HDF5/FINITE ELEMENT DATA

A Standard for Finite Element Data Storage based on HDF5

DRAFT

Version 2006.4 [25th September 2006]

Initiated by Benedikt Oswald. [benedikt.oswald@psi.ch]. Paul Scherrer Institut (PSI). Document history - 2006 sep 25: benedikt oswald, corrections and specification of units group - 2006 aug 25: benedikt oswald, changes and corrections - 2006 aug 21: benedikt oswald, corrections - 2006 aug 17: benedikt oswald, changes of group and data member nomenclature - 2006 aug 10: benedikt oswald, adapted to generalized storage of finite element data - 2006 may 13: benedikt oswald, extensions - 2006 may 01: benedikt oswald, extensions - 2006 mar 22: benedikt oswald, creation.

Abstract

HDF5/FINITE ELEMENT DATA, HDF5/FED in brief, is a data standard for describing the input to and output from computational codes, using the finite element method, based on the hierarchical data format HDF5. It has been motivated by the fact that in most computational electrodynamics applications the same or at least very similar type of input and output data items are required to be read from and written to files. While it it cumbersome to reinvent the wheel, every time a novel electromagnetics code is developed, it would be a definite advantage if there was a flexible file format from which one could expect some fundamental data to be present: e.g. different mesh types, refined versions of a mesh, different boundary meshes, degrees of freedom (DoF) storage for check pointing and restart situations, storage of quantities derived from calculated fields and storage of sampled electromagnetic fields etc. Also, it would prove to be efficient if those items were accessible in a standardized way. From this insight, we have started to define the HDF5/FED data standard which is, on the one hand, flexible and, on the other, sufficient for storing essential data items for, e.g. electrodynamic computations; the main objective being that a code developer can expect a certain minimum standard of HDF5 groups and datasets to be present in a HDF5/FED compliant file and at the same time can access all these data items in a seamless and easy way, by using the HDF5 library routines, both from serial and parallel codes, independent on how the file was written. Therefore, the HDF5/FED data standard has been designed to define HDF5 groups and datasets for storing planar and volume meshes, consisting of triangles, quadrangles, tetrahedra, hexahedra, prisms and pyramids for different mesh refinement levels in a standardized way. It further defines storage of boundary meshes, consisting of triangles and quadrangles on the levels corresponding to the volume mesh levels; at any one specific level as many different boundary meshes can be stored. This is motivated by electromagnetics simulations where there is not only a background boundary that truncates the computational domain but there may also be interior boundaries that separate regions of different physical properties or regions that must be separated by a boundary for modeling reasons, e.g. in the finite-element-boundary-integral (FEBI) approach where fictitious boundaries are needed. Furthermore, the HDF5/FED data standard defines formats for storing quantities derived from the (electromagnetic) fields and also scalar quantities, such as eigenvalues. The HDF5/FED data standard also defines a format for storing degrees of freedom (DoF) associated with any topological entity, such as vertices, edges, triangles, quadrangles, tetrahedral, hexahedra, prisms and pyramids; this feature allows the easy implementation of checkpoint and restart functionality into codes. The HDF5/FED finally defines formats for storing sampled electromagnetic fields for subsequent usage in other analysis or visualization applications. While HDF5/FED has been designed with simplicity and small size in mind, it can be extended to accommodate further constructs, if required by applications. Over all, HDF5/FED is a recipe on which groups and datasets can reasonably expected in a compliant HDF5 file and how groups and datasets containing different electromagnetic data items should be named to be compliant. Finally, an application programming interface is currently developed at PSI using the C++ language. In the future, it is the objective that users use this API when reading and writing HDF5/FED compliant data files.

1 The HDF5/FED standard

1.1 Rationale

Given the flexibility and widespread usage of the HDF5 file format we consider it useful to define a common set of HDF5 groups and datasets for input to and output from, e.g. electromagnetic, codes solving partial differential equations (PDE) with the finite element method; hence the name HDF5/FED. Through this approach it will become easier to implement new electromagnetic solvers because the developer can rely on standardized input and output file formats and there is no need to reinvent the wheel every time a new computational scheme is designed. In this section we discuss features considered worthwhile to be available. A precise list of features present in the current version is given in section 1.2. We foresee the following advantages of this approach:

- no need to write any sort of lexer and parsers programs for reading a specific file format because this is completely handled by the HDF5 library
- apart from the defined groups and datasets the code developer is completely free to organize his own code; he just uses the HD5 library in a seamless way, both in serial and parallel applications; this is because the very same HDF5 data file can be written and read both serial and parallel.
- the format is easily extendable; if more or different data structures are used, more groups and datasets may be added; nevertheless, the code developer can expect that a mesh on a specific level is alway stored the same way and hence can be retrieved the same way.
- HDF5 is an established standard used both in academia and also industry, therefore it is expected to be here to stay.

More to the point, in HDF5 speak, we define groups and datasets that can be expected to be present in input and output files for and from solvers; of course, other groups and and datasets may be defined as desired but, to be compliant with the standard, certain groups and datasets must be present in the HDF5 file. The structure of HDF5 groups and datasets must be flexible enough to accommodate both frequency and time domain electromagnetic approaches and also other ones that may not be classified as either one.

- different types of **planar and volume meshes**, consisting of triangles, quadrangles, tetrahedra, hexahedra, prisms, pyramids etc. and also meshes consisting of mixtures of these element types; different, refined versions of an original mesh might be needed for checkpointing and restart functionality; furthermore, we might wish to store a version of a mesh that has been obtained from an original mesh by partitioning with a load balance tool, typically required in parallel computations;

  nota bene: we want to be able to store type and order of basis function associated with single finite elements and its depending topological entities. We need to be able to store 2-dimensional and also 3-dimensional mesh information.
- different types of **boundary surface meshes**, consisting of triangles and / or quadrangles; even if today typically triangles are most often used, we need to accommodate other types of surface definitions; typical usage of surface meshes are encountered for the application of boundary conditions, such as absorbing boundary conditions (ABC), perfect electric conductors (PEC) and perfect magnetic conductors (PMC); also, surface impedance conditions may be assigned to surface meshes or parts of them; furthermore we need to be able to store different refined versions of a boundary mesh and also boundary meshes that separated internal regions.

nota bene: boundary meshes are only present if we operate in 3-dimensional space; if the problem is formulated in 2-dimensional space the mesh is stored in the first category; a boundary mesh is specified in terms of the coordinates of a 3-dimensional volume mesh.

nota bene: those boundary meshes must also be capable of storing the type of boundary, such as PEC, PML, ABC or surface impedance boundary conditions.

- store lists of **edges** and **faces** associated with meshes; e.g. a tetrahedron's definition not only encompasses a list of its defining vertices but also indices into a list of its associated edges and faces, a relationship which is relevant if we need to store DoF associated with these topological entities.
- store **upward adjacencies**, e.g. to which elements does a face (triangle, quadrangle) belong? So we would store the adjacent elements in the respective face list; furthermore, to which faces and elements does an edge belong to? to which edges, faces and elements does a vertex belong to?
- **material parameters**, e.g. non-dispersive dielectric and magnetic materials; furthermore dispersive dielectrics and magnetic materials must be specifiable; either through the specification of dielectric model parameters, such as Debye, Lorentz and Drude, or by listing the discrete values of dielectric and magnetic properties as a function of frequency or energy, typically used in optical applications.
- **degrees of freedom** (DoF) associated with any topological entity of the mesh; in the case of a time domain simulation we probably want to store DoF sets per time step for example in order to implement checkpoint and restart capabilities
- quantities derived from the field solution, such as **eigenvalues, resonance frequencies, quality factors, radar cross sections** etc.
- **text material** containing for example comments to the analyzed problem.
- specific **simulation parameters**, e.g. time domain signals, frequencies

1.2 Features available in **HDF5/FED** 2006.3

The following features are defined in version 2006.4 of the standard.

- definition of units for length, mass and time
- storage of coordinates, meshes and boundary meshes on different refinement levels
- storage of material parameters
- storage of sampled electric and magnetic fields

1.3 Notation

The notation used for the definition of the HDF5/FED standard is as follows: HDF5 groups are written in **bold sans serif** and datasets contained therein are tabbed to the right and written in standard sans serif typeface. We understand the notation of<L>, <N> and <K> to denote integer numbers, appended to the ASCII string of the respective group's or dataset's name.

1.4 Nomenclature

The nouns nodes, vertices and points are used synonymously to denote locations in 1, 2- or 3-dimensional space. The noun face is used both for triangles and quadrangles. If there is a difference in usage or implementation, the respective terms are used. The noun element is used to denote both planar elements, such as triangles or quadrangles, but also all types of volume elements, such as tetrahedra, hexahedra, prisms and pyramids.

When we speak of the file we mean a file compliant to the **HDF5_FINITE_ELEMENT_DATA** data standard.

1.5 Futures

For this data standard definition to be useful an application programming interface (API) is required. The definition of the API is currently under work and being implemented, respectively. It is the idea to integrate the API definition into this document as soon as its is sufficiently stable.

1.6 Acknowledgements

Benedikt Oswald would like to thank the people whose critical review and suggestions have been essential to make this data standard useful; in particular these are Patrick Leidenberger (IUP, PSI), Andreas Adelmann (PSI) and Peter Bastian (IWR, Heidelberg University).

2 Definition

2.1 Definition of group hierarchy

First, we define the hierarchy of groups which serve as the backbone of HDF5/FED. There is a top level group within which all other groups are embedded.

- Group **HDF5_FINITE_ELEMENT_DATA** within which all following groups are embedded; this is a bracketing group:

  Group **UNITS** : contains an array of strings that specify the units of the data stored in this HDF5/FED file.

  Group **COORD** : contains all coordinates, either in 1D, 2D or 3D; the coordinates are not discriminated against the mesh level on which they are used; as this may vary widely in the applications it is left to the application itself.

  Group **VOLUME_MESH** : contains all volume mesh related data items; in case of 2 dimensions, the volume mesh consists of triangles or quadrangles;

  Group **BOUNDARY_MESH** : contains all boundary mesh related data items; in case of 2 dimensions, the boundary mesh consists of edges;

  Group **MATERIAL** : contains all material definitions

  Group **DOF** : contains degrees of freedom (DoF) associated with topological entities

  Group **FIELD** : contains sampled electric and magnetic fields

2.2 Definition of units in HDF5/FED

- Group **UNITS**

  – Dataset UNITS : is an array of strings; the array contains the string denoting the length units; the second array element contains the unit denoting mass and the third array element contains a string denoting time.

    accepted strings for the length unit are: ATTOMETER, ANGSTROM, NANOMETER, MICROMETER, MILLIMETER, CENTIMETER, DECIMETER, METER, DECAMETER, HECTOMETER, KILOMETER.

    accepted strings for the mass unit are: NANOGRAM, MICROGRAM, MILLIGRAM, GRAM, KILOGRAM, TON.

    accepted strings for the time unit are: ATTOSECOND, FEMTOSECOND, PICOSECOND, NANOSECOND, MICROSECOND, MILLISECOND, SECOND, MINUTE, HOUR, DAY.

2.3 Definition of volume and boundary meshes in HDF5/FED

The mesh group in HDF5/FED is organized as follows. The different element types, such as tetrahedra, hexahedra, prisms and pyramids are based on the numbering scheme of reference elements in the DUNE, cf. `http://hal.iwr.uni-heidelberg.de/dune/contact.html`
Nota bene: the API must provide functions for loading the volume mesh, e.g. tetrahedra, hexahedra, prisms and pyramids, and components derived from it, e.g edges and faces, either in unpartitoned or partitioned format. This is particularly important for parallel calculations. An open source partitioning library, e.g. PARMETIS, should directly be integrated into the data format HDF5/FED.

- Group **COORD**

  - Dataset COORD1D : if the mesh is a 1-dimensional mesh, i.e. a single line and where COORDS1D is a 1-dimensional double type array with as many rows as there are vertices in the most refined version of the mesh; the maximum number of vertices used in the mesh can be derived from the number of rows in the respective two-dimensional arrays ;
  - Dataset COORD2D : if the mesh is a purely 2-dimensional mesh and where COORDS2D is a 2-dimensional double type array of 2 columns and as many rows as there are vertices in the most refined version of the mesh; the maximum number of vertices used in the mesh is derived from the number of rows in the respective two-dimensional arrays;
  - Dataset COORD3D : if the mesh is a 3-dimensional mesh and where COORDS3D is a 2-dimensional double type array of 3 columns and as many rows as there are vertices in the most refined version of the mesh; the maximum number of vertices used in the mesh can be derived from the number of rows in the respective two-dimensional arrays;

- Group **VOLUME_MESH**

  - Dataset MESHDIM : spatial dimension of the main mesh where MESHDIM is a scalar data item, i.e. one single integer number with value either 1, 2 or 3.
  - Dataset NELEM_<L> : number of element types present on level <L> of the mesh where NELEM_<L> is a 2-dimensional integer array with 4 columns, corresponding to the different types of allowed elements, namely tetrahedra, hexahedra, prisms and pyramids, in this order, and as many rows as there are mesh levels.

    in the case of a 2-dimensional mesh, the array still has 4 columns, where the first column stores the number of triangles, the second stores the number of quadrangles and the third and fourth column are no used.

    in the case of a 1-dimensional mesh, the array still has 4 columns, where the first column stores the number of line elements and the remaining three columns are not used.

    nota bene: the mesh level index L varies from 0 to $\#(\text{rows of NELEM}) - 1$.
  - Dataset TETMESH_L<L> : defines a tetrahedral mesh of refinement level L with a two-dimensional integer array of 5 columns and as many rows as there are tetrahedra present in this mesh; the first four columns denote the indices into the respective tetrahedron's corner coordinate array i.e. COORDS3D and the last, fifth column is an index into the list of material parameter associated with the specific tetrahedron.
  - Dataset HEXMESH_L<L> : defines a hexahedral mesh of refinement level L with a two-dimensional integer array of 9 columns and as many rows as there are hexahedra present in this mesh; the first eight columns denote the indices into the respective hexahedron's corner coordinate array i.e. COORDS3D and the last column is an index into the list of material parameter associated with the specific hexahedron.
  - Dataset PRISMATICMESH_L<L> : defines a prismatic mesh, with a triangular base and top, of refinement level L with a two-dimensional integer array of 7 columns and as many rows as there are prisms present in this mesh; the first six columns denote the indices into the respective prism's corner coordinate array i.e. COORDS3D and the last column is an index into the list of material parameter associated with the specific prism.
  - Dataset PYRAMIDMESH_L<L> : defines a prismatic mesh, with a triangular base and top, of refinement level L with a two-dimensional integer array of 6 columns and as many rows as there are pyramids present in this mesh; the first five columns denote the indices into the respective pyramid's corner coordinate array i.e. COORDS3D and the last column is an index into the list of material parameter associated with the specific pyramid.

- **nota bene**: all types of 3- and 2-dimensional meshes can be present simultaneously; if a mesh consists of different type of elements, tetrahedra, hexahedra, prisms, pyramids, then it is the users responsibility to look for the different types of meshes on the same level <N> and to read them into the code; it is the mesh generator's responsibility to make sure that the mesh is consistent if there is more than one type of element present in the mesh.
- Dataset TRIANGLEMESH_L<L> : defines a triangular mesh in 2-dimensional space of refinement level L with a two-dimensional integer array of 4 columns and as many rows as there are triangles present in this mesh; the first three columns denote the indices into the respective triangle's corner coordinate array i.e. COORDS2D and the last column is an index into the list of material parameter associated with the specific triangle.
- Dataset QUADRANGLEMESH_L<L> : defines a quadrangular mesh in 2-dimensional space of refinement level L with a two-dimensional integer array of 5 columns and as many rows as there are quadrangles present in this mesh; the first four columns denote the indices into the respective quadrangle's corner coordinate array i.e. COORDS2D and the last column is an index into the list of material parameter associated with the specific quadrangle.
- **nota bene**: all types of 2-dimensional meshes can be present simultaneously; if a mesh consists of different type of elements, e.g. triangles and quadrangles, then it is the user's responsibility to look for the different types of meshes on the same level <N> and to read them into the code; it is the mesh generator's responsibility to make sure that the mesh is consistent if there is more than one type of element present in the mesh.

• Group **BOUNDARY_MESH**

- Dataset NBOUNDARY_MESH_<L> : number of boundary meshes present on level <L> where NBOUNDARY_MESH_<L> is a 2-dimensional integer array with two columns, corresponding to the number of boundaries built from triangles (first column) and quadrangles (second column) and as many rows as there are mesh levels.
- Dataset BOUNDARY_TRIANGLE_L<L>_K<K> : defines the K-th triangular boundary on refinement level L where BOUNDARY_TRIANGLE_L<L>_K<K> is a two-dimensional integer array of 4 columns and as many rows as there are triangles present in this mesh; the K index starts at 0, according to the C convention; the first three columns denote the indices into the respective triangle's corner coordinate array i.e. COORDS3D and the last column is an index into the list of boundary condition parameters associated with the specific triangle.

  Example: BOUNDARY_TRIANGLE_L3_K2 is the 3rd boundary made from triangles on level 4.
- Dataset BOUNDARY_QUADRANGLE_<L>_<K> : defines the K-th triangular boundary on refinement level L where BOUNDARY_QUADRANGLE_L<L>_K<K> a two-dimensional integer array of 5 columns and as many rows as there are triangles present in this mesh; the K index starts at 0, according to the C convention; the first four columns denote the indices into the respective triangle's corner coordinate array i.e. COORDS3D and the last column is an index into the list of boundary condition parameters associated with the specific triangle.

  Example: BOUNDARY_QUADRANGLE_L5_K1 is the 2nd boundary made from triangles on level 6.
- Dataset BOUNDARY_EDGE_L<L>_K<K> : defines the K-th boundary, made from edges, on refinement level L where BOUNDARY_EDGE_L<L>_K<K> is a two-dimensional

integer array of 3 columns and as many rows as there are edges present in this mesh; the K index starts at 0, according to the C convention; the first three columns denote the indices into the respective edge's corner coordinate array i.e. COORDS2D and the last column is an index into the list of boundary condition parameters associated with the specific triangle.

Example: BOUNDARY_EDGE_L3_K2 is the 3rd boundary made from edges on level 3.

2.4 Definition of material parameters in **HDF5/FED**

Within the group **MATERIAL** there may be different subgroups, corresponding to the different physical problems under study. At present we have defined the group **ELECTROMAGNETIC**, which defines electromagnetic materials parameters, such as physical models for dielectric permittivity, magnetic permeability or electromagnetic material models at discrete frequencies, wavelengths or energy levels.

- **MATERIAL**
  - Group **ELECTROMAGNETIC**: contains two subgroups, **DISCRETE** and **PHYSICAL**, which store the discrete, sampled materials parameters and parameters that fix physical model based material properties.

    every electromagnetic material definition has an associated id which is of type unsigned integer and consecutively numbered from 0 to the number of material properties defined;
  - Dataset MATERIAL_ID<L>_<K> is a two dimensional array of unsigned integers. There are as many rows as the total number of defined materials, including discrete and physical ones. The first row contains the id of the material definition and the second one the type, either discrete or physical.

    Group **DISCRETE**: the are datasets for storing the discrete frequency, wavelength or energy values at which there are complex valued electromagnetic material parameters, for storing complex dielectric permittivity, complex permeability and complex conductivity. Nota bene: either the material parameter are specified to be depending on frequency, wavelength or energy, exclusively. There must be no mixture of these independent parameters in the same file.
    It is the job of the API to provide information on the type and number of material parameters existing in the file.

      Dataset FREQUENCY : a one-dimensional array of double numbers containing the frequencies for which there are electromagnetic material parameters;

      Dataset WAVELENGTH : a one-dimensional array of double numbers containing the wavelengths for which there are electromagnetic material parameters;

      Dataset ENERGY : a one-dimensional array of double numbers containing the energies for which there are electromagnetic material parameters;

      Dataset PERMITTIVITY : a one-dimensional array of complex numbers with as many elements as there are frequencies for which there are material parameters;

      Dataset PERMEABILITY : a one-dimensional array of complex numbers with as many elements as there are frequencies for which there are material parameters;

      Dataset CONDUCTIVITY : a one-dimensional array of complex numbers with as many elements as there are frequencies for which there are material parameters;

    Group **PHYSICAL**: within this group there are groups describing several physically relevant materials models for dielectric and magnetic media. It is the job of the API to provide information on the type and number of material parameters existing in the file.

      Group **DEBYE** : contains datasets for storing Debye material parameters. The order of the Debye model can be derived from the number of weights n, which must be the same for the ϵ_{static}, ϵ_{∞} and f_{relax} datasets and arrays, respectively; i.e. there must be exactly n weights, n ϵ_{static} and n ϵ_{∞} values. The number of Debye components stored in the file must be made availably by the API.

Dataset WEIGHT : a one-dimensional array of double numbers containing the frequencies for which there are electromagnetic material parameters

Dataset EPSILON_STATIC : a one-dimensional array of double numbers containing the frequencies for which there are electromagnetic material parameters

Dataset EPSILON_INFINITY : a one-dimensional array of double numbers containing the frequencies for which there are electromagnetic material parameters

Dataset RELAXATION_FREQUENCY

Group **LORENTZ** : contains datasets for storing Lorentz material parameters

Group **DRUDE** : contains datasets for storing Drude material parameters

2.5 Definition of degrees of freedom (DoF) in HDF5/FED

The notation is the same as for the definition of planar, volume and boundary meshes, cf. section 2.3.

- Group **DOF** : not yet defined

2.6 Definition of sampled fields in **HDF5/FED**

The notation is the same as for the definition of planar, volume and boundary meshes, cf. section 2.3.

- Group **SFIELD**

  Dataset SLOC_L<L>_K<K>_T<T> : the K-th location on level L at which a field was sampled.

  Dataset REAL_FIELD_L<L>_K<K>_T<T> : the K-th field, sampled on level L, at index T where T may indicate time or any other ordering scheme using real numbers.

  Dataset COMPLEX_FIELD_L<L>_K<K>_T<T> : the K-th field, sampled on level L, at index T where T may indicate time or any other ordering scheme using complex numbers.

  the SLOC_L<L>_L<K>_T<T> is a two-dimensional array of doubles with as many rows as there are sampling locations; the number of columns depends on the dimensionality of the mesh; the REAL_FIELD_L<L>_L<K>_T<T> is a two-dimensional array of double numbers with as many rows as there are samples; the number of columns depends on the dimensionality of the mesh;

  the COMPLEX_FIELD_L<L>_K<K>_T<T> is a two-dimensional array of complex numbers with as many rows as there are samples; the number of columns depends on the dimensionality of the mesh;

  There can be, simultaneously, real and complex sampled fields in the file;

  It is the responsibility of the application to deliver as many sampling locations as there are fields sampled at these locations; additionally, the application must make sure that the sampling locations and the sampled fields correspond to each other, on the level of the row index into the arrays storing loctiond and fields

  The API must provide the number of different sampling location and sampled fields and also the number of sampling locations and field on a specfic level for a specific K value.