API - Overview (2/2)

A step further – the basic `MEDLoader` API

- Directly reading/writing a mesh to a file on disk
  - `WriteMesh()` / `ReadMesh()`
- Directly reading/writing a field
  - `WriteField()` / `ReadField()`
  - `WriteFieldUsingAlreadyWrittenMesh()` if you have several fields on a single mesh
- All dealing with MEDCoupling objects (e.g. MEDCouplingUMesh, . . .)
- Only static methods (i.e. no internal state). File reopened each time!

Basic API - Reading a multi-dimensional mesh

- Reading 3 levels of meshes (volumes, faces, vertex)

```
import medcoupling as mc
medFile = 'file.med'
meshName = 'mesh'
mesh3D = mc.ReadUMeshFromFile(medFile, meshName, 0)
mesh2D = mc.ReadUMeshFromFile(medFile, meshName, -1)
mesh1D = mc.ReadUMeshFromFile(medFile, meshName, -2)
```

- Works fine, but:
  - The 3 meshes have 3 independent coordinate arrays
  - Could be *shared* – see later slide on mesh dimension . . .

Outline

Advanced API - Introduction

Use cases

- Dealing with *multiple time-steps*
- Dealing with *multiple mesh dimensions*
  - Typically a volumic mesh (`space dim == mesh dim == 3`)
  - And some boundary conditions on the faces (`mesh dim = 2`)
  - The two mesh share the same `nodes`
- Dealing with mesh *groups*
  - A group is a named set of cells in a given mesh
  - Frequently used for *boundary conditions assignment*
- Dealing with partial support (rare case): *profiles*

Advanced API – Class diagramm

Helps navigate the advanced API

- All names prefixed with % actually start with `MEDFile` (e.g. `MEDFileData`)

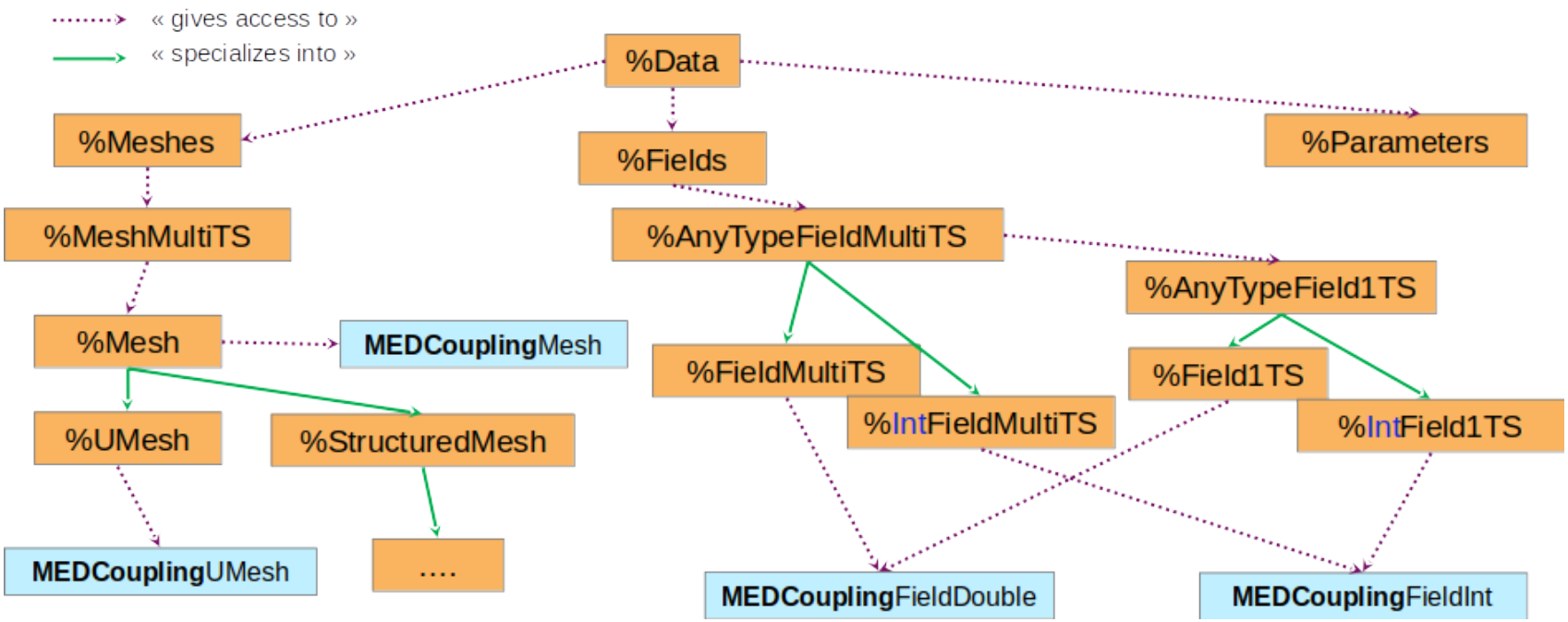


Figure 3: Links with the `MEDCoupling` data model

Advanced API – Important options

Writing a file

- Write options:
  - 2: force writing from scratch (an existing file will be reset)
  - 1: append mode (no corruption risk if file already there).
  - 0: overwrite mode: if the file and the MED object exist, they will be overwritten, otherwise write from scratch

Reading a field (`MEDFileField1TS`, `MEDFileFieldMultiTS`)

- Read *everything* in the file by default
- For large size fields:
  - Set the boolean `loadAll` to `False` in constructors
  - Use `loadArrays()` or `loadArraysIfNecessary()` to load data on demand
  - Use `unloadArrays()` or `unloadArraysWithoutDataLoss()` to free memory

Advanced API - Example

Example

```
medFile = "myWrittenFile.med"
coo = mc.DataArrayDouble([...])
mesh1D = <MEDCoupling object>
mesh1D.setCoords(coo)
mesh2D = <MEDCoupling object>
mesh2D.setCoords(coo)
mesh3D = <MEDCoupling object>
mesh3D.setCoords(coo)
mesh = mc.MEDFileUMesh.New()
mesh.setName("myMesh")
mesh.setMeshAtLevel(0, mesh3D)
mesh.setMeshAtLevel(-1, mesh2D)
mesh.setMeshAtLevel(-2, mesh1D)
mesh.write(medFile, 2)
```

Better than basic API

- All 3 meshes share the same coordinates array
- Saving space and write/load time
- Ensure consistency

Basic/Advanced API - Summary

Basic API is well suited for

- Meta information of a MED-file, without loading everything
  - E.g.: `GetMeshNames`, `GetComponentsNamesOfField`, `GetFieldIterations`
- Reading / Writing single instances of MEDCoupling objects
- Reading / Writing MEDCoupling meshes (without groups nor families)
- Reading / Writing MEDCouplingFieldDouble (without profiles)

You need the advanced API to

- Handle (part of) a MED-file in memory (e.g. to rewrite part of it only, a single time-step, . . .)
- Deal with families and groups
- Deal with field profiles

Outline

Groups and families definition

Families

- **Families do not need to be manipulated directly in normal usage !**
- In a mesh cells are partitioned by *families*
- Each cell has a unique family ID (reverse not true)
- MEDLoader advanced API gives access to families
  - `getFamilies()`, `getFamiliesArr()`, ...

Groups

- A group is a list of families
- MEDLoader advanced API gives access to groups
  - `getGroups()`, ...

Families and Groups - Illustration

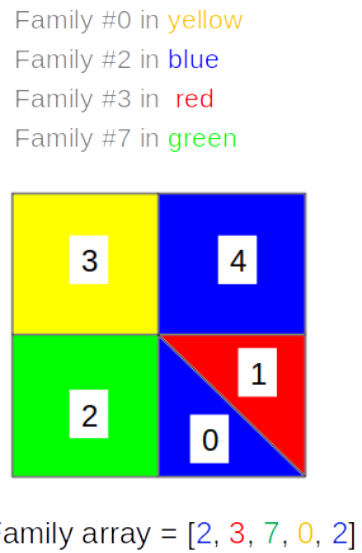


Figure 4: Families and groups example

Table 1: Families and cells associated to groups

Group A	Group B
Families 2 & 3	Families 7 & 3
Cells 0, 1, 4	Cells 2, 1

Recording a Group

Example

```
myCouplingmesh = ...

tabIdCells = mc.DataArrayInt()
tabIdCells.setName("meshGroup")
tabIdCells.setValues(...)

mfum = mc.MEDFileUMesh()
mfum.setName("Name")
mfum.setDescription("description")
mfum.setCoords(myCouplingmesh.getCoords())
mfum.setMeshAtLevel(0, myCouplingmesh)
mfum.setGroupsAtLevel(0, [tabIdCells])
```

Reading a Group

Example

```
mfum = mc.MEDFileUMesh(fname)
gpNames  = mfum.getGroupsNames()
# mc.DataArryInt
myGroupArr = mfum.getGroupArr(0, gpNames[0])
# mc.MEDCouplingUMesh
myGroup = mfum.getGroup(0, gpNames[0])
```

Either get the result:

- as a `DataArrayInt`
- as a sub-mesh

Outline

Conclusion

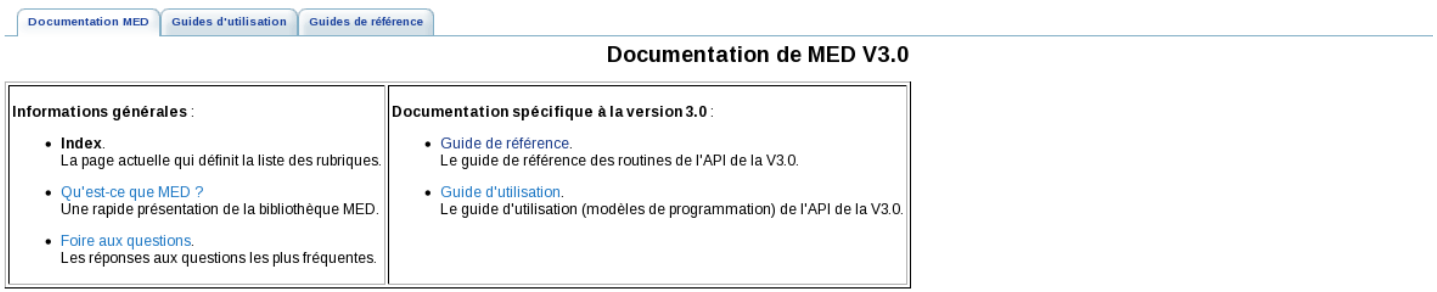
- Any question ?
- Let's get ready for the exercises!

Outline

Advanced API – MED-file format (1/2)

What does a MED file look like?

- A specialization of the HDF5 standard
  - Take a look at http://www.hdfgroup.org/HDF5/
  - Focus on (potentially) big amount of data
  - Parallelism in mind
- Official documentation of the MED-file format available on-line at
  - `${MEDFILE_ROOT_DIR}/share/doc/html/index.html` (in French)
- Reminder: *MED-file* denotes both
  - A file format on disk (I/O)
  - A comprehensive low level C library to read/write MED files
    - Now with a Python wrapping
    - Author: Eric FAYOLLE (EdF R&D)
  - MED-file existed before MEDCoupling!



Advanced API – MED-file format (2/2)

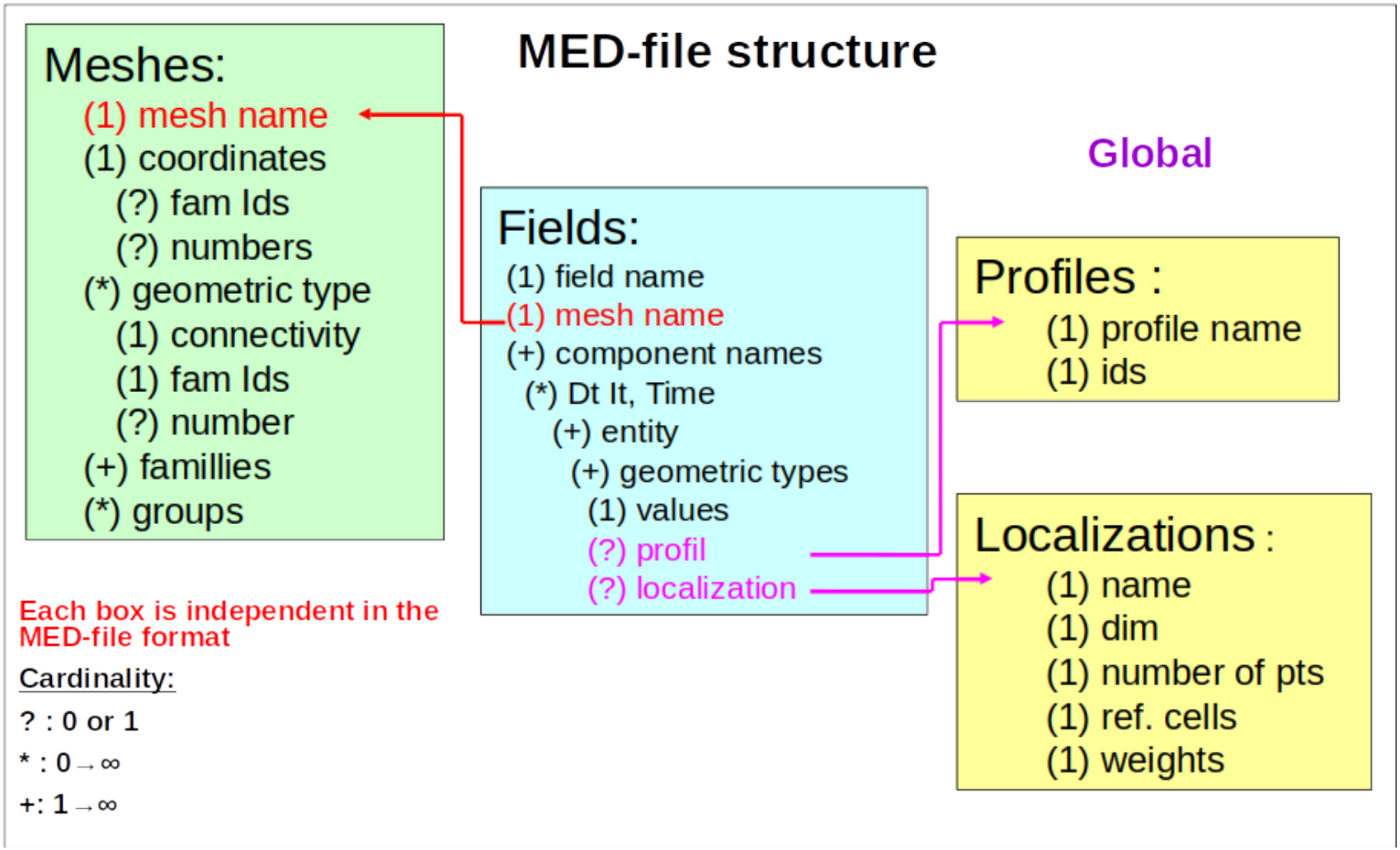


Figure 5: alt text