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], P. Vogel[6], J. Rahmer[7], 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]Department of Experimental Physics 5 (Biophysics), University of Würzburg, Germany
[7]Philips GmbH Innovative Technologies, Research Laboratories, Hamburg, Germany

June 19, 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 and magnetic particle spectroscopy (MPS) data is introduced. The aim of the Magnetic Particle Imaging Data Format (MDF) is to provide a coherent way of exchanging MPI and MPS data acquired with different scanners world wide. 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) and Magnetic Particle Spectroscopy (MPS) data. The Magnetic Particle Imaging Data Format (MDF) is based on the hierarchical document format (HDF) in version 5 (HDF5) [1]. HDF5 is able to store multiple datasets within a single file providing a powerful and flexible data container. To allow an easy 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 providing dynamic access from most programming languages. 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.

Version 2.0.0-preof the MDF is mainly focused on storing raw measurement data, post processed data, system matrices, or reconstruction data together with corresponding sequence parameters and meta data. Though it is possible to combine measurement data and reconstruction data into a single file it is recommended to use a single file for each of the following dataset types:

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

1.1 Datatypes

MPI parameters are stored as regular *HDF5 datasets*. *HDF5 attributes* are not used in the current specification of the MDF. For most data sets 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`. Boolean data is stored as `H5T_NATIVE_INT8`, which we refer to as `Int8`.

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`. The same holds true for the `Integer` data type, which can be any of: `H5T_NATIVE_INT8`, `H5T_NATIVE_INT16`, `H5T_NATIVE_INT32`, and `H5T_NATIVE_INT64`.

For later identification of a data set we store three UUIDs (RFC 4122) [2] in its canonical textual representation as 32 hexadecimal digits, displayed in five groups separated by hyphens `8-4-4-4-12` as for example `ee94cb6d-febf-47d9-bec9-e3afa59bfaf8`. For the generation of the UUIDs we recomend to use version 4.

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.

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 [3] and since that time consistently used in most MPI related publications. The aims of this convention are to report the numbers on a Tesla scale, which most readers with a background in MRI are familiar with and use the correct unit for the magnetic field strength.

1.3 Optional Parameters

The MDF has 9 main groups in the root directory. We distinguish between optional and non-optional groups and optional, non-optional, and conditional parameters. Any optional parameter can be omitted, whereas any non-optional parameter in a non-optional group is mandatory. Conditional parameters are linked to Boolean parameters and have to be provided if that parameter is true and can be omitted if that parameter is false. If a parameter is optional, non-optional, or conditional is indicated by yes, no, or the corresponding Boolean parameter respectively.

If a group is optional all of its parameters may be omitted if this group is not used. The groups `/`, `/study`, `/experiment`, `/tracer`, `/scanner`, `/acquisition` contain mostly metadata and are mandatory. The groups `/measurement`, `/processing`, `/calibration`, and `/reconstruction` are all optional. In case of calibration measurements, the `/calibration` group is mandatory. `/measurement` and `/processing` are both optional but MPI measurement data will contain one of both groups. Reconstruction data is stored in `/reconstruction`. One should not store regular measurements, calibration data, and reconstruction results in a single MDF. Instead, individual files should be used.

1.4 Parameter Extension

Often it is necessary to store additional specific parameters or meta data not covered in the specifications, like for example the temperature of the room in which your MPI device is operated. In this case you are free to add a new parameters to any of the existing groups. Moreover if necessary you are also free to introduce new groups. To be able to distinguish these datasets and groups from the specified ones we recommend to use the prefix _ for all parameters and groups. As an example one could add a new group `_room` and within the dataset `_temperature`.

1.5 Naming Convention

Several parameters within an MDF are linked in the dimensionality. We use short variable names to indicate these connections. The following table describes the meaning of each used variable name

Variable	Meaning: Number of...
A	tracer materials/injections for multi-color MPI
N	acquired frames
O	acquired frames w/o background frames
J	focus-field patches
C	receive channels
D	drive-field channels
F	frequencies describing the drive-field waveform
U	sampling points describing a custom drive-field waveform
V	points sampled at receiver during one patch (product of drive field `period`, `numPeriods`, `numAverages`)
W	sampling points used to store processed data (usually $W = V$ except when a bandwidth reduction has been done)
K	frequencies describing processed data (usually $K = \frac{V}{2} + 1$ if no frequency selection has been done)
L	frequencies describing the transfer function (usually $L = K$)
P	voxels in the reconstructed MPI data set
Q	frames in the reconstructed MPI data set
S	channels in the reconstructed MPI data set

1.6 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.

1.7 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.

1.8 Sanity Check

TODO: move sanity check to `MPIFile.jl`

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 [4, 5, 6], which has to be downloaded from:

`http://julialang.org`.

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

2 Data (group: /, non-optional)

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 can be provided. The actual data is stored in dedicated groups on measurement data, processing 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	3170fdf8-f8e1-4cbf-ac73-41520b41f6ee	no	Universally Unique Identifier (RFC 4122) of MDF file
`time`	`String`	1	yyyy-mm-ddThh:mm:ss.ms	no	UTC creation time of MDF data set

2.1 Study Description (group: /study/, non-optional)

Remarks: A study is supposed to group a series of experiments to support, refute, or validate a hypothesis. The study group may contain `name`, `number`, and `description` of the study.

Parameter	Type	Dim	Unit/Format	Optional	Description
`name`	`String`	1		no	Name of the study
`number`	`Int64`	1		no	Experiment number within study
`uuid`	`String`	1	295258fe-b650-4e5f-96db-b83f11089a6c	no	Universally Unique Identifier (RFC 4122) of study
`description`	`String`	1		no	Short description of the experiment

2.2 Experiment Description (group: `/experiment/`, non-optional)

Remarks: For each experiment within a study `name`, `number`, and `description` may be provided. Additionally, the name of the imaged subject and a flag indicating if data has been obtained via simulations can be stored.

Parameter	Type	Dim	Unit/Format	Optional	Description
`name`	String	1		no	Experiment name
`number`	Int64	1		no	Experiment number
`uuid`	String	1	f96dbc48-1ebd-44c7-b04d-1b45da054693	no	Universally Unique Identifier (RFC 4122) of experiment
`description`	String	1		no	Short description of the experiment
`subject`	String	1		no	Name of the subject that was imaged
`isSimulation`	Int8	1		no	Flag indicating if the data in this file is simulated rather than measured

2.3 Tracer Parameters (group: `/tracer/`, optional)

Remarks: The tracer parameter group contains information about the MPI tracers used during the experiment. For each tracer its `name`, `batch`, `vendor`, its `volume`, its molar `concentration` of `solute` per liter and the time point of injection can be provided.

This version of the MDF can handle two basic scenarios. In the first one static tracer phantoms are used. In this case the phantom contains A distinct tracers. These might be particles of different core sizes, mobile or immobilized particles for example. In this case `injectionTime` is not used. In the second case A boli (e.g. pulsed boli) are administrated during the measurement, in which case the approximate administration volume, tracer type and time point of injection can be provided. Note that the injection clock recording the injection time should be synchronized with the clock, which provides the starting time of the measurement.

In case of a background measurement with no applied tracers in the scanner, the tracer group should be removed. Therefore, it is optional

Parameter	Type	Dim	Unit/Format	Optional	Description
`name`	`String`	A		no	Name of tracer used in experiment
`batch`	`String`	A		no	Batch of tracer
`vendor`	`String`	A		no	Name of tracer supplier
`volume`	`Float64`	A	L	no	Total volume of applied tracer
`concentration`	`Float64`	A	mol(solute)/L	no	Molar concentration of solute per litre
`solute`	`String`	A		no	Solute, e.g. Fe
`injectionTime`	`String`	A	yyyy-mm-ddThh:mm:ss.ms	yes	UTC time at which tracer injection started

2.4 Scanner Parameters (group: `/scanner/`, non-optional)

Remarks: The scanner parameter group contains information about the MPI scanner used, such as the manufacturer, the model, bore size, the field topology, 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		yes	Scanner no
`topology`	`String`	1		no	Scanner topology (e.g. FFP, FFL, MPS)
`boreSize`	`Float64`	1		yes	Scanner model

2.5 Acquisition Parameters (group: `/acquisition/`, non-optional)

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

In MPI a frame groups together all data used to reconstruct a single MPI image/tomogram. In the simplest scenario this data is aquired during one drive field period. If averaging is applied this time increases by `receiver/numAverages`. In a multi patch setting J `offsetFields` or mechanical movements shift the gradient field by `offsetFieldShift` to different spatial positions, were J is the number of patches of a multi patch measurement. At each positions at least one full drive field cycle is used to aquire measurement data. As such a frame may consist of J sub-measurements, each of which is acquired and averaged over `receiver/numAverages` drive field cycles. For instance a Cartesian 2D trajectory with 100 lines would be realized by setting `numPatches` = 100.

Parameter	Type	Dim	Unit/Format	Optional	Description
`startTime`	String	1	yyyy-mm-ddThh:mm:ss.ms	no	UTC start time of MPI measurement
`framePeriod`	Float64	1	s	no	Complete time to acquire data of a full frame (product of drive field `period`, `numPeriods`, `numAverages`, and `numPatches`)
`numPatches`	Integer	1	1	no	Number of patches within a frame denoted by J
`numPeriods`	Integer	1	1	no	Number of drive-field periods per patch
`numAverages`	Integer	1	1	no	Number of block averages per patch
`numFrames`	Integer	1	1	no	Number of available measurement frames N
`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
`offsetFieldShift`	Float64	$J \times 3$	m	yes	Position of the field free point (relative to origin/center)

2.5.1 Drive Field (group: `/acquisition/drivefield/`, non-optional)

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

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 generated drive-fields of channel d can be 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`. it can be set to *sine*, *triangle* or *custom*. If set to *custom*, one can specify a custom waveform using the parameter `customWaveform`. The number of sampling points of the `customWaveform` is denoted by U. 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	$J \times D \times F$	rad	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`	Integer	$D \times F$	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. It is determined by the base frequency and the frequency divider.

2.5.2 Receiver (group: `/acquisition/receiver/`, non-optional)

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 some quantity related to the magnetization dynamic. In most cases these will be a voltage signals induced into the C receive coils, which are proportional to the change of the particle magnetization.

The MPI signal is acquired at V equidistant time points. Note that in most cases the voltages are not measured directly at the receive coils but amplified and filtered first. To be able to compensate these changes the transfer function can optionally be stored in the parameter `transferFunction`. It is stored in frequency space representation where $L = \frac{V}{2} + 1$ is the number of discrete frequency components.

Parameter	Type	Dim	Unit/Format	Optional	Description
`numChannels`	Int64	1		no	Number of receive channels C
`bandwidth`	Float64	1	Hz	no	Bandwidth of the receiver unit
`numSamplingPoints`	Int64	1		no	Number of sampling points within a cycle over all `numPeriods` drive-field periods denoted by V
`transferFunction`	Float64	$C \times L \times 2$		yes	Transfer function of the receive channel

2.6 Measurement (group: `/measurement/`, optional)

Remarks: MPI data is usually acquired by a series of measurements and optional 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. Initially, all data is available in time domain, where the data of a single frame consists of the signal recorded for all patches in each receive channel, i.e. $J \times C \times V$ data points per set with the temporal index being the fastest to access. If several measurements are acquired (indicated by *numFrames*), the frame dimension is the slowest to access. Along this dimension the frames are ordered with respect to the time at which they were acquired starting with the measurement acquired first and stopping with the measurement acquired last. We refer to this data as raw measurement data. In Fourier representation each frame would be stored by $J \times C \times K \times 2$ data points with the last dimension accounting for the complex data and $K = \frac{V}{2} + 1$.

Often it is not convenient to store the raw data but to perform certain processing steps and store the processed data. These steps may lead to a reduction of the number of sampling points from V to W and/or to a reduction of frequency components from L to K depending on the final representation in which the raw/processed data is stored. The most common processing steps are:

1. spectral leakage correction which may be applied to ensure that each individual frame is periodic.
2. background correction, where the background signal in subtracted.
3. Fourier transformation bringing the data from time into the Fourier representation.
4. transfer function correction to obtain the voltage as it is measured directly at the receive coil.
5. frequency selection to reduce the number of frequency components, e.g. bandwidth reduction or selection of high signal frequency components.
6. frame permutation to reorder the frames within the data set.
7. data transposition which is usually applied to Fourier transformed data exchanges the storing order of the data for fast access to the frames.

For each of the steps above there is a corresponding flag within this group indicating if the corresponding processing step has been carried out.

During processing one might want to keep track which of the final N frames belong to background measurements and which do not. Therefore, the binary mask `isBackgroundFrame` can be used. If `isBackgroundFrame` is not provided it is assumed that no background measurements are present. If frequency selection has been performed `frequencySelection` stores the

K frequency components (subset) selected from the set of acquired frequency components $\{1, 2, \ldots L\}$ if a frequency selection has been performed. Frame permutation if performed can be described by a permutation, i.e. a bijective mapping $\sigma : \{1, 2, \ldots, N\} \to \{1, 2, \ldots, N\}$ of the set of frame indices to itself. If such a permutation is performed σ is stored in the one-line notation as $\sigma(1)$, $\sigma(2)$, ..., $\sigma(N)$ in `framePermutation`.

During measurements the analog signal measured is usually converted into $(r_1, \ldots, r_{JV}) \in \mathbb{Z}^J \times \mathbb{Z}^V$ 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. Note that these factors are also used to map (unsigned) integers to a floating point range.

Parameter	Type	Dim	Optional	Description
`unit`	String	1	no	SI unit of the measured quantity, usually V
`dataConversionFactor`	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	$N \times J \times C \times K \times 2$ or $J \times C \times K \times N \times 2$ or $N \times J \times C \times W$ or $J \times C \times W \times N$	no	Processed data
`isSpectralLeakageCorrected`	Int8	1	no	Flag, if spectral leakage correction has been applied
`isBackgroundCorrected`	Int8	1	no	Flag, if a background has been corrected
`isFourierTransformed`	Int8	1	no	Flag, if the data is stored in frequency space
`isTransferFunctionCorrected`	Int8	1	no	Flag, if the transfer function has been corrected
`isFrequencySelection`	Int8	1	no	Flag, if frequencies have been selected
`isFramePermution`	Int8	1	no	Flag, if the order of frames have been changes
`isTransposed`	Int8	1	no	Flag, if Dimension N has been moved
`isBackgroundFrame`	Int8	N	yes	Mask indicating for each of the N frames if it is a background measurement (true) or not
`frequencySelection`	Integer	K	`isFrequencySelection`	Indices of selected frequency components
`framePermutation`	Integer	N	`isFramePermutation`	Permutation performed

2.7 Calibration (group: `/calibration/`, optional)

Remarks: The calibration group describes a calibration experiment. Each of the raw measurements is taken with a calibration sample (delta sample) at a fixed position inside the FOV of the scanner and each of the raw background measurement is taken with the delta sample outside of the FOV of the scanner. Usually, the calibration measurements are not stored as raw measurements but as processed data, where at least averaging, Fourier transformation, frame permutation and Transposition of the data has been performed yielding O processed calibration frames. Which steps have been performed can be documented in the `/processing/` group.

If the measurements were taken on a regular grid of size $N_x \times N_y \times N_z$ the permutation is usually done such that 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. Background measurements are collected at at the end in `/measurement/data`, which in combination with reordering of the measurements allows fast access to the system matrix. If a different storage order is used this can be documented using the optional parameter `order`. For non-regular sampling points there is the possibility to explicitly store all O positions.

Parameter	Type	Dim	Unit/Format	Optional	Description
`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`	`Integer`	3		yes	Number of voxels in each dimension
`order`	`String`	1		yes	Ordering of the dimensions, default is *xyz*
`positions`	`Float64`	$O \times 3$	m	yes	Position of each of the grid points, stored as (x, y, z) triples
`offsetField`	`Float64`	$O \times 3$	$\mathrm{T}\mu_0^{-1}$	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		no	Method used to obtain calibration data. Can for instance be robot, hybrid, or simulation

2.8 Reconstruction Results (group: `/reconstruction/`, optional)

Reconstruction results are stored in the parameter `data` in a $Q \times P \times S$ array, where Q denotes the number of reconstructed frames within the data set, P denotes the number of voxels and S the number of multispectral channels. If no multispectral reconstruction is performed then one may set $S = 1$. Depending on the reconstruction the grid of the reconstruction data can be different from the system matrix grid. Hence, grid parameters are mirrored in the `/reconstruction/` group.

For analysis of the MPI tomograms it is often required to know which parts of the reconstructed tomogram where covered by the trajectory of the field free region and which parts where not. In MPI one refers to the later region not covered by the trajectory as overscan region. Therefore, the optional field `isOverscanRegion` can store for each voxel if it lies within the overscan region or not. If no voxel lies within the overscan region `isOverscanRegion` may be omitted.

Parameter	Type	Dim	Unit/Format	Optional	Description
`data`	Number	$Q \times P \times S$		no	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 scanner origin/center)
`size`	Integer	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
`isOverscanRegion`	Int8	P		yes	mask indicating for each voxel if it lies in the overscan region (true) or not

3 Changelog

3.1 v2.0.0

- Updated Affiliations in the MDF specification.
- Made extensive improvements to the descriptions of fields and groups.
- 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.
- Specified supported data types.
- Added table of all parameters in the MDF.
- Added section describing the possibility to add custom fields to MDF files.
- Added description for optional and non optional groups and conditional optional and non-optional data sets.
- Rename `/date` to `/time` for consistency reasons.
- Remove `/study/reference` since this functionality is now covered by the integrated background measurements.
- Rename `/study/simulation` to `/study/isSimulation` for consistency reasons and changed type to `Int8`.
- Created new group `/experiment` with fields `/experiment/name`, `/experiment/number` and `/experiment/description` to be able to provide more fine grained information on study and experiment.
- Moved `/study/subject` and `/study/isSimulation` to `/experiment/subject` and `/experiment/isSimulation`.
- Renamed `/tracer/time` to `/tracer/injectionTime`.
- Added the possibility store the tracer concentration also for non iron based tracer materials by adding the `/tracer/solute` field and redefining the field `tracer/concentration`.
- Renamed field `/tracer/time` to `/tracer/injectionTime` to be more specific.
- Added the dimension A to all fields of the `tracer` group to be able to describe settings where multiple tracers are used or tracers are administered multiple times.
- Added field `/scanner/boreSize` to describe the scanner.
- Set all fields but `topology` in the `/scanner` group to be optional.
- Rename `/acquisition/time` to `/acquisition/startTime`.
- `/acquisition/gradient` is now optional since it is now mandatory for MPS measurements.
- Added field `/acquisition/numBackgroungFrames` as counter for the number of measurements used for background subtraction.
- Added `/acquisition/offsetField` and `/acquisition/offsetFieldShift` to describe homogeneous offset fields.
- Support for triangle wave forms and fully arbitrary excitation waveforms has been added by adding the fields `/drivefield/phase`, `/drivefield/customWaveform`, `/drivefield/waveform`.
- Support for multiple excitation frequencies on a drive-field channel has been added by intoducing a new dimension F to the fields `/drivefield/strength`, `/drivefield/phase`, `/drivefield/customWaveform`, `/drivefield/waveform`, and `/drivefield/divider`.

- Removed `fieldOfView` and `fieldOfViewCenter` from `/drivefield` group. `/acquisition/offsetField` and `/acquisition/offsetFieldShift` replace `fieldOfViewCenter`. The `fieldOfView` can be derived from the gradient strength and drive field strength.
- Moved `/drivefield/averages` to `/receiver/numAverages`
- Remove `/acquisition/receiver/frequencies` since it can be directly derived from `/acquisition/receiver/bandwidth` and `/acquisition/receiver/numSamplingPoints`.
- **TODO Tobi: record changes to `/measurement` group here.** 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 P was added to this field.
- **TODO Tobi: record changes to `/calibration` group here.**
- Added possibility to mark the overscan region.
- Added new section changelog to the MDF documentation to record the development of the MDF.

3.2 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.

3.3 v1.0.4

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

3.4 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.

3.5 v1.0.2

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

3.6 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] Paul J Leach, Michael Mealling, and Rich Salz. A universally unique identifier (uuid) urn namespace. 2005.

[3] B. Gleich and J. Weizenecker. Tomographic imaging using the nonlinear response of magnetic particles. *Nature*, 435(7046):1214–1217, June 2005.

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

[5] 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.

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