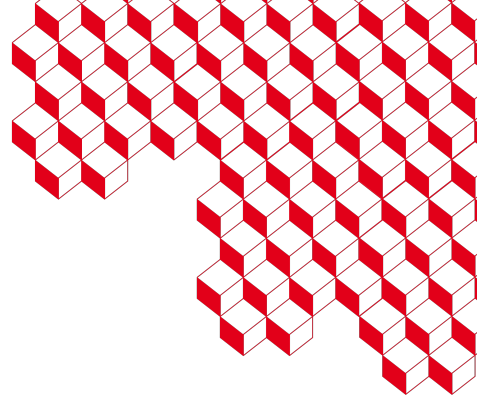
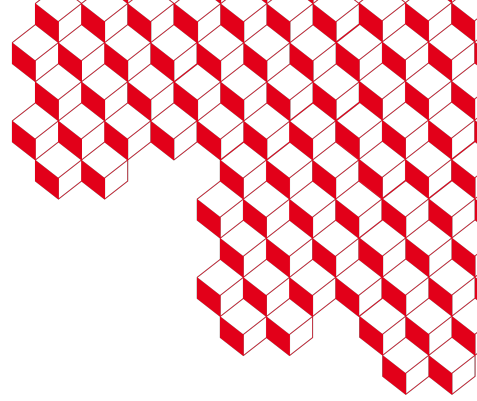


MEDLoader - Reading and writing MED Files

08-03-2024

Aymeric SONOLET, Guillaume BROOKING



MEDLoader - Reading and writing MED Files

08-03-2024

Aymeric SONOLET, Guillaume BROOKING

Outline

Agenda

MEDFile/MEDLoader Overview

Basic API

Advanced API

Appendix

Agenda

- Overview of MEDLoader
- 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

Agenda

MEDFile/MEDLoader Overview

Basic API

Advanced API

Appendix

MEDLoader, from a memory model to a file

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

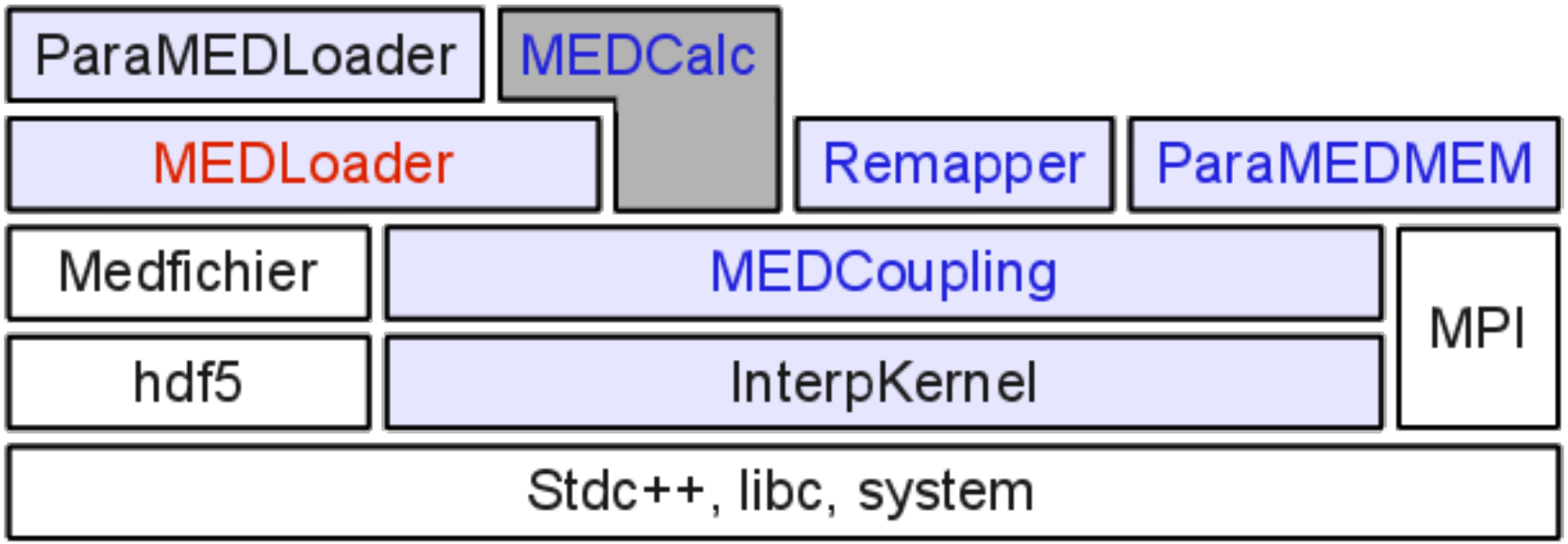


Figure 1: MEDLoader in the MEDCoupling distribution

MEDLoader, not only a “loader”

- **Read/write** meshes and fields from/to **files**
- Target `MED` and `VTK (write only)` formats
- Distinct from core functionnalities of the MEDCoupling library
- Dependencies
  - MEDfile library (low level C/Fortran API, manipulate `.med` files)
  - HDF5
  - MPI (parallel only)

MEDLoader Services

- Read/write in `MED` format
- (limited) Write to `VTK` format
- Geometric algorithms
  - `convertAllToPoly()`
  - `unPolyze()`
  - `zipCoords()`
  - `duplicateNodes()`
  - `...`
- Groups manipulation
  - A collection of entities on the mesh (volumes, faces, points . . .)
- MED file conversion tools
  - Between MED versions, from SAUV, . . .

Differences with MEDCoupling, more flexibility

- `MEDCoupling<Something>` **vs.** `MEDFile<Something>`
  - user-friendly **vs.** capabilities
- Capabilities that requires to use **MEDFile**:
  - A **mesh**:
    - can have `groups` : named collection of entities
    - can have entities of various dimensions
  - A **field**:
    - can have `profiles` : definition only on a part of the mesh
    - can have multiple time-steps (`MEDCoupling` -> single step)
    - can have more than one spatial discretization (`ON_CELLS`, `ON_NODES`, `...`)
  - Tips to stay with `MEDCoupling`:
    - use multiple `MEDCouplingFieldDouble`
    - use `MEDCouplingFieldOverTime`

Rule of thumb: try to do it with the `MEDCoupling` **first**

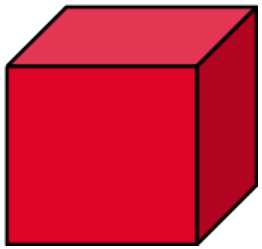
Differences with MEDCoupling, some constraints

- **Mesh** and **fields** are required to be **named** (e.g. `setName()`)
  - With MEDfile, names have a max length of `64`
- **Cells** need to be ordered by **geometric type**
  - Use case: mix of `TRIA3` and `QUAD4` in a mesh (“hybrid mesh”)
  - Related method: `sortCellsInMEDFileFrmt()`
  - Keeping cell types ordered avoids read/write time
- Possible workaround: convert everything to **polyhedrons**
  - Related method: `convertAllToPoly()`
  - Only one cell type (polyhedron)
  - Can be converted back with `unPolyze()`

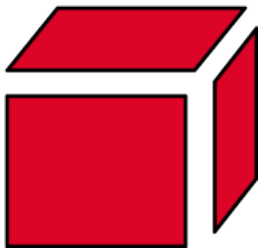
MEDFile - Mesh Dimension

Up to 4 dimensions for a single `MEDFileUMesh`

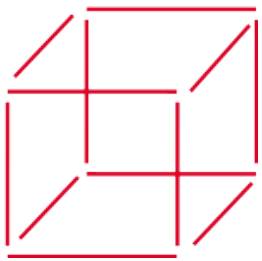
- Start point = most upper level – here 3D
- `levelRelativeToMax = 0` (volumes)



- `levelRelativeToMax = -1` (faces)



- `levelRelativeToMax = -2` (edges)



Outline

Agenda

MEDFile/MEDLoader Overview

Basic API

Advanced API

Appendix

Basic API - Overview (1/2)

Simple VTK export

- `VTK` write

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

- Visualization with `ParaView`

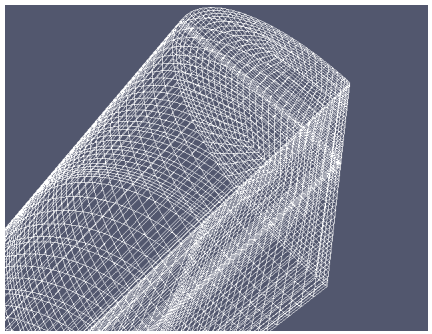


Figure 2: A vtu mesh

- (read is not implemented)

Basic API - Overview (2/2)

A step further - the basic `MEDLoader` API

- reading/writing a **mesh** to a file
  - `WriteMesh()` / `ReadMesh()`
- reading/writing a **field**
  - `WriteField()` / `ReadField()`
  - `WriteFieldUsingAlreadyWrittenMesh()` (several fields, a single mesh)
- **MEDCoupling objects** -> restrictions
- User-friendly : 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
mesh3D = mc.ReadUMeshFromFile("file.med", "mesh", 0)
mesh2D = mc.ReadUMeshFromFile("file.med", "mesh", -1)
mesh1D = mc.ReadUMeshFromFile("file.med", "mesh", -2)
```

- Limitation: the 3 meshes have 3 **independent coordinate arrays**

Outline

Agenda

MEDFile/MEDLoader Overview

Basic API

Advanced API

Appendix

Advanced API - Introduction

Use cases

- multiple time-steps
- multiple mesh dimensions
  - example: a volumic mesh (`space dim == mesh dim == 3`)
  - with boundary conditions on the faces (`mesh dim = 2`)
  - The two mesh share the same `nodes`
- using **groups** (e.g. boundary conditions)
  - A group is a **named collection of cells**
  - Frequently used for **boundary conditions**
- Dealing with partial support: **profiles**

Advanced API – Class diagramm

Helps navigate the advanced API

- Here, % is shortcut for `MEDFile` (e.g. `%Data` -> `MEDFileData`)

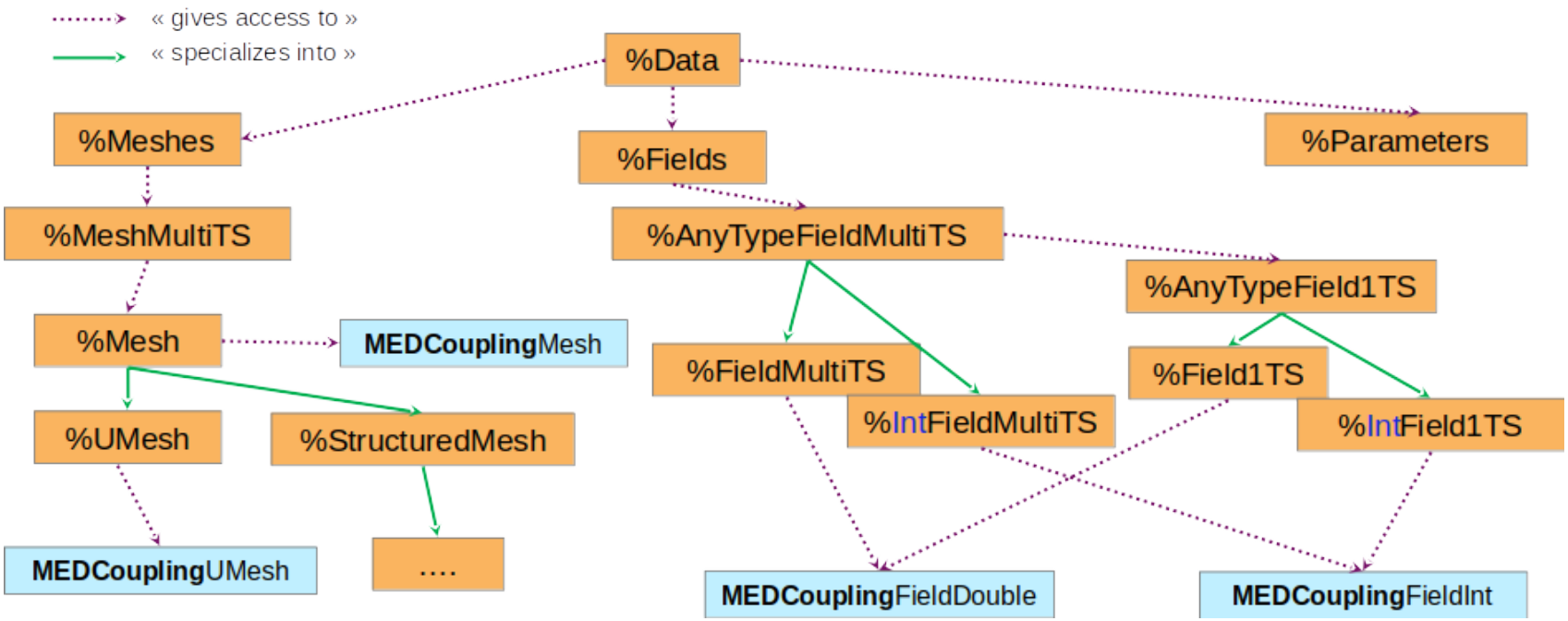


Figure 3: Links with the `MEDCoupling` data model

Advanced API – Important options

Writing a file

- Three **write modes**:
  - `2`: force writing from scratch (an existing file will be reset)
  - `1`: appends to existing file (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
- For example:

```
mesh: MEDFileUMesh
mesh.write("mesh.med", 2)
```

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

- Get meta-info 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

Groups

- A group is a named collection of entities.
- MEDLoader advanced API gives access to groups:
  - `getGroups()`, ...
- The underlying implementation of groups use the concept of families.

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).
- Groups are in fact just a collection of families (not of cells).
- MEDLoader advanced API gives access to families:
  - `getFamilies()`, `getFamiliesArr()`, `...`
- Families allow for efficient boolean operations between groups.

Illustration

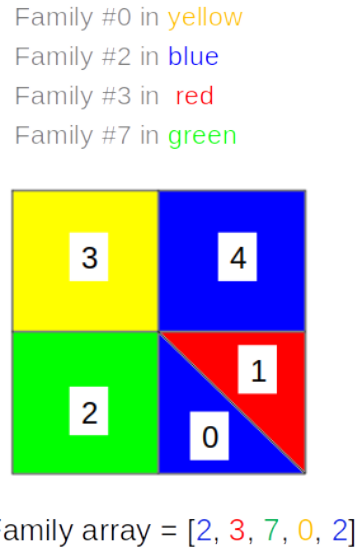


Figure 4: Families and groups example

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

Time-steps

A **field** can have none, one, or multiple **time-step**. Each time-step is a triplet `(time, timestep, sub-timestep)`, where `timestep, sub-timestep` is a unique identifier of the timestep itself.

WARNING: `time` is just an annotation. It is not used to identify the time step.

For example, writing a field at timestep `(4.54, 12, 3)` override the data written at timestep `(3.12, 12, 3)`.

Outline

Agenda

MEDFile/MEDLoader Overview

Basic API

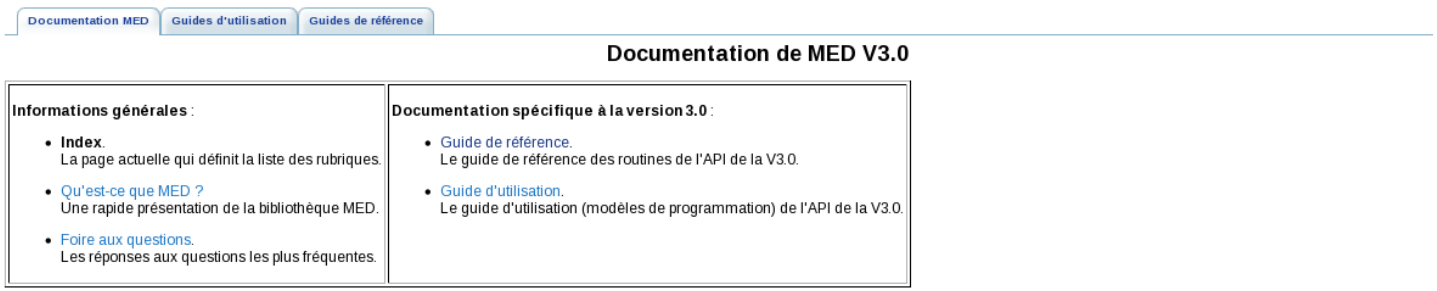
Advanced API

Appendix

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)

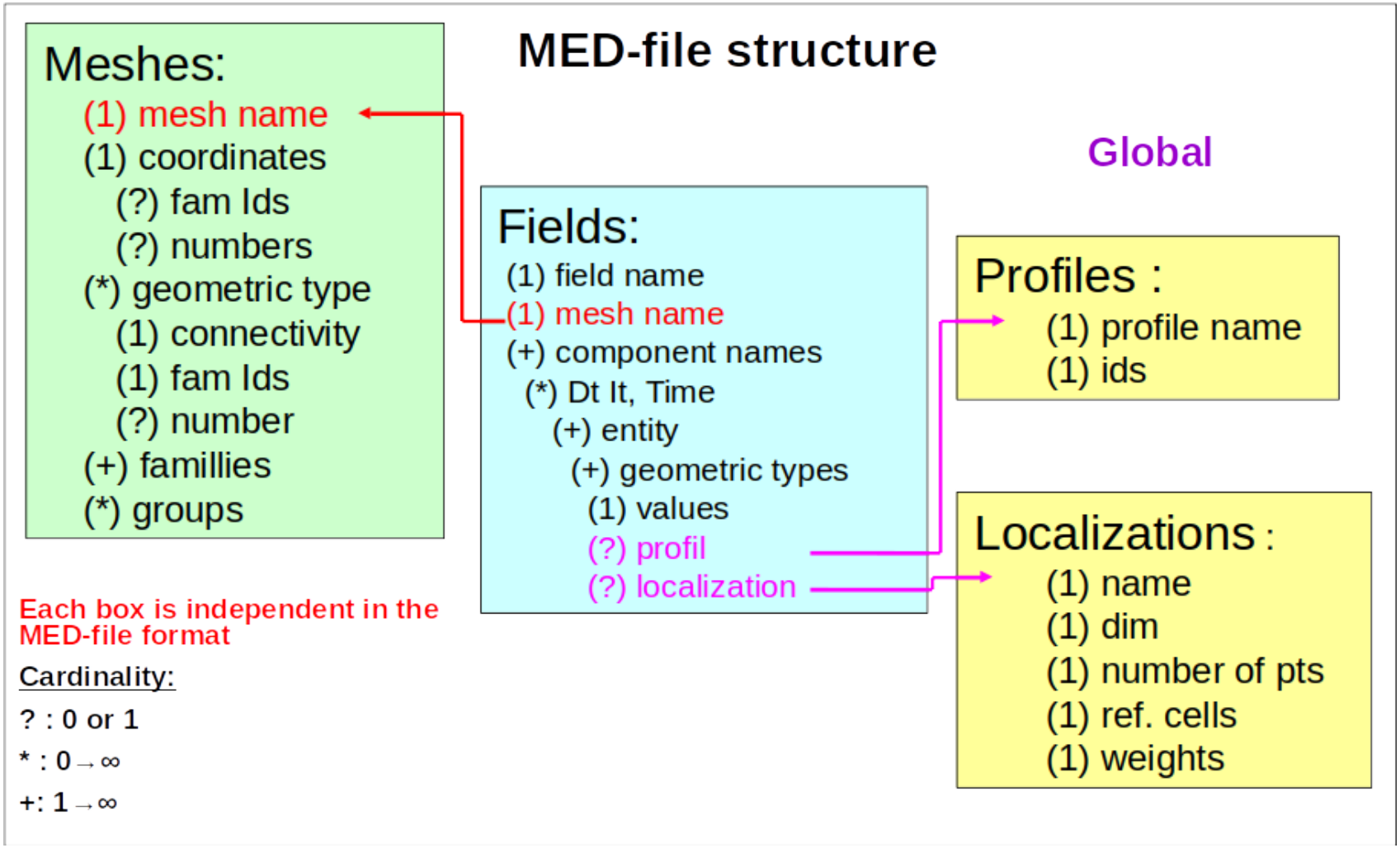


Figure 5: alt text