Amelet-HDF Documentation

Release 1.5.4

Cyril Giraudon

October 15, 2015

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

The purpose of the **Amelet HDF** Specification is to provide a standard for recording and recovering computer data of electromagnetic simulations.

A few well established data format already exist, we can quote :

- CGNS (CFD General Notation System, http://cgns.sourceforge.net/) is recommended for computational fluid dynamics problems (CFD). That's why a lot of electromagnetic concepts are missing
- SILO (a mesh and field I/O library and scientific database https://wci.llnl.gov/codes/silo/)
- MED (in french "Modélisation et Échanges de Données" http://www.code-aster.org/outils/med/), it is the SALOME's data format (http://www.salome-platform.org/home/presentation/overview/). Like CGNS, SALOME provides tools for CFD simulation and aren't adapted to the electromagnetism domain
- netCDF (http://www.unidata.ucar.edu/software/netcdf/)

Besides, **Amelet HDF** Specification is closely related to the Quercy Platform. Quercy is a software platform aiming at :

- Providing knowledge management capabilities
- Providing pre and post processing tools
- Integrating scientific softwares
- Managing software execution

CGNS, SILO and MED specifications give a standard way of describing physical models, mesh definition and many physical concepts. Softwares that can express their native data in CGNS vocabulary are good "customers" for the specification, for data that can not be correctly expressed CGNS offers a "user variable" notion. But for other softwares handling complex structures stranger to the specification, there is no way to use the existing format, even by part.

Quercy Platform expresses data in the form of objects named "infotype", it is possible to add new infotype to the platform and this infotype have to be converted into equivalent concepts into **Amelet HDF**. So, **Amelet HDF** must be sufficiently flexible to express unknown data coming from the new infotypes. This can not be done with CGNS or MED.

The manner **Amelet HDF** accomplishes this task is described is the document.

Basically, **Amelet HDF** specification can express all sort of electromagnetic data, the most important are :

- Mesh (unstructured or structured);
- Numerical array data;
- Material models
- Network and transmission line;
- Electromagnetic sources

This document covers all the aspects of the **Amelet HDF** specification.

Note: **Amelet HDF** is largely inspired from the Amelet project (http://www.predit.prd.fr/predit3/syntheseProjet.fo?inCde=22740)

CHAPTER

TWO

HDF5 FORMAT

2.1 Introduction

Like many scientific data formats (CGNS, MED, SILO), **Amelet HDF** is based upon HDF5.

HDF5 (http://www.hdfgroup.org/HDF5) is a very flexible file format, that is developed by the hdfgroup.

According to the web page: *“HDF5 is a unique technology suite that makes possible the management of extremely large and complex data collections.*“

The main features of HDF5 are :

- The data model can represent very complex data objects
- A portable data format
- A library that runs on all platform and implements a high-level API with C, C++, Fortran and Java interfaces
- Access time and storage optimization
- Tools for viewing the data collection
- A complete documentation and set of examples (tests) for all languages

XML would be a really good candidate to express such a data, queries can be performed by technologies like XPath. Unfortunately, **Amelet HDF** aims at to be scalable, portable and cross language and there is no XML solution for the Fortran world to read/write XML documents.

2.1.1 Editing tools

Furthermore, the hdfgroup provides tools to view or manipulate HDF5 files :

- `hdfview` http://www.hdfgroup.org/hdf-java-html/hdfview/ can :
  - view a file
  - create new file
  - modify the content
  - modify attributes
- `gif2h5` - Converts a GIF file into HDF5
- `h5import` - Imports ASCII or binary data into HDF5
- `h5diff` - Compares two HDF5 files and reports the differences
- `h5repack` - Copies an HDF5 file to a new file with or without compression/chunking

- `h52gif` - Converts an HDF5 file into GIF
- `h5cc, h5fc, h5c++` - Simplifies compiling an HDF5 application
- `h5dump` - Enables the user to examine the contents of an HDF5 file and dump those contents to an ASCII file
- `h5jam/h5unjam` - Add/Remove text to/from user block at the beginning of an HDF5 file.
- `h5ls` - Lists selected information about file objects in the specified format
- `h5repart` - Repartitions a file or family of files
- `h5copy` - Copies objects to a new HDF5 file
- `h5mkgrp` - Makes a group in an HDF5 file
- `h5stat` - Displays object and metadata information for an HDF5 file

The python world offers a very good editing tools of HDF5 documents :

- h5py : "*The HDF5 library is a versatile, mature library designed for the storage of numerical data. The h5py package provides a simple, Pythonic interface to HDF5. A straightforward high-level interface allows the manipulation of HDF5 files, groups and datasets using established Python and NumPy metaphors.*" (http://h5py.alfven.org/)
- pytables is python module to handle HDF5 format as pyh5 does (http://www.pytables.org/moin)
- vitables (http://vitables.berlios.de) is based upon the python pytables module (http://www.pytables.org/moin), is a graphical interface to pytables

Languages for technical computing have also some HDF5 capabilities :

- Matlab provides capabilities to read/write HDF5 with the functions `hdf5info`, `hdf5read` and `hdf5write`.

2.1.2 Data organization

An HDF5 file is hierarchicaly organized like a file system (there are directories and files), the main kinds of objects are :

- Group. It looks like a directory in a file system. It can contain other objects.
- Dataset. It represents a multi-dimension typed matrix an is contained in a group as a file is contained by a directory in a file system.
- Table. It is a special dataset and represents multi-column data.

Each object is located by an absolute or relative path from the root node or from another node.

Each object can be described by attributes, an attribute is a pair key, value. The value of an attribute can be one of all HDF5 supported types : integer, real, boolean, string.

A file can then be represented by a tree structure like directories and files in a file system explorer tool. Group are directories and datasets (and tables) are files :

```
data.h5/
|-- dataset1[@type=a_type]
|-- table1
|-- group1
|    |-- dataset2
|    |-- dataset3
|    |-- table2
|    |-- group2
|    |    |-- table3
|    |    `-- table4
|    `-- table5
|-- table2
`-- dataset3
```

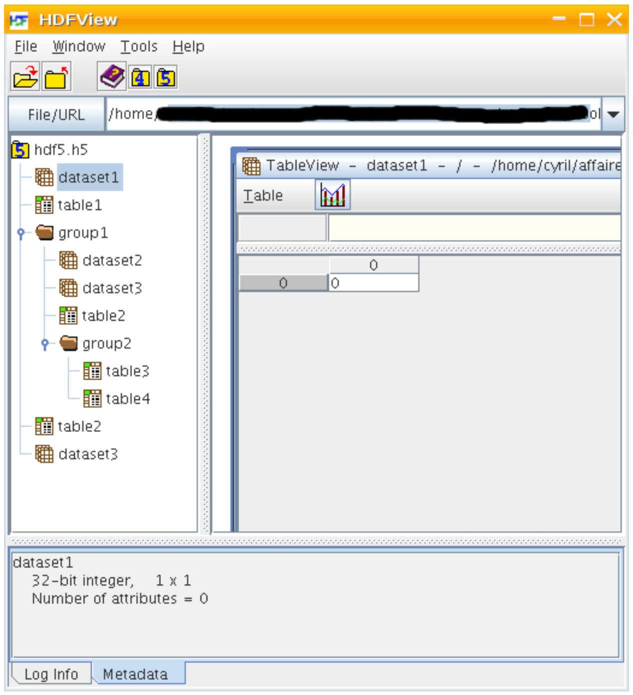
The h5 extension is often associated to HDF5 files. Elements are localized by their absolute path from the root or by their relative path from the parent group, for instance :

- `/group1/group2/table3` is a valid absolute path to reach table3 in group2 in group1
- `group2/table3` is a valid relative path to reach table3 from `/group1`

Therefore, two elements can have the same name if they have not the same parent, `/dataset3` and `/group1/dataset3` can coexist in an HDF5 file.

In this document, attributes of HDF5 elements are represented like XML attributes, they are preceding by `@` and they are all inside square brackets, no quotes are used around the value.

All HDF5 examples can be opened with hdfview (version 2.4), the preceding example opened with it is presented below :



```
dataset1
  32-bit integer,    1 x 1
  Number of attributes = 0
```

Fig. 2.1: HDFView main window

2.2 HDF5 modules

There are two versions of HDF5 in production :

- the version 1.6, the last release is 1.6.4
- the new version 1.8, the last release is 1.8.2. The main feature that comes with the version 1.8 is the Lite API : "The HDF5 Lite API consists of higher-level functions which do more operations per call than the basic HDF5 interface. The purpose is to wrap intuitive functions around certain sets of features in the existing APIs. This version of the API has two sets of functions: dataset and attribute related functions."

The HDF5 format can be read and writen from a library that is also developed by the hdfgroup, this library can be downloaded from http://www.hdfgroup.org/HDF5/release/obtain5.html.

Note: Since **Amelet HDF** specification is dedicated to scientific applications, examples will be given in Fortran language and sometimes in C language.

Amelet HDF can be read almost thanks to the API Lite.

First of all, to manipulate an HDF5 file, the modules which have to be loaded are:

(see example1.f90)

```fortran
! The HDF5 API
use hdf5
! The lite API
use h5lt
```

2.3 Open and close a file

The first step is the initialization of the HDF5 library, then we can open a file :

(see example2.f90)

```fortran
! Variable declaration
character(len=*), parameter :: filename = "data.h5"
integer(hid_t) :: file_id
integer :: hdferr

! Library initialization (native type reading)
call h5open_f(hdferr)

! Generally, if hdferr is negative a problem occured
if (hdferr < 0) then
    print *, "h5open_f, KO"
end if

! Open a file
call h5fopen_f(filename, H5F_ACC_RDONLY_F, file_id, hdferr, H5P_DEFAULT_F)
```

- `H5F_ACC_RDONLY_F` is an HDF5 constant indicating the file is opened in the read only mode
- `file_id` is the file identifier returned by HDF5
- `hdferr` is the error code returned by the function

Note: Take care at the unfamiliar `hid_t` type of `file_id`, fortran type kind must be respected

Finaly, close the file:

```fortran
! Close filename file
call h5fclose_f(file_id, hdferr)
```

As we can see, in Fortran, the last argument is always `hdferr` or whatever integer variable. This argument is the return error code of HDF5 functions. If `hdferr` is negative something went wrong.

It's a good habit to check ''hdferr'' value., though for the sake of clarity it is last time we perform checks in the examples.

2.4 The HDF5 lite API

Amelet HDF is designed to be easily readable by a person. This legibility is found again at source code level. In order to aid in performing this task, HDF5 provides an API for higher-level functions which do more operations per call than the basic HDF5 interface, therefore it becomes straightforward to walk through an **Amelet HDF** file.

For instance, it is possible to read the number of records and the number of fields of a table with a single function :

```fortran
! Table's name
character(len=*), parameter :: table_absolute_name = "/a_table"
! Number of columns (fields) in a table
integer(hsize_t) :: nfields
! Number of rows (records) in a table
integer(hsize_t) :: nrecords
! Error code
integer :: hdferr

call h5tbget_table_info_f(file_id, table_absolute_name, &
                          nfields, nrecords, hdferr)
```

Amelet HDF can be almost entirely read with the lite API, used functions are presented in the next section.

2.4.1 Query for table's information

It is possible to get table's information with the function `h5tbget_table_info_f`. The function returns :

- The number of columns (fields) of a table
- the number of rows (lines) of a table.

(see read-table.f90)

The signature of `h5tbget_table_info_f` is :

```fortran
! The parent id
integer(hid_t) :: loc_id
! Table name
character(len=*), parameter :: table_name = "/a_table"
! Number of columns (fields) in a table
integer(hsize_t) :: nfields
! Number of rows (records) in a table
integer(hsize_t) :: nrecords
! Error code
integer :: hdferr

call h5tbget_table_info_f(file_id, table_name, nfields, nrecords, hdferr)
```

2.4.2 Read the records of a table

Table's records can be read with the function `h5tbread_field_name_f`. **It takes an already allocated buffer** and returns :

- the buffer containing the read values of the named column

(see read-table.f90)

```fortran
! The file id
integer(hid_t) :: file_id
! Table name
character(len=*), parameter :: table_name = "/a_table"
! The field's name to be read
character(len=*), parameter :: field_name = "a_field"
! the reading start row
integer(hsize_t) :: start
! Number of read rows
integer(hsize_t) :: nrecords
! The type size
integer(size_t) :: type_size
! If data are real
real, dimension(nrecords) :: data_buffer
! Error code
integer :: hdferr

call h5tbread_field_name_f(file_id, table_name, field_name, &
                           start, nrecords, type_size, data_buffer, hdferr)
```

2.4.3 Check the presence of an attribute

```fortran
! File id
integer(hid_t) :: file_id
! Element's name
character(len=*), parameter :: element_name = "/an_element"
! Attribute's name
character(len=*), parameter :: attribute_name = "/an_attribute"
! Does attribute exist ?
logical :: attribute_exists
! Error code
integer :: hdferr

call h5aexists_by_name_f(file_id, element_name, attribute_name, &
                         attribute_exists, hdferr, H5P_DEFAULT_F)
```

2.4.4 Read attribute's information

The `h5ltget_attribute_info_f` can read attribute information, it returns :

- The dimensions of the attribute (an attribute can be an array).
- The class identifier
- The size of the datatype in bytes

```fortran
! File id
integer(hid_t) :: file_id
! Element's name
character(len=*), parameter :: element_name = "/an_element"
! Attribute's name
character(len=*), parameter :: attribute_name = "an_attribute"
! Dimensions
integer(hsize_t), dimension(:), allocatable :: dims
! Type class
integer :: type_class
```

```
! Type size in bytes
integer(size_t) :: type_size
! Error code
integer :: hdferr

call h5ltget_attribute_info_f(file_id, element_name, attribute_name, &
                              dims, type_class, type_size, hdferr)
```

2.4.5 Read a string attribute

```
! File id
integer(hid_t) :: file_id
! Element's name
character(len=*), parameter :: element_name = "/an_element"
! Attribute's name
character(len=*), parameter :: attribute_name = "an_attribute"
! Attribute's value
character(len=20) :: attribute_value = ""
! Error code
integer :: hdferr

call h5ltget_attribute_string_f(file_id, element_name, attribute_name, &
                                attribute_value, hdferr)
```

2.4.6 Read an integer attribute

```
! File id
integer(hid_t) :: file_id
! Element's name
character(len=*), parameter :: element_name = "/an_element"
! Attribute's name
character(len=*), parameter :: attribute_name = "an_attribute"
! Attribute's value
integer :: attribute_value
! Error code
integer :: hdferr

call h5ltget_attribute_int_f(file_id, element_name, attribute_name, &
                             attribute_value, hdferr)
```

2.4.7 Read a float attribute

```
! File id
integer(hid_t) :: file_id
! Element's name
character(len=*), parameter :: element_name = "/an_element"
! Attribute's name
character(len=*), parameter :: attribute_name = "an_attribute"
! Attribute's value
real :: attribute_value
! Error code
integer :: hdferr
```

```
call h5ltget_attribute_float_f(file_id, element_name, attribute_name, &
                               attribute_value, hdferr)
```

2.4.8 Read a double attribute

```
! File id
integer(hid_t) :: file_id
! Element's name
character(len=*), parameter :: element_name = "/an_element"
! Attribute's name
character(len=*), parameter :: attribute_name = "an_attribute"
! Attribute's value
double precision :: attribute_value
! Error code
integer :: hdferr

call h5ltget_attribute_double_f(file_id, element_name, attribute_name, &
                                attribute_value, hdferr)
```

2.4.9 Read a dataset's information

The function `h5ltget_dataset_info_f` read dataset's information, it returns :

- The dimensions of the dataset
- The class identifier
- The size of the datatype in bytes

```
! File id
integer(hid_t) :: file_id
! Element's name
character(len=*), parameter :: element_name = "/an_element"
! Dimensions
integer(hsize_t), dimension(*) :: dims
! Type class
integer :: type_class
! Type size in bytes
integer(size_t) :: type_size
! Error code
integer :: hdferr

call h5ltget_dataset_info_f(file_id, element_name, &
                            dims, type_class, type_size, hdferr)
```

2.4.10 Read a float dataset

Dataset's values can be read with the function `h5ltread_dataset_float_f` , the data buffer memory must be allocated before the call.

```
! File id
integer(hid_t) :: file_id
! Element's name
character(len=*), parameter :: element_name = "/an_element"
! Dimensions
```

```
integer(hsize_t), dimension(*) :: dims
! Dateset values
real, dimension(dims) :: dataset_value
! Type class
integer :: type_class
! Type size in bytes
integer(size_t) :: type_size
! Error code
integer :: hdferr

call h5ltread_dataset_float_f(file_id, element_name, &
                              dataset_value, dims, hdferr)
```

2.4.11 Inquire if a dataset exists

`h5ltfind_dataset_f` inquires if a dataset exist. It returns 1 if the dataset exists and returns 0 otherwise.

```
! file or group identifier
integer(hid_t), intent(in) :: loc_id
! name of the dataset
character(len=*), parameter :: dataset_name = "/an_element"
! error code
integer :: hdferr

result = h5ltfind_dataset_f(loc_id, dataset_name, hdferr)
```

2.4.12 Groups functions

In addition, some querry functions about groups are used.

Read the number of members in a group :

```
! file or group identifier
integer(hid_t) :: loc_id
! name of the group
character(len=*), parameter :: group_name = "/an_element"
! number of members in the group
integer :: nmembers
! error code
integer :: hdferr

call h5gn_members_f(loc_id, group_name, nmembers, hdferr)
```

Read the name of the members of a group :

```
! File or group identifier
integer(hid_t) :: loc_id
! Name of the group
character(len=*), parameter :: element_name = "an_element"
! Index of the member
integer :: index
! Name of the member
character(len=*), parameter :: member_name = "an_attribute"
! Possible member types
! H5G_LINK_F if member is a link
! H5G_GROUP_F if member is a group
! H5G_DATASET_F if member is a dataset
! H5G_TYPE_F if member is a type
integer :: member_type
! Error code
integer :: hdferr

call h5gget_obj_info_idx_f(file_id, group_name, index, &
                           member_name, member_type, hdferr)
```

2.5 Integers and reals

By default in **Amelet HDF**, all integers are 32bit integers.

As for as the reals, **Amelet HDF** objects definition doesn't require reals written on more than 32bits. So by default, all reals are 32bit floats and complex are 2x32bit complex (see *The complex type*).

Longer reals can be used in `arraySet` (see *ArraySet*) to take into account the precision of computed numerical data.

Practically, HDFView and h5dump show the data type, it is useful to check when writting data in **Amelet HDF** format.

2.6 The complex type

Natively HDF5 does not propose the complex number type. However it offers a very powerful mechanism to create our own type.

There are two ways to organize a complex number :

- as an array of two elements : A(dim=2) = (r, i) is a two element array and A(0) (A(1) in Fortran) is the real part and A(1) (A(2) in Fortran) is the imaginary part.
- as a dictionary with two key/value pairs with A("r") = r and A("i") = i.

Amelet HDF uses the compound approach, although it is not the simpliest formulation cause it is not accessible from the API lite, it is the strategy followed by some other tools like octave or pytables.

That is to say a complex number is always a compound datatype of two element (r, i).

2.6.1 Read a complex type

Even if a complex type is a compound structure, the too real or double numbers are written as if they were two consecutive elements of an array :

```
! File or group identifier
integer(hid_t) :: loc_id
! Name of the group
character(len=*), parameter :: element_name = "/an_element"
! The complex attribute is a 2 elements array
real, dimension(2) :: complex_attribute = (/0.0, 0.0/)
! Error code
integer :: hdferr

call h5ltget_attribute_float_f(loc_id, element_name, "complex_attribute", &
                               complex_attribute, hdferr)
print *, "\nComplex attribute value :", complex_attribute
```

`complex_attribute` is defined as a two real element array. The `h5ltget_attribute_float_f` function fills in the array with the `r` field and `i` filed of the compound complex attribute structure. Therefore, `complex_attribute(1)` equals `r` and `complex_attribute(2)` equals `i`.

2.7 Table and Dataset

We have seen HDF5 defines tables and datasets. A dataset is a multidimensional matrix, each cell contains data of the same nature (integer, float, ...). A table is like a spreadsheet, it has many columns which can contain different nature data.

In **Amelet HDF**, datasets are used by default when the data's nature are identical even if data can be seen by column.

For example, consider the data structure (name, path), a list of (name, path) can be written with two columns :

name	path
$name1	$path1
$name2	$path2
$name3	$path3

Tables are presented with column headers as HDFView does.

One would create an HDF5 two string column table but since the two columns contain string.

Warning: **Amelet HDF** has made the choice to use a (n x 2) dataset

$name1	$path1
$name2	$path2
$name3	$path3

Note: In fact, a table is a (n x 1) dataset with a compound datatype.

CHAPTER

THREE

THE AMELET HDF FILE

Before anything else, an **Amelet HDF** file is an HDF5 file. To know the file is precisely an **Amelet HDF** instance, the file object has two attributes :

- FORMAT is an HDF5 string attribute, its value is AMELETHDF.
- AMELETHDF_FORMAT_VERSION is an HDF5 string attribute defining the version number of the **Amelet HDF** specification. AMELETHDF_FORMAT_VERSION attribute is composed of three integers separated by colons. The first integer is the major version, the second integer is the minor version and the third integer is a release number. The release number is rarely used.

Example :

```
data.h5[@FORMAT=AMELETHDF
|       @AMELETHDF_FORMAT_VERSION=1.0.0]/
`-- physicalModel
    `-- volume
        |--$water
        `--$soltWater
```

CHAPTER

FOUR

PREDEFINED CATEGORIES

Amelet HDF is a specialization of the HDF5 format, it can then be represented by a tree, for which all groups (branches) can contain groups or leaves (datasets, tables). Branches and leaves can have attributes to precise specific values.

These groups, datasets and tables compose a hierarchical dictionary of electromagnetic data definitions like dielectric materials, impedance, mesh object or numerical data, electromagnetic sources (antenna) and so on.

4.1 The first rule

Amelet HDF relies on a first rule, each definition is contained in a predefined category (a category is an HDF5 group). For instance, three homogeneous material models will be defined in `/physicalModel/volume` :

```
data.h5/
`-- physicalModel
    `-- volume
        |--$water
        `--$soltWater
```

Note: In this document, where there are prefixed HDF5 names with "$", (`$water`, `$soltWater`) the names are example names the user can modify.

4.2 Categories

Amelet HDF information hierarchies can be represented by a tree a human being can easily read

```
data.h5/
|-- label
|-- group
|-- externalElement
|-- globalEnvironment
|-- electromagneticSource
|   |-- planeWave
|   |-- sphericalWave
|   |-- generator
|   |-- dipole
|   |-- sourceOnMesh
|   `-- antenna
|-- physicalModel
|   |-- multiport
|   |-- multilayer
|   |-- slotProperties
|   |-- anisotropic
|   |-- volume
|   |-- aperture
|   |-- shield
|   `-- grid
|-- mesh
|-- floatingType
|-- exchangeSurface
|-- transmissionLine
|   `--transmissionLineElement
|-- network
|   `-- $net1
|       |-- topology
|       |-- tubes
|       |-- junctions
|       `-- connections
|-- link
|-- outputRequest
|-- localizationSystem
|-- extensionTypes
|   `-- $car
`-- simulation
    `-- $sim1
        |-- input
        `-- output
```

The predefined categories are presented underneath and each one represents a concept of the electromagnetic simulation domain :

- **label** The category contains all labels used in the **Amelet HDF** instance.
- **group** This category contains `group` objects. A group contains only element names which have something in common
- **externalElement** The category contains all elements used in the **Amelet HDF** instance but defined in another instance.
- **globalEnvironment** Global data about simulations. If the simulation is in the time domain, “time” defines the computation duration, if the simulation is in the frequency domain, “frequency” defines the input simulation spectrum.
- **electromagneticSource** This category contains all electromagnetic sources definition :
    - **planeWave** A plane wave is defined by a direction of propagation, a linear polarisation angle, an elliptic polarisation angle, a magnitude and a nul phase point
    - **generator** A generator is defined by a magnitude and a linear circuit element
    - **dipole** A dipole is defined by a rotation, a localization, a magnitude, a linear circuit element, a length and a radius.
    - **sourceOnMesh** A source on mesh is defined by an array of values and an exchange Surface
    - **antenna** An antenna is defined by an efficiency, a magnitude, an input impedance, a feeder impedance and a load impedance.
- **physicalModel** In physicalModel are gathered all physical models.
  - **volume** This groups defines homogeneous volume material. A volume material is composed of a relative permittivity, an electric conductivity, a relative permeability and a magnetic conductivity

Predefined volume : `vacuum`

- **multiport** This category contains material models based upon linear circuit elements. All classical element can be used : impedance, admittance, resistance, inductance, capacitance, conductance, s parameters, **multiport circuit** ?

  Predefined multiports : `short circuit`, `open circuit`, `matched multiport`
- **anisotropic** An anisotropic material is defined by volumes or linear multiport. One material model per matrices element
- **multilayer** A multilayer material is a group of layers, a layer is the association of a physicalModel/material and a thickness
- **interface** A interface defines the connection between two media
- **slotProperties** A slot properties defined a slot with a width, a thickness and a material

- **mesh** Electromagnetic simulation methods often consume a discretized space. The space can be discretized with unstructured elements or structured elements. In this category all meshes are gathered.
  - selectorOnMesh Mesh entity are not named in the mesh category (or mesh group), the three tables in this category allow to give a name to mesh entities.
- **link** Once material models and meshes are defined, boundary conditions may have to be set. Links associate material models to mesh entities.
- **floatingType** This category contains floatingType elements which are meaningless in the context of **Amelet HDF** neither a predefined location. `FloatingType` computation results are stored in this category : electric field, magnetic field, current...
- **exchangeSurface** exchangeSurface modelize the data and the mesh to share data between electromagnetic simulations
- **transmissionLine** This category defines a transmission line section
- **network** This category describes all networks in the simulation
- **localizationSystem** Coordinate system definition
- **simulation** Definition of the simulations with input data and output data
- **extensionType** Definition of extensions
- OutputRequest

4.3 The entry point

The general shape of a **Amelet HDF** instance is an HDF5 file with a lot of not empty categories.

The question is the following : what is the first element to be read ?

The file attribute `@entryPoint` is the answer, `@entryPoint` is an HDF5 string attribute that spots the major element of the file. `@entryPoint` is optional.

```
data.h5[@entryPoint=/simulation/$sim1]
`-- simulation/
    `-- $sim1[@module=my-module
        |     @version=1.2]
        |-- parameter
        |-- inputs
        `-- outputs
```

In this example, the first element the reader must open is `data.h5:/simulation/$sim1`.

In addition `@entryPoint` can focus a on group if there is no particular element to read :

```
data.h5[@entryPoint=/floatingType]
`-- floatingType/
    |-- $e_field
    `-- $h_field
```

Here, the **Amelet HDF** instance stores some arraySets, the `@entryPoint` gives the *nature* of the file.

4.4 Physical quantity and unit

4.4.1 International System of Units

The value of a physical quantity Q is expressed as the product of a numerical value {Q} and a physical unit [Q]* (http://en.wikipedia.org/wiki/Physical_quantity) :

Q = {Q} x [Q]

All [Q] are expressed in the SI units (http://en.wikipedia.org/wiki/SI) **except for angular quantities**.

4.4.2 Angular quantities

Angular quantities are stored in `degree` instead of `radian`, it is the scientists preferred unit.

Warning: Angular quantities are stored in `degree`.

4.5 The simulation object

The purpose of the **Amelet HDF** Specification is to provide a standard for recording and recovering computer data of electromagnetic simulations, but what about the simulation itself ? **Amelet HDF** makes an attempt to describe the simulation with its inputs and outputs.

A simulation is an HDF5 group and is localized in the `simulation` category, see for instance `$sim1` simulation below

```
data.h5/
`-- simulation
    `-- $sim1[@module=my-module
        |      @version=1.2]
        |-- parameter
        |-- inputs
        `-- outputs
```

A `simulation` child (i.e. a simulation) has two attributes :

- the `module` attribute : this attribute gives the name of the module or the treatment process that must interpret the file, it is an HDF5 string attribute
- the `version` attribute : this attribute is the version number of the module, it is an HDF5 string attribute

In this preceding example, a “$sim1” simulation is defined, and the module that will be used is “my-module” in version 1.2.

The `simulation` group has three children :

1. `parameter` : `parameter` contains the module’s specific parameters of the simulation like global key words, global options.
2. `inputs` : `inputs` contains all informations used in the simulation, it is an HDF5 string dataset
3. `outputs` : `outputs` contains the informations skeleton of the output created by the simulation, it is an HDF5 string dataset

Warning: The simulation object is the default entry point of an **Amelet HDF** file.

4.5.1 The simulation’s parameters

The simulation’s parameters are module’s specific parameters of a simulation like global key words, global options, all information that is not common.

The module’s specific parameters can not be specified by **Amelet HDF** but **Amelet HDF** allows the user to describe schema data needed by the module.

Simulation parameters are named typed element, the type can be either simple (i.e. a native type) or compound (i.e. a structure) :

- The simple types are :
  - integer
  - real
  - string
  - boolean
- A compound type is a type made of a list of named simple types, example : (param1: integer, param2: real, param3: string)

`/simulation/$simulation/parameter` is an HDF5 group and parameters are written as follows :

- Simple type parameters become simple HDF5 named attributes
- Compound parameters are stocked in named HDF5 tables where columns are the structure’s fields and the table’s name is the parameter’s name.

Example :

Consider the parameters defined by :

- Parameter 1 :
  - name : color
  - type : string
- Parameter 2 :
  - name : temperature
  - type : real
- Parameter 3 :
  - name : listOfCommands

- type : compound
  - name : commandName
  - type : string
  - name : option1
  - type : string
  - name : option2
  - type : string

As a consequence, an instance of **Amelet HDF** looks like :

```
data.h5/
`-- simulation/
    `-- $sim1[@module=my-module
        |     @version=1.2]
        |-- inputs
        |-- outputs
        `-- parameter[@color=black
            |         @temperature=300]
            `-- listOfCommands
```

with the `data.h5:/simulation/$sim1/parameter/listOfCommands` table :

commandName	option1	option2
$command1	$option11	$option12
$command2	$option21	$option22

Note: Logically, simulation parameters of a specific module are not specified by **Amelet HDF**. One could think it would be necessary to add physical nature of parameters to the **Amelet HDF** instance, but the **Amelet HDF** instance is aimed to be read by the module adapter, and this component knows the module's interface.

Parameters physical nature is not mentioned in a simulation software input file.

4.5.2 The simulation's inputs

The simulation's inputs element is a one dimensional HDF5 dataset and contains 20 characters strings.

The possible inputs for a module are defined by the module itself, so there is no surprise by discovering the list of inputs. The module can perform some checks.

For a 3D electromagnetic module, FDTD method for instance, the first input to be read could be `/link/$my-data-on-mesh`, this object associates all physical models to the mesh entities.

For instance see the following `inputs` dataset:

`/link/$my-data-on-mesh`
`/mesh/$gmesh1/$my-car`
`/physicalModel/volume/$iron`
`/electromagneticSource/antenna/$my-antenna-1`
`/electromagneticSource/antenna/$my-antenna-2`
`/localizationSystem/$the-loc`
`/link/$the-data-on-loc`

The simulation takes a mesh ($my-car), a data on mesh ($my-data-on-mesh), a volume material ($iron), two antennas ($my-antenna-1, $my-antenna-2) located with the localization system.

4.5.3 The simulation's outputs

On one hand, simulation's inputs are described in the `inputs` group, on the other hand, many objects will be created while the computation runs and have to be well organized in order to be read correctly by the next process.

Simulation's output group describes those created objects.

The simulation's outputs element is a one dimensional HDF5 dataset and contains 20 characters strings.

For instance see the following table :

```
data.h5/
|-- extensionType/
|   `-- dataSet
|       `-- $data1
|           `-- linksDefinition
|-- floatingType/
|   |-- $e_field
|   |-- $h_field
|   |-- $current
|   `-- $tension
`-- simulation/
    `-- $sim1[@module=my-module
        |     @version=1.2]
        |-- inputs
        `-- outputs
```

with `data.h5:/extensionType/dataSet/$data1/linksDefinition`:

name	specificRole
/floatingType/$e_field	
/floatingType/$h_field	
/floatingType/$current	
/floatingType/$tension	

and `data.h5:/simulation/$sim1/outputs`:

/extensionTypes/dataSet/$output_data
/extensionTypes/dataSet/$output_power

4.5.4 Simulation's parametric elements

A parametric simulation is a simulation defined by a set of combinatorial input data. For instance, for a monochromatic solver the computation on a frequency spectrum (i.e. several frequencies) is a parametric simulation.

In **Amelet HDF**, simulation parametric elements are described in the **optional** `/simulation/$simulation/parametricElements` object. `/simulation/$simulation/parametricElements` is a two column table :

- The first column `name` is a string column which contains the name of the parametric element
- The second column `selected` is an integer column which contains 1 or 0 :
  - 0 : the process which read the simulation object must skip the parametric element
  - 1 : the process which read the simulation object must take into account the parametric element and deal with the creation of as much simulations as required.

```
data.h5/
|-- extensionType/
|   `-- dataSet
|       `-- $data1
|           `-- linksDefinition
|-- globalEnvironment/
|   `-- $ge
|       `-- frequency[@floatingType=vector
|                     @physicalNature=frequency
|                     @unit=hertz]
|-- floatingType/
|   |-- $e_field
|   |-- $h_field
|   |-- $current
|   `-- $tension
`-- simulation/
    `-- $sim1[@module=my-module
        |     @version=1.2]
        |-- parametricElements
        |-- parameter
        |-- inputs
        `-- outputs
```

With `/simulation/$sim1/parametricElements` :

name	selected
`/globalEnvironment/$ge/frequency`	1

In the example, the process will create a number of simulations equals the number of elements of `/globaleEnvironment/$ge/frequency` simulations.

4.5.5 Batch submission system

4.6 The `group` category

This category contains `group` objects. A group contains only element names which have something in common.

In the following example, `data.h5:/group/$em_field_result` is used to gather `/floatingType/$e_field` and `/floatingType/$h_field` for an output request for instance (see *Output requests* for output request definition) :

```
data.h5/
|-- label/
|   `-- outputRequests
|-- mesh/
|   `-- $gmesh1
|       `-- $sphere
|           `-- group
|               |-- $inside[@type=volume]
|               `-- $skin[@type=face]
|-- group
|   |-- $em_field_result
|   `-- $group2
|-- floatingType/
|   |-- $e_field
|   |-- $h_field
|   |-- $current
|   `-- $tension
|-- simulation/
|   `-- $sim1[@module=my-module
|       |     @version=1.2]
|       |-- inputs
|       `-- outputs
`-- outputRequest/
    `-- $outputRequest_group/
        `-- $or1[@subject=/label/predefinedOutputRequests
                 @subject_id=0
                 @object=/mesh/$gmesh1/$sphere/group/$inside
                 @output=/floatingType/$e_field]
```

with `data.h5:/label/outputRequests`:

electromagneticField

and with `data.h5:/group/$em_field_result`:

/floatingType/$e_field
/floatingType/$h_field

CHAPTER

FIVE

FLOATING TYPE STRUCTURES

Array data are whatever data that can be contained in a multidimensional array (also called multidimensional matrix).

An HDF5 dataset is simply a multidimensional array, the values's type of elements in a dataset is the same for all elements. This type can be :

- Integer
- Real
- Double
- String
- Complex

Besides, in physics matrices's dimensions are often linked to physical data. For instance, the first dimension of a temporal pulse current is the time. (The time being a one dimensional array (a vector) stored in a one dimensional HDF5 dataset)

To assure this association each matrix's dimension must be linked to a dimension dataset :

```
data.h5/
|-- current # One dimensional real dataset parameterized by time
`-- time    # One dimensional real dataset, current's abscissa
```

A second example is a current density on a surface at a given frequency, the current is a two dimensional matrix and the two dimensions are x and y that are the cartesian coordinates of a grid :

```
data.h5/
|-- current # Two dimensional real datasets parameterized by x and y
|-- x       # First dimension : one dimensional real dataset
`-- y       # Second dimension : One dimensional real dataset
```

Numeric structures of this kind (one dimensional datasets, multidimensional datasets, all manner to describing a list of numbers) are used in many cases and **can be found out everywhere** in an **Amelet HDF** file, they take the shape of HDF5 elements having an attribute named **floatingType**.

5.1 The `floatingType` category

In **Amelet HDF** `floatingType` are always children of another objects in order to give it a meaning. However, it can happen a `floatingType` instance is orphan, for instance the result of a simulation can be only a time, a real number. The `floatingType` category contains orphan `floatingType`.

Example (see *The SingleReal type* for `singleReal` definition) :

```
data.h5/
|-- floatingType/
|   `-- $flash_duration[@floatingType=singleReal
|                       @label=flash duration
|                       @physicalNature=time
|                       @unit=second
|                       @value=1e-9]
`-- simulation/
    `-- $sim1[@module=my-module
        |     @version=1.2]
        |-- inputs
        `-- outputs
```

with `data.h5:/simulation/$sim1/outputs` :

/floatingType/$flash_duration

5.2 Single types

Generally, **Amelet HDF** tries to use the simplest way to describe an element. For instance, a plane wave is defined by some attributes :

```
data.h5
`-- electromagneticSource
    `-- planeWave
        `-- $a-plane-wave[@xo=0.0
                          @yo=0.0
                          @zo=0.0
                          @theta=0.0
                          @phi=0.0
                          @linearPolarization=0.0]
```

`$a-plane-wave`'s' attributes represents real numbers, their physical nature and unit are set by **Amelet HDF**.

Sometimes, an attribute can be expressed by several ways. Take the electric conductivity of a material model, it could be defined with an HDF5 real attribute :

```
data.h5
`-- physicalModel/
    `-- volume
        `-- $water[@electricConductivity=10e-6]
```

However, the electric conductivity depends on the frequency, and the writing above is an approximation, the valid frequency interval is omitted.

For this case, **Amelet HDF** proposes a little more complex structure : the single types `singleReal` and `singleComplex`.

The electric conductivity is evenly defined with a singleReal :

```
data.h5
`-- physicalModel/
    `-- volume
        `-- $water
            `-- electricConductivity[@floatingType=singleReal]
                                     @value=10e-6]
```

5.2.1 The SingleInteger type

A `singleInteger` is a `floatingType` that represents a integer with optional attributes.

One mandatory attribute :

- `value` : the value of the singleInteger, it is an HDF5 integer attribute.

Optional attributes :

- `label` : the label of the singleInteger
- `physicalNature` : the physical nature of the singleInteger
- `unit` : the unit of the singleInteger
- `comment` : a comment

These attributes are all HDF5 string attributes.

```
data.h5
`-- floatingType/
    `-- $an_integer[@floatingType=singleInteger
                    @value=7]
```

5.2.2 The SingleReal type

A `singleReal` is a `floatingType` that represents a real with optional attributes.

One mandatory attribute :

- `value` : the value of the singleReal, it is an HDF5 real attribute.

Optional attributes :

- `label` : the label of the singleReal
- `physicalNature` : the physical nature of the singleReal
- `unit` : the unit of the singleReal
- `comment` : a comment

These attributes are all HDF5 string attributes.

It is usually used when an element can be defined by a single real number or an array. The electric conductivity definition of a material is a use case of the singleReal type :

```
data.h5
`-- physicalModel/
    `-- volume
        `-- $water
            `-- electricConductivity[@floatingType=singleReal]
                                     @value=10e-6]
```

5.2.3 The singleComplex type

A `singleComplex` is a `floatingType` that represents a complex number with optional attributes :

One mandatory attribute :

- `value` : a two elements array (real part, imaginary part) attribute, it is the value of the complex number.

Optional attributes :

- `label` : the label of the singleComplex
- `physicalNature` : the physical nature of the singleComplex
- `unit` : the unit of the singleComplex
- `comment` : a comment

These attributes are all HDF5 string attributes.

It is usually used when an element can be defined by a single complex number or a complex array. The relative permittivity of a material is a use case of the singleComplex type :

```
data.h5
`-- physicalModel/
    `-- volume
        `-- $water
            `-- relativePermittivity[@floatingType=singleComplex
                                     @value=(80, 0)]
```

5.2.4 The singleString type

For the sake of completness, a `singleString` is a `floatingType` that represents a string with optional attributes :

One mandatory attribute :

- `value` : a string attribute, it is the content of the string

Optional attributes :

- `label` : the label of the singleComplex
- `physicalNature` : the physical nature of the singleComplex
- `unit` : the unit of the singleComplex
- `comment` : a comment

These attributes are all HDF5 string attributes.

As an example, the `date` unit can be expressed as a singleString (see *Dealing with date*)

5.3 Vector

The next `floatingType` element we discuss is the `vector`.

A one dimensional dataset is a vector and can be explicitly defined or implicitly defined :

- A vector is explicitly defined when all values are contained in the HDF5 dataset (A = [1, 2, 3, 4]), `floatingType` equals `vector` in this case
- A vector is implicitly defined when its values can be computed from parameters (A = [1, 10, 2], A is the set of 5 numbers starting at 1 to 10 using a 2 spacing)

Many formulations can be used to implicitly define a `vector` :

- linearListOfReal1 : numberOfValues evenly spaced real numbers from first to last. linearListOfReal1's parameters are :

  - first : the first value of the list
  - last : the last value of the list
  - numberOfValues : the number of values in the list
- linearListOfReal2 : numberOfValues evenly spaced real numbers from first using “step” spacing. linearListOfReal2’s parameters are :
  - first : the first value of the list
  - step : spacing between values
  - numberOfValues : the number of values in the list
- logarithmListOfReal : numberOfValues evenly spaced real numbers on a logarithmic scale. logarithmListOfReal’s parameters are :
  - first : the first value of the list
  - last : the last value of the list
  - numberOfValues : number of values
- perDecadeListOfReal : numberOfValues evenly spaced real numbers on a logarithmic scale per decade. perDecadeListOfReal’s parameters are :
  - first : the first value of the list
  - numberOfDecades : the number of decades in the list
  - numberOfValuesPerDecade : the number of values per decades in the list
- linearListOfInteger2 : numberOfValues evenly spaced integer numbers from first using “step” spacing. linearListOfInteger2’s parameters are :
  - first : the first value of the list
  - step : spacing between values
  - numberOfValues : the number of values in the list

Therefore, a one dimensional HDF5 dataset with the attribute `floatingType` equals :

- `linearListOfReal1` is a linearListOfReal1 list and have also the attributes `first`, `last` and `numberOfValues`. The dataset’s content is not taken into account.
- `linearListOfReal2` is a linearListOfReal2 list and have also the attributes `first`, `step` and `numberOfValues`. The dataset’s content is not taken into account.
- `logarithmListOfReal` is a logarithmListOfReal list and have also the attributes `first`, `last` and `numberOfValues`. The dataset’s content is not taken into account.
- `perDecadeListOfReal` is a perDecadeListOfReal list and have also the attributes `first`, `numberOfDecades` and `numberOfValuesPerDecade`. The dataset’s content is not taken into account.
- `linearListOfInteger2` is a linearListOfInteger2 list and have also the attributes `first`, `last` and `numberOfValues`. The dataset’s content is not taken into account.

For example with the preceding current density data :

```
data.h5/
|-- $current
|-- $x[@floatingType=vector]
|-- $y[@floatingType=linearListOfReal1,
|       @first=1.0,
|       @last=10.0,
|       @numberOfValues=10]
`-- $z
```

`$z` is defined without attribute, when there is no attribute `floatingType`, the element is a `vector`.

5.3.1 Interval

A real interval is classically defined by two real numbers (start, end) and is written `[start, end]`. In **Amelet HDF**, a real interval is described with a `linearListOfReal1` with no `numberOfValues` attribute.

Example :

```
data.h5/
`-- $int1[@floatingType=linearListOfReal1
          @physicalNature=time
          @first=0
          @last=1e-6]
```

`data.h5:/$int1` is a time interval starting from 0 second and terminating at 1e-6 second.

5.4 Rational functions

5.4.1 Introduction

A rational function is any function which can be written as the ratio of two polynomial functions. (http://en.wikipedia.org/wiki/Rational_function).

5.4.2 The rational function

In **Amelet HDF**, `rationalFunction` is a floatingType, it can be found out every where in an instance. A `rationalFraction` is the sum of fractions of different types, the types are :

- Type 1 : $f(p) = B$
- Type 2 : $f(p) = \dfrac{B}{p-A}$
- Type 3 : $f(p) = \dfrac{B}{(p-A)^2}$
- Type 4 : $f(p) = \dfrac{B}{(p-A)^2 + (2\pi F)^2}$
- Type 5 : $f(p) = \dfrac{B(p-A)}{(p-A)^2 + (2\pi F)^2}$

A `rationalFunction` is an named HDF5 table that has four columns : **type**, **A**, **B** and **F**. **type** is an HDF5 integer and the others are HDF5 reals.

- Type 1 : only **B** is read
- Types 2 and 3 : **A** and **B** are used
- Types 4 and 5 : **A**, **B** and **F** are used

Example :

type	A	B	F
1		1	
2	2	3	
3	4	5	
4	6	7	8
5	9	10	11

5.4.3 The general rational function

The general rational function is sometimes used to generalize the representation of frequency dependent properties of dielectric material for instance.

Two **types** of general rational function exist:

- Polynomial
- Partial fraction.

A general rational function $f(j\omega)$ of polynomial type is written as :

$$f(j\omega) = \frac{a_0 + a_1(j\omega) + a_2(j\omega)^2 + ... + a_n(j\omega)^n}{b_0 + b_1(j\omega) + b_2(j\omega)^2 + ... + b_n(j\omega)^n}$$

where a_i, b_i are complex numbers.

A general rational function $f(j\omega)$ of partial fraction type is written as :

$$f(j\omega) = \sum_i (\frac{a_0}{b_0} + \frac{a_1}{j\omega - b_1} + \frac{a_2}{(j\omega - b_2)^2} + ... + \frac{a_i}{(j\omega - b_i)^{k_i}})$$

where a_i, b_i are complex numbers and k_i are integer numbers.

In **Amelet HDF**, `generalRationalFunction` is a floatingType, it can be found out every where in an instance.

A floatingType `generalRationalFunction` has one more attribute:

- `type` : `type` is a HDF5 character attribute which gives the type of the `floatingType`. It can take the values `polynomial` and `partialFraction`

General rational function of polynomial type

A `generalRationalFunction` of polynomial type is a 2 column **dataset** of complex numbers ({r, i}).

The number of lines is the degree of the rational function, example :

degree	numerator	denominator
0	(1, 2)	(2, 4)
1	(1, 1)	(0, 2)
2	(0, 2)	(2, 0)
.	.	.
.	.	.
.	.	.
n	(32, 2)	(2, -12)

The headers and the first column are represented for the explanation, the genuine dataset is :

(1, 2)	(2, 4)
(1, 1)	(0, 2)
(0, 2)	(2, 0)
.	.
.	.
.	.
(32, 2)	(2, -12)

This example defines a nth order general rational function and is equivalent to the following relation :

$$f(j\omega) = \frac{(1+2j) + (1+1j)(j\omega) + (2j)(j\omega)^2 + ... + (32+2j)(j\omega)^n}{(2+4j) + (2j)(j\omega) + (2)(j\omega)^2 + ... + (2-12j)(j\omega)^n}$$

General rational function of partial fraction type

A `generalRationalFunction` of partial fraction type is an named HDF5 **table** that has three columns : **degree** of the denominator, **numerator** and **denominator**. **degree** is an HDF5 integer and the others are HDF5 complex numbers ({r, i}).

Example :

degree	numerator	denominator
0	(5, 0)	(1, 0)
1	(3, 4)	(1, 4)
2	(2, 1)	(1, 4)
1	(0, 2)	(2, 0)

This example is equivalent to this rational fraction:

$$f(j\omega) = 5 + \frac{3+4j}{j\omega - 1 - 4j} + \frac{2+1j}{(j\omega - 1 - 4j)^2} + \frac{2j}{j\omega - 2}$$

5.4.4 The rational

In order to be consistent with the `floatingType dataSet` relative to `singleReal` and `singleComplex`, `rationalFunction` can be used as element of a matrix named `data`. The element of `data` are HDF5 string, names of the `rationalFunction`. The resulting structure is a `floatingType` named `rational`, it is a named HDF5 group and is made up of :

- An HDF5 group named `function` that contains `rationalFunction`
- A `dataSet` named `data` of HDF5 string

The `floatingType rational` is used for the definition of impedance : (the whole definition of a `/physicalModel/multiport` will be seen later)

```
data.h5
    /physicalModel/
    `-- multiport/
        `-- $impedance1[@floatingType=rational
            |            @physicalNature=impedance]
            |-- function
            |   |-- $rationalFraction1[@floatingType=rationalFunction]
            |   |-- $rationalFraction2[@floatingType=rationalFunction]
            |   `-- $rationalFraction3[@floatingType=generalRationalFunction
            |                          @type=polynomial]
            `-- data
```

with `/physicalModel/multiport/$impedance1/function/$rationalFraction1` :

type	A	B	F
1		1	
2	2	3	
3	4	5	
4	6	7	8
5	9	10	11

and `/physicalModel/multiport/$impedance1/function/$rationalFraction3` :

(1, 2)	(2, 4)
(1, 1)	(0, 2)
(0, 2)	(2, 0)
.	.
.	.
.	.
(32, 2)	(2, -12)

and `/physicalModel/multiport/$impedance1/data` :

$rationalFraction1	$rationalFraction3
$rationalFraction2	$rationalFraction1

5.5 DataSet

The `dataSet` structure is another `floatingType` in the **Amelet HDF** specification, it represents a matrix with physical quantity attributes.

A `dataSet` can be used to define a multiport resistance with constant values for instance :

```
data.h5
`-- physicalModel/
    `-- multiport/
        `-- $resistance0[@floatingType=dataSet
                         @physicalNature=resistance
                         @unit=ohm]
```

In the example `$resistance0` is a three ports multiport (given by the dimension of HDF5 dataset), `/physicalModel/multiport/$resistance0` is :

12	13	14
13	12	13
14	13	12

HDF5 provides the API to query the dimension of the dataset.

Note: DataSet is a native HDF5 object.

5.6 ArraySet

An arraySet is another `floatingType` structure, it has an attribute `floatingType` equals `arraySet`. The reason of this floating characteristic is that an arraySet is a structure that can be also used in many locations in a **Amelet HDF** instance.

An arraySet is defined by :

- `data`, an HDF5 multidimensionnal dataset, its type can be :
  - integer
  - real
  - double
  - complex
  - string
- a "ds" group (dimension scale) that contains the HDF5 dataset representing data dimensions. If data's dimension is 3, ds's children are:
  - `dim1`, first `data`'s dimension. `dim1` is a one dimensional dataset.
  - `dim2`, second `data`'s dimension. `dim2` is a one dimensional dataset.
  - `dim3`, third `data`'s dimension. `dim3` is a one dimensional dataset.

  The names `dim1` ... `dimN` is a **Amelet HDF** convention and are the only authorized names.

The first dimension is the most internal loop of `data`.

Example :

```
data.h5/
`-- floatingType/
    `-- $eField[@floatingType=arraySet]
        |-- data              # a (n x m x l) dataset
        `-- ds/
            |-- dim1
            |-- dim2
            `-- dim3
```

In addition, **Amelet HDF** defines the manner to write scientific data the way there is no place for misunderstanding. That's why the physical nature and unit of parameters are set by **Amelet HDF** to limit the complexity of the understanding and the treatment. There is no need the write code to convert the unknown input unit toward another unit.

However, arraySets can be read out of the context of **Amelet HDF**, simulation results can be stored in an arraySet file and there is no way to know the function, the nature and the unit of the data.

As a consequence the arraySet structure must be self-describing and could provide optional attributes to precise the arraySet. The optional attribute hold by `data` (as `data` is a `floatingType=dataSet` are :

- the label (HDF5 string attribute `label`)
- the physical nature : "length", "voltage" (HDF5 string attribute `physicalNature`)
- the unit : "meter", "volt" (HDF5 string attribute `unit`)
- a comment (HDF5 string attribute `comment`)

In addition, the type (or the class) of the values (integer, float...) is not published by an attribute, HDF5 provide the function `H5LTget_dataset_info`, the returned `type_class` is the class of the values.

5.6.1 The dimensions order

The "ds" group contains the HDF5 datasets representing data dimensions : the `ds/dim*`. They contain as well the physical nature of the dimensions, they follow the rule :

`ds/dim1` is always the **fastest-changing dimension**.

5.6.2 C versus fortran - Storage conventions

C and fortran don't use the same storage convention :

- C : the last listed dimension is the fastest-changing dimension and the first-listed dimension is the slowest changing
- Fortran : Fortran stores data by columns, the first-listed dimension is the fastest-changing dimension and the last-listed dimension is the slowest-changing

As a consequence, the array A(20, 100) allocated in C is the array A(100, 20) in fortran. HDF5 uses C storage conventions so an array stored from a C program must be read in the reversed order from a fortran program, the HDF5 wrapper accomplishes the task transparently for the user.

5.6.3 Component parameter

Numerical data can be scalar but can also be vector fields. In this case, a dimension handle the component loop, for this dimension the `@physicalNature` equals `component` and values stored in the dataset are HDF5 string which are the label of each component.

The possible components by coordinate system are :

Coordinate system	Possible components
Cartesian coordinate system	**x**, **y**, **z**
Cylindrical coordinate system	**r**, **theta**, **z**
Spherical coordinate system	**r**, **theta**, **phi**

Example 1

Example of the components Ex, Ey, Ez of an electric field during the time

```
data.h5/
`-- floatingType/
    `-- $dataOne[@floatingType=arraySet
        |        @label=Electric field around a wire]
        |-- data[@label=electric field
        |        @physicalNature=electricField
        |        @unit=voltPerMeter]
        `-- ds/
            |-- dim1[@label=component x y z
            |        @physicalNature=component]
            `-- dim2[@label=the time
                     @physicalNature=time
                     @unit=second]
```

with `data.h5:/floatingType/dataOne/ds/dim1` vector :

index	component
0	x
1	y
2	z

Example 2

Example of the components Ex, Ey of an electric field during the time

```
data.h5/
`-- floatingType/
    `-- $dataOne[@floatingType=arraySet
        |        @label=Electric field around a wire]
        |-- data[@label=electric field
        |        @physicalNature=electricField
        |        @unit=voltPerMeter]
        `-- ds/
            |-- dim1[@label=component x y
            |        @physicalNature=component]
            `-- dim2[@label=the time
                     @physicalNature=time
                     @unit=second]
```

with `data.h5:/floatingType/dataOne/ds/dim1` vector :

index	component
0	x
1	y

5.6.4 Numerical data on mesh

Computations data are often located in space and a mesh is used the associate a data value to a mesh entity. In **Amelet HDF** this association is handled by a dim* child of the arraySet. The `physicalNature` of this dimension must be `meshEntity`.

For a complete description of mesh see *Mesh*.

In this case, the dim* dataset is a string dataset and the values are the name of the mesh entity.

```
data.h5/
|-- mesh/
|   `-- $gmesh1/
|       `-- $mesh1/
|           `-- group/
|               `-- $exchange_surface[@entityType=node]
`-- floatingType
    `-- $dataOne[@floatingType=arraySet
        |        @label=Electric field around the wire]
        |-- data[@label=electric field
        |        @physicalNature=electricField
        |        @unit=voltPerMeter]
        `-- ds/
            |-- dim1[@label=component x y z
            |        @physicalNature=component]
            |-- dim2[@label=mesh elements
            |        @physicalNature=meshEntity]
            `-- dim3[@label=the time
                     @physicalNature=time
                     @unit=second]
```

with `/floatingType/$dataOne/ds/dim2` :

/mesh/$gmesh1/$mesh1/group/$exchange_surface

Numerical data on nodes

The preceding example deals with numerical data on nodes (the `entityType` is `node`), that is to say all data are located on nodes in the node group.

Considering the `meshEntity` dimension, from the first value to last value, values are located from the first node to the last node defined in the group node.

Numerical data on unstructured elements - the `location` attribute

Data may also be located on the elements of a mesh. The following example associates data on the barycenter of `/mesh/$gmesh1/mesh1/group/$exchange_surface` elements :

```
data.h5/
|-- mesh/
|   `-- $gmesh1/
|       `-- $mesh1/
|           `-- group/
|               `-- $exchange_surface[@entityType=face]
`-- floatingType/
    `-- $dataOne[@floatingType=arraySet
        |        @label=Electric field around the wire]
        |-- data[@label=electric field
        |        @physicalNature=electricField
        |        @unit=voltPerMeter]
        `-- ds/
            |-- dim1[@label=component x y z
            |        @physicalNature=component]
            |-- dim2[@label=mesh elements
            |        @physicalNature=meshEntity]
            `-- dim3[@label=the time
                     @physicalNature=time
                     @unit=second]
```

with `/floatingType/$dataOne/ds/dim2` :

/mesh/$gmesh1/mesh1/group/$exchange_surface

The example deals with numerical data on elements (the `entityType` is `face`), that is to say all data are located on elements in the group.

But how are data located within the element ? The answer is in the `location` attribute.

When `physicalNature` is `meshEntity`, the dimension can have an optional attribute named `location`. The `location` attribute is a string attribute which can take the following values :

- `barycenter`, data are located at the barycenter of the elements.
- `node`, data are located at element nodes.
- `middleEdge`, data are located at element middle edges if `entityType=face` or `entityType=volume`
- `centerFace`, data are located at element center face if `entityType=volume`.

If `location` is not present or not understood, its value is `barycenter`.

For this to be possible, implicit and explicit element definition are necessary :

- Unstructured elements are explicitly defined by nodes. So element nodes are explicitly selectable.
- In addition *elementTypes* details the definition of (implicit) subelements (edges for `entityType=face`, edges and faces for `entityType=volume`) of an element, these subelements are numbered.

Implicit subelements permit to select (n-1) dimensional subelements of an n dimensional element.

Important: A simple rule has to be followed to localize data on the mesh :

Data are always located on an explicit or implicit points list.
Data size must be the same as the points list size.

Data are distributed on points in the same order they are defined. If a point is already associated with a value, the process goes on with the next possible location.

Example 1

The first example deals with three reals located on triangles barycenter (one barycenter per element so three elements), the **Amelet HDF** structure looks like the following :

```
data.h5/
|-- mesh/
|   `-- $gmesh1/
|       `-- $mesh1[@type=unstructured]/
|           |-- nodes
|           |-- elementTypes
|           |-- elementNodes
|           `-- group/
|               `-- $three_triangles[@type=element
|                                    @entityType=face]
`-- floatingType/
    `-- $dataOne[@floatingType=arraySet
        |        @label=Electric field on a triangle]
        |-- data[@label=electric field
        |        @physicalNature=electricField
        |        @unit=voltPerMeter]
        `-- ds/
            `-- dim1[@label=mesh elements
                     @physicalNature=meshEntity
                     @location=barycenter]
```

`data.h5:/mesh/$gmesh1/$mesh1/group/$three_triangles` is :

14
23
5

`data.h5:/floatingType/$dataOne/data` is a one-dimensional real array, its length is three and `data.h5:/floatingType/$dataOne/ds/dim1` is :

/mesh/$gmesh1/mesh1/group/$three_triangle

Associations between triangles and data are :

```
triangle 0 in ``$three_triangles`` (14) holds /floatingType/$dataOne/data[0]
triangle 1 in ``$three_triangles`` (23) holds /floatingType/$dataOne/data[1]
triangle 2 in ``$three_triangles`` (5) holds /floatingType/$dataOne/data[2]
```

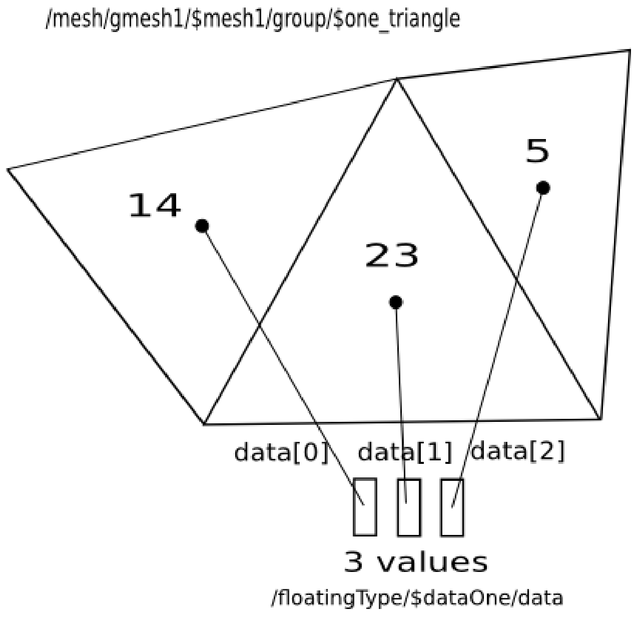

/mesh/gmesh1/$mesh1/group/$one_triangle
14
5
23
data[0]
data[1]
data[2]
3 values
/floatingType/$dataOne/data

Fig. 5.1: Data dispatching on a triangle (tri3)

Example 2

The second example deals with six reals located on two joint quad4 nodes, the **Amelet HDF** structure looks like the following :

```
data.h5/
|-- mesh/
|   `-- $gmesh1/
|       `-- $mesh1[@type=unstructured]/
|           |-- nodes
|           |-- elementTypes
|           |-- elementNodes
|           `-- group/
|               `-- $two_quad4[@type=element
|                              @entityType=face]
`-- floatingType
    `-- $dataOne[@floatingType=arraySet
        |        @label=Electric field on a quad4]
        |-- data[@label=electric field
        |        @physicalNature=electricField
        |        @unit=voltPerMeter]
        `-- ds/
            `-- dim1[@label=mesh elements
                     @physicalNature=meshEntity
                     @location=node]
```

`data.h5:/mesh/$gmesh1/$mesh1/group/$two_quad4` is :

12
27

`data.h5:/mesh/$gmesh1/$mesh1/elementTypes` is respectively for the 12th and 27th element :

...
1
2
5
6
...
2
3
4
5
...

`data.h5:/floatingType/$dataOne/data` is a one-dimensional real array, its length is six and `data.h5:/floatingType/$dataOne/ds/dim1` is :

/mesh/$gmesh1/mesh1/group/$two_quad4

Associations between quad4 and data are :

```
quad4 0 in ``$two_quads4`` (12)
  node 1 holds /floatingType/$dataOne/data[0]
  node 2 holds /floatingType/$dataOne/data[1]
  node 3 holds /floatingType/$dataOne/data[2]
  node 4 holds /floatingType/$dataOne/data[3]
quad4 1 in ``$two_quads4`` (27)
  node 2 holds /floatingType/$dataOne/data[4]
  node 3 holds /floatingType/$dataOne/data[5]
```

Here, the first value in data is located on the first node of the first quad4, the second value on the second node, the third value on the third node of the first quad4, the fourth value on the fourth node. The other values are located on the second quad4 nodes. If the first node of the second quad4 is already associated with a value, we continue. Finally the last value is located on the only node (and last) that has no value of the second quad4.

Note: On the figure, external numbers are quad4 nodes indices defined in `data.h5:/mesh/$gmesh1/$mesh1/elementTypes` and `data.h5:/mesh/$gmesh1/$mesh1/elementNodes` and internal numbers are nodes numbers relative to the element definition see *elementTypes*)

This rule is valid for edges and faces.

Data and finite difference method - the `location` attribute

For finite difference (FD) methods under cartesian coordinate system, computed quantities are not all calculated at the same location :

- Electric fields are located at cell's middle edges
- Magnetic fields are located at cell's center faces

For instance, in a 2D Yee scheme Ex, Ey and Hx components are located as the following picture shows :

In addition structured entities (volumes, surfaces and edges) are defined by too end-corner points (see the gray zone on the picture) :

- (imin, jmin, kmin) are the coordinates of the inferior end-corner point
- (imax, jmax, kmax) are the coordinates of the superior end-corner point

The components contained in a such entity are (the fortran notation for integer range is used, Ex(1:2) is a two elements array with Ex(1) and Ex(2)):

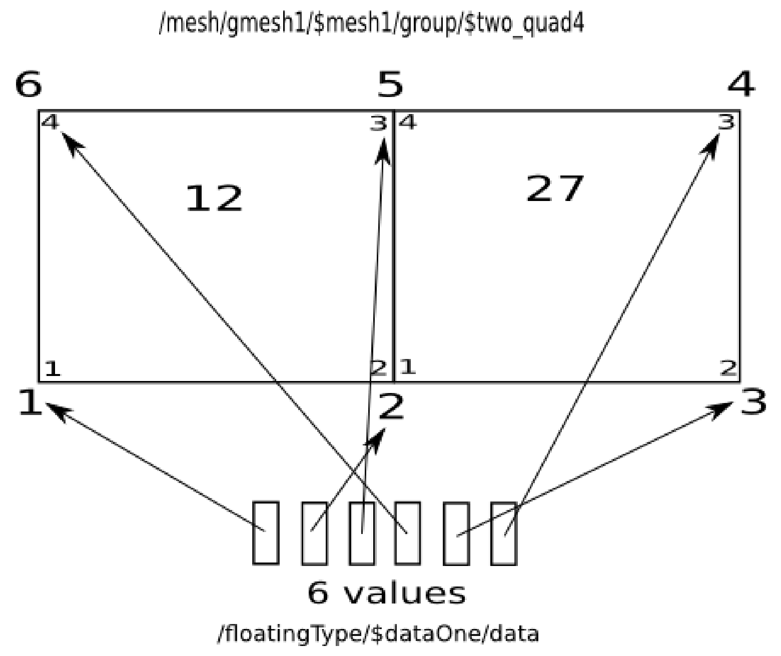

/mesh/gmesh1/$mesh1/group/$two_quad4
6
5
4
4
3
4
3
12
27
1
2
1
2
1
2
3
6 values
/floatingType/$dataOne/data

Fig. 5.2: Data dispatching on quad4 nodes

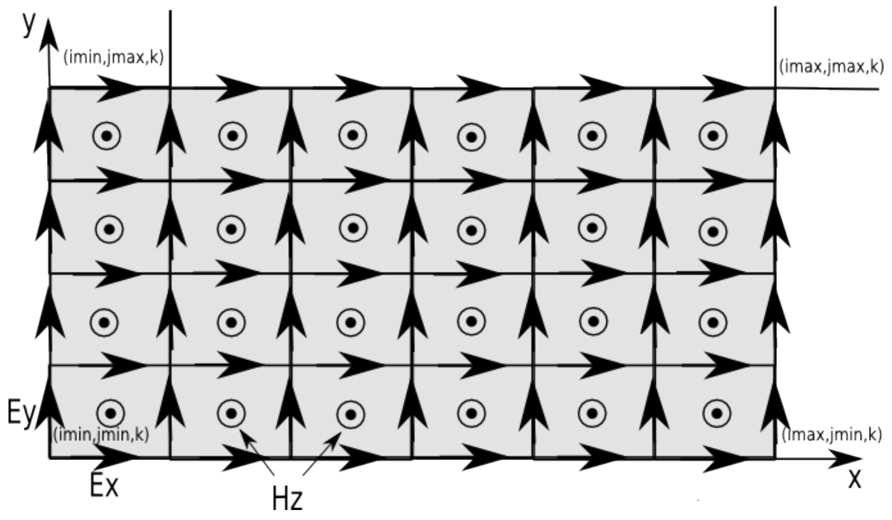


Fig. 5.3: Ex, Ey and Hx location in a 2d Yee grid

- Ex(imin:imax-1, jmin:jmax , kmin:kmax)
- Ey(imin:imax , jmin:jmax-1, kmin:kmax)
- Ez(imin:imax , jmin:jmax , kmin:kmax-1)

and

- Hx(imin:imax , jmin:jmax-1, kmin:kmax-1)
- Hy(imin:imax-1, jmin:jmax , kmin:kmax-1)
- Hz(imin:imax-1, jmin:jmax-1, kmin:kmax)

To store Ex, Ey and Ez in a single arraySet and Hx, Hy and Hz in another arraySet, a `meshEntity` dimension is defined by `x, y, z` and the shape of the regular `data` dataset of the arraySet is [imin:imax, jmin:jmax, kmin:kmax], as a consequence some components are written but should not be consumed :

- Ex(imax, jmin:jmax , kmin:kmax)
- Ey(imin:imax , jmax, kmin:kmax)
- Ez(imin:imax , jmin:jmax , kmax)

and

- Hx(imin:imax , jmax, kmin:kmax-1) and Hx(imin:imax , jmin:jmax-1, kmax)
- Hy(imax, jmin:jmax , kmin:kmax-1) and Hy(imin:imax-1, jmin:jmax , kmax)
- Hz(imax, jmin:jmax-1, kmin:kmax) and Hz(imin:imax-1, jmax, kmin:kmax)

The `location` `attribute`

In the case of finite difference methods, the `location` attribute is `finiteDifference`

Note: Data are written in the same order the coordinate system dimension are defined. (x, y, z, or i, j, k for a cartesian coordinate system)

Example

For the the components from the image, the fields are stored as follows :

```
data.h5/
|-- mesh/
|   `-- $gmesh1/
|       `-- $mesh1[@type=structured]/
|           |-- cartesianGrid/
|           `-- group/
|               `-- $gray_surface[@type=element
|                                 @entityType=face]
`-- floatingType/
    `-- $ex_and_ey[@floatingType=arraySet
        |          @label=Electric field on the gray surface]
        |-- data[@label=electric field
        |        @physicalNature=electricField
        |        @unit=voltPerMeter]
        `-- ds/
           |-- dim1[@label=Ex and Ey
           |        @physicalNature=component]
           `-- dim2[@label=the grey surface
```

```
                         @physicalNature=meshEntity
                         @location=finiteDifference]
```

with `data.h5:/floatingType/$ex_and_ey/ds/dim1` :

index	component
0	x
1	y

and `data.h5:/mesh/$gmesh1/$mesh1/group/$gray_surface` is :

imin	jmin	k	imax	jmax	k

5.6.5 Examples of arraySets

- The material relative permittivity definition (in `/physicalModel/volume`):
  - `data` is a one dimensional complex dataset without unit;
  - `ds/dim1` is a one dimensional real dataset (a vector)
    - `physicalNature` = "frequency"
    - `unit` = "hertz"
- A current pulse in the time domain
  - `data` is a one dimensional real dataset
    - `physicalNature` = "current"
    - `Unit` = "ampere"
  - `ds/dim1` is a one dimensional real dataset
    - `physicalNature` = "time"
    - `unit` = "second"
- An electromagnetic pulse ExEy in the time domain :
  - `data` is a two dimensional real dataset
    - `physicalNature` = "electricField"
    - `unit attribute` = "voltPerMeter"
  - `ds/dim1` is a one dimensional string dataset
    - `physicalNature` = "component"
    - `unit` = ""
    - values of `ds/dim1` are ["x", "y"]
  - `ds/dim2` is a one dimensional real dataset
    - `physicalNature` = "time"
    - `unit` = "second"
- The definition of a resistance found out in the `/physicalModel/resistance` category :
  - `data` is a one dimensional real dataset
    - `physicalNature` = "resistance"

    * `unit` = "ohm"

- `ds/dim1` is a one dimensional real dataset
    * `PhysicalNature` = "frequency"
    * `Unit` = "hertz"

5.6.6 The `complement` group

An arraySet can have an optional `complement` group. If present, this group contains `floatingType` elements aiming at completing the meaning of the arraySet.

Example :

```
data.h5/
`-- floatingType/
    `-- $dataOne[@floatingType=arraySet
        |        @label=Current on the wire]
        |-- complement/
        |   `-- temperature[@floatingType=singleReal
        |                   @label=temperature
        |                   @physicalNature=temperature
        |                   @unit=kelvin
        |                   @value=215]
        |-- data[@label=current
        |        @physicalNature=current
        |        @unit=ampere]
        `-- ds/
            |-- dim1[@label=height
            |        @physicalNature=length
            |        @unit=meter]
            |-- dim2
            |-- dim3
            `-- dim4
```

In this example, `data.h5:/floatingType/$dataOne` is completed by the `singleReal` `temperature`.

5.6.7 ArraySet and Coordinate systems

For simple use cases, **Amelet HDF** defines conventions to express data in coordinate systems instead of creating a mesh and locating data on this mesh's entities with the (component, entity) paradigm.

Note: Images of this section come from Wikipedia (http://www.wikipedia.org)

The convention is based upon the `coordinateSystem` attribute, it can take the following values :

- `x`
- `xy`
- `xyz`
- `rtheta`
- `rhophiz`
- `rthetaphi`

Those values are explained hereafter.

Cartesian coordinate system

Given the definition of the cartesian coodinate systems in 2 and 3 dimensions :

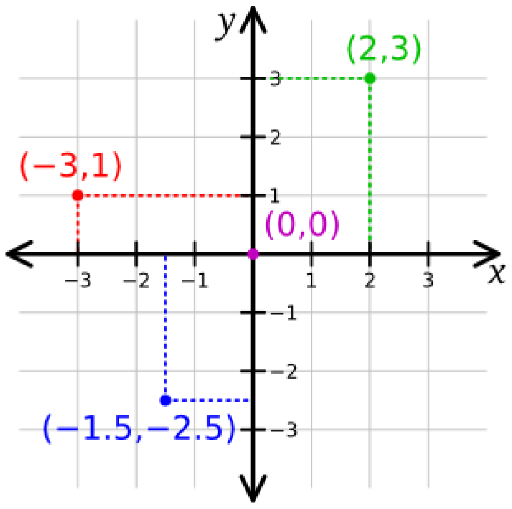


Fig. 5.4: A 2D cartesian coordinate system

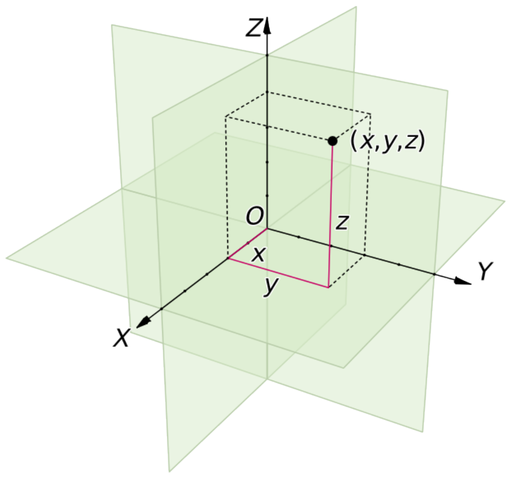


Fig. 5.5: A 3D cartesian coordinate system

- If an arraySet has the `coordinateSystem` attribute equals to `x`, the arraySet must have a dimension with `@label` equals `x`. This dimension represents the X axis of a one-dimensional cartesian coordinate system.
- If an arraySet has the `coordinateSystem` attribute equals to `xy`, the arraySet must have two consecutive dimensions with `@label` equals `x` and `@label` equals `y`. These dimensions represent the X axis and the Y axis of a two-dimensional cartesian coordinate system.
- If an arraySet has the `coordinateSystem` attribute equals to `xyz`, the arraySet must have three consecutive dimensions with `@label` equals `x`, `@label` equals `y` and `@label` equals `z`. These dimensions represent the X axis, the Y axis and the Z axis of a three-dimensional cartesian coordinate system.

Example with a three-dimensional coordinate system :

```
data.h5/
`-- floatingType/
    `-- $e_field[@floatingType=arraySet
        |        @coordinateSystem=xyz
        |        @label=electric field magniture in a box]
        |-- data[@label=electric field magniture
        |        @physicalNature=electricField
        |        @unit=voltPerMeter]
        `-- ds/
            |-- dim1[@label=x
            |        @physicalNature=length
            |        @unit=meter]
            |-- dim2[@label=y
            |        @physicalNature=length
            |        @unit=meter]
            `-- dim3[@label=z
                     @physicalNature=length
                     @unit=meter]
```

Cylindrical and polar coordinate systems

Given the definition of the polar coordinate system :

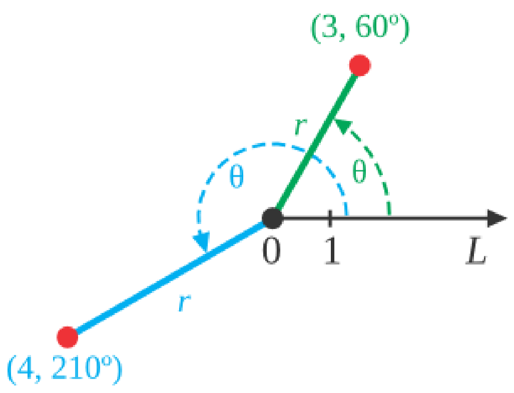


Fig. 5.6: A polar coordinate system

If an arraySet has the `coordinateSystem` attribute equals to `rtheta`, the arraySet must have two consecutive dimensions with `@label` equals `r` and `@label` equals `theta`. These dimensions represent the r parameter and the theta parameter of the polar coordinate system.

Example :

```
data.h5/
`-- floatingType/
    `-- $e_field[@floatingType=arraySet
        |        @coordinateSystem=rtheta
        |        @label=electric field magniture in a circle]
        |-- data[@label=electric field magniture
        |        @physicalNature=electricField
        |        @unit=voltPerMeter]
        `-- ds/
            |-- dim1[@label=r
            |        @physicalNature=length
            |        @unit=meter]
            `-- dim2[@label=theta
                     @physicalNature=angle
                     @unit=degree]
```

Given the definition of the cylindrical coordinate system :

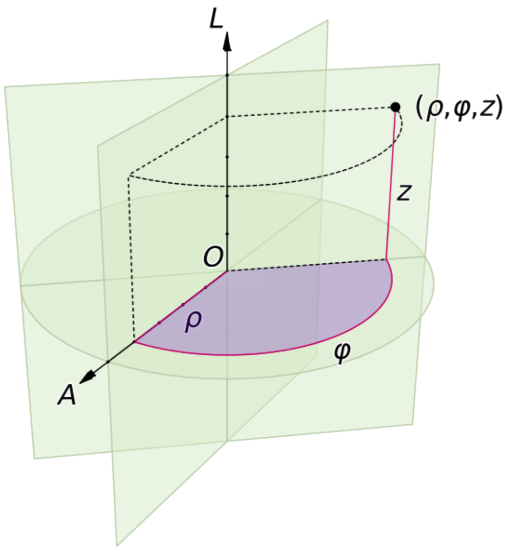


Fig. 5.7: A cylindrical coordinate system

If an arraySet has the `coordinateSystem` attribute equals to `rhophiz`, the arraySet must have three consecutive dimensions with `@label` equals `rho`, `@label` equals `phi` and `@label` equals `z`. These dimensions represent the rho parameter, the `phi` parameter and the `z` parameter of the cylindrical coordinate system.

Example :

```
data.h5/
`-- floatingType/
    `-- $e_field[@floatingType=arraySet
        |        @coordinateSystem=rhophiz
        |        @label=electric field magniture in a cylinder]
        |-- data[@label=electric field magniture
        |        @physicalNature=electricField
        |        @unit=voltPerMeter]
        `-- ds/
            |-- dim1[@label=rho
            |        @physicalNature=length
            |        @unit=meter]
            |-- dim2[@label=phi
            |        @physicalNature=length
            |        @unit=meter]
            `-- dim3[@label=z
                     @physicalNature=angle
                     @unit=degree]
```

Spherical coordinate system

Given the definition of the spherical coordinate system :

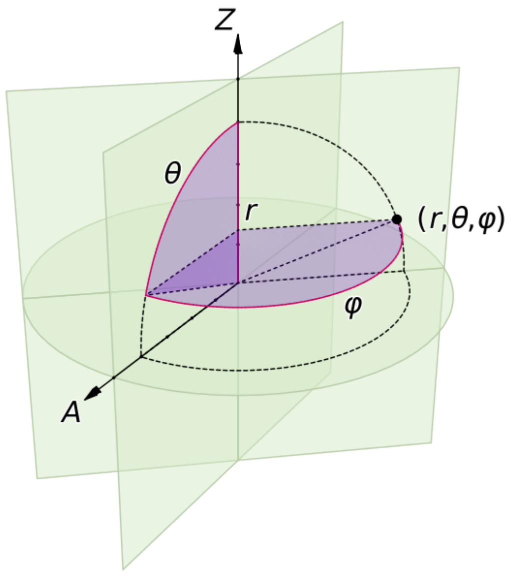


Fig. 5.8: A spherical coordinate system

If an arraySet has the `coordinateSystem` attribute equals to `rthetaphi`, the arraySet must have three consecutive dimensions with `@label` equals `r`, `@label` equals `theta` and `@label` equals `phi`. These dimensions represent the `r` parameter, the `theta` parameter and the `phi` parameter of the spherical coordinate system.

Example :

```
data.h5/
`-- floatingType/
    `-- $e_field[@floatingType=arraySet
        |        @coordinateSystem=rthetaphi
        |        @label=electric field magniture in a sphere]
        |-- data[@label=electric field magniture
        |        @physicalNature=electricField
        |        @unit=voltPerMeter]
        `-- ds/
            |-- dim1[@label=r
            |        @physicalNature=length
            |        @unit=meter]
            |-- dim2[@label=theta
            |        @physicalNature=angle
            |        @unit=degree]
            `-- dim3[@label=phi
                     @physicalNature=angle
                     @unit=degree]
```

5.7 Expression

5.7.1 Overview

Expression is a `floatingType` structure which defines data by a mathematical expression thanks to embedded `floatingTypes`.

A `floatingType` `expression` is an HDF5 group with a mandatory HDF5 string attribute named `expression`, it gives the value of the expression.

As usual, common `floatingType` attributes (`label`, `comment`, `physicalNature` and `unit`) are optional.

Example :

```
data.h5
|
`-- floatingType/
    `-- $U[@floatingType=expression
        |  @expression=$R*$I
        |  @physicalNature=voltage
        |  @unit=volt]
        |-- $R[@floatingType=dataSet
        |      @physicalNature=impedance
        |      @unit=ohm]
        `-- $I[@floatingType=dataSet
               @physicalNature=current
               @unit=ampere]
```

In the example `$U` is defined by the multiplication of `$R` and `$I`.

5.7.2 Simple cases

In the case of common binary operators `+, -, *, /`, **Amelet HDF** defines the following shortcut :

- The expression is reduced to the operator sign
- Operands are named `operand1` and `operand2`

Example : the preceding floatingType `data.h5:/floatingType/$U` can be re-written as :

```
data.h5
|
`-- floatingType/
    `-- $U[@floatingType=expression
        |  @expression=*
        |  @physicalNature=voltage
        |  @unit=volt]
        |-- operand1[@floatingType=dataSet
        |            @physicalNature=impedance
        |            @unit=ohm]
        `-- operand2[@floatingType=dataSet
                     @physicalNature=current
                     @unit=ampere]
```

CHAPTER

SIX

MESH

Computer simulations work well using a discretized space, this discretization of the space is the result of the mesh generation process of a CAD model.

Over-all the mesh data structure is made up of three kinds of elements :

- the nodes. The nodes are points in space and are the foundations of the mesh.
- the elements. The elements are geometrical shapes connecting neighboring nodes.
- the groups. The groups are sets of elements and represent areas, volumes that are important for the simulation.

A mesh can be either *structured* or *unstructured*.

6.1 The Mesh category

In **Amelet HDF** meshes are localized in the `/mesh` category and can be named by whatever word authorized by HDF5, its children are mesh groups.

6.1.1 Mesh group

Meshes are gathered inside "mesh groups". In fact, meshes are often made up of independent mesh part, like hybrid meshes for instance. (An hybrid mesh is mesh composed of unstructured meshes and structured meshes).

A mesh group is named HDF5 group. (see the `data.h5:/mesh/$gmesh1` group below)

Consequently, the name of the system is rather the group's name rather than the mesh name.

Example :

```
data.h5/
`-- mesh/
    |-- $gmesh1/
    `-- $gmesh2/
```

6.1.2 Mesh

Meshes are children of mesh groups with a mandatory attribute : `type`. `type` can take the following values :

- `unstructured` : the mesh is an unstructured mesh.
- `structured` : the mesh is an structured mesh.

Mesh's name and mesh group' name must have a length inferior to 20 characters.

Example of a `/mesh` category :

```
data.h5/
`-- mesh/
    |-- $gmesh1
    |     |-+ $mesh#5
    |     `-+ $mesh_6
    `-- $gmesh2
        |-- $mesh1[@type=unstructured]/
        |   |-- elementNodes
        |   |-- elementTypes
        |   |-- nodes
        |   |-- group
        |   |   |-- $field-location[@type=node]
        |   |   |-- $right-wing[@type=face]
        |   |   `-- $left-wing[@type=face]
        |   |-- groupGroup
        |   |   `-- $wings
        |   `-- selectorOnMesh
        |       |-- nodes
        |       |-- elements
        |       `-- groups
        |-+ $mesh-two
        `-- $mesh-3[@type=structured]/
            |-- cartesianGrid
            |-- group
            |   |-- $field-location[@type=node]
            |   |-- $right-wing[@type=face]
            |   `-- $left-wing[@type=face]
            |-- groupGroup
            |   `-- $wings
            `-- selectorOnMesh
                |-- nodes
                |-- elements
                `-- groups
```

In this example, `data.h5:/mesh/$gmesh2/$mesh1` is an *unstructured* mesh and `data.h5:/mesh/gmesh2/$mesh-3` is a *structured* mesh.

The next section present in details `unstructured` mesh and `structured` meshes.

6.2 Unstructured mesh

An unstructured mesh is a mesh based upon nodes dispatched in the space. Nodes are connected together by edges to form the mesh elements. Mesh elements can be :

- Line or 1d : edge
- Surface or 2d : triangle, quadrilateral, polygon ...
- Volume or 3d : tetrahedron, cube ...

Then elements are gathered to create groups of importance for the simulation. A group can represent a plane wing, a wheel, a dipole, a wire antenna, an output request location.

A mesh is a named group child of the `/mesh` group. A mesh mainly comprises three datasets :

The nodes dataset The nodes dataset contains the definition (x, y, z in 3d) of all nodes

The elementTypes dataset The elementTypes dataset contains the definition of the type of the mesh elements

The elements dataset The elements dataset contains the linked node's indices for all elements

Example :

```
data.h5
`-- mesh/
    `-- $gmesh1
        `-- mesh1[@type=unstructured]/
            |-- nodes
            |-- elementTypes
            `-- elementNodes
```

6.2.1 Attributes

The value of the attribute `type` of an unstructured mesh is `unstructured`.

6.2.2 nodes

All mesh nodes are defined in the `/mesh/$gmesh/$mesh/nodes` dataset. A node is defined by one to three coordinates :

- `x` in a 1d space
- `x`, `y` in a 2d space
- `x`, `y` and `z` in 3d space

x, y, z are real numbers.

The `nodes` dataset is a two dimensional real dataset, its dimensions are (number_of_nodes x number_of_space_dimensions).

The first column is x, the second is y and the third is z. The index of a node is the row's index in which it appears.

The index and columns header are implicit. The index starts at 0.

i	x	y	z
0	0.0	0.0	0.0
1	0.0	1.0	0.0
2	1.0	0.0	2.0

Here, three nodes are defined in a 3d space, the genuine dataset without headers is :

0.0	0.0	0.0
0.0	1.0	0.0
1.0	0.0	2.0

6.2.3 elementTypes

Explicit nodal approach

The elements of an unstructured mesh are defined by the nodal approach, that is to say that each element is defined by a set of nodes. For a given number of nodes, many geometric shapes are possible, then each shape has a type and the shape's type is the element's type.

Warning: Nodes are numbered and numbers are important for element definition and normal computation.

Implicit sub-element

In addition this section introduces the definition of (implicit) sub-elements (edges for `entityType=face`, edges and faces for `entityType=volume`) of an element, these sub-elements are numbered as well. Implicit sub-elements permit to select (n-1) dimensional sub-elements of an n dimensional element.

Warning: Implicit edges and faces are numbered.

Implicit sub-elements can be seen as the way to make a link with the mesh definition by the descendant approach (node, edge, face, volume) .

Predefined shapes types

The predefined shape types are detailed in the following sections.

Note: Node numbers are shown in black, edge numbers in red and face number in blue.

bar2

bar2 cell characteristics are :

- one-dimensional cell
- 2 nodes
- 1 edge
- **code : 1**

Edges are (defined by nodes) :

Edge number	Node 1	Node 2
1	1	2

The following figure shows a bar2 cell :

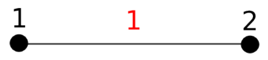


Fig. 6.1: A bar2 cell

bar3

bar3 cell characteristics are :

- one-dimensional cell
- 3 nodes
- 2 edges

- **code : 2**
- comment : 1 node is in the middle of the edge defined by the two endpoints

Edges are (defined by nodes) :

Edge number	Node 1	Node 2
1	1	3
2	3	2

The following figure shows a bar3 cell :

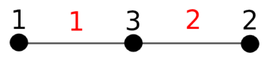


Fig. 6.2: A bar3 cell

tri3

tri3 cell characteristics are :

- two-dimensional cell
- 3 nodes
- 3 edges connecting all nodes each other and 1 face.
- **code : 11**

Edges are (defined by nodes) :

Edge number	Node 1	Node 2
1	1	2
2	2	3
3	3	1

Faces are (defined by nodes) :

Face number	Node 1	Node 2	Node 3
1	1	2	3

The following figure shows a tri3 cell :

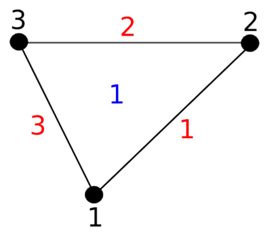


Fig. 6.3: A tri3 cell

tri6

tri6 cell characteristics are :

- two-dimensional cell

- 6 nodes
- 6 edges
- 1 face
- **code : 12**
- comment : 3 nodes are in the middle of the edges

Edges are (defined by nodes) :

Edge number	Node 1	Node 2
1	1	4
2	4	2
3	2	5
4	5	3
5	3	6
6	6	1

Faces are (defined by nodes) :

Face number	Node 1	Node 2	Node 3
1	1	2	3

The following figure shows a tri6 cell :

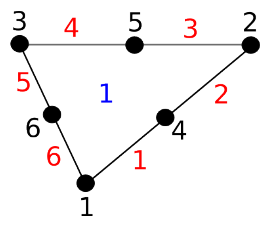


Fig. 6.4: A tri6 cell

quad4

quad4 cell characteristics are :

- two-dimensional cell
- 4 nodes
- 4 edges
- 1 face
- **code : 13**

Edges are (defined by nodes) :

Edge number	Node 1	Node 2
1	1	2
2	2	3
3	3	4
4	4	1

Faces are (defined by nodes) :

Face number	Node 1	Node 2	Node 3	Node 4
1	1	2	3	4

The following figure shows a quad4 cell :

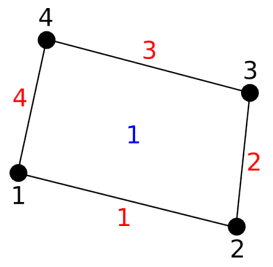


Fig. 6.5: A quad4 cell

quad8

quad8 cell characteristics are :

- two-dimensional cell
- 8 nodes
- 4 edges
- 1 face
- **code : 14**
- comment : 4 nodes in the middle of the edges

Edges are (defined by nodes) :

Edge number	Node 1	Node 2
1	1	5
2	5	2
3	2	6
4	6	3
5	3	7
6	7	4
7	4	8
8	8	1

Faces are (defined by nodes) :

Face number	Node 1	Node 2	Node 3	Node 4
1	1	2	3	4

The following figure shows a quad8 cell :

tetra4

tetra4 cell characteristics are :

- three-dimensional cell
- 4 nodes

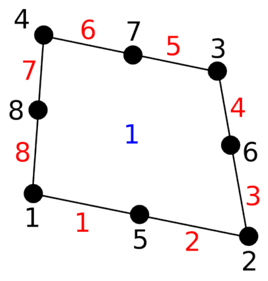


Fig. 6.6: A quad8 cell

- 6 edges connecting all node each other
- 4 faces
- **code : 101**

Edges are (defined by nodes) :

Edge number	Node 1	Node 2
1	1	2
2	2	3
3	3	1
4	1	4
5	2	4
6	3	4

Faces are (defined by nodes) :

Face number	Node 1	Node 2	Node 3
1	1	2	4
2	2	3	4
3	1	4	3
4	1	3	2

The following figure shows a tetra4 cell :

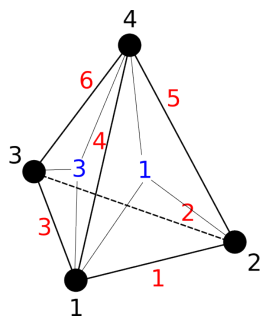


Fig. 6.7: A tetra4 cell

pyra5

pyra5 cell characteristics are :

- three-dimensional cell

- 5 nodes
- 8 edges
- 5 faces
- **code : 102**

Edges are (defined by nodes) :

Edge number	Node 1	Node 2
1	1	2
2	2	3
3	3	4
4	4	1
5	1	5
6	2	5
7	3	5
8	4	5

Faces are (defined by nodes) :

Face number	Node 1	Node 2	Node 3	Node 4
1	1	4	3	2
1	1	2	5	
1	2	3	5	
1	3	4	5	
1	1	5	4	

The following figure shows a pyra5 cell :

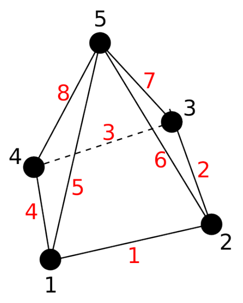


Fig. 6.8: A pyra5 cell

penta6

penta6 cell characteristics are :

- three-dimensional cell
- 6 nodes
- 9 edges
- 5 faces
- **code : 103**

Edges are (defined by nodes) :

Edge number	Node 1	Node 2
1	1	2
2	2	5
3	5	4
4	4	1
5	1	3
6	2	3
7	4	6
8	5	6
9	3	6

Faces are (defined by nodes) :

Face number	Node 1	Node 2	Node 3	Node 4
1	1	4	5	2
2	1	2	3	
3	4	6	5	
4	2	5	6	3
5	1	3	6	4

The following figure shows a penta6 cell :

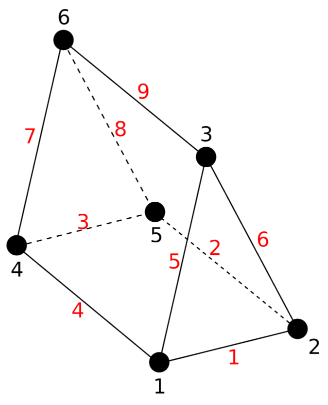


Fig. 6.9: A penta6 cell

hexa8

hexa8 cell characteristics are :

- three-dimensional cell
- 8 nodes
- 12 edges
- 6 faces
- **code : 104**

Edges are (defined by nodes) :

Edge number	Node 1	Node 2
1	1	2
2	2	5
3	5	4
4	4	1
5	1	3
6	2	3
7	4	6
8	5	6
9	3	6

Faces are (defined by nodes) :

Face number	Node 1	Node 2	Node 3	Node 4
1	1	4	3	2
2	1	2	6	5
3	2	3	7	6
4	3	4	8	7
5	1	5	8	4
6	5	6	7	8

The following figure shows a hexa8 cell :

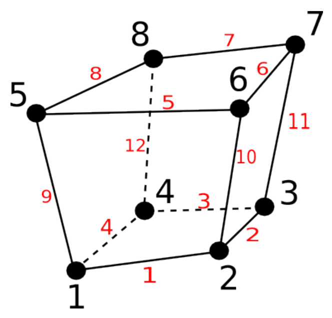


Fig. 6.10: A hexa8 cell

tetra10

tetra10 cell characteristics are :

- three-dimensional cell
- 10 nodes
- 12 edges connecting all node each other
- 4 faces
- **code : 108**

Edges are (defined by nodes) :

Edge number	Node 1	Node 2
1	1	2
2	2	3
3	3	1
4	1	4
5	2	4
6	3	4

Faces are (defined by nodes) :

Face number	Node 1	Node 2	Node 3
1	1	2	4
2	2	3	4
3	1	4	3
4	1	3	2

The following figure shows a tetra10 cell :

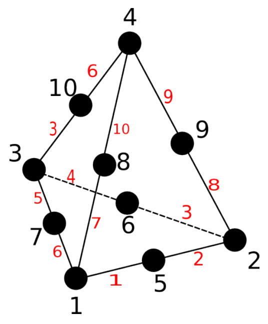


Fig. 6.11: A tetra10 cell

hexa20

hexa20 cell characteristics are :

- three-dimensional cell
- 20 nodes
- 24 edges connecting all node each other
- 4 faces
- **code : 109**

Edges are (defined by nodes) :

Edge number	Node 1	Node 2
1	1	9
2	9	2
3	2	10
4	10	3
5	3	11
6	11	4
7	4	12
8	12	1
9	5	13
10	13	6
11	6	14
12	14	7
13	7	15
14	15	8
15	8	16
16	16	5
17	1	17
18	17	5
19	2	18
20	18	6
21	3	19
22	19	7
23	4	20
24	20	8

Faces are (defined by nodes) :

Face number	Node 1	Node 2	Node 3	Node 4
1	1	4	3	2
2	1	2	6	5
3	2	3	7	6
4	3	4	8	7
5	1	5	8	4
6	5	6	7	8

The following figure shows a hexa20 cell :

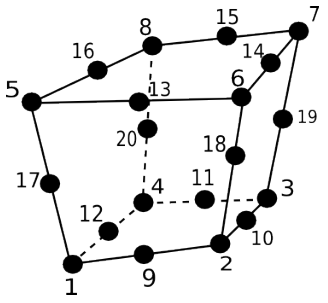


Fig. 6.12: A hexa20 cell

Canonical elements

Besides **Amelet HDF** defines some canonical geometrical elements.

Note: the vector connecting the initial node i with the terminating node j is written $\overrightarrow{N_{ij}}$

- plane
  - A plane is made up of 3 nodes.
    - $\overrightarrow{V_{12}}$ must not be aligned with $\overrightarrow{V_{13}}$
    - The order of nodes gives the plane orientation, the normal is defined by $\vec{n} = \overrightarrow{V_{12}} \times \overrightarrow{V_{13}}$
  - code : 15

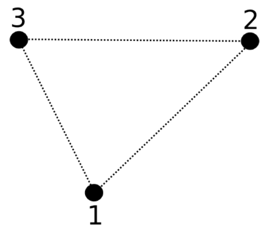


Fig. 6.13: A plane element

- circle
  - A circle is defined by three nodes :
    - The node 1 is the center of the circle
    - The node 2 gives the radius
    - The node 3 gives the plane containing the circle
    - $\overrightarrow{V_{12}}$ must not be aligned with $\overrightarrow{V_{13}}$
    - The order of nodes gives the plane orientation, the normal is defined by $\vec{n} = \overrightarrow{V_{12}} \times \overrightarrow{V_{13}}$
  - code : 16

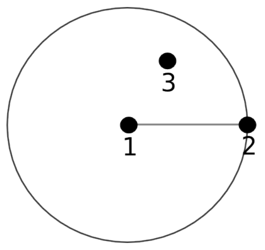


Fig. 6.14: A circle element

- ellipse
  - An ellipse is defined by three nodes :
    - Node 1 and node 2 are the foci
    - The node 3 is on the ellipse and gives the “radius” as well as the plane which contains the ellipse
    - $\overrightarrow{V_{12}}$ must not be aligned with $\overrightarrow{V_{13}}$
    - The order of nodes gives the plane orientation, the normal is defined by $\vec{n} = \overrightarrow{V_{12}} \times \overrightarrow{V_{13}}$

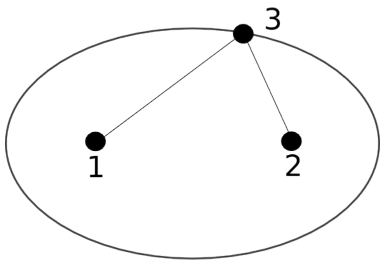


Fig. 6.15: An ellipse element

  - code : 17

- cylinder
  - A cylinder is defined by three nodes :
    * The node 1 is the center of the base circle
    * The node 2 gives the radius of the base circle
    * The node 3 is the center of the top circle.
    * Nodes 1 and 3 are the axis of the cylinder.
    * $\overrightarrow{V_{12}}$ and $\overrightarrow{V_{13}}$ are orthogonal
  - code : 105

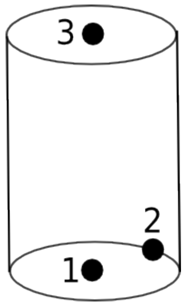


Fig. 6.16: A cylinder element

- cone
  - A cone is defined by four nodes :
    * The node 1 is the center of the base circle
    * The node 2 gives the radius of the base circle
    * The node 3 is the center of the top circle
    * The node 4 gives the radius of the top circle.
    * Nodes 1 and 3 gives the axis of the cone.
    * $\overrightarrow{V_{13}}$ and $\overrightarrow{V_{12}}$ are orthogonal
    * $\overrightarrow{V_{13}}$ and $\overrightarrow{V_{34}}$ are orthogonal
  - code : 106
- sphere

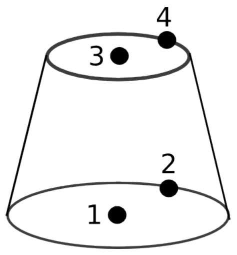


Fig. 6.17: A cone element

- A sphere is defined by two nodes :
    * The node 1 is the center of the sphere
    * The node 2 gives the radius of the sphere
- code : 107

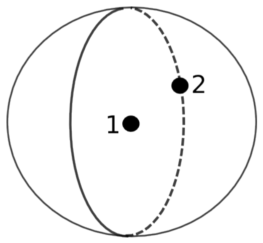


Fig. 6.18: A sphere element

The elementTypes dataset

`/mesh/$gmesh/$mesh/elementTypes` is a one dimensional 8-bits integer dataset and gives the type of all elements in the mesh.

Warning: `/mesh/$gmesh/$mesh/elementTypes` is a 8-bits integer dataset

The size of `elementTypes` is the number of elements.

Note: It is simple to compute the number of elements of a given type by summing the number of time a type appears in `elementTypes`.

index	Element's types
0	1
1	2
2	11

In this example, three elements are defined (two bar2 and one tri3). The index is implicit.

The genuine dataset is

1
2
102

Recap table

1D	
bar2	1
bar3	2
2D	
tri3	11
tri6	12
quad4	13
quad8	14
plane	15
circle	16
ellipse	17
3D	
tetra4	101
pyra5	102
penta6	103
hexa8	104
cylinder	105
cone	106
sphere	107
tetra10	108
hexa20	109

6.2.4 elementNodes

Elements are defined by the nodes. The type of the element gives the number of nodes, the dataset `elementNodes` gives the nodes involved in the element definition. `/mesh/$gmesh/$mesh/elementNodes` is a one dimensional integer dataset.

For each row of `/mesh/$gmesh/$mesh/elementTypes`, there is a matching number_of_nodes. In `/mesh/$gmesh/$mesh/elementNodes` there are number_of_nodes rows indicating the index of each involved nodes (defined in `/mesh/$gmesh/$mesh/nodes`).

Example for the following `/mesh/$gmesh/$mesh/elementTypes` :

1
1
11

and `/mesh/$gmesh/$mesh/elementNodes` is

index	Element's node
0	0
1	1
2	1
3	2
4	0
5	2
6	3

According to the preceding `elementTypes` example :

- The first bar2 is defined by the nodes 0 and 1
- The second bar2 by the nodes 1 and 2

- The tri3 by the nodes 0, 2 et 3.

The genuine dataset is without headers :

0
1
1
2
0
2
3

6.2.5 group

Until now, a mesh is a set of nodes and a set of elements. CAD or topology areas (like plane's wings, thin wires) are groups (of elements for instance). Those groups are `/mesh/$gmesh/$mesh/group`'s children.

`/mesh/$gmesh/$mesh/group` is an HDF5 group and contains HDF5 integer datasets.

These datasets are either sets of nodes indices from `/mesh/$gmesh/$mesh/nodes` (the attribute `type` equals `node`) or sets of elements indices from `/mesh/$gmesh/$mesh/elementTypes` (the attribute `type` equals `element`).

If `type` equals `element` the group has another attribute : `entityType`. `entityType` is an hdf5 string attribute and gives the type of entities stored in the group. `entityType` can take the following values :

- `edge` : the group contains only edge elements.
- `face` : the group contains only surface elements.
- `volume` : the group contains only volume elements.

Name's length of datasets must have less than 20 characters.

Example :

```
data.h5
`-- mesh/
    `-- $gmesh1/
        `-- $plane[@type=unstructured]/
            |-- nodes
            |-- elementTypes
            |-- elementNodes
            `-- group
                |-- $field-location[@type=node]
                |-- $right-wing[@type=element
                |                @entityType=face]
                `-- $left-wing[@type=element
                               @entityType=face]
```

`/mesh/$gmesh1/$mesh1` has three groups :

- `$field-location`, a node group which the location where the field will be computed.
- `$right-wing`, an element group which reprensets the right wing of a plane
- `$left-wing`, an element group which reprensets the left wing of a plane

For example `/mesh/$gmesh1/$mesh1/group/$field-location` is

index	indices of nodes from `/mesh/$gmesh1/$mesh1/nodes`
0	0
1	1
2	1
3	2
4	0
5	2
6	3

The index and headers are reported for convenience.

6.2.6 groupGroup

`groupGroup` is an HDF5 group and contains sets of `group` children. `groupGroup` children are named HDF5 datasets of 20 character string, each groupGroup is a set group's names.

Name's length of sets must have less than 20 characters.

Example :

```
data.h5
`-- mesh/
    `-- $gmesh1/
        `-- $mesh1[@type=unstructured]/
            |-- nodes
            |-- elementTypes
            |-- elementNodes
            |-- group
            |   |-- $field-location[@type=node]
            |   |-- $right-wing[@type=element
            |   |               @entityType=face]
            |   `-- $left-wing[@type=element]
            |                  @entityType=face]
            `-- groupGroup
                `-- $wings
```

and `/mesh/$gmesh1/$mesh1/groupGroup/$wings` is

index	`/mesh/$gmesh1/$mesh1/group`'s children names
0	$right-wing
1	$left-wing

The index and headers are reported for convenience.

Note: It is possible to create groupGroups of groupGroups.

6.3 Structured mesh

The main difference between an unstructured mesh and a structured mesh is that the nodes and the elements of a structured mesh are implicitly defined by a grid. A cartesian grid is characterized by three real vectors of `x`, `y` and `z` . Therefore elements can be located by 3 integers (i, j, k) that are the integer coordinates in the grid.

A structured mesh is a named HDF5 group child of `/mesh` having an attribute `type` equals `structured`.

A structured mesh is composed of three children :

- A mandatory child :
  - a `cartesianGrid` group
- Two optional children :
  - a `group` HDF5 group
  - a `groupGroup` HDF5 group

Structured meshes are mainly used in electromagnetism by the Finite Difference Time Domain method (FDTD).

6.3.1 Cartesian grid

A cartesian grid is defined by 1 to 3 axis (vectors) and is the equivalent structure of the set (nodes, elementTypes, elementNodes) for an unstructured mesh. The nodes are the intersection of axis, the elementTypes is implicit because elements type is bar, face or hexa (depending on the dimension) and elementNodes are all bar, face or hexa located by their coordinates (i, j, k).

If the grid's dimension is 1, the cartesian grid is defined by a child vector (one dimensional dataset) called `x` :

```
data.h5
`-- mesh/
    `-- $gmesh1/
        `-- $structured-mesh-1d[@type=structured]/
            `-- cartesianGrid
                `-- x[@floatingType=vector
                      @physicalNature=length
                      @unit=meter]
```

x is a `floatingType` = `vector`, i.e. one dimensional dataset of reals, its optional attributes are :

- the optional attribute `floatingType` = `vector`
- The optional attribute `physicalNature` value is `length`
- The optional attribute `unit` value is `meter`

This attribute are optional because **Amelet HDF** specification set them.

If the grid dimension is 2, the cartesian grid is defined by 2 children floatingType called x, y :

```
data.h5
`-- mesh/
    `-- $gmesh1/
        `-- $structured-mesh-2d[@type=structured]/
            `-- cartesianGrid
                |-- x[@floatingType=vector
                |     @physicalNature=length
                |     @unit=meter]
                `-- y[@physicalNature=length
                      @unit=meter]
```

If the grid dimension is 3, the cartesian grid is defined by 3 children floatingType called x, y and z :

```
data.h5
`-- mesh/
    `-- $gmesh1
        `-- $structured-mesh-3d[@type=structured]/
            `-- cartesianGrid
                |-- x[@physicalNature=length
                |     @unit=meter]
                |-- y[@physicalNature=length
                |     @unit=meter]
                `-- z[@physicalNature=length
                      @unit=meter]
```

the physical nature of x, y and z is `length` and the unit is `meter`.

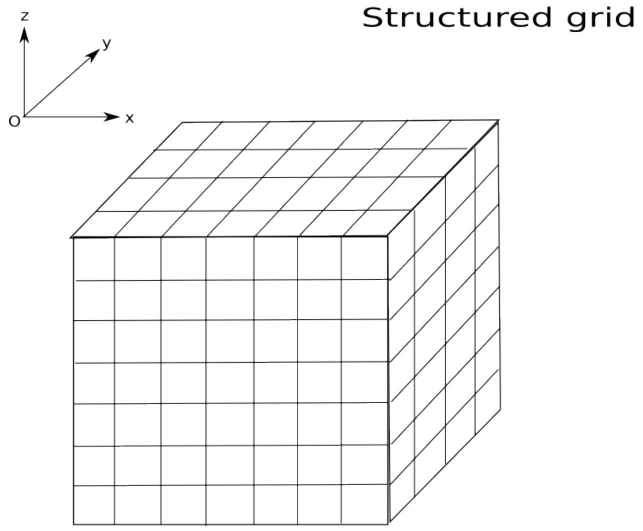


Fig. 6.19: A structured orthogonal grid

Element numbering

Two manners exist to locate/describe elements and structured sets of elements.

- The numbering by indices : Elements are located by their indices in the cartesian grid :
  - i in 1D
  - (i, j) in 2D
  - (i, j, k) in 3D
- The implicit numbering : Elements are located by a conventional numbering rule. Every element has an integer number.

Here is an example on implicit numbering :

- for a `nx` 1D grid : the number of the ith element is its index `i`
- for a `nx * ny` 2D grid, the number of the (i, j) element is `i+(j-1)*nx`
- for a `nx * ny * nz` 3D grid, the number of the (i, j, k)th element is `i + (j-1)*nx + (k-1)*ny*nx`

For a 3D grid, the index increases rapidly. A 5000x1000x1000 grid implies 5 000 000 000 mesh cells. However, the maximum 32 bit integer is 4 294 967 296 and we should use 64 bit integer to express the implicit index. The gain is not obvious relative the complexity of the convention. Therefore, the implicit numbering is not used in **Amelet HDF**.

Moreover, in the manner of unstructured mesh elements (see *elementTypes*) , **Amelet HDF** introduces the sub-element concept to structured mesh. With an explicit numbering, it is possible to select an element's sub-part like a cube's edge.

Node

A node is defined by an integer tuple, the size of the tuple depends on the space dimension :

Space dimension	A's coordinates
1D	i
2D	(i, j)
3D	(i, j, k)

Warning: The indexes of the nodes begins with **0**

The nodes numbering inside a cell is defined in the same manner as :

- the unstructured *bar2* numbering in **1D**
  - A is the point `1`
- the unstructured *quad4* numbering in **2D**
  - A is the point `1`.
  - B is the point `3`.
- the unstructured *hexa8* numbering in **3D**
  - A is the point `1`.
  - B is the point `7`.

Edge

An edge is defined with two aligned nodes A & B. Following the space dimension, A's and B's indices can take values reported in the tabular hereafter :

Space dimension	A's coordinates	B's coordinates
1D	i	i+1
2D	(i, j)	(i+1, j)
	(i, j)	(i, j+1)
3D	(i, j, k)	(i+1, j, k)
	(i, j, k)	(i, j+1, k)
	(i, j, k)	(i, j, k+1)

The edges numbering inside a cell is defined in the same manner as :

- the unstructured *bar2* numbering in **1D**
  - A is the point `1`
- the unstructured *quad4* numbering in **2D**
  - A is the point `1`.
  - B is the point `3`.
- the unstructured *hexa8* numbering in **3D**
  - A is the point `1`.
  - B is the point `7`.

Face

A face is defined with two nodes A & B. Following the space dimension, A's and B's indices can take values reported in the tabular hereafter :

Space dimension	A's coordinates	B's coordinates
2D	(i, j)	(i+1, j+1)
3D	(i, j, k)	(i+1, j+1, k)
	(i, j, k)	(i, j+1, k+1)
	(i, j, k)	(i+1, j, k+1)

The faces numbering inside a cell is defined in the same manner as :

- the unstructured *quad4* numbering in **2D**
  - A is the point `1`.
  - B is the point `3`.
- the unstructured *hexa8* numbering in **3D**
  - A is the point `1`.
  - B is the point `7`.

Volume

A volume is defined with two nodes A & B.

Space dimension	A's coordinates	B's coordinates
3D	(i, j, k)	(i+1, j+1, k+1)

6.3.2 group

`group` is an HDF5 group and contains sets of nodes and sets of elements. `group` children are named HDF5 dataset of integers.

Name's length of sets must have less than 20 characters.

Nodes group

A nodes group of a structured mesh is an HDF5 two dimensional dataset children of `/mesh/$gmesh/$mesh`.

A nodes group has an HDF5 attribute `type`, its value is `node`.

The first dimension is the rows. Each row defines a node by 3 integers representing the coordinates of the node.

Example :

```
data.h5
`-- mesh
    `-- $gmesh1
        `-- $mesh1[@type=structured]/
            |-- cartesianGrid
            `-- group
                `-- $e-field[@type=node]
```

where `data.h5:/mesh/$gmesh1/$mesh1/group/$e-field` is :

i	j	k
1	1	1
8	10	2
15	15	15

Headers are reported for convenience.

Elements group

An elements group of a structured mesh is an HDF5 two dimensional dataset children of `/mesh/$gmesh/$mesh`.

An elements group has an HDF5 attribute `type`, its value is `element`, in addtition, it has the `entityType` attribute. `entityType` is an hdf5 string attribute and gives the type of entities stored in the group. `entityType` can take the following values :

- `edge` : the group contains only edge elements.
- `face` : the group contains only surface elements.
- `volume` : the group contains only volume elements.

The first dimension is the rows. Each row defines 6 integers representing the lowest corner and the highest corner of a parallelepiped.

The lowest corner indices are the `imin`, `jmin`, `kmin` and the highest corner indices are `imax`, `jmax` and `kmax`.

Example :

```
data.h5
`-- mesh/
    `-- $gmesh1/
        `-- $mesh1[@type=structured]/
            |-- cartesianGrid
            `-- group
                `-- $right-wing[@type=element
                                @entityType=volume]
```

where `data.h5:/mesh/$gmesh1/$mesh1/group/$right-wing` is :

imin	jmin	kmin	imax	jmax	kmax
1	1	1	12	10	12
15	15	15	27	25	27

The parallelepiped ((imin, jmin, kmin), (imax, jmax, kmax)) represents either a single element or a set of elements.

If `type` equals `element` the group has another attribute : `entityType`. `entityType` is an hdf5 string attribute and gives the type of entities stored in the group. `entityType` can take the following values :

- `edge` : the group contains only edge elements.
- `face` : the group contains only surface elements.
- `volume` : the group contains only volume elements.

6.3.3 The `normal` group

The normals definition is usually related to the surface concept. However this can be also applied to the edges.

Normals of faces

In **Amelet HDF** specification, normal faces are contained in the `/mesh/$gmesh/$mesh/normal` group in datasets named as the initial face group in `/mesh/$gmesh/$mesh/group`.

The possible values of the `/mesh/$gmesh/$mesh/normal/$normal` dataset are :

- x+ : the normal is along the x axis and positively oriented.
- x- : the normal is along the x axis and negatively oriented.
- y+ : the normal is along the y axis and positively oriented.
- y- : the normal is along the y axis and negatively oriented.
- z+ : the normal is along the z axis and positively oriented.
- z- : the normal is along the z axis and negatively oriented.

Example :

```
data.h5
`-- mesh/
    `-- $gmesh1/
        `-- $mesh1[@type=structured]/
            |-- cartesianGrid
            |-- group
            |   `-- $right-wing[@type=element
            |                   @entityType=face]
            `-- normal/
                `-- $right-wing
```

where `data.h5:/mesh/$gmesh1/$mesh1/group/$right-wing` is :

imin	jmin	kmin	imax	jmax	kmax
1	1	1	12	10	12
15	15	15	27	25	27

and `data.h5:/mesh/$gmesh1/$mesh1/normal/$right-wing` is :

normal
x+
y+

In the example `data.h5:/mesh/$gmesh1/$mesh1/group/$right-wing` are the normals of the face group `data.h5:/mesh/$gmesh1/$mesh1/normal/$right-wing`.

Note: The number of rows of `data.h5:/mesh/$gmesh1/$mesh1/group/$right-wing` and `data.h5:/mesh/$gmesh1/$mesh1/normal/$right-wing` must be **equal**.

Normals of edges

In **Amelet HDF**, structured edges are defined by six integers : imin, jmin, kmin, imax, jmax, kmax. By extension and by misnomer the normals give the direction of edges.

The same conventions as for surfaces are applied.

6.3.4 groupGroup

`groupGroup` is an HDF5 group and contains sets of `group` children. `groupGroup` children are named HDF5 dataset of 20 characters string, each groupGroup is a set group's names.

Name's length of sets must have less than 20 characters.

Example :

```
data.h5
`-- mesh/
    `-- $gmesh1
        `-- mesh1[@type=structured]/
            |-- cartesianGrid
            |-- group
            |   |-- $field-location[@type=node]
            |   |-- $right-wing[@type=element
            |   |               @entityType=cell]
            |   `-- $left-wing[@type=element
            |                  @entityType=cell]
            `-- groupGroup
                `-- $wings
```

and `/mesh/$gmesh1/$mesh1/groupGroup/$wings` is

index	`/mesh/$gmesh1/$mesh1/group`'s children names
0	$right-wing
1	$left-wing

The index is implicit reported for convenience.

Note: It is possible to create groupGroups of groupGroups.

6.4 Selector on mesh

In a mesh, nodes are necessary contained in the `nodes` dataset. However, it could be interesting to spot the middle of an edge or the center of a face... Similarly, it could nice to select a face edge without creating an edge element in `elementTypes`.

This is the role of the `/mesh/$gmesh/$mesh/selectorOnMesh` group localized in mesh instances :

```
data.h5/
`-- mesh
    `-- $gmesh1
        |-- $mesh1
        |   |-- nodes
        |   |-- elementTypes
        |   |-- elementNodes
        |   |-- group
        |   |-- groupGroup
        |   `-- selectorOnMesh
        |-- $mesh-two
        |-- $mesh-3
        |   |-- nodes
        |   |-- elementTypes
        |   |-- elementNodes
        |   |-- group
        |    |-- groupGroup
        |    `-- selectorOnMesh
        |-- $mesh#5
        |-- $mesh_6
        `-- meshLink
```

The `selectorOnMesh` group offers the mean to select and label mesh points and mesh sub-element (The `meshLink` group will be covered in the next section).

`selectorOnMesh` is an HDF5 group which contains two kind of children :

- named tables
- named integer datasets

6.4.1 selectorOnMesh of `pointInElement` type

A `selectorOnMesh` of `pointInElement` type is a named **table** which contains point definitions in element's local coordinate systems. The shape of the table depends on the `unstructured|structured` nature of the mesh.

The table has a string attribute named `type` of value `pointInElement` :

```
data.h5/
`-- mesh/
    `-- $gmesh1/
        `-- $mesh1/
            |-- nodes
            |-- elementTypes
            |-- elementNodes
            |-- group/
            |-- groupGroup/
            `-- selectorOnMesh/
                `-- $point_in_element[@type=pointInElement]
```

Unstructured mesh

It is possible to spot a point in an element thanks to a local coordinate system built by one, two or three vectors (depending on the dimension of the element).

Definition : $\vec{e_{ij}}$ is the vector starting from the node i toward the node j of an element.

- if E is a one-dimensional element, only $\vec{v_1}$ is used and is defined by :

Cell	$\vec{v_1}$
bar2	$\vec{e_{12}}$
bar3	$\vec{e_{12}}$

  In the $(O, \vec{v_1})$ coordinate system (O is node 1), the identification of a point P is realized by the vector $\overrightarrow{OP}$ using a real number α : $\overrightarrow{OP} = \alpha\vec{v_1}$, P must be in the element.

- if E is a two-dimensional element, $(\vec{v_1}, \vec{v_2})$ are used and are defined by :

Cell	$\vec{v_1}$	$\vec{v_2}$
tri3	$\vec{e_{12}}$	$\vec{e_{13}}$
tri6	$\vec{e_{12}}$	$\vec{e_{13}}$
quad4	$\vec{e_{12}}$	$\vec{e_{14}}$
quad8	$\vec{e_{12}}$	$\vec{e_{14}}$

In the $(O, \vec{v_1}, \vec{v_2})$ coordinate system (O is node 1), the identification of a point P is realized by the vector $\overrightarrow{OP}$ using two real numbers α, β : $\overrightarrow{OP} = \alpha\vec{v_1} + \beta\vec{v_2}$, P must be in the element.

- if E is a three-dimension cell, $(\vec{v_1}, \vec{v_2}, \vec{v_3})$ are used and are defined by :

Cell	$\vec{v_1}$	$\vec{v_2}$	$\vec{v_3}$
tetra4	$\vec{e_{12}}$	$\vec{e_{13}}$	$\vec{e_{14}}$
pyra5	$\vec{e_{12}}$	$\vec{e_{14}}$	$\vec{e_{15}}$
penta6	$\vec{e_{12}}$	$\vec{e_{14}}$	$\vec{e_{13}}$
hexa8	$\vec{e_{12}}$	$\vec{e_{14}}$	$\vec{e_{15}}$

In the $(O, \vec{v_1}, \vec{v_2}, \vec{v_3})$ coordinate system (O is node 1), the identification of a point P is realized by the vector $\overrightarrow{OP}$ using three real numbers α, β, γ : $\overrightarrow{OP} = \alpha\vec{v_1} + \beta\vec{v_2} + \gamma\vec{v_3}$, P must be in the element.

Note: The default value of a v* is -1, it is the "not used" value.

Therefore, `pointInElement` `selectorOnMesh` is a four columns HDF5 table.

index	v1	v2	v3

The four columns are :

- `index` : the index of the element in the list of elements (`elementTypes`). `index` is an integer.
- `v1` : the relative distance α along $\vec{v_1}$. `v1` is real
- `v2` : the relative distance β along $\vec{v_2}$. `v2` is real
- `v3` : the relative distance γ along $\vec{v_3}$. `v3` is real

Warning: Distances are normalized

Examples for the mesh `data.h5:/mesh/$gmesh1/mesh1` :

```
data.h5/
`-- mesh
    `-- $gmesh1
        `-- $mesh1[@type=unstructured]
            |-- nodes
            |-- elementTypes
            |-- elementNodes
            |-- group
            |   |-- $field-location[@type=node]
            |   |-- $right-wing[@type=element]
            |   `-- $left-wing[@type=element]
            |-- groupGroup
            |   `-- $wings
            `-- selectorOnMesh
                 `-- $points_in_elements[@type=pointInElement]
```

`/mesh/$gmesh1/$mesh1/elementTypes` is (implicit index are not reported), the mesh is composed of two bar2 element.

1
1

and `data.h5:/mesh/$gmesh1/$mesh1/selectorOnMesh/$points_in_elements` is

index	v1	v2	v3
0	0.5	-1	-1
1	0.5	-1	-1

This example defines the center of the two edges.

Structured mesh

`selectorOnMesh` of `pointInElement` type defines named entities referenced relative to an element E. A point can be localized in the element by a local coordinate system $(\vec{v_1}, \vec{v_2}, \vec{v_3})$ (in the 3D case).

- if E is an edge, $\vec{v_1}$ is used
  - If E is directed along $\overrightarrow{Ox}$,
    - $\vec{v_1} = \overrightarrow{dx}$
  - If E is directed along $\overrightarrow{Oy}$,
    - $\vec{v_1} = \overrightarrow{dy}$
  - If E is directed along $\overrightarrow{Oz}$,
    - $\vec{v_1} = \overrightarrow{dz}$
- if E is a face, $(\vec{v_1}, \vec{v_2})$ are used
  - If E is in xOy
    - $\vec{v_1} = \overrightarrow{dx}$
    - $\vec{v_2} = \overrightarrow{dy}$
  - If E is in xOz
    - $\vec{v_1} = \overrightarrow{dx}$
    - $\vec{v_2} = \overrightarrow{dz}$
  - If E is in yOz
    - $\vec{v_1} = \overrightarrow{dy}$
    - $\vec{v_2} = \overrightarrow{dz}$
- if E is a volume, $(\vec{v_1}, \vec{v_2}, \vec{v_3})$ are used
  - $\vec{v_1} = \overrightarrow{dx}$
  - $\vec{v_2} = \overrightarrow{dy}$
  - $\vec{v_3} = \overrightarrow{dz}$

$\vec{dx}(respectively\vec{dy}$ and $\vec{dz})$ is the oriented dimension of the cell along x-axis, (respectively y-axis z-axis).

Note: The default value of a v* is -1, it is the "not used value".

Therefore, `pointInElement` `selectorOnMesh` is a nine columns HDF5 table.

imin	jmin	kmin	imax	jmax	kmax	v1	v2	v3

The nine columns are :

- `imin` : the i index of the bottom corner node. `imin` is an integer.
- `jmin` : the j index of the bottom corner node. `jmin` is an integer.
- `kmin` : the k index of the bottom corner node. `kmin` is an integer.

- `imax` : the i index of the top corner node. `imax` is an integer.
- `jmax` : the j index of the top corner node. `jmax` is an integer.
- `kmax` : the k index of the top corner node. `kmax` is an integer.
- `v1` : distance in the direction x. `v1` is a real.
- `v2` : distance in the direction y. `v2` is a real.
- `v3` : distance in the direction z. `v3` is a real.

Note: The following rules must be followed :

- (`imax` - `imin`) = 0 or 1
- (`jmax` - `jmin`) = 0 or 1
- (`kmax` - `kmin`) = 0 or 1

One can see that if :

- `imin` = `imax` or `jmin` = `jmax` or `kmin` = `kmax` E is a face
- `imin` = `imax` and (`jmin` = `jmax` or `kmin` = `kmax`) E is an edge (and respectively for the other permutations)

Examples for the mesh `data.h5:/mesh/$gmesh1/$mesh1` :

```
data.h5/
`-- mesh/
    `-- $gmesh1/
        `-- $mesh1[@type=structured]/
            |-- cartesianGrid
            |-- group/
            |   `-- $e-field[@type=node]
            |-- groupGroup/
            `-- selectorOnMesh/
                `-- $points_in_elements[@type=pointInElement]
```

`data.h5:/mesh/$gmesh1/$mesh1/selectorOnMesh/$points_in_elements` is

imin	jmin	kmin	imax	jmax	kmax	v1	v2	v3
1	1	1	2	2	2	0.5	0.5	0.5
1	1	1	1	2	2	0.5	0.5	-1
1	1	1	1	1	2	0.5	-1	-1

The first point is the center of a volume, the second the center of a face and the last is the center of an edge.

6.4.2 `selectorOnMesh` of `edge|face` type

A `selectorOnMesh` of `edge|face` type is a named **dataset** and defines a list of sub-elements.

The dataset has a string attribute named `type` of value :

- `edge` : the dataset references edges
- `face` : the dataset references faces

This kind of `selectorOnMesh` exists only for `unstructured` mesh since it is natural to select whatever entity in a `structured` mesh thanks the `group` concept.

It is possible to spot a sub-element of an element (i.e. an edge, a face) thanks to a local numbering of sub-elements described in *Unstructured mesh*. The local numbering depends on the `type` attribute.

`edge|face selectorOnMesh` is a two columns HDF5 integer **dataset**.

The two columns are :

- The index of the element in the list of elements (in `elementTypes`).
- The number of the sub-element

Examples for the mesh `data.h5:/mesh/$gmesh1/mesh1` :

```
data.h5/
`-- mesh
    `-- $gmesh1
        `-- $mesh1[@type=unstructured]
            |-- nodes
            |-- elementTypes
            |-- elementNodes
            |-- group
            |   |-- $field-location[@type=node]
            |   |-- $right-wing[@type=element]
            |   `-- $left-wing[@type=element]
            |-- groupGroup
            |   `-- $wings
            `-- selectorOnMesh
                 `-- $implicit_edges[@type=edge]
```

`/mesh/$gmesh1/$mesh1/elementTypes` is (implicit index are not reported). Two bar2 and a tetra4 compose the mesh.

1
1
101

and `data.h5:/mesh/$gmesh1/$mesh1/selectorOnMesh/$implicit_edges` is

0	1
1	1
2	3

This example defines a two element edge group :

- The first edge of a bar2, it is the bar2 itself
- The third edge between node 1 and node 3 of a tetra4 cell.

6.5 The meshLink group

Sometimes a mesh can be composed of several meshes or a 2D mesh is merged with a 3D mesh. In these cases, the two meshes must be linked by defining equalities between couples of entities.

This is the task of the `/mesh/$gmesh/meshLink` category. A meshLink is a named HDF5 two dimensional integer *dataset* with three attributes :

- `type`, this is the type of the link defined by an HDF5 string attribute. The type can be `nodes` or `element`
  - If `type` is `node`, the index in the dataset refer to the `/mesh/$gmesh/$mesh/nodes`
  - If `type` is `edge`, the index in the dataset refer to edges in `/mesh/$gmesh/$mesh/elementTypes`
  - If `type` is `face`, the index in the dataset refer to faces in `/mesh/$gmesh/$mesh/elementTypes`

  - If `type` is `volume`, the index in the dataset refer to volumes in `/mesh/$gmesh/$mesh/elementTypes`

- `mesh1`, this is an HDF5 string attribute, it is the name of the first `/mesh/$gmesh/$mesh` implied in the link.
- `mesh2`, this is an HDF5 string attribute, it is the name of the second `/mesh/$gmesh/$mesh` implied in the link.

The name's length must be less than 20 characters.

Example of `/mesh/$gmesh1/meshLink` :

```
data.h5
`-- mesh
    `-- $gmesh1
        |-- $mesh1[@type=unstructured]
        |   |-- nodes
        |   |-- elementTypes
        |   `-- elementNodes
        |-- $mesh2[@type=unstructured]
        |   |-- nodes
        |   |-- elementTypes
        |   `-- elementNodes
        |-- $mesh3[@type=structured]
        |   `-- cartesianGrid
        |-- $mesh4[@type=structured]
        |   `-- cartesianGrid
        `--meshLink
            |-- $ml1[@type=node
            |        @mesh1=/mesh/$gmesh1/$mesh1
            |        @mesh2=/mesh/$gmesh1/$mesh2]
            |-- $ml2[@type=node
            |        @mesh1=/mesh/$gmesh1/$mesh3
            |        @mesh2=/mesh/$gmesh1/$mesh4]
            |-- $ml3[@type=volume
            |        @mesh1=/mesh/$gmesh1/$mesh1
            |        @mesh2=/mesh/$gmesh1/$mesh2]
            `-- $ml4[@type=volume
                     @mesh1=/mesh/$gmesh1/$mesh3
                     @mesh2=/mesh/$gmesh1/$mesh4]
```

The entity in located in the `selectorOnMesh/nodes` table and the `selectorOnMesh/elements` tables in the case of an unstructured mesh.

6.5.1 Link between two unstructured meshes

An element or node link between two unstructured meshes is defined by an HDF5 dataset with a rank of 2 and with dimensions equal number_of_entities x two. The two columns represent :

- i1 : the index i of the node in the first unstructured mesh's `nodes` dataset
- i2 : the index i of the node in the second unstructured mesh's `nodes` dataset

i1	i2
0	0
1	3
2	6

Headers are reported for convenience

6.5.2 Link between two structured meshes

Nodes

A node link between two structured meshes (`/mesh/$gmesh/meshLink/$ml2`) is defined by an HDF5 dataset with a rank of 2 and with dimensions equal number_of_entities x six. The six columns represent :

- i1, j1, k1 : indices of the node in the first mesh
- i2, j2, k2 : indices of the node in the second mesh

i1	j1	k1	i2	j2	k2
1	1	1	2	2	2
2	2	2	3	3	3

Headers are reported for convenience

Element

An element link between two structured meshes (`/mesh/$gmesh/meshLink/$ml4`) is defined by an HDF5 dataset with a rank of 2 and with dimensions equal number_of_entities x 12. The six columns represent :

- imin1, jmin1, kmin1, imax1, jmax1, kmax1 : indices of the element in the first mesh
- imin2, jmin2, kmin2, imax2, jmax2, kmax2 : indices of the element in the second mesh

imin1	jmin1	kmin1	imax1	jmax1	kmax1	imin2	jmin2	kmin2	imax2	jmax2	kmax2
1	1	1	2	2	2	1	1	1	2	2	2
2	2	2	1	1	1	2	2	2	3	3	3

Headers are reported for convenience

6.5.3 Link between one unstructured mesh and one structured mesh

Nodes

An node link between one unstructured mesh and one structured mesh is defined by an HDF5 dataset with a rank of 2 and with dimensions equal number_of_entities x four. The four columns represent :

- i1 : the index of the node in the unstructured mesh's `nodes`
- i2, j2, k2 : indices of the node in the second mesh

i1	i2	j2	k2
1	1	1	1
2	2	2	2

Headers are reported for convenience

Element

An element link between one unstructured mesh and one structured mesh is defined by an HDF5 dataset with a rank of 2 and with dimensions equal number_of_entities x seven. The seven columns represent :

- i1 : the indice of the node in the unstructured mesh's `nodes`
- imin2, jmin2, kmin2, imax2, jmax2, kmax2 : indices of the element in the second mesh

i1	imin2	jmin2	kmin2	imax2	jmax2	kmax2
1	1	1	1	2	2	2
2	2	2	2	3	3	3

Headers are reported for convenience

6.5.4 Link between one structured mesh and one unstructured mesh

Nodes

An node link between one structured mesh and one unstructured mesh is defined by an HDF5 dataset with a rank of 2 and with dimensions equal number_of_entities x four. The four columns represent :

- i1, j1, k1 : indices of the node in the second mesh
- i2 : the index of the node in the unstructured mesh's `nodes`

i1	j1	k1	i2
1	1	1	1
2	2	2	2

Headers are reported for convenience

Element

An element link between one structured mesh and one unstructured mesh is defined by an HDF5 dataset with a rank of 2 and with dimensions equal number_of_entities x seven. The seven columns represent :

- imin1, jmin1, kmin1, imax1, jmax1, kmax1 : indices of the element in the second mesh
- i2 : the index of the node in the unstructured mesh's `nodes`

imin1	jmin1	kmin1	imax1	jmax1	kmax1	i2
1	1	1	2	2	2	1
2	2	2	3	3	3	2

Headers are reported for convenience

CHAPTER

SEVEN

GLOBAL ENVIRONMENT

The global environment category contains mainly the context of a simulation.

Global environments are HDF5 named groups children of `/globalEnvironment`.

Example :

```
data.h5
`-- globalEnvironment
    |-- $ge1
    `-- $ge2
```

`data.h5:/globalEnvironment/$ge1` and `data.h5:/globalEnvironment/$ge1` are two global environment instances.

7.1 Time and frequency domains

If the simulation is in the frequency domain : `/globalEnvironment/$globalEnvironment/frequency` contains the frequencies the computation will be performed at.

If the simulation is in the time domain : `/globalEnvironment/$globalEnvironment/time` is the definition of the simulation time.

`/globalEnvironment/$globalEnvironment/frequency`'s characteristics are :

- is a `floatingType`
- `physicalNature` is `frequency`
- `unit` is `hertz`

`/globalEnvronment/$globalEnvironment/time`'s characteristics are :

- is a `floatingType`
- `physicalNature` is `time`
- `unit` is `second`

Note: If `floatingType` equals `vector`, it is a float vector.

Examples

```
data.h5
`-- globalEnvironment
    `-- $ge1
        `-- time[floatingType=vector]
```

```
data.h5
`-- globalEnvironment
    `-- $ge1
        `-- frequency[@floatingType=logarithmListOfReal1
                      @first=10e3
                      @last=1e9
                      @numberOfValues=100]
```

Note: For some temporal methods only the maximum time (tmax) is relevant, in fact, the simulation is performed for time in the interval [0, tmax], use an *Interval* for this case :

```
data.h5
`-- globalEnvironment
    `-- $ge1
        `-- time[@floatingType=linearListOfReal2
                 @first=0
                 @last=1e-6]
```

7.2 Limit conditions

To gain a lot computation time, it is often necessary to model a system with symmetries. The computation domain is then terminated with particular limit conditions.

Limit conditions can be :

- `electricWall`. An electric wall is positioned as if a perfect electrical conductor was in the mesh.
- `magneticWall`. A magnetic wall is positioned, the mirror effect is relative to the magnetic component.

Limit conditions are given by a `/globalEnvironment/$ge/limitConditions` HDF5 group.

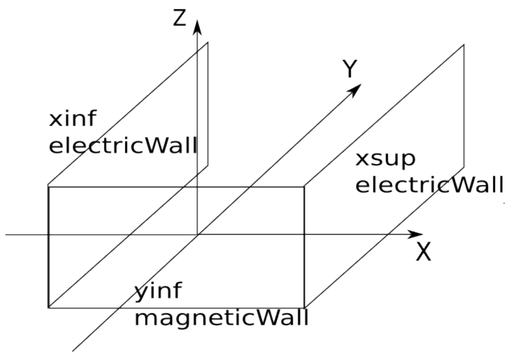
For a cartesian coordinate system, the computation limits are located by six string HDF5 attributes :

- `xinf` locates the inferior-x limit, the first plan orthogonal to the axis-x
- `xsup` locates the superior-x limit
- `xinf` locates the inferior-y limit
- `ysup` locates the superior-y limit
- `zinf` locates the inferior-z limit
- `zsup` locates the superior-z limit

example :

```
data.h5
`-- globalEnvironment
    `-- $ge1
        `-- limitConditons[@xinf=electricWall
                           @xsup=electricWall
                           @yinf=magneticWall]
```

The example defined two electric symmetries and one magnetic symmetry as in the sketch below :


Z
Y
xinf
electricWall
xsup
electricWall
X
yinf
magneticWall

CHAPTER

EIGHT

ELECTROMAGNETIC SOURCES

Amelet HDF defines six kinds of electromagnetic sources :

- Plane wave
- Spherical wave
- Generator
- Dipole
- Antenna
- Source on mesh

They are all associated with an **Amelet HDF** category :

```
electromagneticSource/
|-- planeWave/
|-- generator/
|-- antenna/
`-- sourceOnMesh/
```

8.1 The magnitude element

Elements in `/electromagneticSource` often contain a `magnitude` child that is a `floatingType` and represents the magnitude of a source (current generator, voltage generator, plane wave...).

`magnitude` has two important attributes :

- `delay`. If the magnitude is defined in the time domain, `delay` which is a real HDF5 attribute for a time value in second represents a delay to add to the values of the floatingType.
- `automaticMaximumValue`. `automaticMaximumValue` is a real HDF5 attribute. If it is present, magnitude values must be scaled in order to make the maximum value magnitude equals `automaticMaximumValue`.

8.2 Plane wave

The category `/electromagneticSource/planeWave` contains plane wave definitions.

A plane wave is defined by :

- A null phase point (xo, yo, zo) in meter.
- A direction of propagation in polar coordinates, in degree:

  - θ, where $\theta \in [0, \pi]$
  - φ, where $\varphi \in [0, 2\pi[$
- A polarization type :
  - linear, it is defined by the angle ψ in degree
  - elliptical, it is defined by E_θ and E_φ that are complex electric components
- An electric field magnitude in **volt per meter**

The null phase point (xo, yo, zo) appears as three HDF5 real attributes xo, yo, zo.

In the global cartesian coordinate system, a spherical coordinate system is defined by the two angles (θ, φ) which induce the three unit vectors $(\vec{u}_r, \vec{u}_\theta, \vec{u}_\varphi)$.

The propagation vector is $\vec{k} = -\frac{2\pi f}{c}.\vec{u}_r$

and the coordinates of $(\vec{u}_r, \vec{u}_\theta, \vec{u}_\varphi)$ in the global (x, y, z) coordinate system are :

$$\vec{u}_r = \begin{pmatrix} \sin\theta.\cos\varphi \\ \sin\theta.\sin\varphi \\ \cos\theta \end{pmatrix}, \vec{u}_\theta = \begin{pmatrix} +\cos\theta.\cos\varphi \\ +\cos\theta.\sin\varphi \\ -\sin\theta \end{pmatrix} \text{ and } \vec{u}_\varphi = \begin{pmatrix} -\sin\varphi \\ +\cos\varphi \\ 0 \end{pmatrix}$$

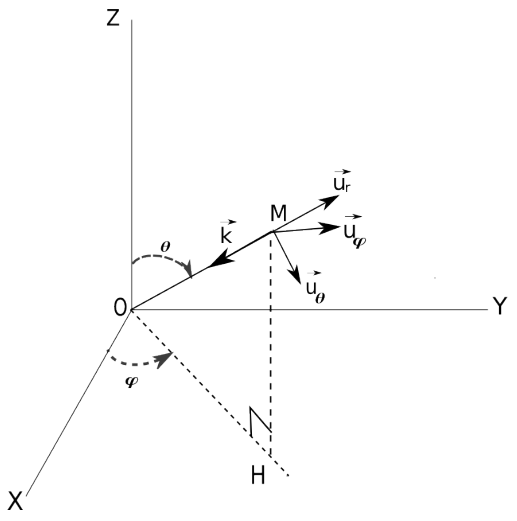


Fig. 8.1: Wave plane definition

The electric field vector is $\vec{E} = E_o e^{-j\vec{k}.\vec{r}}(E_\theta.\vec{u}_\theta + E_\varphi.\vec{u}_\varphi)$ where E_o is the magnitude, E_θ and E_φ are complex numbers (this allows to describe linear polarization and elliptical polarization)

Warning: E_θ and E_φ satisfy the condition $\|E_\theta.\vec{u}_\theta + E_\varphi\vec{u}_\varphi\|$ = 1

The magnetic field is $\vec{H} = \frac{1}{\eta\|\vec{k}\|}.\vec{k} \wedge \vec{E}$, η is the wave impedance of the medium.

In **Amelet HDF** a plane wave is described as follows. A plane wave is a named HDF5 group. The name's length must have less than 20 characters.

The direction of propagation is given by :

- `theta`, a real HDF5 attribute, it is an angle in degree and corresponds to θ
- `phi`, a real HDF5 attribute, it is an angle in degree and corresponds to φ

The polarization can be "linear" or "elliptic" :

- If the polarization is linear, the HDF5 real attribute `linearPolarization` gives the value of the polarization in degree, it corresponds to ψ defined by $E_\theta = \sin\psi$ and $E_\varphi = \cos\psi$.

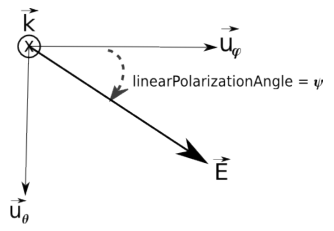


Fig. 8.2: Linear polarization definition

- If the polarization is elliptical, the HDF5 complex attributes `ellipticalPolarizationETheta` (corresponding to E_θ) and `ellipticalPolarizationEPhi` (corresponding to E_φ) give the definition of the polarization.

The `magnitude` is an **Amelet HDF** floatingType in volt per meter.

Example :

```
data.h5
`-- electromagneticSource
    `-- planeWave
        |-- $plane-wave1[@xo=0.0
        |   |            @yo=0.0
        |   |            @zo=0.0
        |   |            @theta=0.0
        |   |            @phi=0.0
        |   |            @linearPolarization=0.0]
        |    `-- magnitude[@floatingType=arraySet]
        `-- $ellipt-wave1[@xo=0.0
            |             @yo=0.0
            |             @zo=0.0
            |             @theta=0.0
            |             @phi=0.0
            |             @ellipticalPolarizationETheta=(0.0, 0.1)
            |             @ellipticalPolarizationEPhi=(0.0, 0.0)]
            `-- magnitude[@floatingType=arraySet]
```

8.3 Spherical wave

The category `/electromagneticSource/sphericalWave` contains spherical wave definitions.

A spherical wave is defined by :

- A null phase point (xo, yo, zo) in meter.
- A power magnitude in **watt**. `magnitude` is a `floatingType`

Example :

```
data.h5
`-- electromagneticSource
    `-- sphericalWave
        `-- $a_spherical_wave[@xo=0.0
```

```
            |                  @yo=0.0
            |                  @zo=0.0]
            `-- magnitude[@floatingType=arraySet]
```

8.4 Generator

Generators are multi-pole electric devices and there are four kinds of generator in **Amelet HDF** :

- the voltage generator
- the current generator
- the power generator
- the power density generator

In **Amelet HDF**, generators are in the `/electromagneticSource/generator` category.

A generator in a named HDF5 group, the name's length must have less than 20 characters.

A generator has two `floatingType` childs :

- `innerImpedance` representing its inner impedance in ohm.
- `magnitude` which is the magnitude of the signal. It is a real **Amelet HDF** parameter, its unit is
  - `volt` if it is a voltage generator
  - `ampere` if is a current generator
  - `watt` if it is a power generator
  - `wattPerCubicMeter` if it is a power density generator

In addition it can have three optional attributes :

- `delay`, HDF5 real attribute in second, it is time delay applied to the `magnitude`
- `initialValue`, an HDF5 real attribute, it is initial value expressed in same unit as the `magnitude`
- `maximumValue`, an HDF5 real attribute, the maximum value is fixed by this attribute, a function is applied to `magnitude` to obtain those values, same unit as `magnitude`.

Example

```
data.h5
|-- physicalModel
|   `-- multiport
|       `-- $imp1
`-- electromagneticSource
    `-- generator
        `-- $a-generator[@type=current]
            |-- innerImpedance[@floatingType=singleComplex
            |                  @physicalNature=impedance
            |                  @value=(10,0)]
            `-- magnitude[@floatingType=singleReal
                             @value=10.0]
```

8.5 Dipole

The category `/electromagneticSource/dipole` contains dipole definitions.

A dipole is a thin wire (the wire radius is `wireRadius` in meter), its `type` can be `electric` (with a `length` in meter) or `magnetic` (with a `radius` in meter). Then, it is localized in the 3d space (`x`, `y`, `z`) and rotated (`theta`, `phi`).

A dipole has a `floatingType` child `loadImpedance` representing its load impedance in ohm.

Example, this **Amelet HDF** instance has two dipoles definition `data.h5:/electromagneticSource/dipole/elec-dipole` and `data.h5:/electromagneticSource/dipole/mag-dipole`

```
data.h5
`-- electromagneticSource
    `-- dipole
        |-- $elec-dipole[@type=electric
        |   |            @x=0.0
        |   |            @y=0.0
        |   |            @z=0.0
        |   |            @theta=0.0
        |   |            @phi=0.0
        |   |            @length=0.2
        |   |            @wireRadius=1e-4]
        |   |-- innerImpedance[@floatingType=singleComplex
        |   |                  @physicalNature=impedance
        |   |                  @value=(10,0)]
        |   `-- magnitude[@floatingType=singleReal
        |                 @delay=0.0
        |                 @value=10.0]
        `-- $mag-dipole[@type=magnetic
            |           @x=0.0
            |           @y=0.0
            |           @z=0.0
            |           @theta=0.0
            |           @phi=0.0
            |           @radius=0.2
            |           @wireRadius=1e-4]
            |-- innerImpedance[@floatingType=singleComplex
            |                  @physicalNature=impedance
            |                  @value=(10,0)]
            `-- magnitude[@floatingType=singleReal]
```

8.6 Antenna

Amelet HDF proposes a general description of an antenna, however some predefined antennas are suitable. An antenna is a named group child of `/electromagneticSource/antenna` category.

An antenna is defined by some electrical characteristics and radiating characteristics.

8.6.1 Electrical properties

The electrical properties are :

- The efficiency. The `efficiency` is an optional real HDF5 attribute.

- The input impedance. The `inputImpedance` is an optional `floatingType` element, its `physicalNature` is `impedance` vs `frequency`.
- The load impedance. The `loadImpedance` is an optional `floatingType` element, its `physicalNature` is `impedance` vs `frequency`.
- The feeder impedance. The `feederImpedance` is an optional `floatingType` element, its `physicalNature` is `impedance` vs `frequency`.
- The magnitude. The `magnitude` is an optional `floatingType` element.

```
data.h5
`-- electromagneticSource
    `-- antenna
        `-- $antenna1[@efficiency=0.5]
            |-- inputImpedance[@floatingType=singleComplex
            |                  @physicalNature=impedance
            |                  @value=(50,0)]
            |-- loadImpedance[@floatingType=singleComplex
            |                 @physicalNature=impedance
            |                 @value=(10,0)]
            |-- feederImpedance[@floatingType=singleComplex
            |                   @physicalNature=impedance
            |                   @value=(10,0)]
            `-- magnitude[@floatingType=arraySet]
```

8.6.2 Radiation properties

The radiation properties of an antenna can be numerical or expressed thanks to a predefined model. The type of an antenna is expressed in a `model` child HDF5 group which has a `type` string attribute :

Example for a rectangular horn :

```
data.h5
`-- electromagneticSource/
    `-- antenna/
        `-- $antenna1[@efficiency=0.5]
            `-- model[@type=rectangularHorn
                      ...]
```

Numerical values

An antenna can be described by **one** of the following quantities :

- The gain
  - it depends on theta, phi and the frequency.
  - `gain` is a `floatingType` element child of the `model` group .
  - `type` equals `gain`
- The effective area
  - It depends on theta, phi and the frequency.
  - `effectiveArea` is a `floatingType` element child of the `model` group.
  - `type` equals `effectiveArea`
- The far field

- It depends on theta, phi and the frequency.
- `farField` is a `floatingType` element child of the `model` group.
- `type` equals `farField`

All these quantity are expressed in the global spherical coordinate system :

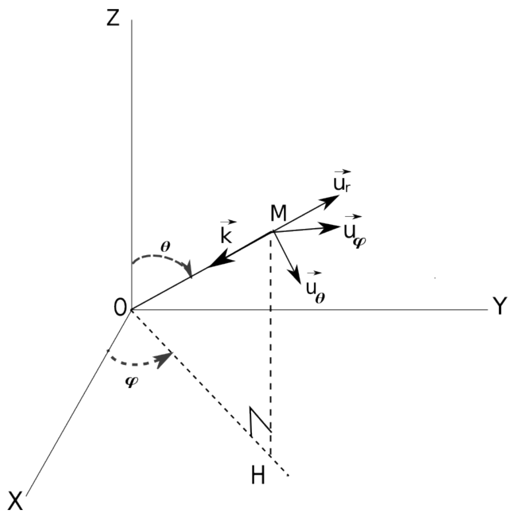


Fig. 8.3: Spherical coordinate system definition

Example of an antenna defined by its gain :

```
data.h5
`-- electromagneticSource/
    `-- antenna/
        `-- $antenna1[@efficiency=0.5]
            `-- model[@type=gain]
                `-- gain[floatingType=arraySet]
```

Predefined antennas

In addition, an antenna can be one of the following predefined types :

- A rectangular horn optionally with a reflector. The horn's aperture is in the xOy plane and radiates toward the positive-z.
  - `apertureLargestDimension`.
    - It is a real HDF5 attribute.
    - It is the aperture's size in the largest dimension
    - A length in meter
  - `apertureSmallestDimension`
    - It is a real HDF5 attribute.
    - It is the aperture's size in the smallest dimension
    - A length in meter
  - `flareAngleLargestDimension`

    * It is a real HDF5
    * It is the flare angle in the largest dimension
    * A angle in degree

  - `flareAngleSmallestDimension`
    * It is a real HDF5
    * It is the flare angle in the smallest dimension
    * A angle in degree

- A circular horn optionally with a reflector. The horn's aperture is in the xOy plane and radiates toward the positive-z .
  - `apertureDiameter`
    * It is a real HDF5 attribute.
    * It is the aperture's diameter
    * A length in meter
  - `flareAngle`
    * It is a real HDF5
    * It is the flare angle of the horn
    * A angle in degree
- A whip antenna. The whip antenna is orthogonal to the xOy plane.
  - `length`
    * It is a real HDF5 attribute.
    * It is the whip length
    * A length in meter
  - `radius`
    * It is a real HDF5 attribute.
    * It is the whip radius
    * A length in meter
- A log periodic antenna. The array of dipole is in yOz plane and radiates toward the positive-z.
  - `AngularAperture`
    * It is a real HDF5
    * It is the angular aperture of the antenna
    * A angle in degree
  - `scaleFactor`
    * It is a real HDF5
    * It is the scale factor between two consecutive dipole
    * without dimension
  - `firstDipoleLength`

    * It is a real HDF5 attribute.
    * It is the length of the first dipole
    * A length in meter

  - `lastDipoleLength`
    * It is a real HDF5 attribute.
    * It is the length of the last dipole
    * A length in meter

The following sketch summarizes the convention orientation for the predefined antennas :

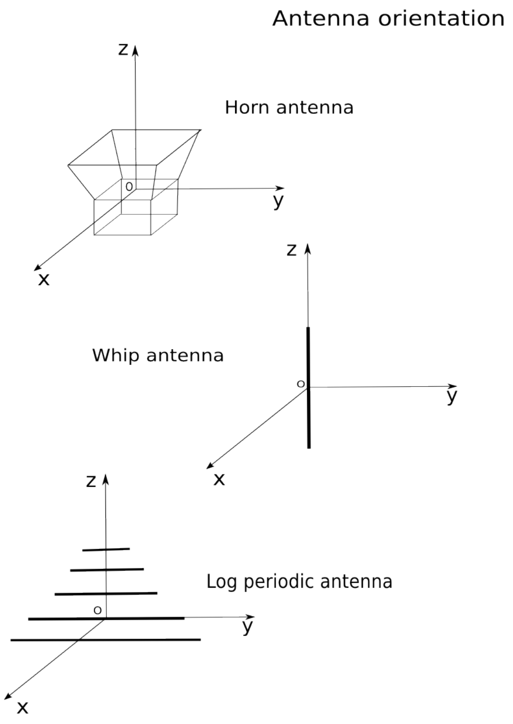


Fig. 8.4: Predefined antenna

- A generic antenna is defined by
  - An `angularAperture` real HDF5 attribute which corresponds to the angular aperture of the antenna.
  - A `pattern` HDF5 string attribute, its possible values are :
    * `omnidirectional`
    * `gaussian`
    * `cosecante`

Note: A generic antenna has an axis-z revolution symmetry

We have seen that a horn antenna could have a reflector. Only parabolic reflector is defined in **Amelet HDF**.

A parabolic reflector is a `parabolicReflector` HDF5 group contained in the `model` group if `type` is `rectangularHorn` or `circularHorn`.

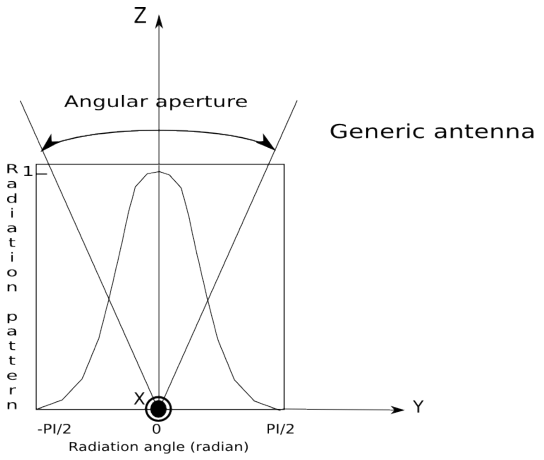


Fig. 8.5: Generic antenna

Parabolic reflector has three mandatory attributes

- `type`. `type` is a string HDF5 attribute and is either `circular` or `rectangular`
  - If `type` is `circular`, the `diameter` attribute is used to give the diameter of the reflector. It is a real HDF5 attribute in meter
  - If `type` is `rectangular`, the `length` and `width` attributes are used to give the length and the width of

    the reflector. It is a real HDF5 attribute in meter
- `focalLength`. It is a real HDF5 attribute in meter, it represents the focal length of the reflector.
- `aspectAngle`

Example for a rectangular horn with a circular reflector :

```
data.h5
`-- electromagneticSource/
    `-- antenna/
        `-- $antenna1[@efficiency=0.5]
            `-- model[@type=rectangularHorn
                |       ...]
                `-- parabolicReflector[@focalLength=0.2
                                       @aspectAngle=1.
                                       @type=circular
                                       @diameter=1.0]
```

8.6.3 Definition by example

A rectangular horn antenna with a circular reflector :

```
data.h5
`-- electromagneticSource
    `-- antenna
        `-- $rectHorn[@efficiency=0.5]
            |-- inputImpedance[@floatingType=singleComplex
            |                  @physicalNature=impedance
            |                  @value=(50,0)]
            |-- loadImpedance[@floatingType=singleComplex
            |                 @physicalNature=impedance
            |                 @value=(10,0)]
            |-- feederImpedance[@floatingType=singleComplex
            |                   @physicalNature=impedance
            |                   @value=(10,0)]
            |-- magnitude[@floatingType=arraySet
            |             @delay=1e-6]                 # measure : time
            |           ...
            `-- model[@type=rectangularHorn
                |     @apertureLargestDimension=0.2    # measure : length
                |     @apertureSmallestDimension=0.1   # measure : length
                |     @flareAngleLargestDimension=0.2  # measure : angle
                |     @flareAngleSmallestDimension=0.1 # measure : angle
                `-- parabolicReflector[@focalLength=0.2
                                       @aspectAngle=1.
                                       @type=circular
                                       @diameter=1.0]
```

A rectangular horn antenna with a rectangular reflector :

```
data.h5
`-- electromagneticSource
    `-- antenna
        `-- $rectHorn[@efficiency=0.5]
            |-- inputImpedance[@floatingType=singleComplex
            |                  @physicalNature=impedance
            |                  @value=(50,0)]
            |-- loadImpedance[@floatingType=singleComplex
            |                 @physicalNature=impedance
            |                 @value=(10,0)]
            |-- feederImpedance[@floatingType=singleComplex
            |                   @physicalNature=impedance
            |                   @value=(10,0)]
            |-- magnitude[@floatingType=arraySet
            |             @delay=1e-6]                  # measure : time
            |           ...
            `-- model[@type=rectangularHorn
                |     @apertureLargestDimension=0.2     # measure : length
                |     @apertureSmallestDimension=0.1    # measure : length
                |     @flareAngleLargestDimension=0.2   # measure : angle
                |     @flareAngleSmallestDimension=0.1  # measure : angle
                `-- parabolicReflector[@focalLength=0.2 # measure : length
                                       @aspectAngle=1.
                                       @type=rectangular
                                       @length=1.0      # measure : length
                                       @with=1.0]       # measure : length
```

A circular horn antenna :

```
data.h5
`-- electromagneticSource
    `-- antenna
        `-- $circHorn[@efficiency=0.5]
            |-- inputImpedance[@floatingType=singleComplex
            |                  @physicalNature=impedance
            |                  @value=(50,0)]
            |-- loadImpedance[@floatingType=singleComplex
            |                 @physicalNature=impedance
            |                 @value=(10,0)]
            |-- feederImpedance[@floatingType=singleComplex
            |                   @physicalNature=impedance
            |                   @value=(10,0)]
            |-- magnitude[@floatingType=arraySet
            |             @delay=1e-6]                 # measure : time
            |           ...
            `-- model[@type=circularHorn
                      @apertureDiameter=0.2            # measure : length
                      @flareAngle=0.2                  # measure : angle
                     ]
```

A log periodic antenna :

```
data.h5
`-- electromagneticSource
    `-- antenna
        `-- $logperiodic[@efficiency=0.5]
            |-- inputImpedance[@floatingType=singleComplex
            |                  @physicalNature=impedance
            |                  @value=(50,0)]
            |-- loadImpedance[@floatingType=singleComplex
            |                 @physicalNature=impedance
            |                 @value=(10,0)]
            |-- feederImpedance[@floatingType=singleComplex
            |                   @physicalNature=impedance
            |                   @value=(10,0)]
            |-- magnitude[@floatingType=arraySet
            |             @delay=1e-6]
            |           ...
            `-- model[@type=logPeriodic
                      @apertureAngle=0.2
                      @scaleFactor=0.1
                      @firstDipoleLength=0.1
                      @lastDipoleLength=0.7]
```

A whip antenna :

```
data.h5
`-- electromagneticSource
    `-- antenna
        `-- $whip[@efficiency=0.5]
            |-- inputImpedance[@floatingType=singleComplex
            |                  @physicalNature=impedance
            |                  @value=(50,0)]
            |-- loadImpedance[@floatingType=singleComplex
            |                 @physicalNature=impedance
            |                 @value=(10,0)]
            |-- feederImpedance[@floatingType=singleComplex
            |                   @physicalNature=impedance
            |                   @value=(10,0)]
            |-- magnitude[@floatingType=arraySet
            |             @delay=1e-6]                 # measure : time
            |           ...
            `-- model[@type=whip
                      @length=0.2                      # measure : length
                      @radius=0.1]                     # measure : angle
```

A generic antenna :

```
data.h5
`-- electromagneticSource
    `-- antenna
        `-- $generic[@efficiency=0.5]
            |-- inputImpedance[@floatingType=singleComplex
            |                  @physicalNature=impedance
            |                  @value=(50,0)]
            |-- loadImpedance[@floatingType=singleComplex
            |                 @physicalNature=impedance
            |                 @value=(10,0)]
            |-- feederImpedance[@floatingType=singleComplex
            |                   @physicalNature=impedance
            |                   @value=(10,0)]
            |-- magnitude[@floatingType=arraySet
            |             @delay=1e-6]
            `-- model[@type=generic
                      @angularAperture=0.1
                      @pattern=gaussian]                # omnidirectional
                                                        # gaussian
                                                        # cosecante
```

An antenna defined by a gain :

```
data.h5
`-- electromagneticSource
    `-- antenna
        `-- $by-gain[@efficiency=0.5]
            |-- inputImpedance[@floatingType=singleComplex
            |                  @physicalNature=impedance
            |                  @value=(50,0)]
            |-- loadImpedance[@floatingType=singleComplex
            |                 @physicalNature=impedance
            |                 @value=(10,0)]
            |-- feederImpedance[@floatingType=singleComplex
            |                   @physicalNature=impedance
            |                   @value=(10,0)]
            |-- magnitude[@floatingType=arraySet
            |             @delay=1e-6                   # measure : time
            |             @initialValue=0]
            `-- model[@type=gain]
                `-- gain[floatingType=arraySet]
```

An antenna defined by the effective area :

```
data.h5
`-- electromagneticSource
    `-- antenna
        `-- $by-area[@efficiency=0.5]
            |-- inputImpedance[@floatingType=singleComplex
            |                  @physicalNature=impedance
            |                  @value=(50,0)]
            |-- loadImpedance[@floatingType=singleComplex
            |                 @physicalNature=impedance
            |                 @value=(10,0)]
            |-- feederImpedance[@floatingType=singleComplex
            |                   @physicalNature=impedance
            |                   @value=(10,0)]
            |-- magnitude[@floatingType=arraySet
            |             @delay=1e-6]                  # measure : time
            `-- model[@type=effectiveArea]
                `-- effectiveArea[@floatingType=arraySet]
```

An antenna defined by the far field :

```
data.h5
`-- electromagneticSource
    `-- antenna
        -- $by-field[@efficiency=0.5]
            |-- inputImpedance[@floatingType=singleComplex
            |                  @physicalNature=impedance
            |                  @value=(50,0)]
            |-- loadImpedance[@floatingType=singleComplex
            |                 @physicalNature=impedance
            |                 @value=(10,0)]
            |-- feederImpedance[@floatingType=singleComplex
            |                   @physicalNature=impedance
            |                   @value=(10,0)]
            |-- magnitude[@floatingType=arraySet
            |             @delay=1e-6]                 # measure : time
            `-- model[@type=farField]
                `-- farField[@floatingType=arraySet]
```

An antenna defined by an exchange surface :

```
data.h5
|-- mesh
|   `-- $gmesh1
|       `-- $mesh1
|-- exchangeSurface
|   `-- $exchange-surface
`-- electromagneticSource
    `-- antenna
        -- $by-field[@efficiency=0.5]
            |-- inputImpedance[@floatingType=singleComplex
            |                  @physicalNature=impedance
            |                  @value=(50,0)]
            |-- loadImpedance[@floatingType=singleComplex
            |                 @physicalNature=impedance
            |                 @value=(10,0)]
            |-- feederImpedance[@floatingType=singleComplex
            |                   @physicalNature=impedance
            |                   @value=(10,0)]
            |-- magnitude[@floatingType=arraySet
            |             @delay=1e-6]                 # measure : time
            `-- model[@type=exchangeSurface
                      @exchangeSurface=/exchangeSurface/$exchange-surface]
```

8.7 Source on mesh

8.7.1 By using a data on mesh

An electromagnetic source can be set by using numerical data on mesh (see *Numerical data on mesh* for details).

```
data.h5
|-- mesh
```

```
|   `-- $gmesh1
|       `-- $mesh1
|           `-- group
|               `-- $group1
`-- electromagneticSource
    `-- sourceOnMesh
        `-- $by-data-on-mesh[@type=arraySet]
            |                 @label=Electric field]
            |-- data[@label=electric field
            |        @physicalNature=electricField
            |        @unit=voltPerMeter]
            `-- ds/
                |-- dim1[@label=component x y z
                |        @physicalNature=component]
                |-- dim2[@label=mesh elements
                |        @physicalNature=meshEntity]
                `-- dim3[@label=the time
                         @physicalNature=time
                         @unit=second]
```

with `/floatingType/$dataOne/ds/dim2` :

/mesh/$gmesh1/$mesh1/group/$group1

In the preceding example, data on mesh named `data.h5:/electromagneticSource/sourceOnMesh/$by-data-on-mesh` is used as an `electromagneticSource` of type `sourceOnMesh`. It relies on the mesh group `data.h5:/mesh/$gmesh1/$mesh1/group/$group1`.

8.7.2 By using an exchange surface

```
data.h5
|-- exchangeSurface
|   `-- $exchange-surface
`-- electromagneticSource
    `-- sourceOnMesh
        `-- $by-ex-surf[@type=exchangeSurface
                        @exchangeSurface=/exchangeSurface/$exchange-surface]
```

CHAPTER

NINE

PHYSICAL MODELS

The material models category contains models used in electromagnetic simulation like :

- Dielectric material
  - Debye material
  - Lorentz material
- Multi layer material for composite elements
- Electrical models
- sub-cellular models
  - Slot model
- Special interfaces

Sub-categories are documented in the next sections.

9.1 Frequency range of validity

For large spectrum simulation, one material model could not be valid for the entire frequency range, so several models have to be used.

Material model have two optional real parameters (two HDF5 real attributes) :

- `frequency_validity_min` is the minimum frequency of validity in Hertz
- `frequency_validity_max` is the maximum frequency of validity in Hertz

For example

```
data.h5
`-- physicalModel/
    `-- volume
        `-- $water[@frequency_validity_min=1e3
                   @frequency_validity_max=1e9]
```

9.2 Predefined model

Amelet HDF predefines some remarkable materials :

- `/physicalModel/perfectElectricConductor`, it is the perfect electric condutor material.
- `/physicalModel/perfectMagneticConductor`, it is the perfect magnetic conductor material.

- `/physicalModel/vacuum`, it represents the EM vacuum.

Note: Predefined material nodes must exist in the Amelet-HDF instance

```
data.h5
`-- physicalModel/
    |-- perfectElectricConductor
    |-- perfectMagneticConductor
    |-- vacuum
    `-- volume
        `-- $water[@frequency_validity_min=1e3
                   @frequency_validity_max=1e9]
```

9.3 Volume

A volume material is a material defined by

- a relative permittivity (dimensionless)
- a relative permeability (dimensionless)
- a electric conductivity (in siemens / meter)
- a magnetic conductivity (in farad / meter), (this conductivity is used in perfectly matched layers (PML) models for instance).
- a volumetric mass density (in kilogram / cubic meter)

Such a material is a named HDF5 group child of `/physicalModel/volume`. The name of the group is the name of the material.

For a material called `$diel1` the tree schema looks like

```
data.h5
`-- physicalModel/
    `-- volume
        `-- $diel1[@volumetricMassDensity=1000.]
            |-- relativePermittivity
            |-- relativePermeability
            |-- electricConductivity
            `-- magneticConductivity
```

The next sections explain how the four components are defined.

9.3.1 Relative permittivity

The relative permittivity is a named HDF5 group, its name is `relativePermittivity` and can be expressed in different manners :

- By a complex value
- By an array of complex values
- By a general rational function
- By a Debye model
- By a Lorentz model

Complex value

If the relative permittivity is defined by a complex number, `relativePermittivity` group is a `floatingType singleComplex`.

`relativePermittivity` has no children.

Example :

```
data.h5
`-- physicalModel/
    `-- volume
        `-- $water
            `-- relativePermittivity[@floatingType=singleComplex
                                     @value=(80,0)]
```

Complex Array Set

If the relative permittivity is defined by an array (frequency, values), `relativePermittivity` group is a floatingType arraySet.

`relativePermittivity` is then an arraySet structure with :

- `relativePermittivity/data` is the relative permittivity values, it's a one dimension HDF5 dataset
- `relativePermittivity/ds/dim1` is the frequency spectrum, it's a one dimensensional HDF5 real dataset

Example

```
data.h5
`-- physicalModel/
    `-- volume
        `-- $water
            `-- relativePermittivity[@floatingType=arraySet]
                |-- data
                `-- ds
                    `-- dim1[@physicalNature=frequency
                             @unit=hertz]
```

General rational function

The relative permittivity can also be defined by a general rational function, that is to say a `floatingType` equals `generalRationalFunction`:

Example

```
data.h5
`-- physicalModel/
    `-- volume
        `-- $diel_gen_rat_func
            `-- relativePermittivity[@floatingType=generalRationalFunction]
```

where `data.h5:/physical/volume/$diel_gen_rat_func/relativePermittivity`:

$a0	$b0
$a1	$b1
$a2	$b2
.	.
.	.
.	.
$an	$bn

Debye model

If the permittivity follows a Debye rule, then the relative permittivity can be a multipole Debye model defined by a list of Debye functions :

$$\hat{\varepsilon}(\omega) = \varepsilon_\infty + (\varepsilon_s - \varepsilon_\infty) \sum_{p=1}^{P} \frac{G_p}{1 + j\omega\tau_p}$$

with the condition that $\sum_{p=1}^{P} G_p = 1$.

where :

- ε_∞ is the infinite frequency permittivity
- ε_s is the static permittivity at zero frequency
- τ_p is the characteristic relaxation time of the medium

In **Amelet HDF** a debye permittivity is an HDF5 group with three attributes and a dataset child :

- `type` attribute equals `debye`, it gives the `type` of the permittivity definition
- `epsilonStatic` is an HDF5 real attribute and gives the static value of the permittivity
- `epsilonLimit` is an HDF5 real attribute and gives the limit value of the permittivity

Then, the list of debye functions is an HDF5 dataset named `listOfFunctions`

```
data.h5
`-- physicalModel/
    `-- volume
        `-- $diel_debye
            `-- relativePermittivity[@type=debye
                |                    @epsilonStatic=3
                |                    @epsilonLimit=80]
                `-- listOfFunctions
```

`listOfFunctions` is a two columns HDF5 dataset of reals :

- G_p is written in the first column
- τ_p is written in the second column

$g1	$tau1
$g2	$tau2
$g3	$tau3

Lorentz model

If the permittivity follows a Lorentz rule, then the relative permittivity can be a multipole Lorentz model defined by a list of Lorentz functions :

$$\hat{\varepsilon}(\omega) = \varepsilon_\infty + (\varepsilon_s - \varepsilon_\infty) \sum_{p=1}^{P} \frac{G_p \omega_p^2}{\omega_p^2 + 2j\omega\delta_p - \omega^2}$$

with the condition that $\sum_{p=1}^{P} G_p = 1$.

where :

- ε_∞ is the infinite frequency permittivity
- ε_s is the static permittivity at zero frequency
- ω_p is the resonant frequency
- δ_p is the dumping coefficient

In **Amelet HDF** a lorentz permittivity is an HDF5 group with three attributes and a dataset child :

- `type` attribute equals `debye`, it gives the `type` of the permittivity definition
- `epsilonStatic` is an HDF5 real attribute and gives the static value of the permittivity
- `epsilonLimit` is an HDF5 real attribute and gives the limit value of the permittivity

Then, the list of Lorentz function is an HDF5 real dataset named `listOfFunctions`

```
data.h5
`-- physicalModel/
    `-- volume
        `-- $diel_lorentz
            `-- relativePermittivity[@type=lorentz
                |                    @epsilonStatic=3
                |                    @epsilonLimit=80]
                `-- listOfFunctions
```

`listOfFunction` has three columns of HDF5 reals:

- G_p is written in the first column
- ω_p is written in the second column
- δ_p is written in the third column

$g1	$omega1	$delta1
$g2	$omega2	$delta2
$g3	$omega3	$delta3

9.3.2 Relative permeability

The relative permeability is a named HDF5 group, its name is `relativePermeability` and can be expressed in different manners as the `relativePermittivity` is expressed :

- By a complex value
- By an array of complex values
- By a general rational function

- By a Debye model
- By a Lorentz model

Example of a `singleComplex` `relativePermeability`

```
data.h5
`-- physicalModel/
    `-- volume
        `-- $water
            `-- relativePermeability[@floatingType=singleComplex]
                                     @value=(1,0)]
```

9.3.3 Electric conductivity

The electric conductivity is a named HDF5 group, its name is `electricConductivity` and can be expressed in different manners :

- By a real value
- By an array of complex values that vary with the frequency

The electric conductivity is expressed in siemens.

Real value

If the electric conductivity is defined by a real number, `electricConductivity` group is a floatingType singleReal.

`electricConductivity` has no children.

Example :

```
data.h5
`-- physicalModel/
    `-- volume
        `-- $water
            `-- electricConductivity[@floatingType=singleReal]
                                     @value=10e-6]
```

Complex ArraySet

If the electric conductivity is defined by an array (frequency, values), `electricConductivity` group is a floatingType arraySet.

`electricConductivity` is then an arraySet structure with :

- `electricConductivity/data` is the electric conductivity values, it's a one dimension HDF5 complex dataset
- `electricConductivity/ds/dim1` is the frequency spectrum, it's a one dimensensional HDF5 real dataset

Example

```
data.h5
`-- physicalModel/
    `-- volume
        `-- $water
```

```
            `-- electricConductivity[@floatingType=arraySet]
                |-- data
                `-- ds
                    `-- dim1[@physicalNature=frequency
                             @unit=hertz]
```

9.3.4 Magnetic conductivity

The magnetic conductivity is a named HDF5 group, its name is `magneticConductivity` and can be expressed in different manners :

- By a real value
- By an array of complex values that vary with the frequency

Real value

If the magnetic conductivity is defined by a real number, `magneticConductivity` group is a floatingType singleReal.

`magneticConductivity` has no children.

Example :

```
data.h5
`-- physicalModel/
    `-- volume
        `-- $water
            `-- magneticConductivity[@floatingType=singleReal]
                                     @value=0]
```

Complex Array Set

If the magnetic conductivity is defined by an array (frequency, values), `magneticConductivity` group is a `floatingType` `arraySet`.

`magneticConductivity` is then an arraySet structure with :

- `magneticConductivity/data` is the magnetic conductivity values, it's a one dimension HDF5 complex dataset
- `magneticConductivity/ds/dim1` is the frequency spectrum, it's a one dimensional HDF5 dataset of reals

Example

```
data.h5
`-- physicalModel/
    `-- volume
        `-- $water
            `-- magneticPermeability[@floatingType=arraySet]
                |-- data
                `-- ds
                    `-- dim1[@physicalNature=frequency
                             @unit=hertz]
```

9.3.5 Volumetric mass density

The volumetric mass density is a HDF5 attribute, its name is `volumetricMassDensity` and is expressed by a real scalar value.

Example :

```
data.h5
`-- physicalModel/
    `-- volume
        `-- $water[@volumetricMassDensity=1000.]
```

9.4 Multi-layer

A multi-layer material is a material made up of several `physicalModel/volume` material layers. Multi-layer materials are stored in the `/physicalModel/multilayer/` category.

A multi-layer material is a named HDF5 table of two columns, each row is a new layer (a `physicalModel/volume` material) and the index is implicit.

Columns definition :

- `physicalModel` : A column of 100 characters strings. It's the absolute name of model defined in `/physicalModel/volume`.
- `thickness` : A column of reals. It's the thickness of the layer in meter.

Example

```
data.h5
`-- physicalModel/
    |-- volume
    |   |-- $vol1
    |   `-- $vol2
    `-- multilayer
        `-- $multilayer1
```

and `/physicalModel/multilayer/$multilayer1` is :

physicalModel	thickness
`/physicalModel/volume/$vol1`	0.1
`/physicalModel/volume/$vol2`	0.2
`/physicalModel/volume/$vol1`	0.1

9.5 Anisotropic material

Anisotropy is the property of being directionally dependent, as opposed to isotropy, which means homogeneity in all directions (http://en.wikipedia.org/wiki/Anisotropy).

Each characteristic (permittivity, permeability ...) of an anisotropic material can be expressed by a 3x3 tensor, see the example for the permittivity :

$$\begin{pmatrix} \varepsilon_{xx} & \varepsilon_{xy} & \varepsilon_{xz} \\ \varepsilon_{yx} & \varepsilon_{yy} & \varepsilon_{yz} \\ \varepsilon_{zx} & \varepsilon_{zy} & \varepsilon_{zz} \end{pmatrix}$$

where elements are the material behaviour in one direction.

In **Amelet HDF**, instead of defining a tensor for all characteritics, each tensor's element is a `physicalModel` and expresses all properties in a given direction. An anisotropic material is then an HDF5 3x3 dataset of 60 character strings, each element of the dataset is a `physicalModel` absolute name.

Example :

```
data.h5
`-- physicalModel/
    |-- volume/
    |   |-- $v1
    |   |-- $v2
    |   `-- $v3
    `-- anisotropic/
        `-- $an1
```

with `/physicalModel/anisotropic/$an1`:

#	0	1	2
0	/physicalModel/volume/$v1	/physicalModel/volume/$v2	/physicalModel/volume/$v3
1	/physicalModel/volume/$v2	/physicalModel/volume/$v1	/physicalModel/volume/$v2
2	/physicalModel/volume/$v3	/physicalModel/volume/$v2	/physicalModel/volume/$v1

=

$$\begin{pmatrix} \text{/physicalModel/volume/\$v1} & \text{/physicalModel/volume/\$v2} & \text{/physicalModel/volume/\$v3} \\ \text{/physicalModel/volume/\$v2} & \text{/physicalModel/volume/\$v1} & \text{/physicalModel/volume/\$v2} \\ \text{/physicalModel/volume/\$v3} & \text{/physicalModel/volume/\$v2} & \text{/physicalModel/volume/\$v1} \end{pmatrix}$$

9.5.1 Volumetric mass density

An anisotropic material definition can be completed by an optional attribute named `volumetricMassDensity` as for volume material.

Example :

```
data.h5
`-- physicalModel/
    |-- volume/
    |   |-- $v1
    |   |-- $v2
    |   `-- $v3
    `-- anisotropic/
        `-- $an1[@volumetricMassDensity=1026.]
```

9.6 Multiport

The category `/physicalModel/multiport` contains electrical models like localized element (resistance ...), multiport models and more complex models like scattering parameters.

There are many sub-types in this category : impedance, admittance, resistance, inductance, capacitance, conductance, s-parameter.

Besides, a multiport can be a scalar, a vector (depending on the frequency) or a matrix, even a matrix depending on the frequency, so the floatingType paradigm is used. By a consequence, a multiport can be defined wherever in a **Amelet HDF** instance (where **Amelet HDF** specifies its usage).

As usual, the unit is set by **Amelet HDF** so the attribute `unit` are optional.

The number of ports of a multiport is given by the dimension of the `floatingType` :

- `singleReal` : the number of ports is 1
- `singleComplex` : the number of ports is 1
- `dataSet` : the number of ports is the dimension of the `dataSet`
- `arraySet` : the number of ports is the dimension of the children `data` element.

9.6.1 Electric potential point

An electric potential point is a point where an electric potential can measured. A voltage can be computed between two electric potential points. In **Amelet HDF**, an electric potential point can be a multiport port, that is why a new physicalNature is introduced : `electricPotentialPoint` physical nature.

For instance, when a multiport impedance is written in **Amelet HDF** by an arraySet, port dimension hold the `electricPotentialPoint` physical nature :

```
data.h5
`-- physicalModel/
    `-- multiport/
        `-- $resistance[@floatinType=arraySet]
            |-- data[@physicalNature=resistance]
            `-- ds
                |-- dim1[@physicalNature=electricPotentialPoint]
                `-- dim2[@physicalNature=electricPotentialPoint]
```

9.6.2 Predefined multiports

Amelet HDF predefines three multiports :

- `/physicalModel/multiport/shortCircuit`, it is the short circuit model short-cut
- `/physicalModel/multiport/openCircuit`, it is the open circuit model short-cut
- `/physicalModel/multiport/matched`, it is the the matched model short-cut

Note: Predefined multiport nodes must exist in Amelet-HDF instances

```
data.h5
`-- physicalModel/
    `-- multiport
        |-- shortCircuit
        |-- openCircuit
        `-- matched
```

9.6.3 Resistance

A resistance is a floatingType element, with a `physicalNature` attribute equals `resistance`.

A resistance can be a singleReal, a dataSet or an arraySet, the unit is `ohm`.

Single real

Example of resistance (a dipole) expressed by a `singleReal` :

```
data.h5
`-- physicalModel/
    `-- multiport/
        `-- $resistance1[@physicalNature=resistance
                         @floatingType=singleReal
                         @unit=ohm
                         @value=80]
```

Real dataset (matrix)

```
data.h5
`-- physicalModel/
    `-- multiport/
        `-- $resistance2[@floatingType=dataSet
                         @physicalNature=resistance]
```

Real ArraySet (depends on the frequency)

Example of resistance expressed by an `arraySet` depending on the frequency :

```
data.h5
`-- physicalModel/
    `-- multiport/
        `-- $resistance2[@floatingType=arraySet]
            |-- data[@physicalNature=resistance
            |        @unit=ohm]
            `-- ds
                `-- dim1[@physicalNature=frequency
                         @unit=hertz]
```

9.6.4 Conductance

A conductance is a floatingType element, with a `physicalNature` attribute equals `conductance`.

A conductance can be a singleReal, a dataSet or an arraySet, the unit is `siemens`.

Single real

Example of conductance expressed by a `singleReal` :

```
data.h5
`-- physicalModel/
    `-- multiport/
        `-- $conductance1[@physicalNature=conductance
                          @floatingType=singleReal
                          @value=1e9]
```

Real Array Set

Example of conductance expressed by an `arraySet` :

```
data.h5
`-- physicalModel/
    `-- multiport/
        `-- $conductance2[@floatingType=arraySet]
            |-- data[@physicalNature=conductance
            |        @unit=siemens]
            `-- ds
                `-- dim1[@physicalNature=frequency
                         @unit=hertz]
```

9.6.5 Inductance

An inductance is a floatingType element, with a `physicalNature` attribute equals `inductance`.

A inductance can be a singleReal or an arraySet, the unit is `henry`.

Single real

Example of inductance expressed by a `singleReal` :

```
data.h5
`-- physicalModel/
    `-- multiport/
        `-- $inductance1[@physicalNature=inductance
                         @floatingType=singleReal
                         @value=1e-3]
```

Real Array Set

Example of inductance expressed by an `arraySet` :

```
data.h5
`-- physicalModel/
    `-- multiport/
        `-- $inductance2[@floatingType=arraySet]
            |-- data[@physicalNature=inductance
            |        @unit=henry]
            `-- ds
                `-- dim1[@physicalNature=frequency
                         @unit=hertz]
```

9.6.6 Capacitance

An capacitance is a floatingType element, with a `physicalNature` attribute equals `capacitance`.

A capacitance can be a singleReal, a dataSet or an arraySet, the unit in farad.

Single real

Example of capacitance expressed by a `singleReal` :

```
data.h5
`-- physicalModel/
    `-- multiport/
        `-- $capacitance1[@physicalNature=capacitance
                          @floatingType=singleReal
                          @value=1e-9]
```

Real Array Set

Example of a capacitance expressed by an `arraySet` :

```
data.h5
`-- physicalModel/
    `-- multiport/
        `-- $capacitance[@floatingType=arraySet]
            |-- data[@physicalNature=capacitance
            |        @unit=farad]
            `-- ds
                `-- dim1[@physicalNature=frequency
                         @unit=hertz]
```

9.6.7 Impedance

"Electrical impedance extends the concept of resistance to AC circuits, describing not only the relative amplitudes of the voltage and current, but also the relative phases". (http://en.wikipedia.org/wiki/Electrical_impedance)

Impedance is measured in ohm and can be expressed with :

- a complex number
- an array of complex number
- some rational functions

Single complex

Example of impedance expressed by a `singleComplex` :

```
data.h5
`-- physicalModel/
    `-- multiport/
        `-- $impedance1[@physicalNature=impedance
                        @floatingType=singleComplex
                        @value=(80,0)]
```

Complex dataSet

```
data.h5
`-- physicalModel/
    `-- multiport/
```

```
        `-- $impedance2[@physicalNature=impedance
                        @floatingType=dataSet]
```

Complex Array Set

Example of impedance expressed by an `arraySet` :

```
data.h5
`-- physicalModel/
    `-- multiport/
        `-- $impedance3[@floatingType=arraySet]
            |-- data[@physicalNature=impedance]
            `-- ds
                `-- dim1[@physicalNature=frequency
                         @unit=hertz]
```

Rational functions

Example of impedance expressed by a rational :

```
data.h5
`-- physicalModel/
    `-- multiport/
        `-- $impedance1[@floatingType=rational
            |           @physicalNature=impedance]
            |-- function
            |   |-- $rational1[@floatingType=rationalFunction]
            |   `-- $rational2[@floatingType=rationalFunction]
            `-- data
```

with `/physicalModel/multiport/$impedance1/function/$rational1` :

type	A	B	F
1		1	
2	2	3	
3	4	5	
4	6	7	8
5	9	10	11

`/physicalModel/multiport/$impedance1/data` is a HDF5 dataset, its characteristics are :

- its rank is 2
- its dimensions are number_of_ports * number_of_ports

For example for a two port elements, `data` is :

ports	1	2
1	$rational1	$rational2
2	$rational2	$rational1

9.6.8 Admittance

The admittance (Y) is the inverse of the impedance (Z), Y is measured in siemens.

Admittances are expressed in the same manner as impedances with a `physicalNature` equals `admittance`.

Example of admittance expressed by an `arraySet` :

```
data.h5
`-- physicalModel/
    `-- multiport/
        `-- $admittance1[@floatingType=arraySet]
            |-- data[@physicalNature=admittance]
            `-- ds
                `-- dim1[@physicalNature=frequency
                         @unit=hertz]
```

Like impedances, admittances can be defined by :

- A complex number
- A complex dataSet
- An array of complex numbers
- Some rational functions

9.6.9 RLC model

RLC model represents an RLC circuit consisting of a resistance, an inductance and a capacitance connected in series or in parallel.

The elements can be connected in different ways :

- type 1:

  R, L and C are in series

```
o---- R ---- L ---- C ----o
```

- type 2:

  C is in parallel with L and R in series

```
     |---- L --- R ----|
o----|                 |----o
     |------- C -------|
```

- type 3:

  L is in parallel with R and C in series

```
     |---- C --- R ----|
o----|                 |----o
     |------- L -------|
```

- type 4:

  R is in series with L and C in parallel

```
            |---- L ----|
o---- R ----|           |----o
            |---- C ----|
```

- type 5:

  R is in parallel with L and C in series

```
     |---- L --- C ----|
o----|                 |----o
     |------- R -------|
```

- type 6:

  L is in series with R and C in parallel

```
            |---- R ----|
o---- L ----|           |----o
            |---- C ----|
```

- type 7:

  C is in series with L and R in parallel

```
            |---- L ----|
o---- C ----|           |----o
            |---- R ----|
```

- type 8:

  R, L et C are in parallel

```
     |---- R ----|
     |           |
o----|---- L ----|----o
     |           |
     |---- C ----|
```

In **Amelet HDF**, RLC circuits are contained in the `/physicalModel/multiport/RLC` subcategory.

An RLC circuit is a group which contains the name of the multiports defining R, L and C elements.

An RLC circuit has four attributes :

- `type`. `type` is an HDF5 integer and can take values in [1..8], it represents the type detailed above.
- `R`. `R` is the name of the resistance, it is an HDF5 string attribute
- `L`. `L` is the name of the inductance, it is an HDF5 string attribute
- `C`. `C` is the name of the conductance, it is an HDF5 string attribute

Example of an RLC circuit :

```
data.h5
`-- physicalModel/
    `-- multiport/
        |-- RLC/
        |   `-- $RLC[@type=1
        |            @R=/physicalModel/multiport/$resistance
        |            @L=/physicalModel/multiport/$inductance
        |            @C=/physicalModel/multiport/$conductance]/
        |-- $resistance[@physicalNature=resistance]
        |-- $inductance[@physicalNature=inductance]
        `-- $conductance[@physicalNature=conductance]
```

9.6.10 Scattering parameters

"Scattering parameters or S-parameters are properties used in electrical engineering, electronics engineering, and communication systems engineering describing the electrical behavior of linear electrical networks when undergoing various steady state stimuli by small signals." (http://en.wikipedia.org/wiki/Scattering_parameters)

Scattering parameters are expressed in the same manner as impedances and are stored in the `/physicalModel/multiport/sParameter` category.

However scattering parameters have an optional attribute :

- `referenceImpedance`. `referenceImpedance` is a real attribute. It represents the reference impedance in `ohm` used to compute the S-parameters.

  If `referenceImpedance` is missing, the reference impedance is the characteristics impedance of the line.

Examples :

```
data.h5
`-- physicalModel/
    `-- multiport/
        `-- sParameter
            |-- $sparam-single[@floatingType=singleComplex,
            |                  @value=(10,2)]
            |-- $sparam[@floatingType=arraySet
            |   |       @referenceImpedance=50]
            |   |-- data
            |   `-- ds
            |       |-- dim1[@label=nbPort
            |       |        @physicalNature=electricPotentialPoint]
            |       |-- dim2[@label=nbPort
            |       |        @physicalNature=electricPotentialPoint]
            |       `-- dim3[@label=frequency
            |                @physicalNature=frequency
            |                @unit=hertz]
            `-- $sparam-rational[@floatingType=rational]/
                |-- function
                |   |-- rational1[@floatingType=rationalFunction]
                |   |-- rational2[@floatingType=rationalFunction]
                |   `-- rational3[@floatingType=rationalFunction]
                `-- data
```

Like impedances, sParameter can be defined by :

- A complex number, the number of ports is 1
- An array of complex numbers, the number of ports is the length of dim1
- Some rational functions, the number of ports is given by the dimensions of `/physicalModel/multiport/sParameter/$sparam-rational/data`

9.6.11 The connection category

The connection category contains line connection definitions like *ideal junction*.

Ideal junction

The `idealJunction` connexion is a way to define ideal connectivity between wire strands.

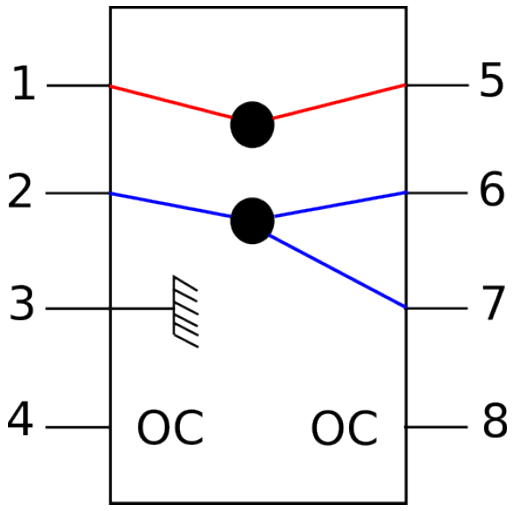


Fig. 9.1: A height ports ideal junction

Suppose height wires are connected to an ideal junction like in the following figure.

Each wire can be in one of three states :

- Connected to another wire (black circle)
- Connected to the ground (short-circuit)
- Not connected at all (open-circuit - OC)

The state set can be sum up in a (n x n) integer matrix C looking like :

X	1	2	3	4	5	6	7	8
1	0	0	0	0	1	0	0	0
2	0	0	0	0	0	1	1	0
3	0	0	1	0	0	0	0	0
4	0	0	0	-1	0	0	0	0
5	1	0	0	0	0	0	0	0
6	0	1	0	0	0	0	1	0
7	0	1	0	0	0	1	0	0
8	0	0	0	0	0	0	0	-1

The matrix' elements are set up with the following rules :

Diagonal elements

- If C(i,i) = 1, the port i is connected to the ground
- If C(i,i) = -1, the port i is an open-circuit
- If C(i,i) = 0, the port is in the default state : not connected to the ground and not an open-circuit. It is connected to some ports

Non diagonal elements

- If C(i,j) = 1, the port i is connected to the port j
- If C(i,j) = 0, the port i is not connected to the port j

In Amelet-HDF, an ideal junction is a number port dimension `floatingType=dataSet` defined by :

- A `type` attribute equals to `idealJunction`

- Integer values, values are defined by the rules above.

Examples :

```
data.h5
`-- physicalModel/
    `-- multiport/
        `-- connection
            `-- $ideal_junction[@type=idealJunction
                                @floatingType=dataSet]
```

9.7 Distributed Multiport

`/physicalModel/multiport/distributed` contains transmission line distributed parameters, i.e the RLCG matrices components. Those parameters are :

- distributed impedance
- distributed admittance
- distributed resistance
- distributed inductance
- distributed capacitance
- distributed conductance

A distributed multiport is a floatingType child of `/physicalModel/multiport/distributed` with a mandatory attribute :

`/physicalModel/multiport/distributed` are expressed in the same way as `/physicalModel/multiport`, except that the `unit` is :

- `ohmPerMeter` for `impedance`
- `siemensPerMeter` for `admittance`
- `ohmPerMeter` for `resistance`
- `henryPerMeter` for `inductance`
- `faradPerMeter` for `capacitance`
- `siemensPerMeter` for `conductance`

All children of these categories can be expressed by :

- A complex number
- A complex dataSet
- An array of complex numbers
- Some rational functions

Example

```
data.h5
`-- physicalModel/
    `-- multiport/
        `-- distributed
            |-- $disImp1[@floatingType=singleComplex
            |            @physicalNature=impedance
            |            @unit=ohmPerMeter
            |            @value=(10,2)]
            |-- $disImp2[@floatingType=dataSet
            |            @physicalNature=impedance
            |            @unit=ohmPerMeter]
            `-- $disImp3[@floatingType=arraySet]
                |-- data[@physicalNature=impedance
                |        @unit=ohmPerMeter]
                `-- ds
                    `-- dim1[@physicalNature=frequency]
```

9.8 Surface

This section describes surface material models, we can see two main types detailed in the next sections :

- the thin dielectric layer model
- the surface impedance boundary condition model

Model or genuine surface, instances can have a `physicalModel` attribute which gives the volume characteristics of the material.

Example

```
data.h5
`-- physicalModel/
    |-- volume/
    |   `-- $mat1
    `-- surface/
        `-- $layer[@type=XXX
                   @physicalModel=/physicalModel/volume/$mat1]
```

9.8.1 Thin dielectric layer

The thin dielectric layer represents a dielectric layer thinner than the cell dimension. Surface impedance boundary models are not used to make wave propagate through the panel but the equivalent medium is computed from the weighting of the layer characteristics and the surrounding medium properties.

Example

```
data.h5
`-- physicalModel/
    |-- volume/
    |   `-- $mat1
    `-- surface/
        `-- $layer[@type=thinDielectricLayer
                   @physicalModel=/physicalModel/volume/$mat1
                   @thickness=1e-3]
```

9.8.2 SIBC

The thin dielectric layer represents a dielectric layer thiner than the cell dimension. Surface impedance boundary models are used to make wave propagate through the panel, the surface impedance is computed by the solver.

Example

```
data.h5
`-- physicalModel/
    |-- volume/
    |   `-- $mat1
    `-- surface/
        `-- $layer[@type=SIBC
                   @physicalModel=/physicalModel/volume/$mat1
                   @thickness=1e-3]
```

9.8.3 Zs

The thin dielectric layer represents a dielectric layer thiner than the cell dimension. Surface impedance boundary models are used to make wave propagate through a panel and the surface impedance is given by the model.

The relation between $\overrightarrow{E}$ and $\overrightarrow{H}$ is :

$$\overrightarrow{E_{tan}}(\vec{r}) = Z_s(\vec{r}, w)\overrightarrow{J}(\vec{r}) = Z_s(\vec{r}, w)[\vec{n}(\vec{r}) \times \overrightarrow{H}(\vec{r})]$$

$\overrightarrow{J}$ is the surface current vector.

Example

```
data.h5
`-- physicalModel/
    |-- volume/
    |   `-- $mat1
    |-- multiport/
    |   `-- $Zs[@floatingType=rational
    |       |   @physicalNature=impedance]
    |       |-- function
    |       |   |-- $Z11[@floatingType=generalRationalFunction
    |                    @type=polynomial]
    |       |   `-- $Z12[@floatingType=generalRationalFunction
    |                    @type=partialFraction]
    |       `-- data
    `-- surface/
        `-- $layer[@type=Zs
                   @Zs=/physicalModel/multiport/$Zs
                   @physicalModel=/physicalModel/volume/$mat1]
```

with `data.h5:/physicalModel/multiport/Zs/function/Z11`:

(5.9545e69, 0)	(2.9773e69, 0)
(9.0191e59, 0)	(1.3921e59, 0)
(1.9344e49, 0)	(1.6254e48, 0)
(1.3458e38, 0)	(7.0897e36, 0)
(3.7609e26, 0)	(1.2717e26, 0)
(4.2033e14, 0)	(8.3344e05, 0)
(138.73, 0)	(1., 0)

with `data.h5:/physicalModel/multiport/Zs/function/Z12`:

degree	A	B
1	(50, 0)	(0.5e-9, 4)
2	(125, 12.5)	(-15.25, 4)
1	(1.e9, 0)	(31, 0)

with `data.h5:/physicalModel/multiport/Zs/data`:

$Z11	$Z12
$Z21	$Z11

9.8.4 ZsZt

The thin dielectric layer represents a dielectric layer thiner than the cell dimension. Surface impedance boundary models are used to make wave propagate through the panel, Zs and Zt are given by the model.

Example

```
data.h5
`-- physicalModel/
    |-- volume/
    |   `-- $mat1
    |-- multiport/
    |   |-- $Zs[@physicalNature=impedance]
    |   `-- $Zt[@physicalNature=impedance]
    `-- surface/
        `-- $layer[@type=ZsZt
                   @Zs=/physicalModel/multiport/$Zs
                   @Zt=/physicalModel/multiport/$Zt
                   @physicalModel=/physicalModel/volume/$mat1]
```

9.8.5 ZsZt2

The thin dielectric layer represents a dielectric layer thiner than the cell dimension. Surface impedance boundary models are used to make wave propagate through the panel, Zs and Zt are given by the model. Front face (1) and back face (2) have different behavior.

Example

```
data.h5
`-- physicalModel/
    |-- volume/
    |   `-- $mat1
    |-- multiport/
    |   |-- $Zs1[@physicalNature=impedance]
    |   |-- $Zs2[@physicalNature=impedance]
    |   |-- $Zt1[@physicalNature=impedance]
    |   `-- $Zt2[@physicalNature=impedance]
    `-- surface/
        `-- $layer[@type=ZsZt2
                   @Zs1=/physicalModel/multiport/$Zs1
                   @Zt1=/physicalModel/multiport/$Zt1
                   @Zs2=/physicalModel/multiport/$Zs2
                   @Zt2=/physicalModel/multiport/$Zt2
                   @physicalModel=/physicalModel/volume/$mat1]
```

9.9 Interface

Objects contained in the category `/physicalModel/interface` define the connection between two media. An interface is a named HDF5 group with two mandatory attributes and one optional attribute :

Mandatory attributes :

- medium1 is an HDF5 string attribute, it is a pointer to a /physicalModel, first medium of the interface
- medium2 is an HDF5 string attribute, it is a pointer to a /physicalModel, second medium of the interface

Optional attribute :

- interface is an HDF5 string attribute, it is a pointer to a /physicalModel, it represents the properties of the interface (infinitely thin, or not meshed).

Below is an example of an interface separating two areas made up of /physicalModel/volume/$diel1 and /physicalModel/volume/$diel2. The interface itself (virtual space between the two media) is an infinitely thin perfectly conducting plane.

```
data.h5
`-- physicalModel/
    |-- volume
    |   |-- $diel1
    |   `-- $diel2
    `-- interface/
        `-- $interface1[@medium1=/physicalModel/volume/$diel1
                        @medium2=/physicalModel/volume/$diel2
                        @interface=/physicalModel/perfectElectricConductor]
```

9.10 Aperture

In the electromagnetic simulation domain, little aperture are often described thanks to sub cellular models associated to linear elements, they don't appear in the mesh as slot but as linear elements. In addition, apertures can be filled (loaded) by a materialModel.

An aperture is a named HDF5 group child of /physicalModel/aperture that have a type attribute. The type is a string HDF5 attribute.

Amelet HDF defines three apertures that are described in the next sections :

- Slot
- Rectangular aperture
- Circular aperture
- Elliptic aperture
- Large aperture
- Measured aperture

9.10.1 Slot

A slot is a named HDF5 group with type equals slot, it has three attributes :

- witdh : the width of the slot, a length in meter, it is a float HDF5 attribute and is mandatory.
- thickness : the thickness of the slot, a length in meter, it is a float HDF5 attribute and is mandatory.
- materialModel : the load material name of the slot, it is a character strings, it is optional.

Example :

```
data.h5
`-- physicalModel/
    |-- volume/
    |   `-- $diel1
    `-- aperture/
        `-- $slot1[@type=slot
                   @width=10e-3
                   @thickness=2e-3
                   @materialModel=/physicalModel/volume/$diel1]
```

9.10.2 Rectangular aperture

A rectangular aperture is a named HDF5 group with `type` equals `rectangular`, it has two attributes :

- `length` : the length of the rectangle, a length in meter, it is a float HDF5 attribute and is mandatory.
- `width` : the width of the rectangle, a length in meter, it is a float HDF5 attribute and is mandatory.
- `thickness` : the thickness of the aperture, a length in meter, it is a float HDF5 attribute and is mandatory.
- `materialModel` : the load material name of the aperture, it is a character strings, it is optional.

Example :

```
data.h5
`-- physicalModel/
    |-- volume/
    |   `-- $diel1
    `-- aperture/
        `-- $rectangularAperture1[@type=rectangular
                                  @length=10e-3
                                  @width=6e-3
                                  @thickness=2e-3
                                  @materialModel=/physicalModel/volume/$diel1]
```

9.10.3 Circular aperture

A circular aperture is a named HDF5 group with `type` equals `circular`, it has two attributes :

- `diameter` : the diameter of the circle, a length in meter, it is a float HDF5 attribute and is mandatory.
- `thickness` : the thickness of the aperture, a length in meter, it is a float HDF5 attribute and is mandatory.
- `materialModel` : the load material name of the aperture, it is a character strings, it is optional.

Example :

```
data.h5
`-- physicalModel/
    |-- volume/
    |   `-- $diel1
    `-- aperture/
        `-- $circularAperture1[@type=circular
                               @diameter=10e-3
                               @thickness=2e-3
                               @materialModel=/physicalModel/volume/$diel1]
```

9.10.4 Elliptic aperture

A elliptic aperture is a named HDF5 group with `type` equals `Elliptic`, it has two attributes :

- `semimajorAxis` : the semimajor axis of the ellipse, a length in meter, it is a float HDF5 attribute and is mandatory.
- `semiminorAxis` : the semiminor axis of the ellipse, a length in meter, it is a float HDF5 attribute and is mandatory.
- `thickness` : the thickness of the aperture, a length in meter, it is a float HDF5 attribute and is mandatory.
- `materialModel` : the load material name of the aperture, it is a character strings, it is optional.

Example :

```
data.h5
`-- physicalModel/
    |-- volume/
    |   `-- $diel1
    `-- aperture/
        `-- $ellipticAperture1[@type=elliptic
                               @semimajorAxis=10e-3
                               @semiminorAxis=6e-3
                               @thickness=2e-3
                               @materialModel=/physicalModel/volume/$diel1]
```

9.10.5 Large aperture

A large aperture relative to the wave length is a named HDF5 group with `type` equals `large`, it has three attribute :

- `surface` : the surface of the aperture, a surface in square meter, it is a float HDF5 attribute and is mandatory.
- `thickness` : the thickness of the aperture, a length in meter, it is a float HDF5 attribute and is mandatory.
- `materialModel` : the load material name of the aperture, it is a character strings, it is optional.

Example :

```
data.h5
`-- physicalModel/
    |-- volume/
    |   `-- $diel1
    `-- aperture/
        `-- $largeAperture1[@type=large
                            @surface=10e-3
                            @thickness=2e-3
                            @materialModel=/physicalModel/volume/$diel1]
```

9.10.6 Measured aperture

A measured aperture is a named HDF5 group with `type` equals `measured`, it has one `floatingType` child called `sigma`. `sigma` is the measured transmission cross section as function of incident field and frequency.

Example :

```
data.h5
`-- physicalModel/
    `-- aperture/
```

```
        `-- $measuredAperture1[@type=measured]
            `-- sigma[@floatingType=arraySet]
                |-- data[@physicalNature=surface
                |        @unit=squareMeter]
                `-- ds/
                    `-- dim1[@physicalNature=frequency
                             @unit=hertz]
```

9.11 Shield

The `shield` category contains shields definitions.

9.11.1 Metal braid shield

A metal braid is a shield for a cable or for a bundle of cables :

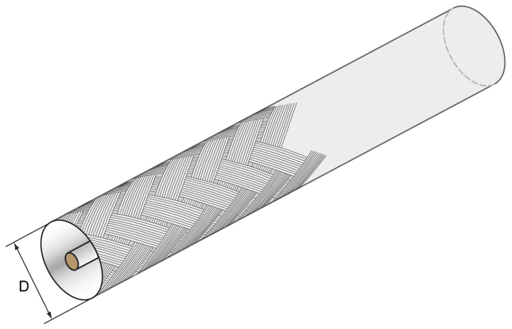


Fig. 9.2: Metal braid representation

A metal braid is completely described by six parameters :

- Diameter D (real number, dimension meters)
- Number of carriers C (i.e. belts of wires) in the braid (integer number)
- Number of wires N in a carrier (integer number)
- Diameter d of a single wire (real number, dimension meters)
- Conductivity σ of the wires (real number, dimension Simens per meters)
- Weave angle α (real number, degrees)

In **Amelet HDF** a metal braid is an HDF5 named group with seven attributes :

- `type` : `type` is an HDF5 string attribute, its value is `metalBraid`
- `braidDiameter` : `braidDiameter` is an HDF5 real attribute and represents D in meters
- `numberOfCarriers` : `numberOfCarriers` is an HDF5 integer attribute and is the number of carriers C.
- `numberOfWiresPerCarrier` : `numberOfWiresPerCarrier` is an HDF5 integer attribute and represents the number of wires N in a carrier
- `wireDiameter` : `wireDiameter` is an HDF5 real attribute and represents d in meters

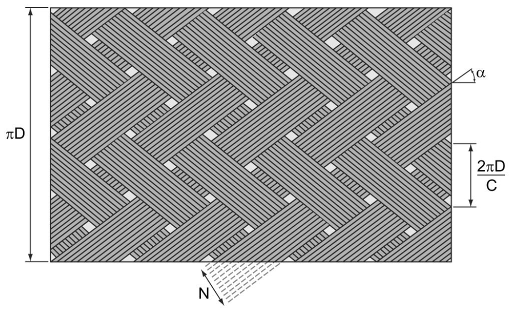


Fig. 9.3: Metal braid chracteristics

- `material` : `material` is an HDF5 string attribute, it contains the name of a `/physicalModel/volume` material
- `weaveAngle` : `weaveAngle` is an HDF5 real number which represents the angle α in degrees

and a `floatingType` child named `Zt`. `Zt` contains the transfer impedance of the shield.

Example :

```
data.h5
`-- physicalModel/
    |-- volume/
    |   `-- $copper
    |       `-- electricConductivity[@floatingType=singleReal
    |                                @value=59.6e6]
    `-- shield/
        `-- $a-metal-braid[@type=metalBraid
            |              @braidDiameter=5e-3
            |              @numberOfCarriers=10
            |              @numberOfWiresPerCarrier=20
            |              @wireDiameter=5e-4
            |              @material=/physicalModel/volume/$copper
            |              @weaveAngle=45]
            `-- Zt[@floatingType=arraySet]
                |-- data
                `-- ds
                    `-- dim1
```

9.12 Grid

The `grid` category contains grids definition. Many kinds of grid exist.

A grid is composed of two materials:

- The grid itself
- A material around the grid

A grid is defined by many parameters:

- Physical characteristics of the surrounding material
- Physical characteristics of the grid
- Dimension of the grid's wires for woven and comb grids (real number in meters):

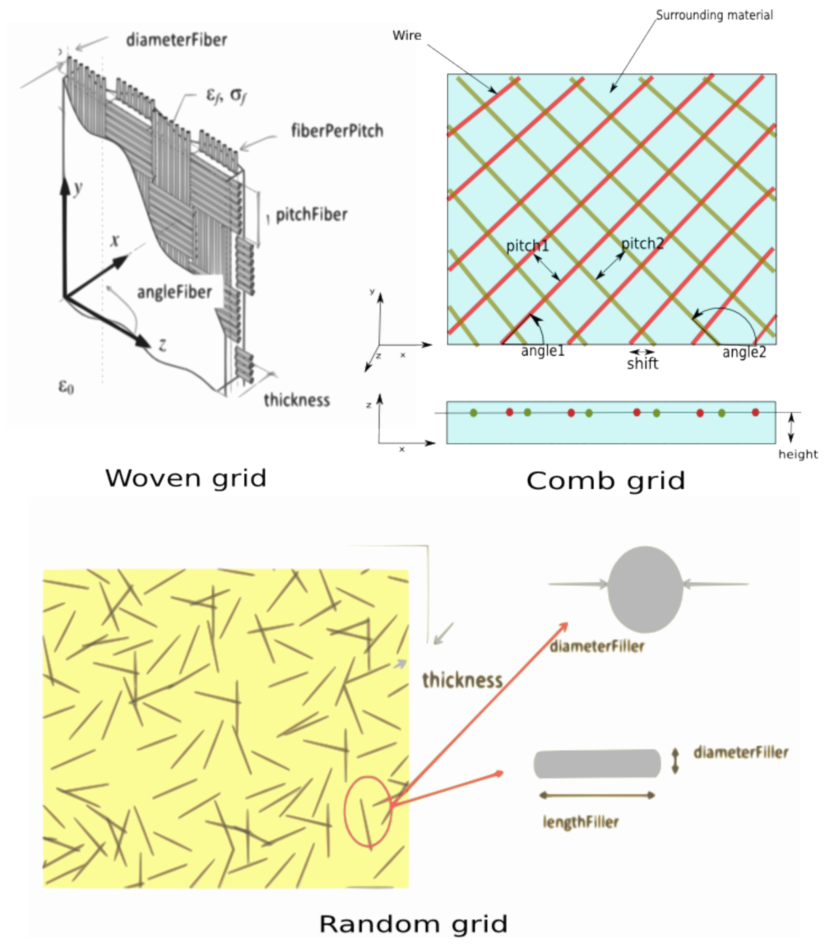


Fig. 9.4: Grids characteristics

  - Diameter if wires have a circular section
  - Thickness and width if wires have a rectangular section
- Texture type (woven, unidirectional, comb, random)

There are specific parameters to describe each kind of grid.

- A woven/unidirectional grid is defined by:
  - Number of fiber per pitch (integer number)
  - Length of a pitch (real number in meters)
  - Angle between the fiber and the Z-axis (real number in degrees)
  - Model type homgeneous or heterogeneous composite
- A comb grid is defined by:
  - Angles relative to the X-axis (real number in degrees)
  - Pitches, distances between two wire centers (real number in meters)
  - Height of the grid from the bottom (z=0) in the context of layer
  - Model type homgeneous or heterogeneous composite
- A random grid:
  - Scale filler : micro or nano (string)
  - Type filler : sphere, rod or disk (string)
  - Volume fraction of filler (real number)
  - Diameter of filler (real number in meter)
  - Length of filler (real number in meter)

In **Amelet HDF** a grid is an HDF5 named group with the following attributes :

- `surroundingMaterial` : `surroundingMaterial` is an HDF5 string attribute, it contains the name of a `/physicalModel/volume` material.
- `gridMaterial` : `gridMaterial` is an HDF5 string attribute, it contains the name of a `/physicalModel/volume` material.
- `textureType` : `textureType` is an HDF5 string attribute and defines the type of grid texture. The accepted values are `woven`, `unilateral`, `comb` and `random`.
- `pitchFiber` : `pitchFiber` is an optional HDF5 real attribute, it represents the length of a pitch. This attribute is present if `textureType` is equal to `woven`.
- `fiberPerPitch` : `fiberPerPitch` is an optional HDF5 integer attribute, it represents the number of fiber per pitch. This attribute is present if `textureType` is equal to `woven`.
- `wireSectionType` : `wireSectionType` is an HDF5 string attribute and defines the type of the wire. The accepted values are `circular` or `rectangular`.
- `diameterWire` : `diameterWire` is an optional HDF5 real attribute and represents the grid's wire diameter in meters. This attribute is present if `wireSectionType` is equal to `circular` or if `textureType` is equal to `random`.
- `thicknessWire` : `thicknessWire` is an optional HDF5 real attribute and represents the grid's wire thickness in meters. This attribute is present if `wireSectionType` is equal to `rectangular`.

- widthWire : widthWire is an optional HDF5 real attribute and represents the grid's wire width in meters. This attribute is present if wireSectionType is equal to rectangular.
- lengthWire : lengthWire is an optional HDF5 real attribute and represents the length of wire in meter which fills the material. This attribute is present if textureType is equal to random.
- scaleFiller : scaleFiller is an optional HDF5 string attribute and represents the scale filled of the material. This attribute can have only two possible values micro or nano.
- typeFiller : typeFiller is an optional HDF5 string attribute and represents the filled type of the material. This attribute can have only three possible values sphere, rod or nano.
- volFractioFiller : volFractioFiller is an optional HDF5 real attribute and represents the ratio of filling volume.
- shift : shift is the optional distance in meters between the two combs measured on the X-axis (see the sketch).
- modelType : modelType is the optional HDF5 string attribute and represents the kind of material. Value can be homogeneous or heterogeneous.

and one or two HDF5 group named comb1 (and optionally comb2 if there are two combs) with attributes as follows :

- Height of the grid from the bottom (z=0) of a layer :
  - relativeHeight represents a relative height relative to the total height of a material layer. It is a dimensionless float between 0 and 1
  - absoluteHeight represents an absolute height from the bottom of a material layer. It is a float number in meters.
- pitch : pitch is an HDF5 real attribute and represents the distance between two wire centers in meters
- angle : angle is an HDF5 real attribute and represents the angle in degrees between wires and the X-axis.

Example of a woven Grid with a thickness of 5 mm :

```
data.h5
`-- physicalModel/
    |-- multilayer/
    |   `-- $multilayerMaterial
    |-- volume/
    |   |-- $resin
    |   |   |-- electricConductivity[@floatingType=singleReal
    |   |   |                        @value=0.0]
    |   |   |-- magneticConductivity[@floatingType=singleReal
    |   |   |                        @value=0.0]
    |   |   |-- relativePermittivity[@floatingType=singleReal
    |   |   |                        @value=2.0]
    |   |   `-- relativePermeability[@floatingType=singleReal
    |   |                            @value=1.0]
    |   `-- $fiber
    |       |-- electricConductivity[@floatingType=singleReal
    |       |                        @value=1.0e5]
    |       |-- magneticConductivity[@floatingType=singleReal
    |       |                        @value=0.0]
    |       |-- relativePermittivity[@floatingType=singleReal
    |       |                        @value=2.0]
    |       `-- relativePermeability[@floatingType=singleReal
    |                                @value=1.0]
    `-- grid/
        `-- $a-wovengrid[@surroundingMaterial=/physicalModel/volume/$resin
             |           @gridMaterial=/physicalModel/volume/$fiber
```

```
            |           @textureType=woven
            |           @pitchFiber=0.1e-3
            |           @fiberPerPitch=10
            |           @modelType=homogeneous]
            `-- comb1[@relativeHeight=0.5
                     @angle=90.0
                     @diameterWire=2.5e-5
                     @wireSectionType=circular]
```

$multilayerMaterial is a table which contains:

physicalModel	thickness
`/physicalModel/grid/$a-wovengrid`	5.0e-3

Example of a two single comb Grid with a thickness of 3 mm :

```
data.h5
`-- physicalModel/
    |-- multilayer/
    |   `-- $multilayerMaterial
    |-- volume/
    |   |-- $resin
    |   |   |-- electricConductivity[@floatingType=singleReal
    |   |   |                         @value=0.0]
    |   |   |-- magneticConductivity[@floatingType=singleReal
    |   |   |                         @value=0.0]
    |   |   |-- relativePermittivity[@floatingType=singleReal
    |   |   |                         @value=2.0]
    |   |   `-- relativePermeability[@floatingType=singleReal
    |   |                             @value=1.0]
    |   `-- $fiber
    |       |-- electricConductivity[@floatingType=singleReal
    |       |                         @value=1.0e5]
    |       |-- magneticConductivity[@floatingType=singleReal
    |       |                         @value=0.0]
    |       |-- relativePermittivity[@floatingType=singleReal
    |       |                         @value=2.0]
    |       `-- relativePermeability[@floatingType=singleReal
    |                                 @value=1.0]
    `-- grid/
        `-- $a-combgrid[@surroundingMaterial=/physicalModel/volume/$resin
            |          @gridMaterial=/physicalModel/volume/$fiber
            |          @textureType=comb
            |          @shift=0.3e-4
            |          @modelType=heterogeneous]
            |-- comb1[@relativeHeight=0.5
            |         @angle=45.0
            |         @pitch=1.0e-4
            |         @thicknessWire=2.5e-5
            |         @widthWire=5.0e-5
            |         @wireSectionType=rectangular]
            `-- comb2[@relativeHeight=0.4
                     @angle=135.0
                     @pitch=1.0e-4
                     @thicknessWire=2.5e-5
                     @widthWire=5.0e-5
                     @wireSectionType=rectangular]
```

$multilayerMaterial is a table which contains:

physicalModel	thickness
`/physicalModel/grid/$a-combgrid`	3.0e-3

Example of a random Grid with a thickness of 4 mm and three kind of wire:

```
data.h5
`-- physicalModel/
    |-- multilayer/
    |   `-- $multilayerMaterial
    |-- volume/
    |   |-- $resin
    |   |   |-- electricConductivity[@floatingType=singleReal
    |   |   |                        @value=0.0]
    |   |   |-- magneticConductivity[@floatingType=singleReal
    |   |   |                        @value=0.0]
    |   |   |-- relativePermittivity[@floatingType=singleReal
    |   |   |                        @value=2.0]
    |   |   `-- relativePermeability[@floatingType=singleReal
    |   |                            @value=1.0]
    |   |-- $fiber1
    |   |   |-- electricConductivity[@floatingType=singleReal
    |   |   |                        @value=1.5e5]
    |   |   |-- magneticConductivity[@floatingType=singleReal
    |   |   |                        @value=0.0]
    |   |   |-- relativePermittivity[@floatingType=singleReal
    |   |   |                        @value=1.0]
    |   |   `-- relativePermeability[@floatingType=singleReal
    |   |
    |   `-- $fiber2
    |       |-- electricConductivity[@floatingType=singleReal
    |       |                        @value=1.0e5]
    |       |-- magneticConductivity[@floatingType=singleReal
    |       |                        @value=0.0]
    |       |-- relativePermittivity[@floatingType=singleReal
    |       |                        @value=1.0]
    |       `-- relativePermeability[@floatingType=singleReal
    |                                @value=1.0]
    `-- grid/
        `-- $a-randomgrid[@surroundingMaterial=/physicalModel/volume/$resin
            |             @textureType=random]
            |-- a-nano-filler[@gridMaterial=/physicalModel/volume/$fiber1
            |                 @scaleFiller=nano
            |                 @typeFiller=rod
            |                 @volFractioFiller=0.1
            |                 @diameterWire=1.e-9
            |                 @lengthWire=5.0e-8]
            |-- a-micro-filler[@gridMaterial=/physicalModel/volume/$fiber2
            |                  @scaleFiller=micro
            |                  @typeFiller=rod
            |                  @volFractioFiller=0.2
            |                  @diameterWire=1.e-5
            |                  @lengthWire=5.0e-5]
            `-- another-micro-filler[@gridMaterial=/physicalModel/volume/$fiber2
                                     @scaleFiller=micro
                                     @typeFiller=rod
                                     @volFractioFiller=0.05
                                     @diameterWire=0.5e-6
                                     @lengthWire=1.0e-5]
```

$multilayerMaterial is a table which contains:

physicalModel	thickness
`/physicalModel/grid/$a-randomgrid`	4.0e-3

CHAPTER

TEN

EXCHANGE SURFACE

In the context of multi-method simulation or multi-scale simulation it often happens that numerical data produced by a computation must be used as input by another computation.

The exchange surface stores numerical `data` on a mesh according the huygens principle or the reciprocity principle.

10.1 Exchange Surface

An exchange surface is a named HDF5 group child of the category `/exchangeSurface`. The name's length must have less than 20 characters.

An exchange surface has two attributes :

- `type`. The `type` can take the following values :
  - `reciprocity`
  - `huygens`
  - `gauss`
- `nature`
  - `outside` : the energy is radiated outside the surface
  - `inside` : the energy is radiated inside the surface

Example an `exchangeSurface` called `/exchangeSurface/$surf1` :

```
data.h5
|-- mesh/
|   `-- $gmesh1/
|       `-- $mesh1[@type=unstructured]
`-- exchangeSurface/
    `-- $surf1[@type=huygens
               @nature=inside]
```

An exchange surface can have several single surfaces children, each single surface contains a quantity component relative to the principle surface.

10.2 Single Surface

An exchange surface can have several single surfaces, a quantity per single surface is stored. Each quantity is then stored in an HDF5 named group child of the exchange surface.

The name's length of the component group must have less than 20 characters.

A single surface is a `floatingType` equals `arraySet` representing numerical data on mesh (see *Numerical data on mesh*).

Example of an exchange surface `/exchangeSurface/$surf1` made up of two single surfaces `/exchangeSurface/$surf1/$surf11` and `/exchangeSurface/$surf1/$surf12` :

```
data.h5
|-- mesh/
|   `-- $gmesh1/
|       `-- $mesh1[@type=unstructured]
|           |-- nodes
|           `-- group
|               |-- $point_cloud[@type=node]
|               `-- $exchange_surface[@type=element
|                                     @entityType=face]
`-- exchangeSurface
    `-- $surf1[@type=huygens
        |      @nature=inside]
        |-- $surf11[@floatingType=arraySet           # single surface
        |    |       @physicalNature=electricField
        |    |       @unit=voltPerMeter]
        |    |-- data
        |    `-- ds
        |        |-- dim1[@physicalNature=component]
        |        `-- dim2[@physicalNature=meshEntity
        |                 @meshEntity=/mesh/$gmesh1/$mesh1/group/$exchange_surface]
        `-- $surf11[@floatingType=arraySet           # single surface
            |       @physicalNature=magneticField
            |       @unit=henryPerMeter]
            |-- data
            `-- ds
                |-- dim1[@physicalNature=component]
                `-- dim2[@physicalNature=meshEntity
                         @meshEntity=/mesh/$gmesh1/$mesh1/group/$exchange_surface]
```

In addition, consider a points cloud representing locations where physical quantities are computed. The huygens principle implies physical quantities are balanced by the a surface and oriented. The approach is to create a mesh with patches, the surface of a patch is the weight, a patch is also oriented, so the single surface is properly defined.

Example :

```
data.h5
|-- mesh/
|   `-- $gmesh1
|       `-- $mesh2[@type=unstructured]
|           |-- nodes
|           |-- elementTypes
|           `-- group
|               `-- $circular_patch[@type=element
|                                   @entityType=face]
`-- exchangeSurface
    `-- $surf1[@type=huygens
        |      @nature=inside]
        |-- $surf11[@floatingType=arraySet           # single surface
        |    |       @physicalNature=electricField
        |    |       @unit=voltPerMeter]
        |    |-- data
        |    `-- ds
        |        |-- dim1[@physicalNature=component]
        |        `-- dim2[@physicalNature=meshEntity
        |                 @meshEntity=/mesh/$gmesh1/$mesh2/group/$circular_patch]
        `-- $surf11[@floatingType=arraySet           # single surface
            |       @physicalNature=magneticField
            |       @unit=henryPerMeter]
            |-- data
            `-- ds
                |-- dim1[@physicalNature=component]
                `-- dim2[@physicalNature=meshEntity
                         @meshEntity=/mesh/$gmesh1/$mesh2/group/$circular_patch]
```

`data.h5:/mesh/$gmesh1/mesh2/group/$circular_patch` is a group of circle elements, their center are the quantities location, the surface is the weight of each value.

CHAPTER

ELEVEN

LINK AND LABEL

In cases of computing methods based on the mesh concept like the finite differences, finite elements or finite volumes, mesh entities have to be associated with physical models like material models or electromagnetic sources.

Sometimes, an object plays a specific role in a simulation, for instance an antenna is the transmitter and another antenna is the receiver and the module must make a difference between the two antennas.

Or else, two objects must be associated, for instance to express a plane incident wave is propagated toward a network tube.

For all those issues, **Amelet HDF** introduces the `Link` concept. In a few words, a `Link` couples two objects and gives the relation a sense.

11.1 Link category

The `/link` category contains all associations between elements of an **Amelet HDF** instance :

Examples :

- Links between physical models and meshes
- Links between physical models and networks
- Links between label and object
- Links between model instances...

```
data.h5
`-- link/
```

In the `link` category, links are gathered in groups. The number of groups and the name of groups are not specified.

Link groups are named HDF5 groups.

```
data.h5
`-- link/
    |-- $link_group1/
    `-- $link_group2/
```

Link groups are described further in the text.

11.2 Link

Finally, link group children (`data.h5:/link/$link_group/$link_instance`) are called link instance.

A link instance is an HDF5 group with two mandatory attributes :

- `subject`. `subject` is an HDF5 string attribute representing the name of an element in the **Amelet HDF** file
- `object`. `object` is an HDF5 string attribute representing the name of an element in the **Amelet HDF** file

Example :

```
data.h5
|-- mesh/
|   `-- $gmesh1/
|       `-- $plane/
|           `-- group
|               `-- $wing
|-- physicalModel/
|   `-- volume/
|       `-- $diel1
`-- link/
    `-- $link_group/
        `-- $link_instance1[@subject=/physicalModel/volume/$diel1
                            @object=/mesh/$gmesh1/$plane/group/$wing]
```

This example shows a link (`data.h5:/link/$link_group/$link_instance`) between a volume material (`$diel1`) and the `$wing` mesh group.

11.2.1 Tables and datasets

Imagine we want to associate a label to a network's tube. We have to select a label in a dataset (/label/$some_labels for instance) and to select a row in a table (/network/$a_network/tubes for instance)

In the case of a table, an attribute is added : the name of this attribute is built with a prefix (`subject_` or `object_`) concatenated with the name of the column. The value of the attribute is the value of the cell.

In the case of a dataset, an attribute is added : the name of this attribute is built with a prefix (`subject_` or `object_`) concatenated with "id". The value of the attribute is the coordinates the cell :

```
data.h5
|-- label/
|   `-- $some_labels
|-- transmissionLine/
|-- network/
|   `-- $a_network/
|       |-- junctions
|       |-- tubes
|       `-- connections
`-- link/
    `-- $link_group/
        `-- $link_instance1[@subject=/label/$some_labels
                            @subject_id=2
                            @object=/network/$a_network/tubes
                            @object_id=$tub_1]
```

with the `data.h5:/label/$some_labels` dataset :

first
last
ground_height

with the `data.h5:/network/$a_network` table :

id	extremity1	extremity2	transmissionLine
...			
$tub_1	$j1	$j2	/transmissionLine/$tl1
$tub2	$j2	$j3	/transmissionLine/$tl2
$tub#3	$j1	$j3	/transmissionLine/$tl3
...			

Here, the link associates the `ground_height` to the tube `$tube_1`

Warning: In @subject_id=2, _id is the cell's index and in @object_id=$tub_1, id is the column's name

Note: If the element to be picked is in a dataset, the `object_id` (or `subject_id`) attribute is used. This attribute contains the coordinates of the element inside the dataset.

For instance, if the dataset is a two dimensional dataset (N x M), `object_id` equals `(n, m)`. `object_id` is a multi value HDF5 integer attribute.

11.2.2 Label link

Links, where the subject is a label, are usually used to set the role of an element in the context of a simulation. For instance, a module would need to know whether an antenna is the transmitter or the receptor.

For this use case, **Amelet HDF** introduces the `/label` category which aims at converting a string label into an **Amelet HDF** element.

The `label` category is an HDF5 group containing only HDF5 dataset :

- The name of the datasets is not specified
- The datasets have only one column.
- The dataset contain 100 character HDF5 strings.

Example of a label table :

```
data.h5
`-- label/
    `-- $label_dataset1
```

with `data.h5:/label/$label_dataset1` :

transmitter
receptor
useSecondOrder

Example of a link using a label :

```
data.h5
|-- label/
|  `-- $label_dataset1
|-- mesh/
|   `-- $gmesh1
|       `-- $plane
|           `-- group
|               `-- $wing
|-- electromagneticSource/
|   |-- antenna
|   |   `-- $antenna1
```

```
`-- link/
    `-- $link_group
        |-- $link_instance1[@subject=/label/$label_dataset1
        |                    @subject_id=0
        |                    @object=/electromagneticSource/antenna/$antenna1]
        `-- $link_instance2[@subject=/label/$label_dataset1
                             @subject_id=2
                             @object=/mesh/$gmesh1/$plane/group/$wing]
```

In this example, the label "transmitter" is linked to the `/electromagneticSource/antenna/$antenna1` antenna thanks to the `data.h5:/link/$link_group/$link_instance1` link and the label "useSecondOrder" is linked to /mesh/$gmesh1/$plane/group/$wing.

Note: Labels are relative to a module and can be defined by the module's developer.

11.2.3 Link options

A link instance can have many options (defined as simulation's parameters), they can simple (i.e. a native type) or compound (i.e. a structure) :

- The simple types are :
  - integer
  - real
  - string
  - boolean
- A compound type is a type made of a list of named simple types, example : (param1: integer, param2: real, param3: string)

A link is an HDF5 group, link's options are written as follows in a link :

- Simple type options become simple HDF5 named attributes
- Compound parameters are stocked in named HDF5 tables where columns are the structure's fields and the table's name is the parameter's name.

For instance, to place a generator on a tube's wire of a network, an integer `idWire` attribute is mandatory to select the wire :

```
data.h5
|-- electromagneticSource/
|   `-- generator
|       `-- $v1
|-- mesh/
|   `-- $gmesh/
|       `-- $umesh
|           `-- group
|               `-- $tube1[@type=element
|                           @entityType=edge]
|-- physicalModel/
|-- transmissionLine/
|-- network/
|   `-- $network1
|       |-- tubes
|       |-- junctions
|       `-- connections
`-- link/
    `-- $link_group
        `-- $link_instance[@subject=/electromagneticSource/generator/$v1
                           @object=/mesh/$gmesh/$umesh/group/$tube1
                           @idWire=1]
```

Note: Options are relative to a module and can be defined by the module's developer.

11.3 Link group

As seen above, in the `link` category links are gathered in groups. The number of groups and the name of groups are not specified.

Link group children are named HDF5 groups which gather genuine link instances.

Each link group has several optional attribute :

- `rootSubject`. In a link group all link subjects can have a common root in their name. `rootSubject` is an HDF5 string attribute representing this common root part.
- `rootObject`. In a link group all link objects can have a common root in their name. `rootObject` is an HDF5 string attribute representing this common root part.
- `type`. `type` is an HDF5 string attribute. This attribute sets the type of all its children.

Example of `rootObject` usage :

```
data.h5
|-- mesh/
|   `-- $gmesh1/
|       `-- $plane/
|           |-- nodes
|           |-- elementTypes
|           |-- elementNodes
|           |-- group
|           |   |-- $phase_origine[@type=node]
|           |   `-- $wing[@type=element]
|           `-- selectorOnMesh
|               |-- $generator_v1[@type=pointInElement]
|               `-- $generator_i1[@type=pointInElement]
|-- electromagneticSource/
|   |-- planeWave
|   |   `-- $pw1
|   `-- generator/
|       |-- $v1
|       `-- $i1
`-- link/
    `-- $link_group[@rootObject=/mesh/$gmesh1/$plane/selectorOnMesh]
        |-- $link_instance1[@subject=/electromagneticSource/generator/$v1
        |                   @object=$generator_v1]
        `-- $link_instance2[@subject=/electromagneticSource/generator/$i1
                            @object=$generator_i1]
```

with `data.h5:/mesh/$gmesh1/$plane/selectorOnMesh/$generator_v1` :

index	v1	v2	v3
23	0.5	-1	-1

with `data.h5:/mesh/$gmesh1/$plane/selectorOnMesh/$generator_i1`:

index	v1	v2	v3
26	0.5	-1	-1

and with `data.h5:/mesh/$gmesh1/$plane/group/$phase_origin`:

124

Note: The `data.h5:/link/$link_group/$link instance` shows that link instance `object` (or `subject`) attribute can still be used, in this case it represents a name part to be added to `rootObject` (or `rootSubject`).

CHAPTER

TWELVE

NETWORKS AND TRANSMISSION LINES

A transmission line is the material medium or structure that forms all or part of a path from one place to another for directing the transmission of energy, such as electromagnetic waves or acoustic waves, as well as electric power transmission. Components of transmission lines include wires, coaxial cables, dielectric slabs, optical fibers, electric power lines, and waveguides (http://en.wikipedia.org/wiki/Transmission_line).

Some practical types of transmission lines are :

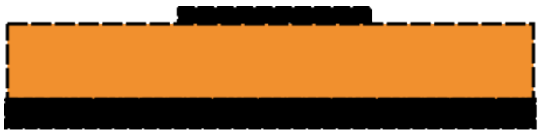
- Coaxial cable
- Microstrip
- Stripline
- Balanced line

12.1 Transmission line

In **Amelet HDF** Transmission lines are contained in the category `/transmissionLine`. A transmission line is a named HDF5 group.

This name most have less than 20 characters.

A transmission line is mainly defined by a cross section :



Cross-section of microstrip geometry. Conductor is separated from ground plane by dielectric substrate.

This cross section is described by an `unstructured` `/mesh` and `/physicalModel/` which are associated thanks to a `dataOnMesh` element.

Therefore, a transmission line group has three attibutes :

- `properties` : it is an HDF5 group that contains the definitions of the distributed properties of the transmission line. This group has three exclusive set of children depending on the `type` attribute. The `type` attribute can take the three following values :
  - If `type` equals `RLCG`, the distributed properties are defined with the four matrix R, L, C and G. Thus, `properties` has four children :
    - R, R is a distributed resistance matrix `floatingType` defined by :
      - `physicalNature` = `resistance`
      - `unit` = `ohmPerMeter`

    * L, L is a distributed inductance matrix `floatingType` defined by :
        · `physicalNature` = `inductance`
        · `unit` = `henryPerMeter`
    * C, C is a distributed capacitance matrix `floatingType` defined by :
        · `physicalNature` = `capacitance`
        · `unit` = `faradPerMeter`
    * G, G is a distributed conductance matrix `floatingType` defined by :
        · `physicalNature` = `conductance`
        · `unit` = `siemensPerMeter`

  – If `type` equals `ZY`, the distributed properties are defined with the two matrices Z and Y Thus, `properties` has two children :
    * Z, Z is a distributed impedance matrix `floatingType` defined by :
        · `physicalNature` = `impedance`
        · `unit` = `ohmPerMeter`
    * Y, Y is a distributed admittance matrix `floatingType` defined by :
        · `physicalNature` = `admittance`
        · `unit` = `siemensPerMeter`

  – If `type` equals `ZcGamma`, the distributed properties are defined with the two matrix Zc and gamma Thus, `properties` has two children :
    * Zc, Zc is the characteristic impedance of the transmission and is a distributed impedance matrix `floatingType` defined by :
        · `physicalNature` = `impedance`
        · `unit` = `ohm`
    * gamma, gamma is the propagation constant per meter and has one attribute :
        · `physicalNature` = `propagationConstant`
        · `unit` = `perMeter`

12.1.1 Properties types

Example of a transmission line `data.h5:/transmissionLine/$tl1` with `type` = `RLCG` :

```
data.h5
`-- transmissionLine
    `-- $tl1
        `-- properties[@type=RLCG]
            |-- R[@floatingType=dataSet
            |     @physicalNature=resistance]
            |-- L[@floatingType=dataSet
            |     @physicalNature=inductance]
            |-- C[@floatingType=dataSet
            |     @physicalNature=capacitance]
            `-- G[@floatingType=dataSet
                  @physicalNature=conductance]
```

Example of a transmission line `data.h5:/transmissionLine/$tl1` with `type` = `ZY` :

```
data.h5
`-- transmissionLine
    `-- $tl1
        `-- properties[@type=ZY]
            |-- Z[@floatingType=dataSet
            |     @physicalNature=impedance]
            `-- Y[@floatingType=dataSet
                  @physicalNature=admittance]
```

Example of a transmission line `data.h5:/transmissionLine/$tl1` with `type` = `ZcGamma` :

```
data.h5
`-- transmissionLine
    `-- $tl1
        `-- properties[@type=ZcGamma]
            |-- Zc[@floatingType=dataSet
            |      @physicalNature=impedance]
            `-- gamma[@floatingType=dataSet]
```

12.1.2 Transmission line elements

Besides, a transmission line is made up of transmission line elements, those elements are conductors, shields ...

Example of a transmissione line `data.h5:/transmissionLine/$tl1` with the `element` group

```
data.h5
`-- transmissionLine
    `-- $tl1
        |-- properties[@type=ZcGamma]
        `-- element                    # Transmission line element group
```

In **Amelet HDF**, elements are named HDF5 group children of the group `element` and have five attributes :

- `type` : it is an HDF5 string attribute that can take the following values :
  - if `type` = `conductor`, the element is a conductor
  - if `type` = `shield`, the element is a shield, a shield is a conductor reference for other conductor
  - if `type` = `dielectric`, the element is a dielectric
- `domain` : the `domain` references the shield domain.
- `rank` : This is the position of the element in the distributed properties matrix
- `referenceElement` : This is the position of the reference conductor

A transmission line element has also an optional child :

- `properties` : this is an HDF5 group, it contain the individual distributed properties for the element (in the case of a measurement for instance) and the transfer distributed properties relative to the shield conductor. `properties` has two string optional attributes : `type` and `transferType`.
  - if `type` = `RLCG`, `properties` has four children R, L, C, G. This elements are defined as in the transmission line distributed properties, they represents the distributed properties of one element.
  - if `transferType` = `ZY`, transfer distributed properties are defined by Zt and Yt. This elements are defined as in the transmission line distributed properties Z and Y, they represents the transfer distributed properties of one element.

  - if `transferType` = `RLCG`, transfer distributed properties are defined by Rt, Lt, Ct and Gt. These elements are defined as in the transmission line distributed properties R, L, C and G, they represents the distributed transfer properties of one element.

Example of a transmission line `data.h5:/transmissionLine/$tl1` which have two elements (the `$ground` and `$elem1`) :

```
data.h5
`-- transmissionLine
    `-- $tl1
        |++ properties
        `-- element
            |-- $ground[@type=conductor
            |           @domain=0]
            `-- $elem1[@type=conductor
                |      @domain=0
                |      @rank=1
                |      @referenceElement=$ground]
                `-- properties[@type=RLCG
                    |          @transfertType=ZY]
                    |-- Zt[@floatingType=dataSet
                    |      @physicalNature=impedance]
                    |-- Yt[@floatingType=dataSet
                    |      @physicalNature=admittance]
                    |-- R[@floatingType=dataSet
                    |     @physicalNature=resistance]
                    |-- L[@floatingType=dataSet
                    |     @physicalNature=inductance]
                    |-- C[@floatingType=dataSet
                    |     @physicalNature=capacitance]
                    `-- G[@floatingType=dataSet
                          @physicalNature=conductance]
```

- `data.h5:/transmissionLine/$tl1/element/$ground` is a `conductor` and is in the domain 0.
- `data.h5:/transmissionLine/$tl1/element/$elem1` is a `conductor` and is also in the domain 0.

12.1.3 TranmissionLineOnMesh link

The section of a transmission line is often associated with a mesh, this association gives the following information :

- the mesh of the transmission line section
- the models that are linked to the mesh entities, this is the role of the `modelsOnSectionLink` attribute. `modelsOnSectionLink` is an HDF5 string attribute which gives the name of the group links defining association between models and the mesh entities.
- the link between the transmission line elements and mesh entities, this is the role of the `elementsOnSection` dataset. `elementsOnSection` is an HDF5 (n x 2) string dataset which contains a list of (transmission line element, named mesh entity) string couple defining the association between transmission line elements and mesh entities.

This association is created thanks a *Link* which links a `transmissionLine` to a `mesh`.

A link group containing only links between `transmissionLine` and `mesh` has the attribute `type` equals `transmissionLineOnMesh` (see *TransmissionLineOnMesh Link*)

Example :

```
data.h5
|-- mesh/
|   `-- $gmesh1
|       `-- $tl1/                                  # Mesh description of $tl1
|           `-- group/
|               |-- $elem1
|               `-- $elem2
|-- physicalModel/
|   `-- volume/
|       `-- $diel1/
|-- link/
|   |-- $models_on_section/                        # Material model / mesh
|   |   |-- $link1[@subject=/physicalModel/volume/$diel1
|   |   |          @object=/mesh/$gmesh1/$mesh1/group/$elem1]
|   |   `-- $link2[@subject=/physicalModel/volume/$diel1
|   |              @object=/mesh/$gmesh1/$mesh1/group/$elem2]
|   `-- $transmissionline_on_mesh[@type=transmissionLineOnMesh]/
|       `-- $link1[@subject=/transmissionLine/$tl1
|           |      @object=/mesh/$gmesh1/$mesh1
|           |      @modelsOnSectionLink=/link/$models_on_section]
|           `-- elementsOnSection
`-- transmissionLine
    `-- $tl1
        `-- element
            |++ $ground
            `++ $elem1
```

with `data.h5:/link/$transmission_line_section/$link1/elementsOnSection`:

$ground	$elem1
$elem1	$elem2

12.1.4 A more complete example

Let the following **Amelet HDF** instance, it defines three transmission lines `/transmissionLine/$tl1`, `/transmissionLine/$tl2` and `/transmissionLine/$tl3`.

```
data.h5
|-- mesh
|   `-- $gmesh1
|       |-- $tl1                                 # Mesh description of
|       |   |-- nodes                            # transmission lines
|       |   |-- elementTypes                     # An unstructured mesh
|       |   |-- elementNodes                     # with three groups
|       |   `-- group                            # of the transmission lines
|       |       |-- $elem1[@type=element         # (cross sections)
|       |       |          @elementType=edge]
|       |       |-- $elem2[@type=element
|       |       |          @elementType=edge]
|       |       `-- $elem3[@type=element
|       |                  @elementType=edge]
|       |++ $tl2
|       `++ $tl3
|-- physicalModel/
|   |-- volume/                               # Material models of
|   |   |-- $diel1                            # transmission lines
|   |   `-- $diel2
|   `-- interface/
|       |-- $interface1
|       `-- $interface2[@medium1=/physicalModel/volume/$diel1
|                       @medium2=/physicalModel/volume/$diel2]
|-- link
|   |-- $tl1_section[@type=dataOnMesh]             # Material model / mesh
|   |   |-- $link_instance1[@subject=/physicalModel/perfectElectricConductor
|   |   |                   @object=/mesh/$gmesh1/$tl1/group/$elem1]
|   |   |-- $link_instance2[@subject=/physicalModel/interface/$interface2
|   |   |                   @object=/mesh/$gmesh1/$tl1/group/$elem2]
|   |   `-- $link_instance3[@subject=/physicalModel/perfectElectricConductor
|   |                       @object=/mesh/$gmesh1/$tl1/group/$elem3]
|   |++ $tl2_section
|   |++ $tl3_section
|   `-- $transmissionline_on_mesh[@type=transmissionLineOnMesh]
|       |-- $tl1[@subject=/transmissionLine/$tl1
|       |   |    @object=/mesh/$gmesh1/$tl1
|       |   |    @modelsOnSectionLink=/link/$tl1_section]
|       |   `-- elementsOnSection
|       |-- $tl2[@subject=/transmissionLine/$tl2
|       |   |    @object=/mesh/$gmesh1/$tl2
|       |   |    @modelsOnSectionLink=/link/$tl2_section
|       |   `-- elementsOnSection
|       `-- $tl3[@subject=/transmissionLine/$tl3
|           |    @object=/mesh/$gmesh1/$tl3
|           |    @modelsOnSectionLink=/link/$tl3_section
|           `-- elementsOnSection
`-- transmissionLine
    |-- $tl1
    |   |-- properties[type=RLCG]
    |   |   |-- R[@floatingType=dataSet
    |   |   |     @physicalNature=resistance]
    |   |   |-- L[@floatingType=dataSet
    |   |   |     @physicalNature=inductance]
    |   |   |-- C[@floatingType=dataSet
    |   |   |     @physicalNature=capacitance]
    |   |   `-- G[@floatingType=dataSet
    |   |         @physicalNature=conductance]
    |   `-- element
    |       |-- $elem1[@type=conductor
    |       |          @domain=0
    |       |          @rank=0]
    |       |-- $elem2[@type=dielectric]
    |       `-- $elem3[@type=shield
    |                  @referenceElement=$elem1
    |                  @internalLine=/transmissionLine/$tl2
    |                  @domain=0
    |                  @rank=1]
    |-- $tl2
    |   |-- properties
    |   `-- element
    |       `-- $elem1[@type=conductor
    |                  @domain=0
    |                  @rank=1]
    `++ $tl3
```

with `data.h5:/link/$transmissionline_on_mesh/$tl1/elementsOnSection`:

$elem1	$elem1
$elem2	$elem2
$elem3	$elem3

This example describes three transmission lines `/transmissionLine/$tl1`, `/transmissionLine/$tl2` and `/transmissionLine/$tl3`. `/transmissionLine/$tl1` has three elements `$elem1`, `$elem2` and `$elem3`. `$elem1` is the electrical reference, `$elem2` is a dielectric element and `$elem3` is a shield.

In addition, all transmission line are associated with a mesh for the section definition, see the `data.h5:/link/transmission_line_on_mesh/$tl1` for `data.h5:/transmissionLine/$tl1` for instance.

12.2 Network

The definition of a network is straightforward. A network is named HDF5 group in the category `network`.

A network has two attributes :

- `type` : it is an mandatory HDF5 string attribute, it gives the `type` of a network. The `type` attributes values are :
  - `simple`. If `type` equals `simple`, the network element defines simple network with no interconnection with another network.
  - `compound`. If `type` equals `compound`, the network defines interconnected network thanks to interconnection tubes and `simple` network.

12.2.1 Simple network

A `simple` network has three children :

- `tubes` : it is an HDF5 table of four columns
  - `id` : an HDF5 string of 20 characters
  - `extremity1` : it is an HDF5 string attribute, it contains the name of the first extremity junction. It is a relative name because the junction is defined inside the network.
  - `extremity2` : it is an HDF5 string attribute, it contains the name of the second extremity junction. It is a relative name because the junction is defined inside the network.
  - `transmissionLine`: it is an HDF5 string attribute, it is the name of the model of transmission line composing the tube.

  Note: If a tube is **zero length**, there is no associated transmission line, the `transmissionLine` field must be equal to the **empty string**.

- `junctions` : it is an HDF5 three column table which defines junctions
  - `id` : `id` is the identifier of the junction, it is an 20 characters HDF5 string
  - `nbPort` : `nbPort` is an integer, it is the number of ports of the multiport which represents the junction.

  Note: A junction can be associated to :

  * A predefined multiport
  * A scalar multiport (`singleReal`, `singleComplex`), the junction's `nbPort` attribute override the implicit value (equals to 1) of `floatingType`

    * A matrix multiport (`dataSet`, `arraySet`), if present the junction's `nbPort` attribute must be equal the dimension of the `floatingType`

  - `multiport` : the name of a `/physicalModel/multiport` object

- `connections` : it is an HDF5 integer table, a connection is a row and is made up of :
  - `idJunction` : a junction identifier
  - `idPort` : a port identifier
  - `idTube` : a tube identifier
  - `idWire` : a wire/transmission line element identifier

Example :

```
data.h5
|-- physicalModel/
|   `-- multiport
|       |-- $j0[@floatingType=singleComplex
|       |       @physicalNature=impedance
|       |       @unit=hertz]
|       |-- $j1[@floatingType=dataSet
|       |       @physicalNature=impedance
|       |       @unit=ohm]
|       |-- $j2[@physicalNature=admittance
|       |   |   @floatingType=arraySet
|       |   |   @unit=ohm]
|       |   |-- data
|       |   '-- ds
|       |       |--dim1[@label=frequency
|       |       |       @physicalNature=frequency
|       |       |       @unit=hertz]
|       |       |--dim2[@label=nbPort
|       |       |       @physicalNature=electricPotentialPoint]
|       |       '--dim3[@label=nbPort
|       |               @physicalNature=electricPotentialPoint]
|       `-- sParameter
|           |--$j3[@floatingType=singleComplex
|           |      @referenceImpedance=50
|           |      @value=(0,0)]
|           '--$j4[@floatingType=dataSet]
|-- transmissionLine/
|   |-- $tl1
|   |-- $tl2
|   `-- $tl3
`-- network/
    `-- $net1[@type=simple]
        |-- tubes
        |-- junctions
        `-- connections
```

The table `/network/$net1/junctions`

The following table shows a part of the table `junctions` :

id	nbPort	multiport
...		
$j0	2	/physicalModel/multiport/$j0
$j1	2	/physicalModel/multiport/$j1
$j2	2	/physicalModel/multiport/$j2
$j3	8	/physicalModel/sParameter/$j3
$j4	8	/physicalModel/sParameter/$j4
$j5	5	/physicalModel/multiport/shortCircuit
...		

The table `/network/$net1/tubes`

The following table shows a part of the table `tubes` :

id	extremity1	extremity2	transmissionLine
...			
$tub_1	$j1	$j2	/transmissionLine/$tl1
$tub2	$j2	$j3	/transmissionLine/$tl2
$tub#3	$j1	$j3	/transmissionLine/$tl3
...			

The numbering of tubes is explicit, that facilitates the possible modifications.

The table `/network/$network/connections`

The following table shows a part of the table `connections` corresponding to the preceding example :

idJunction	idPort	idTube	idWire
...			
$j1	1	$tub_1	1
$j1	2	$tub_1	2
$j1	3	$tub2	1
$j1	4	$tub22	2
...			

The numbering of connections is implicit. idJunction is the name of children of `/network/$network/junctions`. idTube is the identifier from the table `/network/$network/tubes`.

`idPort` is a port of the multiport model representing the junction and `idWire` is a wire number of a tube.

Network topology and tube length

In order to make a real network, tubes must have a length. In the general case, junctions are located in 3D space and the network has a spatial reality, it is the network harness.

That's why a network is associated with a mesh, each tube of the network is linked to a mesh entity.

Consider the following network, in particular the `/network/$net1/tubes` tables and the `/mesh/$gmesh1/$tubes` mesh. They are linked by the `/link/$network_on_mesh/$net1` link :

```
data.h5
|-- mesh
|   `-- $gmesh1
|       |-- $tl1
|       |-- $tl2
|       |-- $tl3
|       `-- $tubes
|           |-- nodes
|           |-- elementTypes
|           |-- elementNodes
|           `-- group
|               |-- $tube1[@type=element
|               |          @elementType=edge]
|               |-- $tube2[@type=element
|               |          @elementType=edge]
|               `-- $tube3[@type=element
|                          @elementType=edge]
|-- transmissionLine/
|   |-- $tl1
|   |-- $tl2
|   `-- $tl3
|-- link
|   `-- $network_on_mesh[@type=networkOnMesh]
|       `-- $net1[@subject=/network/$net1
|           |     @object=/mesh/$gmesh1/$tubes]
|           `-- data
`-- network/
    `-- $net1[@type=simple]
        |-- tubes
        |-- junctions
        `-- connections
```

with `/network/$net1/tubes` :

id	extremity1	extremity2	transmissionLine
$tub_1	$j1	$j2	/transmissionLine/$tl1
$tub2	$j2	$j3	/transmissionLine/$tl2
$tub#3	$j1	$j3	/transmissionLine/$tl3

In this example, a new link `/link/$network_on_mesh/$tub1` is defined, it links a network to a mesh, for doing this association the link contains a child : an HDF5 string dataset named `data`.

The `data` dataset has two columns :

- The first column contains tubes' `id` from the network's `tubes` dataset
- The second column contains named elements from the mesh.

For example, `data.h5:/link/$network_on_mesh/$net1/data` is :

$tub_1	$tube1
$tub2	$tube2
$tub#3	$tube3

For instance, `$tub_1` in the `/network/$net1` network is `$tube1` in the `/mesh/$gmesh1/$tubes` mesh.

A link group containing only links between `network` and `mesh` has the attribute `type` equals `networkOnMesh` (see *NetworkOnMesh Link*)

Fictitious harness

For simple networks, there is sometimes no need to construct a real harness, but the length of tubes must be present. A manner to accomplish that is to build a fake mesh (not real) and to link this mesh to the network with the `harness` attribute equals to `fictitious` (`harness` equals `real` by default).

Practically, for each tube create two nodes so that the distance between the nodes is the length of the tube :

```
data.h5
|-- mesh
|   `-- $gmesh1
|       |-- $tl1
|       `-- $tubes
|           |-- nodes
|           |-- elementTypes
|           |-- elementNodes
|           `-- group
|               `-- $tube1[@type=element
|                          @elementType=edge]
|-- transmissionLine/
|   `-- $tl1
|-- link
|   `-- $network_on_mesh[@type=networkOnMesh]
|       `-- $net1[@subject=/network/$net1
|           |     @object=/mesh/$gmesh1/$tubes
|           |     @harness=fictitious]
|           `-- data
`-- network/
    `-- $net1[@type=simple]
        |-- tubes
        |-- junctions
        `-- connections
```

with `/network/$net1/tubes` :

id	extremity1	extremity2	transmissionLine
$tub_1	$j1	$j2	/transmissionLine/$tl1

with `/mesh/$gmesh1/$tubes/nodes` :

0	0	0
$length	0	0

with `/mesh/$gmesh1/$tubes/elementTypes` :

1

and `/mesh/$gmesh1/$tubes/elementNodes` :

0
1

12.2.2 Compound network

A `compound` network has two children :

- `tubes` : it is an HDF5 table of six columns
  - `id` : an HDF5 string of 20 characters, it is the id of the tube
  - `network1` : it is an HDF5 string attribute, it contains the name of the first extremity network
  - `extremity1` : it is an HDF5 string attribute, it contains the name of the first extremity junction. It is the name of the the junction found out in /network/$network/junctions
  - `network2` : it is an HDF5 string attribute, it contains the name of the second extremity network

  - `extremity2` : it is an HDF5 string attribute, it contains the name of the second extremity junction. It is the name of the the junction found out in /network/$network/junctions
  - `transmissionLine`: it is an HDF5 string attribute, it is the name of the model of transmission line composing the tube.

  Note: If a tube is **zero length**, there is no associated transmission line, the `transmissionLine` field must be equal to the **empty string**.

- `connections` : it is an HDF5 integer table, a connection is a row and is made up of :
  - `idJunction` : it is an HDF5 string, it contains a junction identifier
  - `idPort` : it is an HDF5 integer, it contains a port identifier
  - `idTube` : it is an HDF5 string, it contains a tube identifier
  - `idWire` : it is an HDF5 string, it contains a wire/transmission line element name

Example of a `compound` network $net3:

```
data.h5
|-- mesh
|   `-- $gmesh1
|       |-- $tl1
|       |-- $tl2
|       |-- $tl3
|       `-- $tubes
|           |-- nodes
|           |-- elementTypes
|           |-- elementNodes
|           `-- group
|               |-- $tube1[@type=element
|               |           @elementType=edge]
|               |-- $tube2[@type=element
|               |           @elementType=edge]
|               |-- $tube3[@type=element
|               |           @elementType=edge]
|               `-- $tube4[@type=element
|                           @elementType=edge]
|-- link
|   `-- $dom1
|-- transmissionLine/
|   |-- $tl1
|   |-- $tl2
|   `-- $tl3
`-- network/
    |-- $net1[@type=simple]
    |   |-- tubes
    |   |-- junctions
    |   `-- connections
    |-- $net2[@type=simple]
    |   |-- tubes
    |   |-- junctions
    |   `-- connections
    `-- $net3[@type=compound]
        |-- tubes
        `-- connections
```

with `data.h5:/network/$net3/tubes` :

id	network1	extremity1	network2	extremity2	transmissionLine
$tub1	/network/$net1	$j1	/network/$net2	$j1	/transmissionLine/$tl1

and the table `/network/net3/connections` is :

network	idJunction	idPort	idTube	idWire
/network/$net1	$j1	1	$tub1	1
/network/$net2	$j2	2	$tub1	2

12.3 Locating a model on a network

Consider the preceding simple network (one transmission line, a one tube network and an harness), you would like to put a generator on the second wire of the tube.

The solution is to add a link between a generator and a 1D mesh entity, the `idWire` option allows to select the wire.

```
data.h5
|-- mesh/
|   `-- $gmesh1
|       |-- $tl1
|       `-- $tubes
|           |-- nodes
|           |-- elementTypes
|           |-- elementNodes
|           |-- selectorOnMesh
|           |   `-- $gene1[@type=pointInElement]
|           `-- group
|               `-- $tube1[@type=element
|                          @elementType=edge]
|-- electromagneticSource/
|   `-- generator
|       `-- $gene1
|-- transmissionLine/
|   `-- $tl1
|-- link
|   |-- $data_on_mesh[@type=dataOnMesh]
|   |   `-- $gene1[@subject=/electromagneticSource/generator/$gene1
|   |              @object=/mesh/$gmesh1/$tubes/selectorOnMesh/$gene1
|   |              @idWire=2]
|   `-- $network_on_mesh[@type=networkOnMesh]
|       `-- $net1[@subject=/network/$net1
|           |     @object=/mesh/$gmesh1/$tubes]
|           `-- data
`-- network/
    `-- $net1[@type=simple]
        |-- tubes
        |-- junctions
        `-- connections
```

and `data.h5:/mesh/$gmesh1/$tubes/selectorOnMesh/$gene1` is

index	v1	v2	v3
1	0.5	-1	-1

CHAPTER

THIRTEEN

LOCALIZATION SYSTEM

The localization system category contains only `localizationSystem` elements. A localization system is an HDF5 named group child of `/localizationSystem` with two attributes :

- `reference` is an HDF5 string attribute and is optional. It represents the reference localization system (an absolute localization system has no reference).
- `dimension` is an HDF5 integer attribute and is mandatory. It represents the dimension of the system.

```
data.h5
`-- localizationSystem/
    |-- $locsys[@dimension=3]
    `-- $locsys1[@dimension=3
                 @reference=/localizationSystem/$locsys]
```

A localization system comprises elementary geometric transformations, a transformation can be :

- Scale transformation
- Rotation
- Translation

These elementary transformations are the children of a localization system and have a string attribute `type` equals `scale`, `rotation`, `translation` or `zxzEulerRotation`.

Example :

```
data.h5
`-- localizationSystem/
    |-- $locsys[@dimension=3]/
    `-- $locsys1[@dimension=3
                 @reference=/localizationSystem/$locsys1]/
```

Note: If a localization system has no transformations child, it is equivalent to the reference localization system.

Each transformation moves, rotates or scales the localization relative to the reference localization system in sequence.

13.1 Transformation order

Transformations have to be ordered to give the correct result. The order is stored in a mandatory attribute `rank` owned by all transformations.

```
data.h5
`-- localizationSystem/
    `-- $locsys[@dimension=3]/
        |-- $scale1[@type=scale
        |           @rank=2]
        |-- $rotation1[@type=rotation
        |              @rank=1]
        `-- $translation1[@type=translation
                          @$rank=3]
```

Transformations will then be applied in the following order :

1	$rotation1
2	$scale1
3	$translation1

13.2 Scale transformation

A scale transformation is simply a scale factor, it is a HDF5 named `floatingType` equals `dataSet` of one float element with `type` equals `scale`.

Example :

```
data.h5
`-- localizationSystem/
    `-- $locsys[@dimension=3]/
        `-- $scale1[@type=scale]
```

where `data.h5:/localizationSystem/$locsys/$scale1` :

$$\begin{pmatrix} 2.0 \end{pmatrix}$$

13.3 Rotation

A rotation is defined by a 3x3 matrix :

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

The point coodinates are calculated with :

$$\begin{pmatrix} x'_1 \\ y'_1 \\ z'_1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}$$

Example of a rotation of α around the x axis :

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{pmatrix}$$

In **Amelet HDF**, a rotation is a HDF5 named `floatingType` equals `dataSet` of 3x3 floats elements with `type` equals `rotation`.

Example :

```
data.h5
`-- localizationSystem/
    `-- $locsys[@dimension=3]/
        `-- $rotation1[@type=rotation]
```

where `data.h5:/localizationSystem/$locsys/$rotation1`:

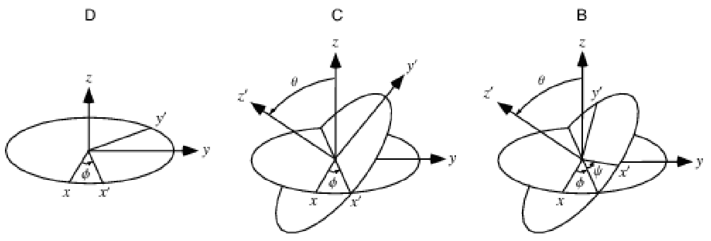
$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{pmatrix}$$

13.4 Euler rotation

Rotation can be expressed with the zxz Euler convention.

Three consecutive rotations D, C, B are applied :

- A rotation around the Z axis, angle ϕ
- A rotation around the X axis , angle $\theta \in [0, \pi]$
- A rotation around the z axis, angle ψ



$$D = \begin{pmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{pmatrix}$$

$$B = \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

And finaly the resultant matrix $A = BCD$

In **Amelet HDF**, an Euler rotation is a HDF5 named `floatingType` equals `vector` of 3 floats elements (ϕ, θ, ψ) with `type` equals `zxzEulerRotation`.

Example :

```
data.h5
`-- localizationSystem/
    `-- $locsys[@dimension=3]/
        `-- $euler1[@type=zxzEulerRotation]
```

where `data.h5:/localizationSystem/$locsys/$euler1` :

$$\begin{pmatrix}\pi & \pi/2 & \pi\end{pmatrix}$$

13.5 Translation

A translation is defined by a vector $T(T_x, T_y, T_z)$.

In **Amelet HDF**, a translation is a HDF5 named `floatingType` equals `vector` of 3 floats elements with `type` equals `translation`.

Example :

```
data.h5
`-- localizationSystem/
    `-- $locsys[@dimension=3]/
        `-- $translation1[@type=translation]
```

where `data.h5:/localizationSystem/$locsys/$translation1` :

$$\begin{pmatrix}1 & 0 & 0\end{pmatrix}$$

CHAPTER

FOURTEEN

PREDEFINED LABELS AND LINKS

Links and labels are general tools to describe an EM simulation. Besides, some concepts are often used accross all numerical methods like the wire radius concept, that's why **Amelet HDF** predefines some labels and some links.

14.1 Predefined labels

Amelet HDF predefined some labels, they are contained in `/label/predefinedLabels` **dataset**.

Example :

```
data.h5
`-- label/
    `-- predefinedLabels
```

where `data.h5:/label/predefinedLabels` is :

$firstLabel
$secondLabel

The predefined labels are described in the next sections.

14.1.1 Wire radius

`wireRadius` label is used to set the radius of a wire. See *Wire radius Link* for its usage.

14.2 Predefined Links

Labels can be involed in links creation, so this fact implies the definition of predefined links.

In addition, other links are predefined that does'nt make participate predefined labels.

14.2.1 Predefined `type` attribute

Link groups can have a `type` attribute. This attribute is an HDF5 string attribute. This attribute allows to give common characteristics to link instances children of groups.

For example, a link group type attribute can be `dataOnMesh`, that is to say all children links associate a `/physicalModel` or an `electromagneticSource` to a `mesh` entity.

Amelet HDF defined several link group `type`.

The `type` attribute is owned by the link group and not by the link. The function of the `type` is to facilitate the research and the storage of simular nature links. In fact, the `type` attribute gives the same information than the reading of the `subject` and the `object` of a link.

14.2.2 Wire radius Link

`wireRadius` is predefied label used to set the radius of a wire. A link built from the `wireRadius` has a mandatory attribute :

- `radius` : `radius` is a real HDF5 attribute, it is a length in meter. `radius` gives the radius of the wire in meter.

Example :

```
data.h5
|-- label/
|   `-- predefinedLabels
|-- mesh/
|   `-- $gmesh1
|       `-- $antenna1
|           `-- group
|               `-- $wire1
`-- link
    `-- $link_group
        `-- $radiusOfWire1[@subject=/label/predefinedLabels
                           @subject_id=0
                           @object=/mesh/$gmesh1/$antenna1/group/$wire1
                           @radius=1e-3]
```

14.2.3 Data on mesh link

If `type` equals `dataOnMesh`, all link instances associate a data to a mesh entity. It is a specialized form of links.

Data can be :

- A `/predefinedLabel` label
- A `/physicalModel` instance
- An `/electromagneticSource` instance

```
data.h5
|-- mesh/
|   `-- $gmesh1
|       `-- $plane
|           |-- nodes
|           |-- elementTypes
|           |-- elementNodes
|           |-- group
|           |   `-- $wing
|           `-- selectorOnMesh
|               |-- elements
|               `-- nodes
|-- electromagneticSource/
|   `-- generator/
|       |-- $v1
|       `-- $i1
`-- link/
```

```
    `-- $link_group[@type=dataOnMesh
        |           @rootObject=/mesh/$gmesh1/$plane/selectorOnMesh]/
        |-- $link_instance1[@subject=/electromagneticSource/generator/$v1
        |                   @object=$generator_v1]
        `-- $link_instance2[@subject=/electromagneticSource/generator/$i1
                            @object=$generator_i1]
```

Where `data.h5:/mesh/$gmesh1/$plane/selectorOnMesh/$generator_v1` is

index
23

Where `data.h5:/mesh/$gmesh1/$plane/selectorOnMesh/generator_i1` is

index
26

In this example, the link group `data.h5:/link/$link_group` is a specialized link group for `dataOnMesh` links.

14.2.4 Data in localization system link

`dataInLocalizationSystem` `type` allows to gather links associating data to a localization system in order to locate and to orient models in the space (an antenna for instance).

```
data.h5
|-- electromagneticSource/
|   `-- antenna/
|       `-- $antenna1
|-- localizationSystem
|   `++ $locasys[@dimension=3]
`-- link
    `-- $link_group[@type=dataInLocalizationSystem]
        `-- $v1-location[@subject=/electromagneticSource/antenna/$antenna1]
                         @object=/localizationSystem/$locasys]
```

In this example, `data.h5:/electromagneticSource/antenna/$antenna1` is located and oriented thanks to `data.h5:/localizationSystem/$locasys`.

Without the `type` attribute, `subject` and `object` are enough to understand the link's nature.

14.2.5 Specific role link

If `type` equals `specificRole`, all link instances associate a label to an object. It is a specialized form of links.

Example :

```
data.h5
|-- label/
|   `-- predefinedLabels
|-- mesh/
|   `-- $gmesh1
|       `-- $antenna1
|           `-- group
|               `-- $wire1
`-- link
    `-- $link_group[@type=specificRole]
        `-- $transmitter[@subject=/label
```

```
                                   @subject_label=transmitter
                                   @object=/mesh/$gmesh1/$antenna1/group/$wire1]
```

14.2.6 TransmissionLineOnMesh Link

A "`transmissionLineOnMesh`" link associates a transmissionLine to a mesh.

It has one optional attribute :

- `modelsOnSectionLink` is an HDF5 string attribute and contains the name of the link group defining the section of the transmission line

and an optional child dataset named `elementsOnSection` :

- **`elementsOnSection` is an HDF5 (n x 2) string dataset which** contains a list of (transmission line element, named mesh entity) string couple defining the association between transmission line elements and mesh entities.

```
data.h5
|-- mesh/
|   `-- $tl11
|-- transmissionLine/
|   `-- $tl1
`-- link/
    `-- $transmissionline_on_mesh[@type=transmissionLineOnMesh]/
        `-- $link1[@subject=/transmissionLine/$tl1
            |      @object=/mesh/$gmesh1/$tl1
            |      @modelsOnSectionLink=/link/$models_on_section]
            `-- elementsOnSection
```

with `elementsOnSection` :

$tl_elem1	$mesh_elem1
$tl_elem2	$mesh_elem2
$tl_elem3	$mesh_elem3

`$tl_*` and `$mesh_*` are named element of `/transmissionLine/$tl1` and `/mesh/$gmesh1/$tl1`.

14.2.7 NetworkOnMesh Link

In this example, a new link `/link/$network_on_mesh/$tub1` is defined, it links a network to a mesh, for doing this association the link contains a child : an HDF5 string dataset named `data`.

The `data` dataset has two columns :

- The first column contains tubes' `id` from the network's `tubes` dataset
- The second column contains named elements from the mesh.

The link has an optional attribute name `harness` :

- `harness` is an HDF5 string attribute that can be equal to :
- `real`, in this case the mesh defined in the link is a `real` harness.
- `fictitious``in this case the mesh defined in the link is a ``fictitious` harness. Nodes, named elements are not properly located in space, but tube's length is correct.

```
data.h5
|-- mesh/
|   `-- $gmesh1
|       `-- $net1
|-- network/
|   `-- $net1
`-- link/
    `-- $network_on_mesh[@type=networkOnMesh]
        `-- $net1[@subject=/network/$net1
            |     @object=/mesh/$gmesh1/$net1
            |     @harness=fictitious]
            `-- data
```

$net_tube1	$mesh_tube1
$net_tube2	$mesh_tube2
$net_tube3	$mesh_tube3

`$net_*` and `$mesh_*` are named element of `/network/$net1` and `/mesh/$gmesh1/$net1`.

14.2.8 Model on a network link

The problem is to put a model (a generator for example) on a wire of a network tube.

The way to accomplish this is to add a link between a generator and a 1D mesh entity, a `idWire` option allows to select the wire.

```
data.h5
|-- mesh
|   `-- $gmesh1
|       `-- $net1
|-- electromagneticSource/
|   `-- generator
|       `-- $gene1
|-- transmissionLine/
|   `-- $tl1
|-- link
|   |-- $data_on_mesh[@type=dataOnMesh]
|   |   `-- $gene1[@subject=/electromagneticSource/generator/$gene1
|   |              @object=/mesh/$gmesh1/$net1/selectorOnMesh/$gene1
|   |              @idWire = 2]
|   `-- $network_on_mesh[@type=networkOnMesh]
|       `-- $net1[@subject=/network/$net1
|           |     @object=/mesh/$gmesh1/$tubes]
|           `-- data
`-- network/
    `-- $net1
```

with `data.h5:/mesh/$gmesh1/$net1/selectorOnMesh/$gene1`:

index	v1	v2	v3
1	0.5	-1	-1

CHAPTER

FIFTEEN

OUTPUT REQUESTS

In the context of a simulation, the `outputRequest` category contains all output requests, that is to say all results the user would like to get after the simulation run.

The simulation category follows the same rules as the `link` category. The main schema is :

```
data.h5
`-- outputRequest/
    `-- $outputRequest_group/
        `-- $outputRequest_instance
```

15.1 The output request instance

15.1.1 Subject and object

An output request instance is the association between a label and an element. The output request labels are picked in a label dataset in the `/label` category :

```
data.h5
|-- label/
|   `-- $outputRequest
`-- outputRequest/
    `-- $outputRequest_group/
        `-- $outputRequest_instance
```

where `data.h5:/label/$outputRequest` (the name is not specified by **Amelet HDF**) is :

E field computation
H field computation

Then a link instance is built between a label and a mesh entity (or whatever element in the **Amelet HDF** instance :

```
data.h5
|-- label/
|   `-- $outpuRequest
|-- mesh/
|   `-- $gmesh1
|       `-- $plane
|           `-- group
|               `-- $wing
`-- outputRequest/
    `-- $outputRequest_group/
        `-- $outputRequest_instance[@subject=/label/$outputRequest
```

```
                                    @subject_id=1
                                    @object=/mesh/$gmesh1/$plane/group/$wing]
```

15.1.2 The `output` attribute

Output results (numerical or not) are **Amelet HDF** objects with an absolute name like all objects. The `output` attribute of `data.h5:/outputRequest/$outputRequest` is an HDF5 string attribute which contains the name of the object result. For instance :

```
data.h5
|-- label/
|   `-- $outpuRequest
|-- mesh/
|   `-- $gmesh1
|       `-- $plane
|           `-- group
|               `-- $wing
|-- floatingType/
|   |-- $e_field
|   `-- $h_field
`-- outputRequest/
    `-- $outputRequest_group/
        |-- $outputRequest_instance1[@subject=/label/$outputRequest
        |                            @subject_id=0
        |                            @object=/mesh/$gmesh1/$plane/group/$wing
        |                            @output=/floatingType/$e_field]
        `-- $outputRequest_instance2[@subject=/label/$outputRequest
                                     @subject_id=1
                                     @object=/mesh/$gmesh1/$plane/group/$wing
                                     @output=/floatingType/$h_field]
```

In this example, the computed electric field will be store in `data.h5:/floatingType/$e_field` and the magnetic field in `data.h5:/floatingType/$h_field`

Note: In the simulation input file, `data.h5:/floatingType/$e_field` is empty. In the simulation output file, `data.h5:/floatingType/$e_field` contains the result data.

15.2 The predefined output requests

Amelet-HDF predefines some output resquests, they are detailed in the next sections.

15.2.1 The predefined outputRequest dataset

The predefined output request label list is :

```
data.h5
`-- label/
    `-- predefinedOutputRequests
```

This dataset contains the following labels :

0	electricField
1	magneticField
2	powerDensity
3	planeWaveDecomposition
4	current
5	voltage
6	power
7	sParameter
8	zParameter
9	yParameter
10	theveninVoltageGenerator
11	nortonCurrentGenerator
12	couplingCrossSection
13	radarCrossSection

(The first column of the array is informative)

15.2.2 Electromagnetic field computation

This section contains output request examples concerning electromagnetic field computation.

Electric field computation in a volume

The electric field computation request in a volume can be written as :

```
data.h5
|-- label/
|   `-- predefinedOutputRequests
|-- mesh/
|   `-- $gmesh1
|       `-- $sphere
|           `-- group
|               |-- $inside[@type=volume]
|               `-- $skin[@type=face]
|-- floatingType/
|   `-- $e_field
`-- outputRequest/
    `-- $outputRequest_group/
        `-- $or1[@subject=/label/predefinedOutputRequests
                 @subject_id=0
                 @object=/mesh/$gmesh1/$sphere/group/$inside
                 @output=/floatingType/$e_field]
```

H field computation in a volume

The magnetic field computation request in a volume can be written as :

```
data.h5
|-- label/
|   `-- predefinedOutputRequests
|-- mesh/
|   `-- $gmesh1
|       `-- $sphere
|           `-- group
|               |-- $inside[@type=volume]
|               `-- $skin[@type=face]
|-- floatingType/
|   `-- $h_field
`-- outputRequest/
    `-- $outputRequest_group/
        `-- $or1[@subject=/label/predefinedOutputRequests
                 @subject_id=1
                 @object=/mesh/$gmesh1/$sphere/group/$inside
                 @output=/floatingType/$h_field]
```

E field computation on surface

The electric field computation request on a surface can be written as :

```
data.h5
|-- label/
|   `-- predefinedOutputRequests
|-- mesh/
|   `-- $gmesh1
|       `-- $sphere
|           `-- group
|               |-- $diameter[@type=edge]
|               |-- $inside[@type=volume]
|               `-- $skin[@type=face]
|-- floatingType/
|   `-- $e_field
`-- outputRequest/
    `-- $outputRequest_group/
        `-- $or1[@subject=/label/predefinedOutputRequests
                 @subject_id=0
                 @object=/mesh/$gmesh1/$sphere/group/$skin
                 @output=/floatingType/$e_field]
```

H field computation on surface

The magnetic field computation request on a surface can be written as :

```
data.h5
|-- label/
|   `-- predefinedOutputRequests
|-- mesh/
|   `-- $gmesh1
|       `-- $sphere
|           `-- group
|               |-- $diameter[@type=edge]
|               |-- $inside[@type=volume]
|               `-- $skin[@type=face]
|-- floatingType/
|   `-- $h_field
`-- outputRequest/
    `-- $outputRequest_group/
        `-- $or1[@subject=/label/predefinedOutputRequests
                 @subject_id=1
                 @object=/mesh/$gmesh1/$sphere/group/$skin
                 @output=/floatingType/$h_field]
```

E field computation on line

The electric field computation request on a surface can be written as :

```
data.h5
|-- label/
|   `-- predefinedOutputRequests
|-- mesh/
|   `-- $gmesh1
|       `-- $sphere
|           `-- group
|               |-- $diameter[@type=edge]
|               |-- $inside[@type=volume]
|               `-- $skin[@type=face]
|-- floatingType/
|   `-- $e_field
`-- outputRequest/
    `-- $outputRequest_group/
        `-- $or1[@subject=/label/predefinedOutputRequests
                 @subject_id=0
                 @object=/mesh/$gmesh1/$sphere/group/$diameter
                 @output=/floatingType/$e_field]
```

H field computation on line

The magnetic field computation request on a line can be written as :

```
data.h5
|-- label/
|   `-- predefinedOutputRequests
|-- mesh/
|   `-- $gmesh1
|       `-- $sphere
|           `-- group
|               |-- $diameter[@type=edge]
|               |-- $inside[@type=volume]
|               `-- $skin[@type=face]
|-- floatingType/
|   `-- $h_field
`-- outputRequest/
    `-- $outputRequest_group/
        `-- $or1[@subject=/label/predefinedOutputRequests
                 @subject_id=0
                 @object=/mesh/$gmesh1/$sphere/group/$diameter
                 @output=/floatingType/$h_field]
```

Power density computation

The power computation request can be written as :

```
data.h5
|-- label/
|   `-- predefinedOutputRequests
|-- mesh/
|   `-- $gmesh1
|       `-- $sphere
|           `-- group
|               |-- $diameter[@type=edge]
|               |-- $inside[@type=volume]
|               `-- $skin[@type=face]
|-- floatingType/
|   `-- $power_density
`-- outputRequest/
    `-- $outputRequest_group/
        `-- $or1[@subject=/label/predefinedOutputRequests
                 @subject_id=2
                 @object=/mesh/$gmesh1/$sphere/group/$volume
                 @output=/floatingType/$power_density]
```

15.2.3 Plane wave decomposition

The plane wave decomposition request can be written as :

```
data.h5
|-- label/
|   `-- predefinedOutputRequests
|-- mesh/
|   `-- $gmesh1
|       `-- $sphere
|           `-- group
|               |-- $diameter[@type=edge]
|               |-- $inside[@type=volume]
|               `-- $skin[@type=face]
|-- floatingType/
|   |-- $magnitude
|   |-- $polarization
|   `-- $propagation_vector
|-- group/
|   `-- $wave_decomposition
`-- outputRequest/
    `-- $outputRequest_group/
        `-- $or1[@subject=/label/predefinedOutputRequests
                 @subject_id=3
                 @object=""
                 @output=/group/$wave_decomposition]
```

and after the run `data.h5:/group/$wave_decomposition` will be :

/floatingType/$magnitude
/floatingType/$polarization
/floatingType/$propagation_vector

15.2.4 Cable

This section presents the output request on wire and cable structures.

Wire current

```
data.h5
|-- label/
```

```
|   `-- predefinedOutputRequests
|-- mesh/
|   `-- $gmesh1
|       `-- $sphere
|           |-- nodes
|           |-- elementTypes
|           |-- elementNodes
|           |-- selectorOnMesh
|           |   `-- $current_sensor
|           `-- group
|               |-- $diameter[@type=edge]
|               |-- $inside[@type=volume]
|               `-- $skin[@type=face]
|-- floatingType/
|   `-- $current
`-- outputRequest/
    `-- $outputRequest_group/
        `-- $or1[@subject=/label/predefinedOutputRequests
                 @subject_id=4
                 @object=/mesh/$gmesh1/$sphere/selectorOnMesh/$current_sensor
                 @output=/floatingType/$current]
```

and `data.h5:/mesh/$gmesh1/$sphere/selectorOnMesh/$current_sensor`

index	v1	v2	v3
23	0	0	0

Here is an example with a network structure :

```
data.h5
|-- label/
|   `-- predefinedOutputRequests
|-- mesh/
|   `-- $gmesh1
|       |-- $tl1
|       `-- $tubes
|           |-- nodes
|           |-- elementTypes
|           |-- elementNodes
|           |-- selectorOnMesh
|           |   `-- $current_sensor
|           `-- group
|               `-- $tube1[@type=element
|                          @elementType=edge]
|-- transmissionLine/
|   `-- $tl1
|-- outputRequest/
|   `-- $outputRequest_group/
|       `-- $or1[@subject=/label/predefinedOutputRequests
|                @subject_id=4
|                @object=/mesh/$gmesh1/$sphere/selectorOnMesh/$current_sensor
|                @output=/floatingType/$current
|                @idWire=2]
|-- floatingType/
|    `-- $current
`-- network/
    `-- $net1[@type=simple]
        |-- tubes
        |-- junctions
        `-- connections
```

and `data.h5:/mesh/$gmesh1/$sphere/selectorOnMesh/$current_sensor`

index	v1	v2	v3
23	0	0	0

Wire voltage

```
data.h5
|-- label/
|   `-- predefinedOutputRequests
|-- mesh/
|   `-- $gmesh1
|       `-- $sphere
|           |-- nodes
|           |-- elementTypes
|           |-- elementNodes
|           |-- selectorOnMesh
|           |   `-- $voltage_sensor
|           `-- group
|               |-- $diameter[@type=edge]
|               |-- $inside[@type=volume]
|               `-- $skin[@type=face]
|-- floatingType/
|   `-- $voltage
`-- outputRequest/
    `-- $outputRequest_group/
        `-- $or1[@subject=/label/predefinedOutputRequests
                 @subject_id=5
                 @object=/mesh/$gmesh1/$sphere/selectorOnMesh/$voltage_sensor
                 @output=/floatingType/$voltage]
```

and `data.h5:/mesh/$gmesh1/$sphere/selectorOnMesh/$voltage_sensor`

index	v1	v2	v3
23	0	0	0

Wire Power

```
data.h5
|-- label/
|   `-- predefinedOutputRequests
|-- mesh/
|   `-- $gmesh1
|       `-- $sphere
|           |-- nodes
|           |-- elementTypes
|           |-- elementNodes
|           |-- selectorOnMesh
|           |   `-- $power_sensor
|           `-- group
|               |-- $diameter[@type=edge]
|               |-- $inside[@type=volume]
|               `-- $skin[@type=face]
|-- floatingType/
```

```
|   `-- $power
`-- outputRequest/
    `-- $outputRequest_group/
        `-- $or1[@subject=/label/predefinedOutputRequests
                 @subject_id=6
                 @object=/mesh/$gmesh1/$sphere/selectorOnMesh/$power_sensor
                 @output=/floatingType/$power]
```

and `data.h5:/mesh/$gmesh1/$sphere/selectorOnMesh/$power_sensor`

index	v1	v2	v3
23	0	0	0

Bundle Current

To request the current on a cable bundle :

```
data.h5
|-- label/
|   `-- predefinedOutputRequests
|-- mesh/
|   `-- $gmesh1
|       |-- $tl1
|       `-- $tubes
|           |-- nodes
|           |-- elementTypes
|           |-- elementNodes
|           |-- selectorOnMesh
|           |   `-- $current_sensor
|           `-- group
|               `-- $tube1[@type=element
|                          @elementType=edge]
|-- transmissionLine/
|   `-- $tl1
|-- outputRequest/
|   `-- $outputRequest_group/
|       `-- $or1[@subject=/label/predefinedOutputRequests
|                @subject_id=4
|                @object=/mesh/$gmesh1/$tubes/selectorOnMesh/$current_sensor
|                @output=/floatingType/$current]
|-- floatingType/
|   `-- $current
`-- network/
    `-- $net1[@type=simple]
        |-- tubes
        |-- junctions
        `-- connections
```

and `data.h5:/mesh/$gmesh1/$sphere/selectorOnMesh/$current_sensor`

index	v1	v2	v3
23	0	0	0

No wire is specified.

Bundle Power

To request the power on a cable bundle :

```
data.h5
|-- label/
|   `-- predefinedOutputRequests
|-- mesh/
|   `-- $gmesh1
|       |-- $tl1
|       `-- $tubes
|           |-- nodes
|           |-- elementTypes
|           |-- elementNodes
|           |-- selectorOnMesh
|           |   `-- $power_sensor
|           `-- group
|               `-- $tube1[@type=element
|                          @elementType=edge]
|-- transmissionLine/
|   `-- $tl1
|-- outputRequest/
|   `-- $outputRequest_group/
|       `-- $or1[@subject=/label/predefinedOutputRequests
|                @subject_id=6
|                @object=/mesh/$gmesh1/$tubes/selectorOnMesh/$power_sensor
|                 @output=/floatingType/$power]
|-- floatingType/
|   `-- $power
`-- network/
    `-- $net1[@type=simple]
        |-- tubes
        |-- junctions
        `-- connections
```

and `data.h5:/mesh/$gmesh1/$sphere/selectorOnMesh/$power_sensor`:

index	v1	v2	v3
23	0	0	0

No wire is specified.

15.2.5 Electric port

Port current

To request the current at a given port :

```
data.h5
|-- label/
|   `-- predefinedOutputRequests
|-- physicalModel/
|   `-- multiport
|       |-- $j1
|       `-- $j2
|-- floatingType/
|   `-- $current
|-- outputRequest/
|    `-- $outputRequest_group/
|        `-- $or1[@subject=/label/predefinedOutputRequests
|                 @subject_id=4
|                 @object=/network/$net1/junctions
|                  @object_id=$j1
|                  @output=/floatingType/$current
|                  @idPort=2]
`-- network/
    `-- $net1[@type=simple]
        |-- tubes
        |-- junctions
        `-- connections
```

The multi port is a junction : `/network/$net1/junctions#$j1` and the port number is 2 :

id	nbPort	multiport
$j1	2	/physicalModel/multiport/$j1
$j2	2	/physicalModel/multiport/$j2

Port voltage

To request the voltage at a given port :

```
data.h5
|-- label/
|   `-- predefinedOutputRequests
|-- physicalModel/
|   `-- multiport
|       |-- $j1
|       `-- $j2
|-- floatingType/
|   `-- $voltage
|-- outputRequest/
|    `-- $outputRequest_group/
|        `-- $or1[@subject=/label/predefinedOutputRequests
|                  @subject_id=5
|                  @object=/network/$net1/junctions
|                  @object_id=$j1
|                  @output=/floatingType/$voltage
|                  @idPort=2]
`-- network/
    `-- $net1[@type=simple]
        |-- tubes
        |-- junctions
        `-- connections
```

The multi port is a junction : `/network/$net1/junctions#$j1` and the port number is 2.

id	nbPort	multiport
$j1	2	/physicalModel/multiport/$j1
$j2	2	/physicalModel/multiport/$j2

Port Power

To request the voltage at a given port :

```
data.h5
|-- label/
|   `-- predefinedOutputRequests
|-- physicalModel/
|   `-- multiport
|       |-- $j1
|       `-- $j2
|-- floatingType/
|   `-- $current
|-- outputRequest/
|    `-- $outputRequest_group/
|        `-- $or1[@subject=/label/predefinedOutputRequests
|                 @subject_id=6
|                 @object=/network/$net1/junctions
|                 @object_id=$j1
|                 @output=/floatingType/$current
|                 @idPort=2]
`-- network/
    `-- $net1[@type=simple]
        |-- tubes
        |-- junctions
        `-- connections
```

The multi port is a junction : `/network/$net1/junctions#$j1` and the port number is 2 :

id	nbPort	multiport
$j1	2	/physicalModel/multiport/$j1
$j2	2	/physicalModel/multiport/$j2

Total current

```
data.h5
|-- label/
|   `-- predefinedOutputRequests
|-- physicalModel/
|   `-- multiport
|       |-- $j1
|       `-- $j2
|-- floatingType/
|   `-- $current
|-- outputRequest/
|    `-- $outputRequest_group/
|        `-- $or1[@subject=/label/predefinedOutputRequests
|                 @subject_id=4
|                 @object=/network/$net1/junctions
|                 @object_id=$j1
|                 @output=/floatingType/$current]
`-- network/
    `-- $net1[@type=simple]
        |-- tubes
        |-- junctions
        `-- connections
```

`data.h5:/network/$net1/junctions#$j1` is :

id	nbPort	multiport
$j1	2	/physicalModel/multiport/$j1
$j2	2	/physicalModel/multiport/$j2

No wire is specified.

Total power

```
data.h5
|-- label/
|   `-- predefinedOutputRequests
|-- physicalModel/
|   `-- multiport
|       |-- $j1
|       `-- $j2
|-- floatingType/
|   `-- $power
|-- outputRequest/
|    `-- $outputRequest_group/
|        `-- $or1[@subject=/label/predefinedOutputRequests
|                 @subject_id=6
|                 @object=/network/$net1/junctions
|                 @object_id=$j1
|                 @output=/floatingType/$power]
`-- network/
    `-- $net1[@type=simple]
        |-- tubes
        |-- junctions
        `-- connections
```

`data.h5:/network/$net1/junctions#$j1` is :

id	nbPort	multiport
$j1	2	/physicalModel/multiport/$j1
$j2	2	/physicalModel/multiport/$j2

No wire is specified.

15.2.6 Equivalent circuit

S parameters

Z parameters

Y parameters

Thevenin voltage generator

Norton current generator

Voltage and current generators

15.2.7 3D object power absorption capabability

Coupling cross-section

CHAPTER

SIXTEEN

EXTENSION TYPES

16.1 Introduction

Amelet HDF is closely related the Quercy, it is a software platform aiming at :

- Integrating scientific softwares;
- Providing knowledge management capabilities;
- Managing computation execution;
- Providing pre and post processing tools;
- and lot of other cool stuff

Quercy expresses data in the form of objects called “infotype” (defined by meta-data). Infotypes are classes, informations are infotype instances. Informations are stored in a relational database.

Infotypes instances can be involved in the simulation process, as input and output, so each infotype have to be converted into equivalent concepts into **Amelet HDF**. Therefore, it exists a sort of bijection between infotypes and **Amelet HDF** for predefined concepts.

For unknown infotypes or infotypes added by users, **Amelet HDF** must be sufficiently flexible to express unknown data coming from this new infotypes.

For example, imagine a module taken a car instance made of four wheel instances. The infotypes “car” and “wheel” are user infotypes so there is no equivalent concept in **Amelet HDF**. Each wheel can be composed of a known material and a pressure array (pressure / speed), **Amelet HDF** must expose this structure of a kind in a consistent manner.

The next sections show the definition of Quercy infotypes and the machinery to serialize them into **Amelet HDF** vocabulary.

16.2 Infotype’s definition

An infotype is like a class in object oriented programming, it is defined by a name and some properties. Properties are called meta-data.

16.2.1 The meta-data

Simple types

A meta-data is named attribute with some other characteristics.

- Type : `boolean`, `integer`, `real`, `string`

- Physical nature : for instance `resistance`, `length`
- Unit : `ohm`, `meter`
- Possible values : if the meta-data is a string, the possible values can be a list of words, if the meta-data is an integer, the possible values can be an interval...
- Default value : the value taken by the meta-data at the creation.

Example, below is the definition of a 3d cartesian grid :

- nx is the number of cells along the x axis
- ny is the number of cells along the y axis
- nz is the number of cells along the z axis
- dx is the step cell along the x axis
- dy is the step cell along the y axis
- dz is the step cell along the z axis

Infotype characteristics :

Specification name	User interface name	Aggregate
CartesianGrid	Cartesian grid	False

Meta-data characteristics :

Name	Type	Physical Nature	Unit	Possible Values	Default Value
nx	integer	Null	Null	N	1
ny	integer	Null	Null	N	1
nz	integer	Null	Null	N	1
dx	real	length	meter	R+	1
dy	real	length	meter	R+	1
dz	real	length	meter	R+	1

Note:

- N represents all Integers
- R represents all Reals
- R+ represents all positive Reals

Nested Lists

Sometimes, the meta-data is a named list. A list can be compared to an HDF5 table. A list is defined by columns of the same nature than simple types meta-data.

For instance, the children of a family can be a list of (string, integer) couple for (name, age), each child is defined in a row :

Infotype characteristics :

Specification name	User interface name	Aggregate
Familly	Familly	False

The word "Aggregate" will be explained in the next section.

Meta-data characteristics :

Name	Type	Physical Nature	Unit	Possible Values	Default Value
father	string	Null	Null	S	dad
mother	string	Null	Null	S	mum
children	listOfChildren	Null	Null	N	1

Note: S represents all Strings

listOfChildren characteristics :

Name	Type	Physical Nature	Unit	Possible Values	Default Value
name	string	Null	Null	S	son
age	integer	time	year	N	10

And finally, a family instance "Simpsons" could be defined by

- the meta-data are :

meta-data	value
father	Brian
mother	Julia

- and the children are :

name	age
john	10
charly	13

The "Simpsons" family is composed of Brian, Julia and two children :

- john, 10 years old
- charly, 13 years old

16.2.2 Quantum and aggregate

The simplest form of infotype is the quantum. A quantum is composed of simple meta-data.

However, a meta-data can be an infotype instance. An infotype that contains infotypes is an aggregate.

In Quercy, the structure of an aggregate can be visualized by a tree view, think of a car and its four wheels

```
$my-car[@infotype=car]
|-- $my-wheel1[@infotype=wheel,
|   |          @radius=12]
|   |-- $pressure[@infotype=arraySet]
|   `-- $material[@infotype=classicalMaterial]
|-- $my-wheel2[@infotype=wheel,
|   |          @radius=12]
|   |-- $pressure[@infotype=arraySet]
|   `-- $material[@infotype=classicalMaterial]
|-- $my-wheel3[@infotype=wheel,
|   |          @radius=12]
|   |-- $pressure[@infotype=arraySet]
|   `-- $material[@infotype=classicalMaterial]
`-- $my-wheel4[@infotype=wheel,
    |          @radius=12]
    |-- $pressure[@infotype=arraySet]
    `-- $material[@infotype=classicalMaterial]
```

$my-car is an instance of car and have four aggregated informations instances of wheel. Each wheel have two aggregated informations and one simple real meta-data :

- `pressure`, it is an instance of arraySet
- `material`, it is an instance of classicalMaterial.
- `radius` is a meta-data for the wheel infotype, it is a real and its value is 12.

This hierarchy of object can be rewritten with an XML syntax

```
<car name="$my-car">
    <wheel
        name="$my-wheel1"
        radius="12">
        <arraySet name="pressure"/>
        <classicalMaterial name="material">
    </wheel>
    <wheel
        name="$my-wheel2"
        radius="12">
        <arraySet name="pressure"/>
        <classicalMaterial name="material"
    </wheel>
    <wheel
        name="$my-wheel3"
        radius="12">
        <arraySet name="pressure"/>
        <classicalMaterial name="material">
    </wheel>
    <wheel
        name="$my-wheel4"
        radius="12">
        <arraySet name="pressure"/>
        <classicalMaterial name="material">
    </wheel>
</car>
```

Consequently, a hiereachy data structure that can be mapped in XML can be adapted to the infotype point of view.

16.3 The serialization

Users infotypes are serialized in the **Amelet HDF** `extensionTypes` category.

The hierarchy level of an infotype's definition can be highly deep (an aggregate can contain aggregates that can contain aggregates ...). This hierarchy level can not be reproduced in **Amelet HDF** without danger : absolute name can become very long. In addition, the "category" formalism can not be respected.

The manner users infotypes are serialized is described now. Each Quercy infotype (and not information) (for example car and wheel) are stored in `extensionTypes/car` and `extensionTypes/wheel` categories.

Each infotype instance become an HDF5 group children of their infotype category. For example, all informations "car" are stored in `/extensionTypes/car` and all informations "wheel" are stored in `/extensionTypes/wheel`.

16.3.1 Simple type meta-data

The simple types Quercy meta-data becomes HDF5 attributes :

- Boolean meta-data : the boolean meta-data are converted into HDF5 integer attributes. The name of the attribute is the specification name name of the meta-data (ASCII name).

  - False becomes 0
  - True becomes 1

- Integer meta-data : the integer meta-data are converted into HDF5 integer attributes. The name of the attribute is the specification name of the meta-data (ASCII name).
- Real meta-data : the real meta-data are converted into HDF5 float attributes. The name of the attribute is the specification name of the meta-data (ASCII name).
- String meta-data : the string meta-data are converted into HDF5 string attributes. The name of the attribute is the specification name of the meta-data (ASCII name).

16.3.2 Nested lists

The nested lists definition is very closed to HDF5 tables, the conversion is straightforward.

a nested list becomes an HDF5 table. The name of the table is the specification name of the list (ASCII name). Each list's field become a columns in the table and the type of the column follow the rules of simple type meta-data.

For example, a nested list `children` defined by :

name	age
john	10
charly	13

is converted into a table, child of `/extensionType/family`, named `children` of two columns

```
/extensionType/
`-- family/
    `-- $Simpsons[@father=Brian,
        |         @mother=Julia]/
        `-- children
```

The first column is called `name` and contains strings, the second columns is called `age` and contains integer.

Aggregation

For aggregate instances, a `linksDefinition` **dataset** reproduces the hierarchy schema. The `linksDefinition` dataset is a (n x 2) HDF5 string dataset :

- the first column contains `name` of the element in the **Amelet HDF** instance.
- the second column contains the optional name/role of the element in the parent.

```
data.h5
`-- extensionType/
    |-- car/
    |   `-- $my-car/
    |       `-- linksDefinitions
    `-- wheel/
        |-- $my-wheel1[@radius=12]
        |   `-- linksDefinition
        |-- $my-wheel2[@radius=12]
        |   `-- linksDefinition
        |-- $my-wheel3[@radius=12]
        |   `-- linksDefinition
        `-- $my-wheel4[@radius=12]
            `-- linksDefinition
```

with /extensionTypes/car/$my-car/linksDefinition :

/extensionType/wheel/$my-wheel1	" "
/extensionType/wheel/$my-wheel2	" "
/extensionType/wheel/$my-wheel3	" "
/extensionType/wheel/$my-wheel4	" "

Sometimes, aggregated informations are named. For instance, the car's wheels must be identified separately and can be picked by their name :

- nose_right_wheel
- nose_left_wheel
- rear_right_wheel
- rear_left_wheel

The **Amelet HDF** conversion gives in this case a new linksDefinition /extensionTypes/car/$my-car/linksDefinition:

/extensionType/wheel/$my-wheel1	nose_right_wheel
/extensionType/wheel/$my-wheel2	nose_left_wheel
/extensionType/wheel/$my-wheel3	rear_right_wheel
/extensionType/wheel/$my-wheel4	rear_left_wheel

Finally, /extensionTypes/wheel/my-wheel1/linksDefinition looks like :

/physicalModel/volume/$wood	material
/floatingType/$pressure	pressure

16.4 Predefined extension types

16.4.1 floatingType

Scalar real or complex numbers can be produced by computations. Amelet-HDF proposes the floatingType category to store genuine floating floatingType.

Example :

```
data.h5
`-- extensionType/
    `-- floatingType/
        |-- $time[@floatingType=singleReal
        |         @physicalNature=time
        |         @unit=second
        |         @value=10]
        `-- $probability[@floatingType=singleReal
                         @value=0.05]
```

16.4.2 unstructuredEdge

Introduction

Amelet HDF describe unstructured meshes with the nodes approach. A convention could provide a way to edges of a mesh, but all edges consumer should implements the algorithm. An another means is to store the edges.

Once the `edges` are computed, the element definition by edge is also interesting for some kind of softwares and stored in `elementEdges`.

The referenced mesh is given by the attribute `mesh`.

An edge is defined by two nodes (node 1, node 2), so `edges` is an integer dateset of number_of_edges X 2 elements. The referenced mesh `elementTypes` dataset gives the number of edges for each element. So for an element in `elementTypes`, there are n rows in `elementEdges` for its description by edges.

`elementEdges` is an HDF5 integer dataset of one column, each row is the number of an edge.

```
data.h5
|-- mesh
|   `-- $gmesh1
|       `-- $mesh1
|           |-- nodes
|           |-- elementTypes
|           `-- elementNodes
`-- extensionType/
    `-- unstructuredEdge/
        `-- $edge-mesh1[@mesh=/mesh/$gmesh1/$mesh1]
            |-- edges
            `-- elementEdges
```

with `data.h5:/extensionType/unstructuredEdge/$edge-mesh1/edges` :

implicit index	node1	node2
0	0	1
1	1	2
2	3	4

and with `data.h5:/extensionType/unstructuredEdge/$edge-mesh1/elementEdges` :

implicit index	edge
0	0
1	1
2	2

Headers are implicit in this case.

The `unstructredEdge/$unstructuredEdge/meshLink`

As for the `/mesh`, equalities between couples of entities of different meshes have to be defined, that 's why `/extensionType/unstructured/meshLink`. This sub-category work as `/mesh/$gmesh/meshLink`.

Example of `/extensionType/unstructured/meshLink` :

```
data.h5
|-- mesh
|   `-- $gmesh1
|       |-- $mesh1
|       |   |-- nodes
|       |   |-- elementTypes
|       |   `-- elementNodes
|       |-- $mesh2
|       |   |-- nodes
|       |   |-- elementTypes
|       |   `-- elementNodes
|       `--meshLink
`-- extensionType/
```

```
    `-- unstructuredEdge/
        |-- $edge-mesh1[@mesh=/mesh/$gmesh1/$mesh1]
        |   |-- edges
        |   `-- elementEdges
        |-- $edge-mesh2[@mesh=/mesh/$gmesh1/$mesh2]
        |   |-- edges
        |   `-- elementEdges
        `-- meshLink
            `-- $ml1[@type=edge
                     @mesh1=/extensionType/unstructuredEdge/$edge-mesh1
                     @mesh2=/extensionType/unstructuredEdge/$edge-mesh2]
```

CHAPTER

SEVENTEEN

PARAMETERIZED ATTRIBUTES

17.1 Introduction

To get the point of parameterized attributes begin with an example : the `/electromagneticSource/planeWave` definition.

We have seen that a plane wave is defined by some attributes and a `magnitude` arraySet :

```
data.h5
`-- electromagneticSource
    `-- planeWave
        `-- $a-plane-wave[@xo=0.0
            |             @yo=0.0
            |             @zo=0.0
            |             @theta=0.0
            |             @phi=0.0
            |             @linearPolarization=0.0]
            `-- magnitude[@floatingType=arraySet]
```

All attributes are simple :

- the physical nature is given by **Amelet HDF**
- the unit is fixed
- the value is a scalar

However, it could happen someone would like to define a plane wave with varying parameters :

- `theta` and/or `phi` could take their values in an interval [0, PI] because a module can take into account several directions of propagation in a single simulation
- Make the origin point move during a simulation

For these reasons, **Amelet HDF** offers a flexible way to override attributes definition without blurring the simplicity of the format.

17.2 The _param group

Given a simple attribute of the above example, `@xo` for example, we want it to take many values. The solution is to create a `_param` group in `electromagneticSource/planeWave/a-plane-wave` element and add an `xo` floating type arrayset to `_param`

```
data.h5
`-- electromagneticSource
    `-- planeWave
        `-- a-plane-wave[@yo=0.0
            |            @zo=0.0
            |            @theta=0.0
            |            @phi=0.0
            |            @linearPolarization=0.0]
            |-- magnitude[@floatingType=arraySet]
            `-- _param
                `-- xo[@floatingType=arraySet]
                    |-- data[@physicalNature=length
                    |        @unit=meter]
                    `-- ds
                        `-- dim1[@physicalNature=time
                                 @unit=second]
```

The element `a-plane-wave/_param/xo` simply overrides the definition `a-plane-wave/@xo`.

The general rule is that *a container's attribute can be overridden by adding an element with the same name to the ''_param'' group child of the container.*

CHAPTER

EIGHTEEN

MULTIPLE FILES CAPABILITIES

There are a lot of use cases for which an **Amelet HDF** instance could be split up into several files :

- The mesh are in an another file because it has been generated and stored separately. It is also better for the re-usability of objects.
- A large arraySet has been generated by a simulation.

Amelet HDF offers a flexible means to handle multiple files.

18.1 The externalElement category

The `/externalElement` category is an HDF5 group, its children are HDF5 string (n x 3) datasets :

- The first column contains elements names used in the **Amelet HDF** instance but defined in another (external) **Amelet HDF** instance.
- The second column contains the file's name the element is defined in.
- The third column contains the name of the elements in the external file.

Note: The name of the children of `/externalElement` is not specified by **Amelet HDF**.

Example, consider the following **Amelet HDF** instances data.h5 and mesh.h5 :

```
data.h5/
|-- externalElement/
|   `-- $external_element
|-- simulation/
|-- link/
|   `-- $link_group[@type=dataOnMesh]
|       `-- link_instance[@subject=/physicalModel/volume/$diel1
|                         @object=/mesh/$gmesh1/$plane/group/$right-wing]
`-- physicalModel/
    `-- volume
        `-- $diel1
```

where `data.h5:/externalElement/$external_element` is :

/mesh/$gmesh1/$plane	mesh.h5	/mesh/$gmesh1/$mesh1
/mesh/$gmesh1/$wing	mesh.h5	/mesh/$gmesh1/$mesh2

and :

```
mesh.h5/
`-- mesh
    `-- $gmesh1
        |-- $mesh2
        `-- $mesh1
            |-- elementNodes
            |-- elementTypes
            |-- nodes
            |-- group
            |   |-- $field-location[@type=node]
            |   |-- $right-wing[@type=element]
            |   `-- $left-wing[@type=element]
            |-- groupGroup
            |   `-- $wings
            `-- selectorOnMesh
                |-- nodes
                `-- elements
```

The mesh `data.h5:/mesh/$gmesh1/$mesh1` declared (and used) in the instance data.h5 is really defined in the instance `mesh.h5:/mesh/$gmesh1/$mesh1`.

From this point, elements which appear in `data.h5` with a name beginning with `/mesh/$gmesh1/$plane` must be read from `data.h5:/mesh/$gmesh1/$mesh1`.

Note: With `externalElement`, it is possible to know all elements defined externally in one glance.

18.2 the linkDefinition attribute

A named element can have one optional attribute which drives toward the genuine location of its definition :

This attribute is `linkDefinition` : It is the absolute name of the element definition. It is an HDF5 200 character string attribute.

Example, consider the following **Amelet HDF** instances data.h5 containing two arraySet `data.h5:/floatingType/e_field` and `data.h5:/floatingType/h_field`:

```
data.h5/
`-- floatingType
    |-- e_field
    |   |-- data
    |   `-- ds
    |       |-- dim1
    |       `-- dim2
    `-- h_field
        |-- data
        `-- ds
            |-- dim1[@linkDefinition=/floatingType/e_field/ds/dim1]
            `-- dim2[@linkDefinition=/floatingType/e_field/ds/dim2]
```

In this example, the dimension of `data.h5:/floatingType/h_field` are defined from the dimension of `data.h5:/floatingType/e_field`.

CHAPTER

NINETEEN

LIST OF PHYSICAL QUANTITIES

19.1 The attribute `physicalNature`

In **Amelet HDF** data's measure is given by the attribute `physicalNature`.

Permitted measures are reported in *Summary of allowed measures*.

19.2 SI unit

All quantities are expressed in SI unit to avoid lecture conversions except angles that are expressed in degree.

Warning: Angles are expressed in degrees.

19.3 Dealing with date

Globally, a date is a format to express the time. The Si unit for the time is the second and in this form it often defines a duration from a zero time. For computing dates, the zero time is the POSIX timestamp but a date can be a day far in the past with the following classical definition :

- year, the year can be negative
- month
- day
- hour in [0, 24] format.
- minute
- second
- microsecond

In Amlet-HDF a date is written is a string following this pattern :

```
year/month/day/hour/minute/second/microsecond/dayInTheWeek
```

The day in the week are :

- monday
- tuesday

- wednesday
- thirsday
- friday
- saturday
- sunday

Note: a "-" replaces not used elements.

Warning: A fictitious unit is introduced to write date : `date`

Example :

Sunday 16 November 2003 is written :

```
data.h5
`-- floatingType/
    `-- a_date[@floatingType=singleString
               @physicalNature=time
               @unit=date
               @value=2003/11/16/-/-/-/-/sunday]
```

or Wednesday 25 November 2009, 11:08:40 is written :

```
data.h5
`-- floatingType/
    `-- a_date[@floatingType=singleString
               @physicalNature=time
               @unit=date
               @value=2009/11/25/11/08/40/-/wednesday]
```

In addition, this format can be used in vector or arraySet like this :

```
data.h5
`-- floatingType/
    `-- pressure[@floatingType=arraySEt
        |        @physicalNature=pressure
        |        @unit=pascal
        |        @comment=the pressure]
        |        @unit=date]
        |-- data
        `-- ds
            `-- dim1[@floatingType=vector
                     @physicalNature=time
                     @unit=date
                     @label=day]
```

with `date.h5:/floatingType/pressure/ds/dim1` is a string dataset containing :

2009/11/25/11/08/40/-/wednesday
2009/11/25/12/08/40/-/wednesday
2009/11/25/13/08/40/-/wednesday

19.4 component physicalNature

`component` physical nature is used in the definition of some *ArraySet*'s dimension. In the case where a dimension describes vector components like `Ex, Ey, Ez` *Component parameter* defines components stores in `data`.

The `component` physical nature has no unit.

Example of the components Ex, Ey, Ez of an electric field during the time :

```
data.h5/
`-- floatingType
    `-- dataOne[@floatingType=arraySet
        |       @label=Electric field around a wire]
        |-- data[@label=electric field
        |        @physicalNature=electricField
        |        @unit=voltPerMeter]
        `-- ds
            |-- dim1[@label=component x y z
            |        @physicalNature=component]
            `-- dim2[@label=the time
                     @physicalNature=time
                     @unit=second]
```

with `data.h5:/floatingType/dataOne/ds/dim1` vector :

index	component
0	x
1	y
2	z

19.5 meshEntity physicalNature

`meshEntity` physical nature is used in the definition of *Numerical data on mesh* in the case where an *ArraySet*'s dimension corresponds to a mesh group. Mesh groups contain mesh entities.

The `meshEntity` physical nature has no unit.

Example :

```
data.h5/
|-- mesh/
|   `-- $gmesh1/
|       `-- $mesh1/
|           `-- group
|               `-- $efield_surface
`-- floatingType
    `-- $dataOne[@floatingType=arraySet
        |        @label=Electric field around the wire]
        |-- data[@label=electric field
        |        @physicalNature=electricField
        |        @unit=voltPerMeter]
        `-- ds
            |-- dim1[@label=component x y z
            |        @physicalNature=component]
            `-- dim2[@label=mesh elements
                     @physicalNature=meshEntity]
```

with `/floatingType/$dataOne/ds/dim2` :

/mesh/$gmesh1/mesh1/group/$exchange_surface

19.6 electricPotentialPoint physicalNature

`electricPotentialPoint` is used in the definition of some *ArraySet*'s dimensions where the nature of the dimension corresponds to an electric potential point like :

- Multi-port ports
- Transmission line wires
- Thevenin generator
- Norton generator

The `electricPotentialPoint` physical nature has no unit.

19.7 Summary of allowed measures

Allowed values for the attribute `physicalNature` are reported in the following tabular :

Measure	Unit
admittance	siemens (S) / siemensPerMeter (S/m)
angle	degree (°)
angularVelocity	degreePerSecond (°/s)
capacitance	farad (F) / faradPerMeter (F/m)
component	no unit
conductance	siemens (S) / siemensPerMeter (S/m)
couplingCrossSection	squareMeter (m^2)
electricalConductivity	siemensPerMeter (S/m)
electricCharge	coulomb (C)
electricCurrent	ampere (A)
electricCurrentDensity	amperePerSquareMeter (A/m^2)
electricField	voltPerMeter (V/m)
electricPotentialPoint	no unit
energyFluxDensity	wattPerSquareMeter (W/m^2)
frequency	hertz (Hz)
impedance	ohm (Ω) / ohmPerMeter (Ω/m)
inductance	henry (H) / henryPerMeter (H/m)
length	meter (m)
mass	kilogram (kg)
magneticConductivity	faradPerMeter (F/m)
magneticField	amperePerMeter (A/m)
meshEntity	no unit
permeability	henryPerMeter (H/m)
permittivity	faradPerMeter (F/m)
power	watt (W)
powerDensity	wattPerCubicMeter (W/m^3)
propagationConstant	perMeter (1/m)

Continued on next page

Table 19.1 – continued from previous page

Measure	Unit
pressure	pascal (Pa)
resistance	ohm (Ω) / ohmPerMeter (Ω/m)
surface	squareMeter (m^2)
temperature	kelvin (K)
time	second (s) / date
volume	cubicMeter (m^3)
voltage	volt (V)
volumetricMassDensity	kilogramPerCubicMeter(kg/m^3)

CHAPTER

TWENTY

ELEMENTS SUMMARY

Path	HDF5 element nature	HDF5 Data type
file	File	
file/@entryPoint	Attribute	String
/mesh	Group	
/mesh/$gmesh1/$mesh	Group	
/mesh/$gmesh1/$mesh[@type]	Attribute	String
`structured`		
`unstructured`		
/mesh/$gmesh1/$mesh/nodes	Dataset (n x 3)	Float
/mesh/$gmesh1/$mesh/elementTypes	Dataset (n x 1)	To be defined
/mesh/$gmesh1/$mesh/elementNodes	Dataset (n x 1)	Float
/mesh/$gmesh1/$mesh/group	Group	
/mesh/$gmesh1/$mesh/group/$group	Dataset (n x 1)	Integer
/mesh/$gmesh1/$mesh/group/$group/@type	Attribute	String
`node`		
`element`		
/mesh/$gmesh1/$mesh/groupGroup	Group	
/mesh/$gmesh1/$mesh/groupGroup/$groupGroup	Group	
/mesh/$gmesh1/$mesh/cartesianGrid	Group	
/mesh/$gmesh1/$mesh/cartesianGrid/x	floatingType = vector	Float
/mesh/$gmesh1/$mesh/cartesianGrid/y	floatingType = vector	Float
/mesh/$gmesh1/$mesh/cartesianGrid/z	floatingType = vector	Float
/mesh/$gmesh1/$mesh/normal	Group	
/mesh/$gmesh1/$mesh/normal/$group	Dataset (n x 1)	Integer (16 bit ?)
/mesh/$gmesh1/$mesh/selectorOnMesh	Group	
/mesh/$gmesh1/$mesh/selectorOnMesh/$nodes	Table	
unstructured mesh case		(Integer)
structured mesh case		(Integer x 3)
/mesh/$gmesh1/$mesh/selectorOnMesh/$elements	Table	
unstructured mesh case		(Integer, Float x 3)
structured mesh case		(Integer x 6, Float x 3)
/mesh/$gmesh1/meshLink	Group	
/mesh/$gmesh1/meshLink/$meshLink	Dataset	
/mesh/$gmesh1/meshLink/$meshLink/@type	Attribute	String
`node`		
`edge`		
`face`		
`volume`		

Continued on next page

Table 20.1 – continued from previous page

Path	HDF5 element nature	HDF5 Data type
/mesh/$gmesh1/meshLink/$meshLink/@mesh1	Attribute	String
/mesh/$gmesh1/meshLink/$meshLink/@mesh2	Attribute	String
/globalEnvironment/time	floatingType -> vector	Float
/globalEnvironment/frequency	floatingType -> vector	Float
/electromagneticSource/planeWave	Group	
.		
.		
.		

CHAPTER

TWENTYONE

INDICES AND TABLES

- genindex
- search