MDF: Magnetic Particle Imaging Data Format

T. Knopp[1,2], T. Viereck[3], G. Bringout[4], M. Ahlborg[5], A. von Gladiß[5], C. Kaethner[5], J. Rahmer[6], M. Möddel[1,2]

[1]Section for Biomedical Imaging, University Medical Center Hamburg-Eppendorf, Germany
[2]Institute for Biomedical Imaging, Hamburg University of Technology, Germany
[3]Institute of Electrical Measurement and Fundamental Electrical Engineering, TU Braunschweig, Germany
[4]Physikalisch-Technische Bundesanstalt, Berlin, Germany
[5]Institute of Medical Engineering, University of Lübeck, Germany
[6]Philips GmbH Innovative Technologies, Research Laboratories, Röntgenstraße 24-26, 22315 Hamburg, Germany

May 30, 2017

Version **2.0.0-pre**

Abstract

Magnetic particle imaging (MPI) is a tomographic method to determine the spatial distribution of magnetic nanoparticles. In this document a file format for the standardized storage of MPI data is introduced. The aim of the Magnetic Particle Imaging Data Format (MDF) is to provide a coherent way of exchanging MPI data acquired with different MPI scanners worldwide. The focus of the file format is on sequence parameters, raw measurement data, calibration data, and reconstruction data. The format is based on the hierarchical document format (HDF) in version 5 (HDF5).

1 Introduction

The purpose of this document is to introduce a file format for exchanging Magnetic Particle Imaging (MPI) data. The Magnetic Particle Imaging Data Format (MDF) is based on the hierarchical document format (HDF) in version 5 (HDF5) [1]. HDF5 allows to store multiple datasets within a single file and is thus very flexible to use. To allow the exchange of MPI data, one has to specify a naming scheme within HDF5 files which is the purpose of this document. In order to create and access HDF5 data, an Open Source C library is available. For most programming languages bindings to this library exist. Matlab supports HDF5 by the functions `h5read` and `h5write`. For Python the `h5py` package exists. The Julia programming language provides access to HDF5 files via the `HDF5` package. For languages based on the .NET framework the `HDF5DotNet` library is available.

In this initial version of the file format the focus is on sequence parameters, raw measurement data, calibration data, and reconstruction data. The format can store three different dataset types

1. Measurement data
2. System calibration data
3. Reconstruction data

It is possible to combine measurement data and reconstruction data into a single file. However, calibration data has to be stored in an independent HDF5 file.

1.1 Datatypes

For most parameters a fixed datatype is used, i.e. the drive-field amplitudes are stored as `H5T_NATIVE_DOUBLE` values. For our convinience we refer to the HDF5 datatypes `H5T_STRING`, `H5T_NATIVE_DOUBLE` and `H5T_NATIVE_INT64` as `String`, `Float64` and `Int64`. The datatype of the measurement data and the calibration data offers more freedom and is denoted by `Number`, which can be any of the following HDF5 data types: `H5T_NATIVE_FLOAT`, `H5T_NATIVE_DOUBLE`, `H5T_NATIVE_INT8`, `H5T_NATIVE_INT16`, `H5T_NATIVE_INT32`, and `H5T_NATIVE_INT64`.

MPI parameters are stored as regular *HDF5 datasets*. *HDF5 attributes* are not used in the current specification of the MDF.

Since storing complex data in HDF5 is not standardized, we extend the dimensionality of an existing array and store the real and imaginary part in the last dimension with size 2 (index 0 = real part, index 1 = imaginary part). In this way the real and imaginary part of a complex datum is stored sequentially on disk. When loading the data it is possible to cast it to a complex array in most programming languages.

1.2 Units

Physical quantities are given in SI units with one exception. The field strength is reported in $\mathrm{T}\mu_0^{-1} = 4\ \pi\ \mathrm{Am}^{-1}\mu_0^{-1}$. This convention has been proposed in the first MPI publication and since that time consistently used in most MPI related publications. The aim of this convention is to report the numbers on a Tesla scale, which most readers with a background in MRI are familiar with, but, on the other hand still use the correct unit for the magnetic field strength.

1.3 Sanity Check

In order to check if a generated MDF file is valid, we provide a sanity check script that can be found in the gitub repository:

`https://github.com/MagneticParticleImaging/MDF`

The code is written in the Julia programming language [2, 3, 4], which has to be downloaded from:

`http://julialang.org`.

More detailed instructions can be found in the `README` of the repository.

1.4 Contact

If you find mistakes in this document or the specified file format or if you want to discuss extensions to this specification, please open an issue on GitHub:

`https://github.com/MagneticParticleImaging/MDF`

As the file format is versionized it will be possible to extend it for future needs of MPI. The current version discussed in this document is version 2.0.0-pre.

1.5 arXiv

As of version 1.0.1 the most recent release of these specifications can also be also found on the arXiv:

`http://arxiv.org/abs/1602.06072`

If you use MDF please cite us using the arXiv reference, which is also available for download as `MDF.bib` from GitHub.

2 Data (group: `/`)

Remarks: Within the root group metadata about the file itself is stored. Within several subgroups, metadata about the experimental setting, the MPI tracer, and the MPI scanner are stored. The actual data is stored in dedicated groups on measurement data, calibration data, and/or reconstruction data.

Parameter	Type	Dim	Unit/Format	Optional	Description
`version`	String	1	0.1	no	Version of the file format
`uuid`	String	1	f81d4fae-7dec-11d0-a765-00a0c91e6bf6	no	Universally Unique Identifier (RFC 4122)
`time`	String	1	yyyy-mm-ddThh:mm:ss.ms	no	UTC creation time of MDF data set

2.1 Study Description (group: `/study/`)

Remarks: The study description group describes the experimental setting under which the MPI data was recorded. The study field may be used as a name tag for several experiments, which are related. The dataset at hand may then be described by a number and a short description. Additionally, the name of the imaged subject can be provided and the starting time of the MPI measurement can be provided.

The reference field may be used to indicate, if the background signal of the scanner was recorded.

Parameter	Type	Dim	Unit/Format	Optional	Description
`name`	String	1		yes	Name of the study
`experiment`	String	1		yes	Experiment number within study
`description`	String	1		yes	Short description of the experiment
`subject`	String	1		yes	Name of the subject that was imaged
`isSimulation`	Int64	1		yes	Flag indicating if the data in this file is simulated rather than measured
`isCalibration`	Bool	1		no	Flag indicating if data belongs to a system matrix calibration measurement

2.2 Tracer Parameters (group: `/tracer/`)

Remarks: The tracer parameter group contains information about the MPI tracer used during the experiment such as the tracer name, its vendor, the tracer concentration, and the total volume applied.

Note that the injection clock recording the injection time should be synchronized with the clock, which provides the starting time of the measurement.

Parameter	Type	Dim	Unit/Format	Optional	Description
`name`	`String`	1		yes	Name of tracer used in experiment
`batch`	`String`	1		yes	Batch of tracer
`vendor`	`String`	1		yes	Name of tracer supplier
`volume`	`Float64`	1	L	yes	Total volume of applied tracer
`concentration`	`Float64`	1	mol(solute)/L	yes	Molar concentration of solute per litre
`solute`	`String`	1		yes	Solute, e.g. Fe
`injectionTime`	`String`	1	yyyy-mm-ddThh:mm:ss.ms	yes	UTC time at which tracer injection started

2.3 Scanner Parameters (group: `/scanner/`)

Remarks: The scanner parameter group contains information about the MPI scanner used such as the manufacturer, the model, and the facility where the scanner is installed.

Parameter	Type	Dim	Unit/Format	Optional	Description
`facility`	`String`	1		no	Facility where the MPI scanner is installed
`operator`	`String`	1		no	User who operates the MPI scanner
`manufacturer`	`String`	1		no	Scanner manufacturer
`model`	`String`	1		no	Scanner model
`topology`	`String`	1		no	Scanner topology (e.g. FFP or FFL)

2.4 Acquisition Parameters (group: /acquisition/)

Remarks: The acquisition parameter group can describe different imaging protocols and trajectory settings. The corresponding data is organized into general information, a subgroup containing data on the particle excitation and a subgroup containing data on the receive channels.

In general each MPI dataset consists of the measurement data of N frames. A frame groups all data together that will be used to reconstruct an image/volume. On certain MPI scanners the drive-field field-of-view (FOV) can be shifted either by magnetic fields or by mechanical movement. Therefore, a frame may consist of J sub-measurements. For instance a Cartesian 2D trajectory with 100 lines would be realized by setting `numPatches` = 100.

The center of the drive-field is specified in the parameter `fieldOfViewCenter`. The shift of the FOV is induced by a certain offset field that can be reported in the parameter `offsetField`

Parameter	Type	Dim	Unit/Format	Optional	Description
`startTime`	String	1	yyyy-mm-ddThh:mm:ss.ms	no	UTC start time of MPI measurement
`numFrames`	Int64	1	1	no	Number of available measurement frames, denoted by N
`numBackgroundFrames`	Int64	1	1	no	Number of available background measurement frames, denoted by M
`framePeriod`	Float64	1	s	no	Complete time to acquire data of a full frame (product of drive field `period`, `numPatches`, and `numAverages`)
`numPatches`	Int64	1	1	no	Number of patches within a frame denoted by J
`gradient`	Float64	$J \times 3$	$\mathrm{Tm}^{-1}\mu_0^{-1}$	yes	Gradient strength of the selection field in x, y, and z directions
`offsetField`	Float64	$J \times 3$	$\mathrm{T}\mu_0^{-1}$	yes	Offset field applied for each patch in the measurement sequence
`fieldOfView`	Float64	$J \times 3$	m	yes	Approximate size of the area/volume captured by the trajectory
`fieldOfViewCenter`	Float64	$J \times 3$	m	yes	Position of the field free point (relative to origin/center)

2.4.1 Drive Field (group: /acquisition/drivefield/)

Remarks: The drive field subgroup describes the details on the imaging protocol and trajectory settings. On the lowest level each MPI scanner contains D channels for particle excitation. Since most drive-field parameters may change from patch to patch they have a leading J dimension.

These excitation signals are usually sinusoidal and can be described by D amplitudes (drive field strengths), D phases, a base frequency, and D dividers. In a more general setting the drive-field in channel d is described by

$$H_d(t) = \sum_{l=1}^{F} A_l \Lambda_l(2\pi f_l t + \varphi_l)$$

where F is the number of frequencies on the channel, A_l is the drive-field strength, ϕ_l is the phase, f_l is the frequency (described by the base frequency and the divider), and Λ_l is the waveform. The waveform is specified by a dedicated parameter `waveform`. If `waveform` is set to *custom*, one can specify a custom waveform using the parameter `customWaveform`. The triangle is defined to be a 2π periodization of the triangle function:

$$\Lambda_{\text{tri}}(t) = \left|t + \frac{\pi}{2}\right| - \frac{\pi}{2} \quad \text{for} \quad -\frac{3}{2}\pi \leq t \leq \frac{\pi}{2}$$

Parameter	Type	Dim	Unit/Format	Optional	Description
`numChannels`	Int64	1	1	no	Number of drive field channels, denoted by D
`strength`	Float64	$J \times D \times F$	$\mathrm{T}\mu_0^{-1}$	no	Applied drive field strength
`phase`	Float64	$J \times D \times F$	$\mathrm{T}\mu_0^{-1}$	no	Applied drive field phase φ in radians in the range $[-\pi, \pi)$
`baseFrequency`	Float64	1	Hz	no	Base frequency to derive drive field frequencies
`customWaveform`	Float64	$D \times F \times U$	1	yes	Custom waveform table
`divider`	Int64	$D \times F$	1	no	Divider for drive fields frequencies (`baseFrequency` / `divider`)
`waveform`	String	$D \times F$	1	no	Waveform type: *sine*, *triangle* or *custom*
`period`	Float64	1	s	no	Drive field trajectory period

2.4.2 Receiver (group: `/acquisition/receiver/`)

Remarks: The receiver subgroup describes details on the MPI receiver. For a multi-patch sequence it is assumed, that signal acquisition only takes place during particle excitation. During each drive-field cycle, C receive channels record the superposition of the change of the particle magnetization at Z equidistant time points. The transfer function can optionally be stored in the parameter `transferFunction`. It is stored in frequency space representation where $K = \frac{Z}{2} + 1$ is the number of discrete frequency components.

Parameter	Type	Dim	Unit/Format	Optional	Description
numChannels	Int64	1		no	Number of receive channels C
numAverages	Int64	1		no	Internal block averaging over a number of drive field cycles
bandwidth	Float64	1	Hz	no	Bandwidth of the receiver unit
numSamplingPoints	Int64	1		no	Number of sampling point within one drive-field period denoted by Z
transferFunction	Float64	$C \times K \times 2$		yes	Transfer function of the receive channel

2.5 Measurement (group: /measurement/)

Remarks: MPI data is usually acquired by a series of N measurements and M background measurements. Here we refer to background measurements as MPI data captured, when any signal generating material, e.g. a phantom or a delta sample is removed from the scanner bore. Both the N measurements and the M background measurements can be stored in an arbitrary order, e.g. with respect to the spatial position of a delta sample if the data corresponds to a calibration measurement. To be able to recover the time order in which the measurements were taken each of the $N+M$ data sets can be assigned an integer number $o = 1, 2, \dots, N+M$ ordering the measurements and background measurements with respect to time. I.e. data set o_1 is acquired prior to data set o_2, if and only if $o_1 < o_2$.

The resulting N measurements and M background measurements should be stored in time domain, where the data of a single frame consists of the signal recorded for all patches in each receive channel, i.e. $J \times C \times Z$ data points per set with the temporal index being the fastest to access. If several measurements are acquired (indicated by *numFrames*), the N measurements and M background measurements are concatenated along the slowest dimension respectively.

During measurements the analog signal measured is usually converted into $(r_1, \dots, r_{JZ}) \in \mathbb{Z}^J \times \mathbb{Z}^Z$ integer values per channel $c \in C$ and frame using analog to digital converters. Often this raw data is stored instead of the physical quantities they represent. To bring the raw values into a physical representation one can map $r_i \mapsto (a_c r_i + b_c)U$, where a_c and b_c are the characteristic dimensionless scaling factor and offset the receive channel $c \in C$ and U is the corresponding unit of measurement, i.e. usually voltages.

Parameter	Type	Dim	Unit/Format	Optional	Description
unit	String	1		yes	SI unit of the measured quantity, usually V
rawDataConversion	Number	$C \times 2$		yes	Dimension less scaling factor and offset (a_c, b_c) to convert raw data into a physical quantity with corresponding unit of measurement unit
data	Number	$N \times J \times C \times Z$		yes	Measurement data stored in time domain representation
dataTimeOrder	Int64	N		yes	Time ordering number for measurements
backgroundData	Number	$M \times J \times C \times Z$		yes	Background measurements stored in time domain representation
backgroundDataTimeOrder	Int64	M		yes	Time ordering number for background measurements

2.6 Calibration (group: /calibration/)

Remarks: To handle system matrix data efficiently, Fourier transformation and extensive reordering of the raw time domain data is required. Therefore, a post processed and background corrected system matrix can be stored separately.

This post processing is done as follows:

- First, the background corrected time domain data is Fourier transformed along the Z dimension into a $N \times J \times C \times K$ complex valued dataset.
- Second, this dataset is brought into a real valued representation of $N \times J \times C \times K \times 2$ real valued data points.
- Third, the data is partially transposed into the form $J \times C \times K \times N \times 2$.

This representation grants a quick access to the frequency dimensions $J \times C \times K$ and therefore a fast frequency selection without having read the entire system matrix. For this selection often the signal to noise characteristics of the frequency components is used, which can be stored as well, without having to post process the raw measurement data repeatedly.

Each of the N calibration measurements is taken with a a calibration sample (delta sample) at a fixed position inside the FOV of the scanner. Each background measurement is taken with the delta sample outside of the FOV of the scanner. Usually, the calibration measurements are not stored in the order in which they were taken, but with respect to the corresponding spatial position of the delta sample. If a regular grid of size $N_x \times N_y \times N_z$ is used for sampling, by default the $N_x N_Y N_z = N$ measurements are ordered with respect to their x position first, second with respect to their y position and last with respect to their z position. If a different ordering is used this can be documented using the optional parameter `order`. For non-regular sampling points there is the possibility to explicitly store all N positions.

Parameter	Type	Dim	Unit/Format	Optional	Description
systemMatrixData	Number	$J \times C \times K \times N \times 2$		yes	Stores the background corrected system matrix in Fourier representation with the last dimension storing the complex data.
snr	Float64	$J \times C \times K$		yes	Signal-to-noise estimate for recorded frequency components
fieldOfView	Float64	3	m	yes	Field of view of system matrix
fieldOfViewCenter	Float64	3	m	yes	Center of the system matrix (relative to origin/center)
size	Int64	3		yes	Number of voxels in each dimension
order	String	1		yes	Ordering of the dimensions, default is xyz
positions	Float64	$N \times 3$	m	yes	Position of each of the grid points, stored as (x, y, z) triples
offsetField	Float64	$N \times 3$	$\mathrm{T}\mu_0^{-1}$	yes	Applied offset field strength to emulate a spatial position (x, y, z)
deltaSampleSize	Float64	3	m	yes	Size of delta sample used for calibration scan
method	String	1		yes	Method used to obtain calibration data. Can for instance be robot, hybrid, or simulation

2.7 Reconstruction Results (group: `/reconstruction/`)

Reconstruction results are stored in the parameter `data`. Dependent on the number of individual channels S obtained by the reconstruction the results can be stored in a $L \times P$ array for $S = 1$ or in a $L \times P \times S$ array for $S > 1$. Since the grid of the reconstruction data can be different than the system matrix grid, the grid parameter are mirrored in the reconstruction parameter group.

Usually the data is stored in a real data format but it is also possible to store complex data if the reconstruction output is complex.

Parameter	Type	Dim	Unit/Format	Optional	Description
`data`	Number	$L \times P \times S$		yes	Reconstructed data
`fieldOfView`	Float64	3	m	yes	Field of view of reconstructed data
`fieldOfViewCenter`	Float64	3	m	yes	Center of the reconstructed data (relative to origin/center)
`size`	Int64	3		yes	Number of voxels in each dimension
`order`	String	1		yes	Ordering of the dimensions, default is *xyz*
`positions`	Float64	$P \times 3$	m	yes	Position of each of the grid points, stored as (x, y, z) tripels

2.8 Changelog

2.8.1 v2.0

- Updated Affiliations in the MDF specification.
- Improved the description of various fields.
- Added definition for triangle function.
- Infrastructure for storing background data directly with MDF data has been added for both regular measurements and calibration measurements.
- In v1.x the MDF allowed certain fields to have varying dimensions depending on the context. This has been removed such that starting from v2.0 all dimensions have to be specified. This change should make implementations handling MDF files less complex.
- Support for multiple excitation frequencies on a drive-field channel has been added. Additionally we added support for fully arbitrary excitation waveforms.
- Added the possibility store the tracer concentration also for non iron based tracer materials by adding the `solute` field to the `tracer` group.
- Specified supported data types for the storage of measurement data and reconstruction data.
- Raw data is now always stored in time domain representation for both measurement data and background measurements.
- Added possibility to store the transformation from raw data to a physical representation with units.
- Signal to noise ratios can now be stored for each patch indivisually. To do so the dimension J was added to this field.
- Restructured storage of raw measurement data.
- Made fields `gradient`, `fieldOfView` and `fieldOfViewCenter` optional.
- Introduced the field `dataTimeOrder` and `backgroundDataTimeOrder` to store the order in which measurements and background measurements were taken.
- Added field `numBackgroundFrames` to `aquisition` group.
- Added field `isCalibration` to `study` group to indicate if data corresponds to a calibration measurement or not
- Moved `numAverages` field to `receiver` subgroup, `offsetfield` field to `acquisition` group and `fieldOfViewCenter` field to `acquisition` group.
- Remove `/study/reference` since this functionality is now covered by the integrated background measurements.
- Rename `/study/simulation` to `/study/isSimulation` for consistency reasons.
- Remove `/acquisition/receiver/frequencies` since it can be directly derived from `/acquisition/receiver/bandwidth` and `/acquisition/receiver/numSamplingPoints`.
- Rename `/date` to `/time` for consistency reasons.
- Rename `/acquisition/time` to `/acquisition/startTime`.
- Rename `/tracer/time` to `/tracer/injectionTime`.

2.8.2 v1.0.5

- Added the possibility to store different channels of reconstructed data.
- Added support for receive channels with different characteristics (e.g. bandwidth).
- Made dataset `/acquisition/receiver/frequencies` optional.
- Extended the description on the data types, which are used to store data.
- Added references for Julia and HDF5 to the specifications.

2.8.3 v1.0.4

- Clarify that HDF5 datasets are used to store MPI parameters.

2.8.4 v1.0.3

- Updated Affiliations in the MDF specification.
- Included data download into the Python and Matlab example code.
- Changes in the Python and Matlab example code to be better comparable to the Julia example code.

2.8.5 v1.0.2

- Added reference to arXiv paper and bibtex file for reference.

2.8.6 v1.0.1

- A sanity check within the Julia code shipped alongside the specifications.
- An update to the specification documenting the availability of a sanity check.
- Updated MDF files on https://www.tuhh.de/ibi/research/mpi-data-format.html.
- Updated documentation to the Julia, Matlab and Python reconstruction scripts.
- Improved Julia reconstruction script, automatically downloading the required MDF files.

References

[1] The HDF Group. Hierarchical Data Format, version 5, 1997-2016. http://www.hdfgroup.org/HDF5/.

[2] Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman. Julia: A fast dynamic language for technical computing. *CoRR*, abs/1209.5145, 2012.

[3] Jeff Bezanson, Jiahao Chen, Stefan Karpinski, Viral B. Shah, and Alan Edelman. Array operators using multiple dispatch: a design methodology for array implementations in dynamic languages. *CoRR*, abs/1407.3845, 2014.

[4] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A fresh approach to numerical computing. *CoRR*, abs/1411.1607, 2014.