1 H5MD format specification version (work version)

1.1 Objective

H5MD stands for "HDF5 for molecular data". H5MD is a specification to store molecular simulation data and is based on the HDF5 file format (The HDF Group, n.d.). The primary goal is to facilitate the portability of said data amongst scientific simulation and analysis programs.

1.2 File format

H5MD structures are stored in the HDF5 file format version 0 or later. It is recommended to use the HDF5 file format version 2, which includes the implicit tracking of the creation and modification times of the file and of each of its objects.

1.3 Notation and naming

HDF5 files are organized into groups and datasets, summarized as *objects*, which form a tree structure with the datasets as leaves. Attributes can be attached to each object. The H5MD specification adopts this naming and uses the following notation to depict the tree or its subtrees:

`\-- item` An object within a group, that is either a dataset or a group. If it is a group itself, the objects within the group are indented by five spaces with respect to the group name.

`+-- attribute` An attribute, that relates either to a group or a dataset.

`\-- data: <type>[dim1][dim2]` A dataset with array dimensions `dim1` by `dim2` and of type `<type>`. The type is taken from `Enumeration`, `Integer`, `Float` or `String` and follows the HDF5 Datatype classes. If the type is not mandated by H5MD, `<type>` is indicated. A scalar dataspace is indicated by `[]`.

`(identifier)` An optional item.

`<identifier>` An optional item with unspecified name.

H5MD defines a structure called *H5MD element* (or *element* whenever there is no confusion). An element is either a time-dependent group or a single dataset (see time-dependent data below), depending on the situation.

1.4 General organization

H5MD defines an organization of the HDF5 file or a part thereof into groups, datasets, and attributes. The root level of the H5MD structure may coincide with the root of the HDF5 file or be an arbitrary group inside the HDF5 tree. A number of groups are defined at the H5MD root level. Several levels of subgroups may exist inside the H5MD structure, allowing the storage and description of subsystems.

The H5MD structure is allowed to possess non-specified groups, datasets, or attributes that contain additional information such as application-specific parameters or data structures, leaving scope for future extensions. Only the `h5md` group is mandatory at the H5MD root level. All other root groups are optional, allowing the user to store only relevant data. Inside each group, every group or dataset is again optional, unless specified differently.

H5MD supports equally the storage of time-dependent and time-independent data, i.e., data that change in the course of the simulation or that do not. The choice between those storage types is not made explicit for the elements in the specification, it has to be made according to the situation. For instance, the species and mass of the particles are often fixed in time, but in chemically reactive systems this might not be appropriate.

1.4.1 Time-dependent data

Time-dependent data consist of a series of samples (or frames) referring to multiple time steps. Such data are found inside a single dataset and are accessed via dataset slicing. In order to link the samples to the time axis of the simulation, H5MD defines a *time-dependent H5MD element* as a group that contains, in addition to the actual data, information on the corresponding integer time step and on the physical time. The structure of such a group is:

```
<element>
 \-- step
 \-- (time)
 \-- value: <type>[variable][...]
```

value A dataset that holds the data of the time series. It uses a simple dataspace whose rank is given by 1 plus the tensor rank of the data stored. Its shape is the shape of a single data item prepended by a `[variable]` dimension that allows the accumulation of samples during the course of time. For instance, the data shape of scalars has the form `[variable]`, `D`-dimensional vectors use `[variable][D]`, etc. The first dimension of `value` must match the unique dimension of `step` and `time`.

If several H5MD elements are sampled at equal times, `step` and `time` of one element may be hard links to the `step` and `time` datasets of a different element. If two elements are sampled at different times (for instance, if one needs the positions more frequently than the velocities), `step` and `time` are unique to each of them.

The storage of step and time information follows one of the two modes below, depending on the dataset layout of `step`.

1.4.1.1 Explicit step and time storage

```
<element>
 \-- step: Integer[variable]
 \-- (time: type[variable])
 \-- value: <type>[variable][...]
```

step A dataset with dimensions `[variable]` that contains the time steps at which the corresponding data were sampled. It is of `Integer` type to allow exact temporal matching of data from one H5MD element to another. The values of the dataset are in monotonically increasing order.

time An optional dataset that is the same as the `step` dataset, except it is `Float` or `Integer`-valued and contains the simulation time in physical units. The values of the dataset are in monotonically increasing order.

1.4.1.2 Fixed step and time storage

```
<element>
 \-- step: Integer[]
     +-- (offset: type[])
 \-- (time: type[])
     +-- (offset: type[])
 \-- value: <type>[variable][...]
```

step A scalar dataset of `Integer` type that contains the increment of the time step between two successive rows of data in `value`.

offset A scalar attribute of type `Integer` corresponding to the first sampled value of `step`.

time An optional scalar dataset that is the same as the `step` dataset, except that it is `Float` or `Integer`-valued and contains the increment in simulation time, in physical units.

offset A scalar attribute of the same type as `time` corresponding to the first sampled value of `time`.

For this storage mode, the explicit value $s(i)$ of the step corresponding to the i-th row of the dataset `value` is $s(i) = i \times \text{step} + \text{offset}$ where offset is set to zero if absent. The corresponding formula for the time $t(i)$ is identical: $t(i) = i \times \text{time} + \text{offset}$. The index i is zero-based.

1.4.2 Time-independent data

H5MD defines a *time-independent H5MD element* as a dataset. As for the `value` dataset in the case of time-dependent data, data type and array shape are implied by the stored data, where the `[variable]` dimension is omitted.

1.4.3 Storage order of arrays

All arrays are stored in C-order as enforced by the HDF5 file format (see § 3.2.5 in ("HDF5 User's guide," n.d.)). A C or C++ program may thus declare `r[N][D]` for the array of particle coordinates while the Fortran program will declare a `r(D,N)` array (appropriate index ordering for a system of `N` particles in `D` spatial dimensions), and the HDF5 file will be the same.

1.5 Storage of particles and tuples lists

1.5.1 Storage of a list of particles

A list of particles is an H5MD element:

```
<list_name>: Integer[N]
 +-- particles_group: Object reference
```

where `list_name` is a dataset of `Integer` type and dimensions `[N]`, N being the number of particle indices stored in the list. `particles_group` is an attribute containing an HDF5 Object Reference as defined by the HDF5 file format (see § 6.5.2). `particles_group` must refer to one of the groups in `/particles`.

If a *fill value* (see § 6.6 in ("HDF5 User's guide," n.d.)) is defined for `list_name`, the particles indices in `list_name` set to this value are ignored.

If the corresponding `particles_group` does not possess the `id` element, the values in `list_name` correspond to the indexing of the elements in `particles_group`. Else, the values in `list_name` must be put in correspondence with the equal values in the `id` element.

1.5.2 Storage of tuples

A list of tuples is an H5MD element:

```
<tuples_list_name>: Integer[N,T]
 +-- particles_group: Object reference
```

where `N` is the length of the list and `T` is the size of the tuples. Both `N` and `T` may indicate variable dimensions. `particles_group` is an attribute containing an HDF5 Object Reference, obeying the same rules as for the lists of particles.

The interpretation of the values stored within the tuples is done as for a list of particles.

If a *fill value* (see § 6.6 in ("HDF5 User's guide," n.d.)) is defined, tuples with at least one entry set to this value are ignored.

1.5.3 Time-dependence

As the lists of particles and tuples above are H5MD elements, they can be stored either as time-dependent groups or time-independent datasets.

As an example, a time-dependent list of pairs is stored as:

```
<pair_list_name>
   +-- particles_group: Object reference
   \-- value: Integer[variable,N,2]
   \-- step: Integer[variable]
```

The dimension denoted by `N` may be variable.

1.6 H5MD root level

The root level of an H5MD structure holds a number of groups and is organized as follows:

```
H5MD root
 \-- h5md
 \-- (particles)
 \-- (observables)
 \-- (connectivity)
 \-- (parameters)
```

`h5md` A group that contains metadata and information on the H5MD structure itself. It is the only mandatory group at the root level of H5MD.

`particles` An optional group that contains information on each particle in the system, e.g., a snapshot of the positions or the full trajectory in phase space. The size of the stored data scales linearly with the number of particles under consideration.

`observables` An optional group that contains other quantities of interest, e.g., physical observables that are derived from the system state at given points in time. The size of stored data is typically independent of the system size.

`connectivity` An optional group that contains the connectivity between particles.

`parameters` An optional group that contains application-specific, custom data such as control parameters or simulation scripts.

In subsequent sections, the examples of HDF5 organization may start at the group level, omitting the display of `H5MD root`.

1.7 H5MD metadata

A set of global metadata describing the H5MD structure is stored in the `h5md` group as attributes. The contents of the group is:

```
h5md
 +-- version: Integer[2]
 \-- author
 |    +-- name: String[]
 |    +-- (email: String[])
 \-- creator
      +-- name: String[]
      +-- version: String[]
```

version An attribute, of `Integer` datatype and of simple dataspace of rank 1 and size 2, that contains the major version number and the minor version number of the H5MD specification the H5MD structure conforms to.

The version $x.y.z$ of the H5MD specification follows semantic versioning (Preston-Werner, n.d.): A change of the major version number x indicates backwards-incompatible changes to the file structure. A change of the minor version number y indicates backwards-compatible changes to the file structure. A change of the patch version number z indicates changes that have no effect on the file structure and serves to allow for clarifications or minor text editing of the specification.

As the z component has no impact on the content of a H5MD file, the `version` attribute contains only x and y.

author A group that contains metadata on the person responsible for the simulation (or the experiment) as follows:

name An attribute, of fixed-length string datatype and of scalar dataspace, that holds the author's real name.

email An optional attribute, of fixed-length string datatype and of scalar dataspace, that holds the author's email address of the form `email@domain.tld`.

creator A group that contains metadata on the program that created the H5MD structure as follows:

name An attribute, of fixed-length string datatype and of scalar dataspace, that stores the name of the program.

version An attribute, of fixed-length string datatype and of scalar dataspace, that yields the version of the program.

1.7.1 Modules

The H5MD specification can be complemented by modules specific to a domain of research. A module may define additional data elements within the H5MD structure, add conditions that the data must satisfy, or define rules for their semantic interpretation. Multiple modules may be present, as long as their prescriptions are not contradictory. Each module is identified by a name and a version number.

The modules that apply to a specific H5MD structure are stored as subgroups within the group `h5md/modules`. Each module holds its version number as an attribute, further module-specific information may be stored:

```
h5md
 \-- (modules)
      \-- <module1>
```

```
 |    +-- version: Integer[2]
 \-- <module2>
 |    +-- version: Integer[2]
 \-- ...
```

version An attribute, of `Integer` datatype and of simple dataspace of rank 1 and size 2, that contains the major version number and the minor version number of the module.

The version *x.y.z* of an H5MD module follows semantic versioning (Preston-Werner, n.d.) and again only the components x and y are stored, see `h5md/version` in "H5MD metadata."

1.8 Particles group

Information on each particle, i.e., particle trajectories, is stored in the `particles` group. The `particles` group is a container for subgroups that represent different subsets of the system under consideration, and it may hold one or several subgroups, as needed. These subsets may overlap and their union may be incomplete, i.e., not represent all particles of the simulation volume. The subgroups contain the trajectory data for each particle as time-dependent or time-independent data, depending on the situation. Each subgroup contains a specification of the simulation box, see below. For each dataset, the particle index is accommodated by the second (first, in the case of time-independence) array dimension.

The contents of the `particles` group assuming `N` particles in `D`-dimensional space could be the following:

```
particles
 \-- <group1>
      \-- box
      \-- (position)
      |    \-- step: Integer[variable]
      |    \-- time: Float[variable]
      |    \-- value: <type>[variable][N][D]
      \-- (image)
      |    \-- step: Integer[variable]
      |    \-- time: Float[variable]
      |    \-- value: <type>[variable][N][D]
      \-- (species: Enumeration[N])
      \-- ...
```

The following identifiers for H5MD elements are standardized:

position An element that describes the particle positions as coordinate vectors of `Float` or `Integer` type.

If the component k of `box/boundary` (see below) is set to `none`, the data indicate for each particle the component k of its absolute position in space. If the component k of `box/boundary` is set to `periodic`, the data indicate for each particle the component k of the absolute position in space of an *arbitrary* periodic image of that particle.

image An element that represents periodic images of the box as coordinate vectors of `Float` or `Integer` type and allows one to compute for each particle its absolute position in space. If `image` is present, `position` must be present as well. For time-dependent data, the `step`

and `time` datasets of `image` must equal those of `position`, which must be accomplished by hard-linking the respective datasets.

If the component k of `box/boundary` (see below) is set to `none`, the values of the corresponding component k of `image` serve as placeholders. If the component k of `box/boundary` is set to `periodic`, for a cuboid box, the component k of the absolute position of particle i is computed as $R_{ik} = r_{ik} + L_k a_{ik}$, where $\vec{r}_i$ is taken from `position`, $\vec{a}_i$ is taken from `image`, and $\vec{L}$ from `box/edges`.

`velocity` An element that contains the velocities for each particle as a vector of `Float` or `Integer` type.

`force` An element that contains the total forces (i.e., the accelerations multiplied by the particle mass) for each particle as a vector of `Float` or `Integer` type.

`mass` An element that holds the mass for each particle as a scalar of `Float` type.

`species` An element that describes the species for each particle, i.e., its atomic or chemical identity, as a scalar of `Enumeration` or `Integer` data type. Particles of the same species are assumed to be identical with respect to their properties and unbonded interactions.

`id` An element that holds a scalar identifier for each particle of `Integer` type, which is unique within the given particle subgroup. The `id` serves to identify particles over the course of the simulation in the case when the order of the particles changes, or when new particles are inserted and removed. If `id` is absent, the identity of the particles is given by their index in the `value` datasets of the elements within the same subgroup.

A *fill value* (see § 6.6 in ("HDF5 User's guide," n.d.)) may be defined for `id/value` upon dataset creation. When the identifier of a particle is equal to this user-defined value, the particle is considered non-existing, the entry serves as a placeholder. This permits the storage of subsystems whose number of particles varies in time. For the case of varying particle number, the dimension denoted by `[N]` above may be variable.

`charge` An element that contains the charge associated to each particle as a scalar, of `Integer` or `Float` type.

`charge` has the optional attribute `type` of fixed-length string datatype and of scalar dataspace, possible values are `effective` and `formal`. In the case `effective`, the charge is part of an effective description of the interactions with the precise meaning depending on the underlying empirical force fields or coarse-grained models.

In the case `formal`, the charge is the so-called "formal charge" assigned to an atom (see http://en.wikipedia.org/wiki/Formal_charge) and must be of `Integer` type. This case corresponds to the entries in PDB files (see definition in the PDBx/mmCIF dictionary http://mmcif.wwpdb.org/dictionaries/mmcif_pdbx_v40.dic/Items/_atom_site.pdbx_formal_charge.html).

If none of `effective` or `formal` describes the data properly, the attribute `type` may be omitted.

1.9 Simulation box

The specification of the simulation box is stored in the group `box`, which must be contained within each of the subgroups of the `particles` group. Storing the box information at several places reflects the fact that different subgroups may be sampled at different time grids. This way, the box

information remains associated to a group of particles. A specific requirement for `box` groups inside `particles` is that the `step` and `time` datasets exactly match those of the corresponding `position` groups, which must be accomplished by hard-linking the respective datasets.

The spatial dimension and the boundary conditions of the box are stored as attributes to the `box` group, e.g., :

```
particles
 \-- <group1>
      \-- box
           +-- dimension: Integer[]
           +-- boundary: String[D]
           \-- (edges)
```

`dimension` An attribute that stores the spatial dimension `D` of the simulation box and is of `Integer` datatype and scalar dataspace.

`boundary` An attribute, of fixed-length string datatype and of simple dataspace of rank 1 and size `D`, that specifies the boundary condition of the box along each dimension. The values in `boundary` are either `periodic` or `none`:

`periodic` The simulation box is periodically continued along the given dimension and serves as the unit cell for an infinite tiling of space.

`none` No boundary condition is imposed. This summarizes the situations of open systems (i.e., an infinitely large box) and closed systems (e.g., due to an impenetrable wall). For those components where `boundary` is set to `none`, the corresponding values of `edges` serve as placeholders.

Information on the geometry of the box edges is stored as an H5MD element, allowing for the box to be fixed in time or not. Supported box shapes are the cuboid and triclinic unit cell, for other shapes a transformation to the triclinic shape may be considered (Bekker 1997). If all values in `boundary` are `none`, `edges` may be omitted.

`edges` A `D`-dimensional vector or a $D \times D$ matrix, depending on the geometry of the box, of `Float` or `Integer` type. If `edges` is a vector, it specifies the space diagonal of a cuboid-shaped box. If `edges` is a matrix, the box is of triclinic shape with the edge vectors given by the rows of the matrix.

For a time-dependent box, a cuboid geometry is encoded by a dataset `value` (within the H5MD element) of rank 2 (1 dimension for the time and 1 for the vector) and a triclinic geometry by a dataset `value` of rank 3 (1 dimension for the time and 2 for the matrix).

For a time-independent box, a cuboid geometry is encoded by a dataset `edges` of rank 1 and a triclinic geometry by a dataset of rank 2.

For instance, a cuboid box that changes in time would appear as:

```
particles
 \-- <group1>
      \-- box
           +-- dimension: Integer[]
           +-- boundary: String[D]
           \-- edges
                \-- step: Integer[variable]
```

```
                \-- time: Float[variable]
                \-- value: <type>[variable][D]
```

where `dimension` is equal to `D`. A triclinic box that is fixed in time would appear as:

```
particles
 \-- <group1>
      \-- box
           +-- dimension: Integer[]
           +-- boundary: String[D]
           \-- edges: <type>[D][D]
```

where `dimension` is equal to `D`.

1.10 Observables group

Macroscopic observables, or more generally, averages of some property over many particles, are stored in the root group `observables`. Observables representing only a subset of the particles may be stored in appropriate subgroups similarly to the `particles` tree. Each observable is stored as an H5MD element. The shape of the corresponding dataset (the element itself for time-independent data and `value` for time-dependent data) is the tensor shape of the observable, prepended by a `[variable]` dimension for time-dependent data.

The contents of the observables group has the following structure:

```
observables
 \-- <observable1>
 |    \-- step: Integer[variable]
 |    \-- time: Float[variable]
 |    \-- value: <type>[variable]
 \-- <observable2>
 |    \-- step: Integer[variable]
 |    \-- time: Float[variable]
 |    \-- value: <type>[variable][D]
 \-- <group1>
 |    \-- <observable3>
 |         \-- step: Integer[variable]
 |         \-- time: Float[variable]
 |         \-- value: <type>[variable][D][D]
 \-- <observable4>: <type>[]
 \-- ...
```

1.11 Connectivity group

The connectivity information is stored as tuples in the group `/connectivity`. The tuples are pairs, triples, etc. as needed and may be either time-independent or time-dependent. Several connectivity elements may be defined for any particles group. A connectivity element may only refer to a single particle group.

The tuples of particles are interpreted according to the section Storage of particles and tuples lists.

1.12 Parameters group

The `parameters` group stores application-specific, custom data such as control parameters or simulation scripts. The group consists of groups, datasets, and attributes; the detailed structure, however, is left unspecified.

The contents of the `parameters` group could be the following:

```
parameters
 +-- <user_attribute1>
 \-- <user_data1>
 \-- <user_group1>
 |    \-- <user_data2>
 |    \-- ...
 \-- ...
```

1.13 References

2 Modules

2.1 Thermodynamic observables

2.1.1 Objective

This module defines a set of thermodynamic observables commonly output by molecular simulation programs.

2.1.2 Module name and version

The name of this module is `thermodynamics`. The module version is 1.0.0.

2.1.3 Observables

Thermodynamic observables are stored in the `observables` group for global properties or in `observables/<group>` for subsystems, similarly to the `particles` group. The groups have the following contents:

```
observables
 \-- <group>
      +-- dimension: Integer[]
      \-- particle_number
      \-- (pressure)
      \-- (temperature)
      \-- (density)
      \-- ...
```

`dimension` A scalar attribute of `Integer` type that gives the dimension of the space embedding the subsystem.

`particle_number` The number of particles in the subsystem stored as scalar H5MD element of `Integer` datatype.

The following H5MD elements are optional, of scalar character, and use the `Float` datatype.

`pressure` The pressure of the subsystem.

temperature The (instantaneous) temperature of the subsystem as inferred from the kinetic energy.
density The number density of the subsystem stored as `Float` or `Integer` datatype.
potential_energy The potential energy of the subsystem.
kinetic_energy The kinetic energy of the subsystem.
internal_energy The internal energy of the subsystem, typically the sum of potential and kinetic energy.
enthalpy The enthalpy of the subsystem.

The latter 4 quantities are stored as per-particle averages. The corresponding extensive quantities (scaling linearly with the system size) are obtained by multiplication with the number of particles. Per-volume averages follow by multiplication with the number density if present.

2.2 Units

2.2.1 Objective

This module defines how physical units are attached to dimensionful H5MD elements.

2.2.2 Module name and version

The name of this module is `units`. The module version is 1.0.0.

2.2.3 Unit system definition

The `units` group possesses, in addition to the `version` attribute, a `system` attribute that defines the unit system in use. `system` is of scalar dataspace and fixed-length string datatype.

2.2.4 Unit attribute

The datasets of any H5MD element that have a physical dimension may carry an attribute `unit` to indicate the physical unit of the respective data. In general, this refers to the dataset itself for time-independent elements, or to the datasets `value` and `time` in the time-dependent case:

```
<element>
 \-- step: Integer[variable]
 \-- time: Float[variable]
 |    +-- (unit: String[])
 \-- value: <type>[variable][...]
      +-- (unit: String[])
```

The attribute `unit` is of scalar dataspace and fixed-length `String` datatype using the ASCII character set.

2.2.5 Unit string

The `unit` string consists of a sequence of unit factors separated by a space. A unit factor is either a number (an integer or a decimal fraction) or a unit symbol optionally followed by a non-zero, signed integer indicating the power to which this factor is raised. Each unit symbol may occur only once. There may also be at most one numeric factor, which must be the first one.

Examples:

- "nm+3" stands for cubic nanometers

- “um+2 s-1” stands for micrometers squared per second
- “60 s” stands for a minute
- “10+3 m” stands for a kilometer

2.2.6 The “SI” unit system

The “SI” unit system (“The International System of Units (SI)” 2006) defines SI base units (§2.1), SI derived units (§2.2), and SI prefixes (§3.1).

Table 1: SI base unit symbols and names.

dimension	symbol	unit name
length	m	meter
mass	kg	kilogram
time	s	second
electric current	A	ampere
temperature	K	kelvin
amount of substance	mol	mole
luminous intensity	cd	candela

Table 2: SI derived unit symbols, names and conversion rules.

dimension	symbol	unit name | conversio	n
plane angle	rad	radian | 1 rad = 1	m m^{-1}
solid angle	sr	steradian | 1 sr = 1	m^2 m^{-2}
frequency	Hz	hertz | 1 Hz = 1	s^{-1}
force	N	newton | 1 N = 1 m	kg s^{-2}
pressure/stress	Pa	pascal | 1 Pa = 1	N m^{-2}
energy/work	J	joule | 1 J = 1 N	m
power	W	watt | 1 W = 1 J	s^{-1}
electric charge	C	coulomb | 1 C = 1 A	s
voltage	V	volt | 1 V = W A	$^{-1}$
capacitance	F	farad | 1 F = C V	$^{-1}$
electric resistance	ohm	ohm | 1 Ω = V A	$^{-1}$
electric conductance	S	siemens | 1 S = 1 A	V^{-1}
magnetic flux	Wb	weber | 1 Wb = 1	V s
magnetic flux density	T	tesla | 1 T = 1 W	b m^{-2}
inductance	H	henry | 1 H = 1 W	b A^{-1}
Celsius temperature	degC	degree Celsius | 0 °C = 27	3.15 K
luminous flux	lm	lumen | 1 lm = 1	cd sr
illuminance	lx	lux | 1 lx = 1	lm m^{-2}
radioactivity	Bq	becquerel | 1 Bq = 1	s^{-1}
absorbed dose	Gy	gray | 1 Gy = 1	J kg^{-1}
dose equivalent	Sv	sievert | 1 Sv = 1	J kg^{-1}
catalytic activity	kat	katal | 1 kat = 1	mol s^{-1}

Table 3: SI prefixes.

prefix	symbol	factor
exa-	E	10^{18}
peta-	P	10^{15}
tera-	T	10^{12}
giga-	G	10^{9}
mega-	M	10^{6}
kilo-	k	10^{3}
hecto-	h	10^{2}
deca-	da	10^{1}
deci-	d	10^{-1}
centi-	c	10^{-2}
milli-	m	10^{-3}
micro-	u	10^{-6}
nano-	n	10^{-9}
pico-	p	10^{-12}
femto-	f	10^{-15}
atto-	a	10^{-18}

2.2.7 References

3 H5MD Developers

H5MD has been started by

- Pierre de Buyl
- Peter Colberg
- Felix Höfling

With further developments being brought in by other developers, "H5MD Developers" refers to the three above developers and to:

- Olaf Lenz
- Konrad Hinsen

Bekker, H. 1997. "Unification of Box Shapes in Molecular Simulations." *J. Comput. Chem.* 18 (15): 1930–42. https://doi.org/10.1002/(SICI)1096-987X(19971130)18:15<1930::AID-JCC8>3.0.CO;2-P.

"HDF5 User's guide." n.d. http://www.hdfgroup.org/HDF5/doc/UG.

Preston-Werner, Tom. n.d. "Semantic Versioning." http://semver.org/spec/v2.0.0.html.

The HDF Group. n.d. "Hierarchical Data Format Version 5." http://www.hdfgroup.org/HDF5.

"The International System of Units (SI)." 2006. 8th ed. Paris: Bureau International des Poids et Mesures. http://www.bipm.org/en/publications/si-brochure/.