ISMRMRD

ISMRMRD Working Group

Jun 14, 2023

GENERAL

A prerequisite for sharing magnetic resonance (imaging) reconstruction algorithms and code is a common raw data format. The ISMRMRD project describes such a common raw data format, which attempts to capture the data fields that are required to describe the magnetic resonance experiment with enough detail to reconstruct images. The repository also contains a C/C++ library for working with the format. This standard was developed by a subcommittee of the ISMRM Sedona 2013 workshop and is described in detail in:

Inati SJ, Naegele JD, Zwart NR, Roopchansingh V, Lizak MJ, Hansen DC, Liu CY, Atkinson D, Kellman P, Kozerke S, Xue H, Campbell-Washburn AE, Sørensen TS, Hansen MS. ISMRM Raw data format: A proposed standard for MRI raw datasets. Magn Reson Med. 2017 Jan;77(1):411-421.

Please cite this paper if you use the format.

CHAPTER

ONE

MAGNETIC RESONANCE DATA (MRD) FORMAT

The Magnetic Resonance Data (MRD) format is a vendor neutral standard for describing data from MR acquisitions and reconstructions. It consists of 4 major components:

1. An *MRD header* containing general metadata describing the acquisition, including MR system details and k-space sampling. The header contains a small set of mandatory parameters common to all MR acquisitions, but is extensible to parameters for specialized acquisitions such as b-values, venc, magnetization preparation durations, etc. The MRD header is in XML format and described by an XML schema file.
2. *Raw k-space data* is stored as individual readout acquisitions. Each readout contains the *complex raw data* for all channels, a fixed *AcquisitionHeader* for metadata including *encoding loop counters*, and optionally corresponding *k-space trajectory* information. Most datasets will be comprised of many acquisitions, each stored individually with its own AcquisitionHeader, optional trajectory, and raw data.
3. *Image data* is stored as either sets of 2D or 3D arrays with a fixed *ImageHeader* of common properties and an extensible set of image *MetaAttributes*. Images can be organized into series of common types and multi-channel data is supported for non-coil-combined images.
4. Physiological data such as electrocardiograms, pulse oximetry, or external triggering sources are stored as individual *waveforms* along with a fixed *WaveformHeader* for metadata.

CHAPTER

TWO

MRD FILE FORMAT

MRD data can be stored in various formats, but the HDF5 format is commonly used due to its good compatibility across programming languages and platforms. HDF5 is a hierarchical data format (much like a file system), which can contain multiple variable organized in groups (like folders in a file system). The variables can contain arrays of data values, custom defined structs, or simple text fields. Interface libraries are provided for C++, Python, and MATLAB to simplify usage. MRD HDF5 files can also be opened using standard HDF tools such as HDFView or HDF5 packages such as h5py for Python or the built-in h5read and associated functions in MATLAB.

The extension `.mrd` is used to indicate an HDF5 file containing MRD formatted data as follows:

```
/dataset/xml                text of MRD header
/dataset/data               array of raw data (data + AcquisitionHeader + optional␣
↪trajectory)
/dataset/waveforms          array of waveform (e.g. PMU) data
/dataset/image_0/data       array of image data
/dataset/image_0/header     array of ImageHeaders
/dataset/image_0/attributes array of image MetaAttributes (xml text)
/dataset/config             text of configuration parameters for reconstruction or␣
↪image analysis (optional)
/dataset/config_file        file name of configuration parameters for reconstruction or␣
↪image analysis (optional)
```

All data from a complete acquisition are stored in a group (`dataset` in the above example). An MRD file may contain multiple acquisitions in separate groups, usually in the case of related or dependent acquisitions.

2.1 Reading MRD data in Python

The ismrmrd-python library provides a convenient interface for working with MRD files. It can either be compiled from source or installed from a pip package using the command `pip install ismrmrd`. The following code shows an example of getting the number of readout lines from a dataset and reading the first line of k-space data:

```
>>> import ismrmrd
>>> dset = ismrmrd.Dataset('data.mrd')
>>> nacq = dset.number_of_acquisitions()
>>> acq = dset.read_acquisition(0)
>>> dset.close()
>>> print(acq.getHead())
version: 1
flags: 0
measurement_uid: 281
```

(continues on next page)

(continued from previous page)

```
scan_counter: 3
acquisition_time_stamp: 23187655
physiology_time_stamp: 4304495, 0, 0
number_of_samples: 256
available_channels: 2
active_channels: 2
channel_mask: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
discard_pre: 0
discard_post: 0
center_sample: 128
encoding_space_ref: 0
trajectory_dimensions: 0
sample_time_us: 15.0
position: 0.0, 0.0, 0.0
read_dir: -0.9999999403953552, 5.960464477539063e-08, 0.0
phase_dir: 5.960464477539063e-08, 0.9999999403953552, 0.0
slice_dir: 0.0, 0.0, 1.0
patient_table_position: 0.0, 0.0, -1374995.0
idx: kspace_encode_step_1: 2
kspace_encode_step_2: 0
average: 0
slice: 0
contrast: 0
phase: 0
repetition: 0
set: 0
segment: 0
user: 0, 0, 0, 0, 0, 64, 0, 0

user_int: 0, 0, 0, 0, 0, 0, 0, 0
user_float: 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0

>>> acq.data.shape
(2, 256)
```

The basic h5py package Python can also be used to read the files:

```
>>> import h5py
>>> import numpy as np
>>> f = h5py.File('data.mrd')
>>> acq = np.array(f['dataset']['data'][0])
>>> { k: acq['head'][k] for k in acq['head'].dtype.fields.keys() }

{'version': array(1, dtype=uint16),
'flags': array(64, dtype=uint64),
'measurement_uid': array(0, dtype=uint32),
'scan_counter': array(0, dtype=uint32),
'acquisition_time_stamp': array(0, dtype=uint32),
'physiology_time_stamp': array([0, 0, 0], dtype=uint32),
'number_of_samples': array(512, dtype=uint16),
'available_channels': array(8, dtype=uint16),
'active_channels': array(8, dtype=uint16),
```

(continues on next page)

(continued from previous page)

```
  'channel_mask': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=uint64),
  'discard_pre': array(0, dtype=uint16),
  'discard_post': array(0, dtype=uint16),
  'center_sample': array(256, dtype=uint16),
  'encoding_space_ref': array(0, dtype=uint16),
  'trajectory_dimensions': array(0, dtype=uint16),
  'sample_time_us': array(5., dtype=float32),
  'position': array([0., 0., 0.], dtype=float32),
  'read_dir': array([0., 0., 0.], dtype=float32),
  'phase_dir': array([0., 0., 0.], dtype=float32),
  'slice_dir': array([0., 0., 0.], dtype=float32),
  'patient_table_position': array([0., 0., 0.], dtype=float32),
  'idx': array((0, 0, 0, 0, 0, 0, 0, 0, 0, [0, 0, 0, 0, 0, 0, 0, 0]),
      dtype=[('kspace_encode_step_1', '<u2'), ('kspace_encode_step_2', '<u2'), ('average
↪', '<u2'), ('slice', '<u2'), ('contrast', '<u2'), ('phase', '<u2'), ('repetition', '<u2
↪'), ('set', '<u2'), ('segment', '<u2'), ('user', '<u2', (8,))]),
  'user_int': array([0, 0, 0, 0, 0, 0, 0, 0], dtype=int32),
  'user_float': array([0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)}
```

2.2 Reading MRD data in MATLAB

A MATLAB package is also provided in this repository to facilitate easy usage of MRD files. To use it, add the `matlab` folder in this repository to the MATLAB path. The following code shows an example of getting the number of readout lines from a dataset and reading the first line of k-space data:

```
>> dset = ismrmrd.Dataset('data.mrd');
>> nacq = dset.getNumberOfAcquisitions()
nacq =
   128
>> acq = dset.readAcquisition(1)
acq =
  Acquisition with properties:

    head: [1×1 ismrmrd.AcquisitionHeader]
    traj: {[0×256 single]}
    data: {[256×2 single]}
>> dset.close();
>> acq.head
ans =
  AcquisitionHeader with properties:

                  version: 1
                    flags: 64
          measurement_uid: 281
             scan_counter: 1
    acquisition_time_stamp: 23187639
     physiology_time_stamp: [3×1 uint32]
         number_of_samples: 256
        available_channels: 2
           active_channels: 2
```

(continues on next page)

(continued from previous page)

```
              channel_mask: [16×1 uint64]
               discard_pre: 0
              discard_post: 0
             center_sample: 128
          encoding_space_ref: 0
       trajectory_dimensions: 0
             sample_time_us: 15
                   position: [3×1 single]
                   read_dir: [3×1 single]
                  phase_dir: [3×1 single]
                  slice_dir: [3×1 single]
     patient_table_position: [3×1 single]
                        idx: [1×1 struct]
                   user_int: [8×1 int32]
                 user_float: [8×1 single]
                      FLAGS: [1×1 struct]
```

MATLAB also provides native HDF5 support which can be used to read the data without an external library:

```
>> data = h5read('data.mrd', '/dataset/data')
data =
  struct with fields:

    head: [1×1 struct]
    traj: {128×1 cell}
    data: {128×1 cell}

>> data.head
ans =
  struct with fields:

                    version: [128×1 uint16]
                      flags: [128×1 uint64]
            measurement_uid: [128×1 uint32]
               scan_counter: [128×1 uint32]
     acquisition_time_stamp: [128×1 uint32]
      physiology_time_stamp: [3×128 uint32]
          number_of_samples: [128×1 uint16]
         available_channels: [128×1 uint16]
            active_channels: [128×1 uint16]
               channel_mask: [16×128 uint64]
                discard_pre: [128×1 uint16]
               discard_post: [128×1 uint16]
              center_sample: [128×1 uint16]
          encoding_space_ref: [128×1 uint16]
       trajectory_dimensions: [128×1 uint16]
             sample_time_us: [128×1 single]
                   position: [3×128 single]
                   read_dir: [3×128 single]
                  phase_dir: [3×128 single]
                  slice_dir: [3×128 single]
     patient_table_position: [3×128 single]
```

(continues on next page)

(continued from previous page)

```
                        idx: [1×1 struct]
                   user_int: [8×128 int32]
                 user_float: [8×128 single]
>>
```

CHAPTER

THREE

MRD HEADER

The flexible data structure is defined by the xml schema definition in schema/ismrmrd.xsd. An example of an XML file for a Cartesian 3D acquisition can be found schema/ismrmrd_example.xml.

The most critical elements for image reconstruction are contained in the <encoding> section of the document, which describes the encoded spaced and also the target reconstructed space. Along with the <encodingLimits>, this section allows the reconstruction program to determine matrix sizes, oversampling factors, partial Fourier, etc. In the example above, data is acquired with two-fold oversampling in the read-out (x) direction, which is reflected in the larger matrix size in the encoded space compared to the reconstruction space. The field of view is also twice as large in the encoded space. For the first phase encoding dimension (y), we have a combination of oversampling (20%), reduced phase resolution (only 83 lines of k-space acquired, and partial Fourier sampling, which is reflected in the asymmetric center of the encoding limits of the <kspace_encoding_step_1>. Specifically, the data lines would be placed into the encoding space like this::

```
0                                    70                                    139
|------------------------------------|-------------------------------------|
                      ******************************************************
                      ^                ^                                    ^
                      0               28                                   83
```

After FFT, only the central 116 lines are kept, i.e. there is a reduced field of view in the phase encoding direction. Center and encoding limits for the readout dimension is not given in the XML header. This is to accommodate sequences where the center of the readout may change from readout to readout (alternating directions of readout). There is a field on the individual data headers (see below) to indicate the center of the readout.

An experiment can have multiple encoding spaces and it is possible to indicate on each acquired data readout, which encoding space the data belongs to (see below).

In addition to the defined field in the xml header, it is possible to add an arbitrary number of user defined parameters to accommodate special sequence parameters. Please consult the xml schema to see how user parameters are defined. Briefly, the XML header can have a section at the end which looks like:

```
<userParameters>
  <userParameterLong>
    <name>MyVar1</name>
    <value>1003</value>
  </userParameterLong>
  <userParameterLong>
    <name>MyVar2</name>
    <value>1999</value>
  </userParameterLong>
  <userParameterDouble>
    <name>MyDoubleVar</name>
```

(continues on next page)

(continued from previous page)

```
    <value>87.6676</value>
  </userParameterDouble>
</userParameters>
```

CHAPTER

FOUR

RAW ACQUISITION DATA

Raw k-space data is stored in MRD format as individual readout acquisitions. Each readout contains the complex raw data for all channels, a fixed AcquisitionHeader, and optionally corresponding k-space trajectory information. Most datasets will be comprised of many acquisitions, each stored indvidiually with its own *AcquisitionHeader*, optional *trajectory*, and *raw data*.

4.1 AcquisitionHeader

An MRD AcquisitionHeader accompanies each readout containing metadata common to most data. It is of a fixed size and thus fields cannot be added, removed, or otherwise repurposed. It contains the following information:

Field	Description	Type	Offset
version	Major version number (currently 1)	uint16	0 bytes
flags	A bit mask of common attributes applicable to individual acquisition readouts	uint64	2 bytes
measurement_uid	Unique ID corresponding to the readout	uint32	10 bytes
scan_counter	Zero-indexed incrementing counter for readouts	uint32	14 bytes
acquisition_time_stamp	Clock time stamp (e.g. milliseconds since midnight)	uint32	18 bytes
physiology_time_stamp	Time stamps relative to physiological triggering, e.g. ECG, pulse oximetry, respiratory. Multiplicity defined by ISMRMRD_PHYS_STAMPS (currently 3)	uint32 (x3)	22 bytes
number_of_samples	Number of digitized readout samples	uint16	34 bytes
available_channels	Number of possible receiver coils (channels)	uint16	36 bytes
active_channels	Number of active receiver coils	uint16	38 bytes
channel_mask	Bit mask indicating active coils (64*16 = 1024 bits)	uint64 (x16)	40 bytes
discard_pre	Number of readout samples to be discarded at the beginning (e.g. if the ADC is active during gradient events)	uint16	168 bytes
discard_post	Number of readout samples to be discarded at the end (e.g. if the ADC is active during gradient events)	uint16	170 bytes
center_sample	Index of the readout sample corresponing to k-space center (zero indexed)	uint16	172 bytes
encoding_space_ref	Indexed reference to the encoding spaces enumerated in the MRD (xml) header	uint16	174 bytes
trajectory_dimensions	Dimensionality of the k-space trajectory vector (e.g. 2 for 2D radial (kx, ky), 0 for no trajectory data)	uint16	176 bytes
sample_time_us	Readout bandwidth, as time between samples in microseconds	float (32 bit)	178 bytes
position	Center of the excited volume, in (left, posterior, superior) (LPS) coordinates relative to isocenter in millimeters	float (32 bit) (x3)	182 bytes
read_dir	Directional cosine of readout/frequency encoding	float (32 bit) (x3)	194 bytes
phase_dir	Directional cosine of phase encoding (2D)	float (32 bit) (x3)	206 bytes
slice_dir	Directional cosine of slice normal, i.e. cross-product of read_dir and phase_dir	float (32 bit) (x3)	218 bytes
patient_table_position	Offset position of the patient table, in LPS coordinates	float (32 bit) (x3)	230 bytes
idx	Encoding loop counters, as defined *below*	uint16 (x17)	242 bytes
user_int	User-defined integer parameters, multiplicity defined by ISMRMRD_USER_INTS (currently 8)	int32 (x8)	276 bytes
user_float	User-defined float parameters, multiplicity defined by ISMRMRD_USER_FLOATS (currently 8)	float (32 bit) (x8)	308 bytes
	Total	**340 bytes**	

A reference implementation for serialization/deserialization of the AcquisitionHeader can be found in serialization.cpp.

4.1.1 MRD EncodingCounters

MR acquisitions often loop through a set of counters (e.g. phase encodes) in a complete experiment. The following encoding counters are referred to by the `idx` field in the AcquisitionHeader.

Field	Format	Description	Type	Offset
kspace_encode_step_1	uint16	Phase encoding line	uint16	0 bytes
kspace_encode_step_2	uint16	Partition encoding	uint16	2 bytes
average	uint16	Signal average	uint16	4 bytes
slice	uint16	Slice number (multi-slice 2D)	uint16	6 bytes
contrast	uint16	Echo number in multi-echo	uint16	8 bytes
phase	uint16	Cardiac phase	uint16	10 bytes
repetition	uint16	Counter in repeated/dynamic acquisitions	uint16	12 bytes
set	uint16	Sets of different preparation, e.g. flow encoding, diffusion weighting	uint16	14 bytes
segment	uint16	Counter for segmented acquisitions	uint16	16 bytes
user	uint16 (x8)	User defined counters, multiplicity defined by ISMRMRD_USER_INTS (currently 8)	uint16 (x8)	18 bytes
		Total	**34 bytes**	

A reference implementation for serialization/deserialization of the EncodingCounters can be found in serialization.cpp.

4.1.2 MRD AcquisitionFlags

The `flags` field in the AcquisitionHeader is a 64 bit mask that can be used to indicate specific attributes of the corresponding readout. One usage of these flags is to trigger the processing of data when a condition is met, e.g. the last readout for the current slice. The following flags are defined in the `ISMRMRD_AcquisitionFlags` enum of ismrmrd.h:

```
    ISMRMRD_ACQ_FIRST_IN_ENCODE_STEP1             =  1,
    ISMRMRD_ACQ_LAST_IN_ENCODE_STEP1              =  2,
    ISMRMRD_ACQ_FIRST_IN_ENCODE_STEP2             =  3,
    ISMRMRD_ACQ_LAST_IN_ENCODE_STEP2              =  4,
    ISMRMRD_ACQ_FIRST_IN_AVERAGE                  =  5,
    ISMRMRD_ACQ_LAST_IN_AVERAGE                   =  6,
    ISMRMRD_ACQ_FIRST_IN_SLICE                    =  7,
    ISMRMRD_ACQ_LAST_IN_SLICE                     =  8,
    ISMRMRD_ACQ_FIRST_IN_CONTRAST                 =  9,
    ISMRMRD_ACQ_LAST_IN_CONTRAST                  = 10,
    ISMRMRD_ACQ_FIRST_IN_PHASE                    = 11,
```

(continues on next page)

(continued from previous page)

```
    ISMRMRD_ACQ_LAST_IN_PHASE                       = 12,
    ISMRMRD_ACQ_FIRST_IN_REPETITION                 = 13,
    ISMRMRD_ACQ_LAST_IN_REPETITION                  = 14,
    ISMRMRD_ACQ_FIRST_IN_SET                        = 15,
    ISMRMRD_ACQ_LAST_IN_SET                         = 16,
    ISMRMRD_ACQ_FIRST_IN_SEGMENT                    = 17,
    ISMRMRD_ACQ_LAST_IN_SEGMENT                     = 18,
    ISMRMRD_ACQ_IS_NOISE_MEASUREMENT                = 19,
    ISMRMRD_ACQ_IS_PARALLEL_CALIBRATION             = 20,
    ISMRMRD_ACQ_IS_PARALLEL_CALIBRATION_AND_IMAGING = 21,
    ISMRMRD_ACQ_IS_REVERSE                          = 22,
    ISMRMRD_ACQ_IS_NAVIGATION_DATA                  = 23,
    ISMRMRD_ACQ_IS_PHASECORR_DATA                   = 24,
    ISMRMRD_ACQ_LAST_IN_MEASUREMENT                 = 25,
    ISMRMRD_ACQ_IS_HPFEEDBACK_DATA                  = 26,
    ISMRMRD_ACQ_IS_DUMMYSCAN_DATA                   = 27,
    ISMRMRD_ACQ_IS_RTFEEDBACK_DATA                  = 28,
    ISMRMRD_ACQ_IS_SURFACECOILCORRECTIONSCAN_DATA   = 29,
    ISMRMRD_ACQ_COMPRESSION1                        = 53,
    ISMRMRD_ACQ_COMPRESSION2                        = 54,
    ISMRMRD_ACQ_COMPRESSION3                        = 55,
    ISMRMRD_ACQ_COMPRESSION4                        = 56,
    ISMRMRD_ACQ_USER1                               = 57,
    ISMRMRD_ACQ_USER2                               = 58,
    ISMRMRD_ACQ_USER3                               = 59,
    ISMRMRD_ACQ_USER4                               = 60,
    ISMRMRD_ACQ_USER5                               = 61,
    ISMRMRD_ACQ_USER6                               = 62,
    ISMRMRD_ACQ_USER7                               = 63,
    ISMRMRD_ACQ_USER8                               = 64
```

4.2 k-space Trajectory

k-space trajectory information is optionally included with each readout, with dimensionality specified by the `trajectory_dimensions` field in the AcquisitionHeader. Common values are `2` for 2D radial (kx, ky), `3` for 3D radial (kx, ky, kz). Trajectory information is omitted if `trajectory_dimensions` is set to `0`.

Trajectory data is organized by looping through the dimensions first then the samples:

- For 2D trajectory data:
- For 3D trajectory data:

4.3 Raw Data

MR acquisition raw data are stored as complex valued floats. Data from all receiver channels are included in a single readout object. Data is organized by looping through real/imaginary data, samples, then channels:

CHAPTER

FIVE

IMAGE DATA

MRD images are stored as a combination of *image data*, a fixed *ImageHeader* of common properties, and an extensible set of *MetaAttributes*. Images can stores as individual 2D images or 3D volumes and may include multiple channels for individual receiver coils.

5.1 ImageHeader

Field	Description	Type	Offset
version	Major version number (currently 1)	uint16	0 bytes
data_type	Data type of the image data, e.g. short, float, complex float, etc., as defined in *MRD Image Data Types*	uint16	2 bytes
flags	A bit mask of common attributes applicable to individual images	uint64	4 bytes
measurement_uid	Unique ID corresponding to the image	uint32	12 bytes
matrix_size	Number of pixels in each of the 3 dimensions in the image	uint16 (x3)	16 bytes
field_of_view	Physical size (in mm) in each of the 3 dimensions in the image	float (32 bit) (x3)	22 bytes
channels	Number of receiver channels in image data (stored in the 4th dimension)	uint16	34 bytes
position	Center of the excited volume, in (left, posterior, superior) (LPS) coordinates relative to isocenter in millimeters. NB this is different than DICOM's ImageOrientationPatient, which defines the center of the first (typically top-left) voxel.	float (32 bit) (x3)	36 bytes
read_dir	Directional cosine of readout/frequency encoding. If the image is flipped or rotated to bring them into standard DICOM orientation, ***this field still corresponds to the acquisition readout/frequency direction***, but the `ImageRowDir` must be set in the MetaAttributes.	float (32 bit) (x3)	48 bytes
phase_dir	Directional cosine of phase encoding (2D). If the image is flipped or rotated to bring them into standard DICOM orientation, ***this field still corresponds to the 2D phase encoding direction***, but the `ImageColumnDir` must be set in the MetaAttributes.	float (32 bit) (x3)	60 bytes
slice_dir	For 3D data, the directional cosine of 3D phase encoding direction. For 3D data, the slice normal, i.e. cross-product of `read_dir` and `phase_dir`. If the image is flipped or rotated to bring them into standard DICOM orientation, ***this field still corresponds to the 3D phase encoding direction***, but the `ImageSliceDir` must be set in the MetaAttributes.	float (32 bit) (x3)	72 bytes
patient_table_position	Offset position of the patient table, in LPS coordinates	float (32 bit) (x3)	84 bytes
average	Signal average	uint16	96 bytes
slice	Slice number (multi-slice 2D)	uint16	98 bytes
contrast	Echo number in multi-echo	uint16	100 bytes
phase	Cardiac phase	uint16	102 bytes
repetition	Counter in repeated/dynamic acquisitions	uint16	104 bytes
set	Sets of different preparation, e.g. flow encoding, diffusion weighting	uint16	106 bytes
acquisition_time_stamp	Clock time stamp (e.g. milliseconds since midnight)	uint32	108 bytes
physiol-	Time stamps relative to physiological triggering, e.g. ECG, pulse oximetry, respiratory. Multiplicity defined by ISMRMRD_PHYS_STAMPS (currently 3)	uint32 (x3)	112 bytes

A reference implementation for serialization/deserialization of the ImageHeader can be found in serialization.cpp.

5.1.1 Data Types

The `data_type` field of the ImageHeader describes the data type and precision of the image data. The following types are supported:

Value	Name	Type	Size
1	MRD_USHORT	uint16_t	2 bytes
2	MRD_SHORT	int16_t	2 bytes
3	MRD_UINT	uint32_t	4 bytes
4	MRD_INT	int32_t	4 bytes
5	MRD_FLOAT	float	4 bytes
6	MRD_DOUBLE	double	8 bytes
7	MRD_CXFLOAT	complex float	2 * 4 bytes
8	MRD_CXDOUBLE	complex double	2 * 8 bytes

5.1.2 Image Types

The `image_type` field of the ImageHeader is an enum describing the image type with the following values:

Value	Name
1	MRD_IMTYPE_MAGNITUDE
2	MRD_IMTYPE_PHASE
3	MRD_IMTYPE_REAL
4	MRD_IMTYPE_IMAG
5	MRD_IMTYPE_COMPLEX
6	MRD_IMTYPE_RGB

A value of 6 is used for 8-bit RGB color images, which have the following settings:

- `image_type` is set to `MRD_IMTYPE_RGB`
- `data_type` is set to `MRD_USHORT`
- `channels` is set to 3, representing the red, green, and blue channels of the RGB image
- image data values are in the range 0-255 (8-bit color depth)

5.2 MetaAttributes

Image metadata can be stored in the extensible MRD MetaContainer format. This is serialized as XML text such as:

```
<ismrmrdMeta>
    <meta>
        <name>DataRole</name>
        <value>Image</value>
        <value>AVE</value>
        <value>NORM</value>
        <value>MAGIR</value>
```

(continues on next page)

(continued from previous page)

```
    </meta>
    <meta>
        <name>ImageNumber</name>
        <value>1</value>
    </meta>
</ismrmrdMeta>
```

A variable number of “meta” elements can be defined, each with a single name and one or more value sub-elements. The following table lists standardized attributes which should be used when appropriate, but custom “meta” elements can also be added.

MRD Element Name	Format	DICOM Tag	Interpretation
DataRole	text array	N/A	Characteristics of the image. A value of "Quantitative" indicates that pixel values in the image are parametric and to be interpreted directly (e.g. T1 values, velocity, etc.). If this role is present, pixel values are not further modified in the ICE chain, e.g. by normalization.
SeriesDescription	text array	SeriesDescription	Brief characteristics of the image. The DICOM SeriesDescription (0008,103E) field is constructed by combining this array of values, delimited by "_" (underscores).
SeriesDescriptionAdditional	text array	SeriesDescription	Brief characteristics of the image. The existing DICOM SeriesDescription (0008,103E) field is appended each string in this array, delimited by "_" (underscores).
ImageComments	text array	ImageComments	Remarks about the image. This array of values is stored in the DICOM ImageComment (0020,4000) field, delimited by "_" (underscores).
ImageType	text array	ImageType	Characteristics of the image. This array of values is appended to the DICOM ImageType (0008,0008) field starting in position 4, delimited by "\" (backslash).
ImageRowDir	double array	N/A	A (1x3) vector in indicating the direction along row dimension. For images reconstructed from raw data and not undergoing any flipping or rotating to bring them into standard DICOM orientation, this value is equivalent to the AcquisitionHeader read_dir field.
ImageColumnDir	double array	N/A	A (1x3) vector in indicating the direction along column dimension. For images reconstructed from raw data and not undergoing any flipping or rotating to bring them into standard DICOM orientation, this value is equivalent to the AcquisitionHeader phase_dir field.
RescaleIntercept	double	RescaleIntercept	Intercept for image pixel values, used in conjunction with RescaleSlope. Pixel values are to be interpreted as: ***value = RescaleSlope*pixelValue + RescaleIntercept***. This value is set in the DICOM RescaleIntercept (0028,1052) field.
RescaleSlope	double	RescaleSlope	Scaling factor for image pixel values, used in conjunction with RescaleIntercept. Pixel values are to be interpreted as: ***value = RescaleSlope*pixelValue + RescaleIntercept***. This value is set in the DICOM RescaleSlope (0028,1053) field.
WindowCenter	long	WindowCenter	The window center in the rendered image, used in conjunction with WindowWidth. If RescaleIntercept and RescaleSlope are defined, WindowCenter and WindowWidth are applied to rescaled values. This value is set in the DICOM WindowCenter (0028,1050) field.
WindowWidth	long	WindowWidth	The window center in the rendered image, used in conjunction with WindowCenter. If RescaleIntercept and RescaleSlope are defined, WindowCenter and WindowWidth are applied to rescaled values. This value is set in the DICOM WindowWidth (0028,1051) field.
LUTFileName	text	PhotometricInterpretation, RedPaletteColorLookupTable, RedPaletteColorLookupTable, RedPaletteColorLookupTable	Path to a color lookup table file to be used for this image. LUT files must be in Siemens .pal format and stored in C:\MedCom\config\MRI\ColorLUT. If a value is provided, the DICOM field PhotometricInterpretation (0028,0004) is set to "PALETTE COLOR"
EchoTime	double	EchoTime	Echo time of the image in ms. This value is set in the DICOM EchoTime (0018,0081) field.
InversionTime	dou-	InversionTime	Inversion time of the image in ms. This value is set in the DICOM

5.3 Image Data

Image data is organized by looping through `matrix_size[0]`, `matrix_size[1]`, `matrix_size[2]`, then `channels`. For example, 2D image data would be formatted as:

CHAPTER

SIX

PHYSIOLOGICAL WAVEFORMS

Physiological monitoring data such as electrocardiograms, pulse oximetry, or external triggering may accompany MR acquisitions. These physiological data are stored in MRD as a combination of a fixed *WaveformHeader* and the *raw physiological waveforms*.

6.1 WaveformHeader

The WaveformHeader contains metadata associated with a set of waveform data and has the following fields:

Field	Description	Type	Offset
version	Version number	uint16_t	0 bytes
flags	Bit field with flags	uint64_t	8 bytes (not 2!)
measurement_uid	Unique ID for this measurement	uint32_t	16 bytes
scan_counter	Number of the acquisition after this waveform	uint32_t	20 bytes
time_stamp	Starting timestamp of this waveform	uint32_t	24 bytes
number_of_samples	Number of samples acquired	uint16_t	28 bytes
channels	Active channels	uint16_t	30 bytes
sample_time_us	Time between samples in microseconds	float	32 bytes
waveform_id	*ID matching types specified in XML header*	uint16_t	36 bytes
	Total	**40 bytes (2 bytes padding at the end!)**	

A reference implementation for serialization/deserialization of the WaveformHeader can be found in serialization.cpp.

6.1.1 Waveform IDs

The `waveform_id` field in the WaveformHeader describes the type of physiological data stored. The following ID numbers are standardized:

Value	Name
0	ECG
1	Pulse Oximetry
2	Respiratory
3	External Waveform 1
4	External Waveform 2

For each type of `waveform_id` included in the dataset, a corresponding `WaveformInformation` entry is found in the MRD header to describe the data interpretation. For example:

```
<waveformInformation>
    <waveformId>0</waveformName>
    <waveformName>ECG1</waveformName>
    <waveformTriggerChannel>4</waveformTriggerChannel>
</waveformInformation>
```

Physiological data used for triggering may have an associated "trigger" channel as detected by the MR system. The `waveformTriggerChannel` indicates the channel index (0-indexed) which contains the detected triggers and is omitted if no trigger data is present.

Waveform ID numbers less than 1024 are reserved while numbers greater than or equal to 1024 can be used to define custom physiological data. For custom waveform_ids, corresponding WaveformInformation entries should be added to the MRD header to describe the data interpretation. For example:

```
<waveformInformation>
    <waveformId>1024</waveformName>
    <waveformName>CustomName</waveformName>
</waveformInformation>
```

6.2 Waveform Data

Waveform data is sent as an uint32_t array, ordered by looping through samples and then through channels:

CHAPTER

SEVEN

SESSION PROTOCOL

The MR Data (MRD) streaming protocol describes the communication of data (*k-space*, *image*, or *waveform*) between a client and a server pair. It consists of a series of *messages* that are sent through a TCP/IP socket in sessions with the protocol defined as follow:

1. The server is started and listens for incoming connections on a designated TCP port (9002 by default). A client initiates a session by connecting to the TCP port above.
2. The client sends a configuration message to indicate analysis that should be performed by the server. The message may be either (but not both):
   - *MRD_MESSAGE_CONFIG_FILE*, corresponding to the name of a config file that exists on the server
   - *MRD_MESSAGE_CONFIG_TEXT*, configuration parameters for the server, in XML text format.
3. The client sends the MRD acquisition parameter header, *MRD_MESSAGE_HEADER*, containing information pertaining to the entire acquisition. This information is sent as XML formatted text that conforms to a standardized ismrmrd.xsd schema.
4. The client sends raw k-space, image, waveform or text data. Not all types of data may be sent, depending on the analysis pipeline to be performed. For example, image processing pipelines may not contain k-space or waveform data. Data of each type must sent in order by acquisition time, but the order between messages of different types is not guaranteed. For eample, a waveform message corresponding to a specific time may be sent before or after the raw k-space data from that time. The data types are:
   - *MRD_MESSAGE_ACQUISITION*: Data from a single k-space readout, including a fixed raw data *AcquisitionHeader* and optional k-space trajectory information.
   - *MRD_MESSAGE_IMAGE*: Image data as a 2D or 3D array, including both a fixed image data *ImageHeader* and a flexible set of image *MetaAttributes* formatted as an XML text string.
   - *MRD_MESSAGE_WAVEFORM*: Waveform data such as physiological monitoring (ECG, pulse oximeter, respiratory motion) including a fixed waveform *WaveformHeader*.
   - *MRD_MESSAGE_TEXT*: Informational text for the other party. This text may provide logging information about the status of the analysis or client, but is optional and should not be used for workflow control.
5. At any point after a config message is received, the server may send back raw k-space, image, waveform, or text data as described above.
6. When all data has been sent, the client sends *MRD_MESSAGE_CLOSE* to indicate that no further data will be sent by the client.
7. When the server has sent all data from its side, it also sends *MRD_MESSAGE_CLOSE*. This usually occurs after the client has sent MRD_MESSAGE_CLOSE, but can also occur if the server encounters an unrecoverable error and no further data can be processed.
8. The TCP session may be closed by either side and the MRD streaming session is complete.

CHAPTER

EIGHT

MESSAGE TYPES

8.1 ID 1: MRD_MESSAGE_CONFIG_FILE

ID	Config File Name
2 bytes	1024 bytes
unsigned short	char

This message type is used to send the file name of a configuration file (local on the server file system) to be used during reconstruction. The file name must not exceed 1023 characters and is formatted as a null-terminated, UTF-8 encoded char string.

8.2 ID 2: MRD_MESSAGE_CONFIG_TEXT

ID	Length	Config Text
2 bytes	4 bytes	length * 1 byte
unsigned short	uint32_t	char

Alternatively, the text contents of a configuration file can be sent directly via the data stream. The length is sent as an uint32_t. Configuration text is sent as a UTF-8 encoded char string.

In addition to specifying a configuration "preset" to be executed on the server, it is often desirable to modify specific parameters of the configuration, such as filter strength or the toggling of intermediate outputs for debugging purposes. While individual parameters are specific to a given pipeline and server, the format of this configuration is standardized to enable interoperable communications between various clients and servers.

8.3 ID 3: MRD_MESSAGE_HEADER

ID	Length	XML Header Text
2 bytes	4 bytes	length * 1 byte
unsigned short	uint32_t	char

Metadata for MRD datasets are stored in a flexible XML formatted *MRD header*. The header length is sent as an uint32_t and the text is sent as a UTF-8 encoded char string.

8.4 ID 4: MRD_MESSAGE_CLOSE

ID
2 bytes
unsigned short

This message type consists only of an ID with no following data. It is used to indicate that all data related to an acquisition/reconstruction has been sent. The client will send this message after sending the last data (raw, image, or waveform) message. The server will also send this message after sending the last data (raw, image, or waveform) message back to the client.

8.5 ID 5: MRD_MESSAGE_TEXT

ID	Length	Text
2 bytes	4 bytes	length * 1 byte
unsigned short	uint32_t	char

Informational (logging) text can be sent using this message type, typically from the reconstruction side to the acquisition/client side. The length of message text is sent as an uint32_t while the text is sent as a UTF-8 encoded char string. These messages are optional and their timing is not guaranteed.

8.6 ID 1008: MRD_MESSAGE_ACQUISITION

ID	Fixed Raw Data Header	Trajectory	Raw Data
2 bytes	340 bytes	number_of_samples * trajectory_dimensions * 4 bytes	number_of_channels * number_of_samples * 8 bytes
unsigned short	mixed	float	float

This message type is used to send raw (k-space) acquisition data. A separate message is sent for each readout. A fixed *AcquisitionHeader* contains metadata such as *encoding loop counters*. Three fields of the data header must be parsed in order to read the rest of the message:

- **trajectory_dimensions**: defines the number of dimensions in the k-space trajectory data component. For 2D acquisitions (kx, ky), this is set to 2, while for 3D acquisitions (kx, ky, kz), this is set to 3. If set to 0, the trajectory component is omitted.
- **number_of_samples**: number of readout samples.
- **active_channels**: number of channels for which raw data is acquired.

Trajectory data is organized by looping through the dimensions first then the samples:

- For 2D trajectory data:
- For 3D trajectory data:

Raw data is organized by looping through real/imaginary data, samples, then channels:

8.7 ID 1022: MRD_MESSAGE_IMAGE

ID	Fixed Image Header	Attribute Length	Attribute Data	Image Data
2 bytes	198 bytes	8 bytes	length * 1 byte	matrix_size[0] * matrix_size[1] * matrix_size[2] * channels * sizeof(data_type)
unsigned short	mixed	uint_64	char	data_type

Image data is sent using this message type. The fixed image header contains metadata including fields such as the *ImageType* (magnitude, phase, etc.) and indices such as slice and repetition number. It is defined by the *ImageHeader* struct. Within this header, there are 3 fields that must be interpreted to parse the rest of the message:

- **matrix_size**: This 3 element array indicates the size of each dimension of the image data.
- **channels**: This value indicates the number of (receive) channels for which image data is sent
- **data_type**: This value is an MRD_DataTypes enum that indicates the type of data sent. The following types are supported:

Value	Name	Type	Size
1	MRD_USHORT	uint16_t	2 bytes
2	MRD_SHORT	int16_t	2 bytes
3	MRD_UINT	uint32_t	4 bytes
4	MRD_INT	int32_t	4 bytes
5	MRD_FLOAT	float	4 bytes
6	MRD_DOUBLE	double	8 bytes
7	MRD_CXFLOAT	complex float	2 * 4 bytes
8	MRD_CXDOUBLE	complex double	2 * 8 bytes

Attributes are used to declare additional image metadata that is not present in the fixed image header. In general, this data is sent as a UTF-8 encoded char string (not null-terminated), with the length sent first as an uint_64 (not uint_32!). These are interpreted as an XML formatted set of image *MetaAttributes*.

Image data is organized by looping through `matrix_size[0]`, `matrix_size[1]`, `matrix_size[2]`, then `channels`. For example, 2D image data would be formatted as:

8.8 ID 1026: MRD_MESSAGE_WAVEFORM

ID	Fixed Waveform Header	Waveform Data
2 bytes	40 bytes	channels * number of samples * bytes
unsigned short	mixed	uint32_t

This message type is used to send arbitrary waveform data (e.g. physio signals, gradient waveforms, etc.). The fixed waveform data header contains metadata as defined by the *WaveformHeader*.

The `channels` and `number_of_samples` members fields must be parsed in order to read the rest of the message. Waveform data is sent as an uint32_t array, ordered by looping through samples and then through channels:

CHAPTER

NINE

BUILD INSTRUCTIONS

The ISMRM Raw Data format is described by an XML schema and some C-style structs with fixed memory layout and as such does not have dependencies. However, it uses HDF5 files for storage and a C++ library for reading and writing the ISMRMRD files is included in this distribution. Furthermore, since the XML header is defined with an XML schema, we encourage using XML data binding when writing software using the format. To compile all components of this distribution you need:

- HDF5 (version 1.8 or higher) libraries.
- Boost
- Pugixml
- Cmake build tool
- Git if you would like to use the source code archive
- FFTW if you would like to compile some of the example applications
- Doxygen if you would like to generate API documentation

It is only necessary to install the dependencies if you wish to develop compiled C/C++ software, which uses the ISMRMRD format. The format can be read in Matlab or Python without installing any additional software.

9.1 Linux installation

The dependencies mentioned above should be included in most linux distributions. On Ubuntu you can install all required dependencies with::

```
sudo apt-get -y install doxygen git-core graphviz libboost-all-dev libfftw3-dev libhdf5-
↪serial-dev libxml2-utils libpugixml-dev
```

After installation of dependencies, the library can be installed with:

```
git clone https://github.com/ismrmrd/ismrmrd
cd ismrmrd/
mkdir build
cd build
cmake ../
make
sudo make install
```

This will install the library in `/usr/local/` by default. To specify an alternative installation directory, pass `-D CMAKE_INSTALL_PREFIX=<install dir>` to `cmake`.

9.2 External use of ISMRMRD C++ library in other projects

To use ISMRMRD for your externally developed projects, add the following to your CMakeLists.txt file:

```
find_package( ISMRMRD REQUIRED )
link_directories( ${ISMRMRD_LIBRARY_DIRS} )
include_directories( ${ISMRMRD_INCLUDE_DIRS} )
target_link_libraries( mytarget ${ISMRMRD_LIBRARIES} )
```

then when configuring your package use set the following cmake variables (command line variant shown):

```
cmake -DISMRMRD_DIR:PATH=<path to build/install tree of ISMRMRD> <path to my source tree>
```

CHAPTER

TEN

C++ SUPPORT LIBRARY

To enable easy prototyping of C++ software using the ISMRMRD data format, a simple C++ wrapper class is provided (defined in dataset.h).

Using this wrapper, C++ applications can be programmed as:

```
// Open dataset
ISMRMRD::Dataset d(datafile.c_str(), "dataset", false);

std::string xml;
d.readHeader(xml);
ISMRMRD::IsmrmrdHeader hdr;
ISMRMRD::deserialize(xml.c_str(),hdr);

// Do something with the header

unsigned int number_of_acquisitions = d.getNumberOfAcquisitions();
ISMRMRD::Acquisition acq;
for (unsigned int i = 0; i < number_of_acquisitions; i++) {
    // Read one acquisition at a time
    d.readAcquisition(i, acq);

    // Do something with the data
}
```

Since the XML header is defined in the schema/ismrmrd.xsd file, it can be parsed with numerous xml parsing libraries. The ISMRMRD library includes an API that allows for programmatically deserializing, manipulating, and serializing the XML header. See the code in the utilities directory for examples of how to use the XML API.

CHAPTER

ELEVEN

C++ EXAMPLE APPLICATIONS

The distribution includes two example applications, one that creates a simple 2D single-channel dataset from scratch and one that reconstructs this dataset (you need FFTW installed to compile these test applications). The data generation application can be found in utilities/generate_cartesian_shepp_logan.cpp:

To reconstruct this synthetic dataset, you can use the test reconstruction application utilities/recon_cartesian_2d.cpp.

CHAPTER

TWELVE

EXTERNAL USE OF ISMRMRD C++ LIBRARY IN OTHER PROJECTS

To use ISMRMRD for your externally developed projects, add the following to your CMakeLists.txt file:

```
find_package( ISMRMRD REQUIRED )
include_directories( ${ISMRMRD_INCLUDE_DIR} )
target_link_libraries( mytarget ISMRMRD::ISMRMRD )
```

then when configuring your package use set the following cmake variables (command line variant shown):

```
cmake <path to my source tree>
```

CHAPTER

THIRTEEN

ISMRMRD API

The project includes both a C and A C++ API:

13.1 C API

typedef struct ISMRMRD::*ISMRMRD_Image* **ISMRMRD_Image**

typedef void (***ismrmrd_error_handler_t**)(const char *file, int line, const char *function, int code, const char *msg)

EXPORTISMRMRD int ismrmrd_init_acquisition_header (ISMRMRD_AcquisitionHeader *hdr)

EXPORTISMRMRD ISMRMRD_Acquisition * ismrmrd_create_acquisition ()

EXPORTISMRMRD int ismrmrd_free_acquisition (ISMRMRD_Acquisition *acq)

EXPORTISMRMRD int ismrmrd_init_acquisition (ISMRMRD_Acquisition *acq)

EXPORTISMRMRD int ismrmrd_cleanup_acquisition (ISMRMRD_Acquisition *acq)

EXPORTISMRMRD int ismrmrd_copy_acquisition (ISMRMRD_Acquisition *acqdest,
const ISMRMRD_Acquisition *acqsource)

EXPORTISMRMRD int ismrmrd_make_consistent_acquisition (ISMRMRD_Acquisition *acq)

EXPORTISMRMRD size_t ismrmrd_size_of_acquisition_traj (const ISMRMRD_Acquisition *acq)

EXPORTISMRMRD size_t ismrmrd_size_of_acquisition_data (const ISMRMRD_Acquisition *acq)

EXPORTISMRMRD int ismrmrd_init_image_header (ISMRMRD_ImageHeader *hdr)

EXPORTISMRMRD ISMRMRD_Image * ismrmrd_create_image ()

```
EXPORTISMRMRD int ismrmrd_free_image (ISMRMRD_Image *im)

EXPORTISMRMRD int ismrmrd_init_image (ISMRMRD_Image *im)

EXPORTISMRMRD int ismrmrd_cleanup_image (ISMRMRD_Image *im)

EXPORTISMRMRD int ismrmrd_copy_image (ISMRMRD_Image *imdest,
const ISMRMRD_Image *imsource)

EXPORTISMRMRD int ismrmrd_make_consistent_image (ISMRMRD_Image *im)

EXPORTISMRMRD size_t ismrmrd_size_of_image_attribute_string (const ISMRMRD_Image *im)

EXPORTISMRMRD size_t ismrmrd_size_of_image_data (const ISMRMRD_Image *im)

EXPORTISMRMRD ISMRMRD_NDArray * ismrmrd_create_ndarray ()

EXPORTISMRMRD int ismrmrd_free_ndarray (ISMRMRD_NDArray *arr)

EXPORTISMRMRD int ismrmrd_init_ndarray (ISMRMRD_NDArray *arr)

EXPORTISMRMRD int ismrmrd_cleanup_ndarray (ISMRMRD_NDArray *arr)

EXPORTISMRMRD int ismrmrd_copy_ndarray (ISMRMRD_NDArray *arrdest,
const ISMRMRD_NDArray *arrsource)

EXPORTISMRMRD int ismrmrd_make_consistent_ndarray (ISMRMRD_NDArray *arr)

EXPORTISMRMRD size_t ismrmrd_size_of_ndarray_data (const ISMRMRD_NDArray *arr)

EXPORTISMRMRD bool ismrmrd_is_flag_set (const uint64_t flags, const uint64_t val)

EXPORTISMRMRD int ismrmrd_set_flag (ISMRMRD_UNALIGNED uint64_t *flags,
const uint64_t val)

EXPORTISMRMRD int ismrmrd_set_flags (ISMRMRD_UNALIGNED uint64_t *flags,
const uint64_t val)

EXPORTISMRMRD int ismrmrd_clear_flag (ISMRMRD_UNALIGNED uint64_t *flags,
const uint64_t val)

EXPORTISMRMRD int ismrmrd_clear_all_flags (ISMRMRD_UNALIGNED uint64_t *flags)
```

EXPORTISMRMRD bool ismrmrd_is_channel_on (const uint64_t channel_mask[ISMRMRD_CHANNEL_MASKS], const uint16_t chan)

EXPORTISMRMRD int ismrmrd_set_channel_on (uint64_t channel_mask[ISMRMRD_CHANNEL_MASKS], const uint16_t chan)

EXPORTISMRMRD int ismrmrd_set_channel_off (uint64_t channel_mask[ISMRMRD_CHANNEL_MASKS], const uint16_t chan)

EXPORTISMRMRD int ismrmrd_set_all_channels_off (uint64_t channel_mask[ISMRMRD_CHANNEL_MASKS])

int **ismrmrd_push_error**(const char *file, const int line, const char *func, const int code, const char *msg)

EXPORTISMRMRD void ismrmrd_set_error_handler (ismrmrd_error_handler_t)

EXPORTISMRMRD const char * ismrmrd_strerror (int code)

EXPORTISMRMRD int ismrmrd_sign_of_directions (float const read_dir[3], float const phase_dir[3], float const slice_dir[3])

EXPORTISMRMRD void ismrmrd_directions_to_quaternion (float const read_dir[3], float const phase_dir[3], float const slice_dir[3], float quat[4])

EXPORTISMRMRD void ismrmrd_quaternion_to_directions (float const quat[4], float read_dir[3], float phase_dir[3], float slice_dir[3])

ISMRMRD_PUSH_ERR(code, msg)

struct **ISMRMRD_Image**

Public Members

ISMRMRD_ImageHeader **head**

char ***attribute_string**

void ***data**

13.2 C++ API

bool **operator==**(const EncodingCounters &ec1, const EncodingCounters &ec2)

template<typename T> EXPORTISMRMRD ISMRMRD_DataTypes get_data_type ()

class **FlagBit**

Public Functions

inline **FlagBit**(unsigned short b)

inline bool **isSet**(const uint64_t &m) const

Public Members

uint64_t **bitmask_**

class **AcquisitionHeader** : public ISMRMRD::ISMRMRD_AcquisitionHeader

Public Functions

AcquisitionHeader()

bool **operator==**(const *AcquisitionHeader* &acq) const

bool **isFlagSet**(const ISMRMRD_AcquisitionFlags val) const

void **setFlag**(const ISMRMRD_AcquisitionFlags val)

void **clearFlag**(const ISMRMRD_AcquisitionFlags val)

void **clearAllFlags**()

bool **isChannelActive**(uint16_t channel_id) const

void **setChannelActive**(uint16_t channel_id)

void **setChannelNotActive**(uint16_t channel_id)

void **setAllChannelsNotActive**()

class **Acquisition**

Public Functions

Acquisition()

Acquisition(uint16_t num_samples, uint16_t active_channels = 1, uint16_t trajectory_dimensions = 0)

Acquisition(const *Acquisition* &other)

Acquisition &**operator=**(const *Acquisition* &other)

~Acquisition()

uint16_t **version**() const

uint64_t **flags**() const

uint32_t &**measurement_uid**()

uint32_t &**scan_counter**()

uint32_t &**acquisition_time_stamp**()

uint32_t (&**physiology_time_stamp**())[ISMRMRD_PHYS_STAMPS]

const uint16_t &**number_of_samples**()

uint16_t &**available_channels**()

const uint16_t &**active_channels**()

const uint64_t (&**channel_mask**())[ISMRMRD_CHANNEL_MASKS]

uint16_t &**discard_pre**()

uint16_t &**discard_post**()

uint16_t &**center_sample**()

uint16_t &**encoding_space_ref**()

const uint16_t &**trajectory_dimensions**()

float &**sample_time_us**()

float (&**position**())[3]

float (&**read_dir**())[3]

float (&**phase_dir**())[3]

float (&**slice_dir**())[3]

float (&**patient_table_position**())[3]

ISMRMRD_EncodingCounters &**idx**()

int32_t (&**user_int**())[ISMRMRD_USER_INTS]

float (&**user_float**())[ISMRMRD_USER_FLOATS]

uint32_t **measurement_uid**() const

uint32_t **scan_counter**() const

uint32_t **acquisition_time_stamp**() const

const uint32_t (&**physiology_time_stamp**() const)[ISMRMRD_PHYS_STAMPS]

uint16_t **number_of_samples**() const

uint16_t **available_channels**() const

uint16_t **active_channels**() const

const uint64_t (&**channel_mask**() const)[ISMRMRD_CHANNEL_MASKS]

uint16_t **discard_pre**() const

uint16_t **discard_post**() const

uint16_t **center_sample**() const

uint16_t **encoding_space_ref**() const

uint16_t **trajectory_dimensions**() const

float **sample_time_us**() const

const float (&**position**() const)[3]

const float (&**read_dir**() const)[3]

const float (&**phase_dir**() const)[3]

const float (&**slice_dir**() const)[3]

const float (&**patient_table_position**() const)[3]

const ISMRMRD_EncodingCounters &**idx**() const

const int32_t (&**user_int**() const)[ISMRMRD_USER_INTS]

const float (&**user_float**() const)[ISMRMRD_USER_FLOATS]

void **resize**(uint16_t num_samples, uint16_t active_channels = 1, uint16_t trajectory_dimensions = 0)

size_t **getNumberOfDataElements**() const

size_t **getNumberOfTrajElements**() const

size_t **getDataSize**() const

size_t **getTrajSize**() const

const *AcquisitionHeader* &**getHead**() const

void **setHead**(const *AcquisitionHeader* &other)

const complex_float_t ***getDataPtr**() const

complex_float_t ***getDataPtr**()

complex_float_t &**data**(uint16_t sample, uint16_t channel)

void **setData**(complex_float_t *data)

complex_float_t ***data_begin**()

const complex_float_t ***data_begin**() const

complex_float_t ***data_end**()

const complex_float_t ***data_end**() const

const float ***getTrajPtr**() const

float ***getTrajPtr**()

float &**traj**(uint16_t dimension, uint16_t sample)

void **setTraj**(float *traj)

float ***traj_begin**()

const float ***traj_begin**() const

float ***traj_end**()

const float ***traj_end**() const

bool **isFlagSet**(const uint64_t val) const

void **setFlag**(const uint64_t val)

void **clearFlag**(const uint64_t val)

void **clearAllFlags**()

inline bool **isFlagSet**(const *FlagBit* &val) const

inline void **setFlag**(const *FlagBit* &val)

inline void **clearFlag**(const *FlagBit* &val)

bool **isChannelActive**(uint16_t channel_id) const

void **setChannelActive**(uint16_t channel_id)

void **setChannelNotActive**(uint16_t channel_id)

void **setAllChannelsNotActive**()

Protected Attributes

ISMRMRD_Acquisition **acq**

Friends

friend class Dataset

class **ImageHeader** : public ISMRMRD::ISMRMRD_ImageHeader

Public Functions

ImageHeader()

bool **isFlagSet**(const uint64_t val) const

void **setFlag**(const uint64_t val)

void **clearFlag**(const uint64_t val)

void **clearAllFlags**()

template<typename **T**>

class **Image**

Public Functions

Image(uint16_t matrix_size_x = 0, uint16_t matrix_size_y = 1, uint16_t matrix_size_z = 1, uint16_t channels = 1)

Image(const *Image* &other)

Image &**operator=**(const *Image* &other)

~Image()

void **resize**(uint16_t matrix_size_x, uint16_t matrix_size_y, uint16_t matrix_size_z, uint16_t channels)

uint16_t **getMatrixSizeX**() const

void **setMatrixSizeX**(uint16_t matrix_size_x)

uint16_t **getMatrixSizeY**() const

void **setMatrixSizeY**(uint16_t matrix_size_y)

uint16_t **getMatrixSizeZ**() const

void **setMatrixSizeZ**(uint16_t matrix_size_z)

uint16_t **getNumberOfChannels**() const

void **setNumberOfChannels**(uint16_t channels)

void **setFieldOfView**(float fov_x, float fov_y, float fov_z)

float **getFieldOfViewX**() const

void **setFieldOfViewX**(float f)

float **getFieldOfViewY**() const

void **setFieldOfViewY**(float f)

float **getFieldOfViewZ**() const

void **setFieldOfViewZ**(float f)

void **setPosition**(float x, float y, float z)

float **getPositionX**() const

void **setPositionX**(float x)

float **getPositionY**() const

void **setPositionY**(float y)

float **getPositionZ**() const

void **setPositionZ**(float z)

void **setReadDirection**(float x, float y, float z)

float **getReadDirectionX**() const

void **setReadDirectionX**(float x)

float **getReadDirectionY**() const

void **setReadDirectionY**(float y)

float **getReadDirectionZ**() const

void **setReadDirectionZ**(float z)

void **setPhaseDirection**(float x, float y, float z)

float **getPhaseDirectionX**() const

void **setPhaseDirectionX**(float x)

float **getPhaseDirectionY**() const

void **setPhaseDirectionY**(float y)

float **getPhaseDirectionZ**() const

void **setPhaseDirectionZ**(float z)

void **setSliceDirection**(float x, float y, float z)

float **getSliceDirectionX**() const

void **setSliceDirectionX**(float x)

float **getSliceDirectionY**() const

void **setSliceDirectionY**(float y)

float **getSliceDirectionZ**() const

void **setSliceDirectionZ**(float z)

void **setPatientTablePosition**(float x, float y, float z)

float **getPatientTablePositionX**() const

void **setPatientTablePositionX**(float x)

float **getPatientTablePositionY**() const

void **setPatientTablePositionY**(float y)

float **getPatientTablePositionZ**() const

void **setPatientTablePositionZ**(float z)

uint16_t **getVersion**() const

ISMRMRD_DataTypes **getDataType**() const

uint32_t **getMeasurementUid**() const

void **setMeasurementUid**(uint32_t measurement_uid)

uint16_t **getAverage**() const

void **setAverage**(uint16_t average)

uint16_t **getSlice**() const

void **setSlice**(uint16_t slice)

uint16_t **getContrast**() const

void **setContrast**(uint16_t contrast)

uint16_t **getPhase**() const

void **setPhase**(uint16_t phase)

uint16_t **getRepetition**() const

void **setRepetition**(uint16_t repetition)

uint16_t **getSet**() const

void **setSet**(uint16_t set)

uint32_t **getAcquisitionTimeStamp**() const

void **setAcquisitionTimeStamp**(uint32_t acquisition_time_stamp)

uint32_t **getPhysiologyTimeStamp**(unsigned int stamp_id) const

void **setPhysiologyTimeStamp**(unsigned int stamp_id, uint32_t value)

uint16_t **getImageType**() const

void **setImageType**(uint16_t image_type)

uint16_t **getImageIndex**() const

void **setImageIndex**(uint16_t image_index)

uint16_t **getImageSeriesIndex**() const

void **setImageSeriesIndex**(uint16_t image_series_index)

float **getUserFloat**(unsigned int index) const

void **setUserFloat**(unsigned int index, float value)

int32_t **getUserInt**(unsigned int index) const

void **setUserInt**(unsigned int index, int32_t value)

uint64_t **getFlags**() const

void **setFlags**(const uint64_t flags)

bool **isFlagSet**(const uint64_t val) const

void **setFlag**(const uint64_t val)

void **clearFlag**(const uint64_t val)

void **clearAllFlags**()

ImageHeader &**getHead**()

const *ImageHeader* &**getHead**() const

void **setHead**(const *ImageHeader* &head)

void **getAttributeString**(std::string &attr) const

const char *__getAttributeString__() const

void **setAttributeString**(const std::string &attr)

void **setAttributeString**(const char *attr)

size_t **getAttributeStringLength**() const

T ***getDataPtr**()

const *T* ***getDataPtr**() const

size_t **getNumberOfDataElements**() const

size_t **getDataSize**() const

T ***begin**()

T ***end**()

T &**operator()**(uint16_t x, uint16_t y = 0, uint16_t z = 0, uint16_t channel = 0)

Protected Attributes

ISMRMRD_Image **im**

Friends

friend class Dataset

template<typename **T**>

class **NDArray**

Public Functions

NDArray()

NDArray(const std::vector<size_t> dimvec)

NDArray(const *NDArray*<*T*> &other)

~NDArray()

NDArray<*T*> &**operator=**(const *NDArray*<*T*> &other)

uint16_t **getVersion**() const

ISMRMRD_DataTypes **getDataType**() const

uint16_t **getNDim**() const

const size_t (&**getDims**())[ISMRMRD_NDARRAY_MAXDIM]

size_t **getDataSize**() const

void **resize**(const std::vector<size_t> dimvec)

size_t **getNumberOfElements**() const

T ***getDataPtr**()

const *T* ***getDataPtr**() const

T ***begin**()

T ***end**()

T &**operator()**(uint16_t x, uint16_t y = 0, uint16_t z = 0, uint16_t w = 0, uint16_t n = 0, uint16_t m = 0, uint16_t l = 0)

Protected Attributes

ISMRMRD_NDArray **arr**

Friends

friend class Dataset

13.3 Meta API

EXPORTISMRMRD void deserialize (const char *xml, MetaContainer &h)

EXPORTISMRMRD void serialize (const MetaContainer &h, std::ostream &o)

class **MetaValue**

Public Functions

inline **MetaValue**()

inline **MetaValue**(const char *s)

inline **MetaValue**(long l)

inline **MetaValue**(double d)

inline *MetaValue* &**operator=**(const char *s)

inline *MetaValue* &**operator=**(long l)

inline *MetaValue* &**operator=**(double d)

inline long **as_long**() const

inline double **as_double**() const

inline const char ***as_str**() const

Protected Functions

inline void **set**(const char *s)

inline void **set**(long l)

inline void **set**(double d)

Protected Attributes

long **l_**

double **d_**

std::string **s_**

class **MetaContainer**

to

template<class **T**>
inline void **set**(const char *name, *T* value)

template<class **T**>
inline void **append**(const char *name, *T* value)

inline void **remove**(const char *name)

inline size_t **length**(const char *name) const

as long

inline long **as_long**(const char *name, size_t index = 0) const

as double

inline double **as_double**(const char *name, size_t index = 0) const

as string

map_t **map_**

inline const char ***as_str**(const char *name, size_t index = 0) const

inline const *MetaValue* &**value**(const char *name, size_t index = 0) const

inline bool **empty**() const

inline *map_t*::iterator **begin**()

inline *map_t*::iterator **end**()

inline *map_t*::const_iterator **begin**() const

inline *map_t*::const_iterator **end**() const

Public Functions

inline **MetaContainer**()

Protected Types

typedef std::map<std::string, std::vector<*MetaValue*>> **map_t**

13.4 XML API

enum **TrajectoryType**

Values:

enumerator **CARTESIAN**

enumerator **EPI**

enumerator **RADIAL**

enumerator **GOLDENANGLE**

enumerator **SPIRAL**

enumerator **OTHER**

enum **MultibandCalibrationType**

Values:

enumerator **SEPARABLE2D**

enumerator **FULL3D**

enumerator **OTHER**

enum **DiffusionDimension**

Values:

enumerator **AVERAGE**

enumerator **CONTRAST**

enumerator **PHASE**

enumerator **REPETITION**

enumerator **SET**

enumerator **SEGMENT**

enumerator **USER_0**

enumerator **USER_1**

enumerator **USER_2**

enumerator **USER_3**

enumerator **USER_4**

enumerator **USER_5**

enumerator **USER_6**

enumerator **USER_7**

enum **WaveformType**

Values:

enumerator **ECG**

enumerator **PULSE**

enumerator **RESPIRATORY**

enumerator **TRIGGER**

enumerator **GRADIENTWAVEFORM**

enumerator **OTHER**

EXPORTISMRMRD void deserialize (const char *xml, IsmrmrdHeader &h)

EXPORTISMRMRD void serialize (const IsmrmrdHeader &h, std::ostream &o)

EXPORTISMRMRD std::ostream & operator<< (std::ostream &os, const IsmrmrdHeader &)

```
EXPORTISMRMRD bool operator== (const IsmrmrdHeader &, const IsmrmrdHeader &)

EXPORTISMRMRD bool operator!= (const IsmrmrdHeader &lhs, const IsmrmrdHeader &rhs)

EXPORTISMRMRD bool operator== (const SubjectInformation &lhs,
const SubjectInformation &rhs)

EXPORTISMRMRD bool operator!= (const SubjectInformation &lhs,
const SubjectInformation &rhs)

EXPORTISMRMRD bool operator== (const StudyInformation &lhs, const StudyInformation &rhs)

EXPORTISMRMRD bool operator!= (const StudyInformation &lhs, const StudyInformation &rhs)

EXPORTISMRMRD bool operator== (const ReferencedImageSequence &lhs,
const ReferencedImageSequence &rhs)

EXPORTISMRMRD bool operator!= (const ReferencedImageSequence &lhs,
const ReferencedImageSequence &rhs)

EXPORTISMRMRD bool operator== (const MeasurementInformation &lhs,
const MeasurementInformation &rhs)

EXPORTISMRMRD bool operator!= (const MeasurementInformation &lhs,
const MeasurementInformation &rhs)

EXPORTISMRMRD bool operator== (const CoilLabel &lhs, const CoilLabel &rhs)

EXPORTISMRMRD bool operator!= (const CoilLabel &lhs, const CoilLabel &rhs)

EXPORTISMRMRD bool operator== (const AcquisitionSystemInformation &lhs,
const AcquisitionSystemInformation &rhs)

EXPORTISMRMRD bool operator!= (const AcquisitionSystemInformation &lhs,
const AcquisitionSystemInformation &rhs)

EXPORTISMRMRD bool operator== (const ExperimentalConditions &lhs,
const ExperimentalConditions &rhs)

EXPORTISMRMRD bool operator!= (const ExperimentalConditions &lhs,
const ExperimentalConditions &rhs)

EXPORTISMRMRD bool operator== (const MatrixSize &lhs, const MatrixSize &rhs)

EXPORTISMRMRD bool operator!= (const MatrixSize &lhs, const MatrixSize &rhs)
```

```
EXPORTISMRMRD bool operator== (const FieldOfView_mm &lhs, const FieldOfView_mm &rhs)

EXPORTISMRMRD bool operator!= (const FieldOfView_mm &lhs, const FieldOfView_mm &rhs)

EXPORTISMRMRD bool operator== (const EncodingSpace &lhs, const EncodingSpace &rhs)

EXPORTISMRMRD bool operator!= (const EncodingSpace &lhs, const EncodingSpace &rhs)

EXPORTISMRMRD bool operator== (const Limit &lhs, const Limit &rhs)

EXPORTISMRMRD bool operator!= (const Limit &lhs, const Limit &rhs)

EXPORTISMRMRD bool operator== (const EncodingLimits &lhs, const EncodingLimits &rhs)

EXPORTISMRMRD bool operator!= (const EncodingLimits &lhs, const EncodingLimits &rhs)

EXPORTISMRMRD bool operator== (const UserParameterLong &lhs,
const UserParameterLong &rhs)

EXPORTISMRMRD bool operator!= (const UserParameterLong &lhs,
const UserParameterLong &rhs)

EXPORTISMRMRD bool operator== (const UserParameterDouble &lhs,
const UserParameterDouble &rhs)

EXPORTISMRMRD bool operator!= (const UserParameterDouble &lhs,
const UserParameterDouble &rhs)

EXPORTISMRMRD bool operator== (const UserParameterString &lhs,
const UserParameterString &rhs)

EXPORTISMRMRD bool operator!= (const UserParameterString &lhs,
const UserParameterString &rhs)

EXPORTISMRMRD bool operator== (const UserParameters &lhs, const UserParameters &rhs)

EXPORTISMRMRD bool operator!= (const UserParameters &lhs, const UserParameters &rhs)

EXPORTISMRMRD bool operator== (const TrajectoryDescription &lhs,
const TrajectoryDescription &rhs)

EXPORTISMRMRD bool operator!= (const TrajectoryDescription &lhs,
const TrajectoryDescription &rhs)
```

```
EXPORTISMRMRD bool operator== (const AccelerationFactor &lhs,
const AccelerationFactor &rhs)

EXPORTISMRMRD bool operator!= (const AccelerationFactor &lhs,
const AccelerationFactor &rhs)

EXPORTISMRMRD bool operator== (const ParallelImaging &lhs, const ParallelImaging &rhs)

EXPORTISMRMRD bool operator!= (const ParallelImaging &lhs, const ParallelImaging &rhs)

EXPORTISMRMRD bool operator== (const Multiband &lhs, const Multiband &rhs)

EXPORTISMRMRD bool operator!= (const Multiband &lhs, const Multiband &rhs)

EXPORTISMRMRD bool operator== (const MultibandSpacing &lhs, const MultibandSpacing &rhs)

EXPORTISMRMRD bool operator!= (const MultibandSpacing &lhs, const MultibandSpacing &rhs)

EXPORTISMRMRD bool operator== (const Encoding &lhs, const Encoding &rhs)

EXPORTISMRMRD bool operator!= (const Encoding &lhs, const Encoding &rhs)

EXPORTISMRMRD bool operator== (const SequenceParameters &lhs,
const SequenceParameters &rhs)

EXPORTISMRMRD bool operator!= (const SequenceParameters &lhs,
const SequenceParameters &rhs)

EXPORTISMRMRD bool operator== (const WaveformInformation &lhs,
const WaveformInformation &rhs)

EXPORTISMRMRD bool operator!= (const WaveformInformation &lhs,
const WaveformInformation &rhs)

EXPORTISMRMRD bool operator== (const threeDimensionalFloat &lhs,
const threeDimensionalFloat &rhs)

EXPORTISMRMRD bool operator!= (const threeDimensionalFloat &lhs,
const threeDimensionalFloat &rhs)

EXPORTISMRMRD bool operator== (const MeasurementDependency &lhs,
const MeasurementDependency &rhs)

EXPORTISMRMRD bool operator!= (const MeasurementDependency &lhs,
const MeasurementDependency &rhs)
```

EXPORTISMRMRD bool operator== (const GradientDirection &lhs,
const GradientDirection &rhs)

EXPORTISMRMRD bool operator!= (const GradientDirection &lhs,
const GradientDirection &rhs)

EXPORTISMRMRD bool operator== (const Diffusion &lhs, const Diffusion &rhs)

EXPORTISMRMRD bool operator!= (const Diffusion &lhs, const Diffusion &rhs)

template<typename **T**>

class **Optional**

Public Functions

inline **Optional**()

inline **Optional**(const *T* &v)

inline *Optional* &**operator=**(const *Optional* &o)

inline *Optional* &**operator=**(const *T* &v)

inline *T* ***operator->**()

inline *T* &**operator***()

inline const *T* ***operator->**() const

inline const *T* &**operator***() const

inline **operator bool**() const

inline bool **is_present**() const

inline bool **has_value**() const noexcept

inline *T* &**value**() &

inline const *T* &**value**() const &

inline *T* &&**value**() &&

inline const *T* &&**value**() const &&

inline *T* &**get**() &

inline *T* &&**get**() &&

inline const *T* &**get**() const &

inline const *T* &&**get**() const &&

template<class **U**>

inline *T* **value_or**(*U* &&default_value) const &

template<class **U**>
inline *T* **value_or**(*U* &&default_value) &&

inline bool **operator==**(const *Optional*<*T*> &other) const

inline bool **operator==**(const *T* &val) const

inline *T* &**operator()**()

inline void **set**(const *T* &v)

Protected Attributes

bool **present_**

T **value_**

struct **threeDimensionalFloat**

Public Members

float **x**

float **y**

float **z**

struct **SubjectInformation**

Public Members

Optional<std::string> **patientName**

Optional<float> **patientWeight_kg**

Optional<float> **patientHeight_m**

Optional<std::string> **patientID**

Optional<std::string> **patientBirthdate**

Optional<std::string> **patientGender**

struct **StudyInformation**

Public Members

Optional<std::string> **studyDate**

Optional<std::string> **studyTime**

Optional<std::string> **studyID**

Optional<std::int64_t> **accessionNumber**

Optional<std::string> **referringPhysicianName**

Optional<std::string> **studyDescription**

Optional<std::string> **studyInstanceUID**

Optional<std::string> **bodyPartExamined**

struct **MeasurementDependency**

Public Members

std::string **dependencyType**

std::string **measurementID**

struct **ReferencedImageSequence**

Public Members

std::string **referencedSOPInstanceUID**

struct **MeasurementInformation**

Public Members

Optional<std::string> **measurementID**

Optional<std::string> **seriesDate**

Optional<std::string> **seriesTime**

std::string **patientPosition**

Optional<*threeDimensionalFloat*> **relativeTablePosition**

Optional<std::int64_t> **initialSeriesNumber**

Optional<std::string> **protocolName**

Optional<std::string> **sequenceName**

Optional<std::string> **seriesDescription**

std::vector<*MeasurementDependency*> **measurementDependency**

Optional<std::string> **seriesInstanceUIDRoot**

Optional<std::string> **frameOfReferenceUID**

std::vector<*ReferencedImageSequence*> **referencedImageSequence**

struct **CoilLabel**

Public Members

std::uint16_t **coilNumber**

std::string **coilName**

struct **AcquisitionSystemInformation**

Public Members

Optional<std::string> **systemVendor**

Optional<std::string> **systemModel**

Optional<float> **systemFieldStrength_T**

Optional<float> **relativeReceiverNoiseBandwidth**

Optional<std::uint16_t> **receiverChannels**

std::vector<*CoilLabel*> **coilLabel**

Optional<std::string> **institutionName**

Optional<std::string> **stationName**

Optional<std::string> **deviceID**

Optional<std::string> **deviceSerialNumber**

struct **ExperimentalConditions**

Public Members

std::int64_t **H1resonanceFrequency_Hz**

struct **MatrixSize**

Public Functions

inline **MatrixSize**()

inline **MatrixSize**(std::uint16_t x, std::uint16_t y)

inline **MatrixSize**(std::uint16_t x, std::uint16_t y, std::uint16_t z)

Public Members

std::uint16_t **x**

std::uint16_t **y**

std::uint16_t **z**

struct **FieldOfView_mm**

Public Members

float **x**

float **y**

float **z**

struct **EncodingSpace**

Public Members

MatrixSize **matrixSize**

FieldOfView_mm **fieldOfView_mm**

struct **Limit**

Public Functions

inline **Limit**()

inline **Limit**(std::uint16_t minimum, std::uint16_t maximum, std::uint16_t center)

Public Members

std::uint16_t **minimum**

std::uint16_t **maximum**

std::uint16_t **center**

struct **EncodingLimits**

Public Members

Optional<*Limit*> **kspace_encoding_step_0**

Optional<*Limit*> **kspace_encoding_step_1**

Optional<*Limit*> **kspace_encoding_step_2**

Optional<*Limit*> **average**

Optional<*Limit*> **slice**

Optional<*Limit*> **contrast**

Optional<*Limit*> **phase**

Optional<*Limit*> **repetition**

Optional<*Limit*> **set**

Optional<*Limit*> **segment**

std::array<*Optional*<*Limit*>, ISMRMRD_USER_INTS> **user**

struct **UserParameterLong**

Public Members

std::string **name**

std::int64_t **value**

struct **UserParameterDouble**

Public Members

std::string **name**

double **value**

struct **UserParameterString**

Public Members

std::string **name**

std::string **value**

struct **UserParameters**

Public Members

std::vector<*UserParameterLong*> **userParameterLong**

std::vector<*UserParameterDouble*> **userParameterDouble**

std::vector<*UserParameterString*> **userParameterString**

std::vector<*UserParameterString*> **userParameterBase64**

struct **TrajectoryDescription**

Public Members

std::string **identifier**

std::vector<*UserParameterLong*> **userParameterLong**

std::vector<*UserParameterDouble*> **userParameterDouble**

std::vector<*UserParameterString*> **userParameterString**

Optional<std::string> **comment**

struct **AccelerationFactor**

Public Members

std::uint16_t **kspace_encoding_step_1**

std::uint16_t **kspace_encoding_step_2**

struct **MultibandSpacing**

Public Members

std::vector<float> **dZ**

struct **Multiband**

Public Members

std::vector<*MultibandSpacing*> **spacing**

float **deltaKz**

std::uint32_t **multiband_factor**

MultibandCalibrationType **calibration**

std::uint64_t **calibration_encoding**

struct **ParallelImaging**

Public Members

AccelerationFactor **accelerationFactor**

Optional<std::string> **calibrationMode**

Optional<std::string> **interleavingDimension**

Optional<*Multiband*> **multiband**

struct **Encoding**

Public Members

EncodingSpace **encodedSpace**

EncodingSpace **reconSpace**

EncodingLimits **encodingLimits**

TrajectoryType **trajectory**

Optional<*TrajectoryDescription*> **trajectoryDescription**

Optional<*ParallelImaging*> **parallelImaging**

Optional<std::int64_t> **echoTrainLength**

struct **GradientDirection**

Public Members

float **rl**

float **ap**

float **fh**

struct **Diffusion**

Public Members

float **bvalue**

GradientDirection **gradientDirection**

struct **SequenceParameters**

Public Members

Optional<std::vector<float>> **TR**

Optional<std::vector<float>> **TE**

Optional<std::vector<float>> **TI**

Optional<std::vector<float>> **flipAngle_deg**

Optional<std::string> **sequence_type**

Optional<std::vector<float>> **echo_spacing**

Optional<*DiffusionDimension*> **diffusionDimension**

Optional<std::vector<*Diffusion*>> **diffusion**

Optional<std::string> **diffusionScheme**

struct **WaveformInformation**

Public Members

std::string **waveformName**

WaveformType **waveformType**

Optional<*UserParameters*> **userParameters**

struct **IsmrmrdHeader**

Public Members

Optional<std::int64_t> **version**

Optional<*SubjectInformation*> **subjectInformation**

Optional<*StudyInformation*> **studyInformation**

Optional<*MeasurementInformation*> **measurementInformation**

Optional<*AcquisitionSystemInformation*> **acquisitionSystemInformation**

ExperimentalConditions **experimentalConditions**

std::vector<*Encoding*> **encoding**

Optional<*SequenceParameters*> **sequenceParameters**

Optional<*UserParameters*> **userParameters**

std::vector<*WaveformInformation*> **waveformInformation**

INDEX

D

I

M

O

T

W