JSON metadata – Manual

Responsible developer: Evelyne Balteau
Developed as part of the hMRI toolbox, including the following developers: Martina Callaghan, Siawoosh Mohammadi, Antoine Lutti, Karsten Tabelow, Nikolaus Weiskopf, Lars Ruthotto, Tobias Leutritz.

1. Introduction

BIDS recommendations suggest using JSON-encoded metadata to store useful information alongside brain imaging data sets. These metadata include acquisition parameters contained in the DICOM header of MR images, which are otherwise often discarded during DICOM to NIFTI image conversion.

The tools described in this manual have the following goals:

- retrieving the full DICOM header as a readable and searchable Matlab structure (metadata),
- during DICOM to NIFTI conversion, storing the metadata structure as separate JSON file (alongside the NIFTI image file) and/or as extended header into the NIFTI image file,
- handling metadata: mainly get, set and search tools.

The use of these tools is described in more detail below and good housekeeping suggestions are provided to make use of the metadata structure to track acquisition and processing steps, including processing parameters, tools and software versions, etc.

The current implementation rests on the SPM12 implementation of DICOM tools (`spm_dicom_header(s)`, `spm_dicom_essentials`, `spm_dicom_convert`) and relies on `spm_jsonread` and `spm_jsonwrite` for handling JSON-encoded metadata.

2. DICOM headers – `spm_dicom_header`

The following scripts are unchanged: `spm_dicom_header`, `spm_dicom_headers` and `spm_dicom_essentials`.

3. DICOM to NIFTI conversion – `spm_dicom_convert`

An additional input argument has been added to deal with the new JSON metadata options. If JSON metadata storage is enabled, DICOM header information is stored as a JSON-formatted structure either as a separate JSON file or as extended NIFTI header (or both). If omitted or disabled, the DICOM to NIFTI conversion proceeds using exactly the same implementation as before. The output NIFTI images, extended or not, have a valid NIFTI format compatible with standard NIFTI readers (MRICron, SPM)[1].

1 Note that FSL has trouble reading extended NIFTI images since it does not even read the offset of the NIFTI object and assumes the default 352 bytes offset instead. As a result, NIFTI files containing extended headers are misread in FSL.

USAGE:

```
spm_dicom_convert(hdr, opts, root_dir, format, out_dir, json)
Inputs are as before except for the last json input:
json - a structure with fields:
    extended -> metadata stored as extended nii header included in
            the nii image [default: false]
    separate -> metadata stored in separate json file [default:
            false]
    anonym -> 'basic' (default), 'full', 'none' (see
            anonymise_metadata.m)
```

EXAMPLES:

```
files2convert = spm_select(Inf,'^*.IMA');
outpth = spm_select(1,'dir','Select output directory');
hdr = spm_dicom_headers(files2convert);
% for conversion output identical to previous SPM12 versions (no
% metadata stored during conversion):
files_converted = spm_dicom_convert(hdr,'all','flat','nii',outpth);
% for conversion with creation of a separate JSON metadata file:
json = struct('extended',false,'separate',true);
files_converted = spm_dicom_convert(hdr,'all','flat','nii',outpth,json);
% for conversion with insertion of the JSON metadata as NIFTI
% extended header:
json = struct('extended',true,'separate',false);
files_converted = spm_dicom_convert(hdr,'all','flat','nii',outpth,json);
% metadata can be stored simultaneously as a separate JSON file and
% extended NIFTI header:
json = struct('extended',true,'separate',true);
files_converted = spm_dicom_convert(hdr,'all','flat','nii',outpth,json);
```

4. DICOM to NIFTI conversion – SPM Batch

The above modifications have been reflected in the SPM Batch GUI (Batch > SPM > Tools > DICOM Import, or directly from the DICOM Import button) by adding the following options:

```
JSON metadata format: Metadata (acquisition parameters, processing steps,
    …) can be stored in JSON format. The metadata can be stored as a
    separate JSON file and/or as an extended header (included in the
    image file, for nii output format only).
    One of the following options must be selected:
        - None
        - Separate JSON file
        - Extended nii header
        - Both

Anonymisation: If JSON metadata are to be stored, you might want to make
    sure that patient confidentiality is preserved.
    WARNING: effective anonymisation cannot be guaranteed, since it
    depends on the way patient data have been registered. USE WITH CARE!
    Two levels of anonymisation are available:
```

```
      - full anonymisation: no patient information is kept at all. NOTE:
        the patient ID is kept in the converted file name.
      - basic anonymisation: patient ID (assuming it does not contain
        his name), age (years at the time of the data acquisition), sex,
        size and weight are kept; patient name, date of birth and DICOM
        filename (often containing the patient name) are removed.
    One of the following options must be selected:
        - None
        - Full
        - Basic

Content of JSON metadata: To reduce storage space (but is it still a
    concern?), metadata can be limited to a basic set of parameters.
    One of the following options must be selected:
        - Full (recommended)
        - Essentials
```

5. The metadata structure

FORMAT OF METADATA STRUCTURE

- Metadata are simple Matlab structures – hence quite flexible and modular.
- The Matlab structure is converted into JSON (a text format transcription of the Matlab structure) to be stored in the NIFTI header or as a separate JSON file.
- The JSON transcription is easily readable and searchable by opening the NIFTI or JSON file with a text editor.
- JSON string ↔ Matlab structure conversion done using `spm_jsonread/write`.
- NIFTI files with extended headers can be read without any trouble with SPM and MRIcron (existing bug with FSL though, see footnote[1]).

CONTENT AND STRUCTURE OF THE METADATA

The structure of the metadata is divided into the following two main fields:

- **Acquisition parameters** (**hdr.acqpar**): this branch contains the *original DICOM header*. It should be kept unchanged and is created when importing DICOM images to NIFTI. This might be dropped if the processed image relies on several input images (e.g. when creating a map from mtflash and b1mapping acquisitions). Note that by default, the patient name, date of birth and DICOM file name (often containing the patient name) are removed, while patient age (years), sex, size and weight are kept, for confidentiality reasons.

- **History** (**hdr.history**): is a *nested* structure containing
    - **procstep**: a structure describing the current processing step and containing:
        - **descrip**: a brief description of the processing applied (e.g. DICOM to NIFTI conversion, realign, create map, …)
        - **version**: version number of the processing applied (traceability!)
        - **procpar**: processing parameters (relevant parameters used to process the data – e.g. for fieldmap-based EPI undistortion: EPI readout duration, TEs of

the field mapping acquisition, ...). Basically all the parameters included into the matlab batch "job" *except for the input images*.

- **input**. An array listing the input images used for the processing. Each input (hdr.history.input(i)) is a structure containing:
  - **filename**: the filename of the input image
  - **history**: the history structure from the input image
- **output**: a structure containing
  - **imtype**: image type of the current output (e.g. R1, FA, MT, ADC, …)
  - **units**: either physical units (sec, 1/sec, …), standardized units (e.g. percent units = p.u.) or arbitrary units (a.u.)…

6. Metadata handling

Storing these metadata as a JSON-encoded structure is a good thing, but better if you can read, write, update, use these metadata ;)… Tools for metadata handling have been implemented and added in the `SPM12/metadata` directory. Type `help <function>` in Matlab for detailed syntax.

Here is the list of scripts and their summary descriptions:

`init_metadata`: used in `spm_dicom_convert` to initialize the metadata structure
`get_metadata`: returns the Matlab structure described above.
`set_metadata`: to write (insert, modify, overwrite) metadata in JSON files and extended NIFTI headers. In general, metadata are initialized during the DICOM to NIFTI conversion and further modified as the processing progresses.
`get_metadata_val`: returns pairs of structure fields and values agreeing with search criteria
`find_field_name`: recursively searches a Matlab structure for a specific field name
`has_extended_header`: returns whether a NIFTI file has extended header or not
`anonymise_metadata`: used in `init_metadata` to remove confidential fields.
`get_jhdr_and_offset`: used in `init_metadata` and `set_metadata`, can be considered as private method since should be of no use otherwise.
`write_extended_header`: used in `init_metadata` and `set_metadata`, can be considered as private method since should be of no use otherwise.

The following scripts are used during DICOM to NIFTI conversion and should not be needed otherwise. Just reported here for completeness:

`reformat_spm_dicom_header`: used to reformat CSA and ASCII fields in the DICOM header output from `spm_dicom_header(s)`.
`tidy_CSA`: reformat the CSA fields into standard Matlab structure.
`read_ASCII`: read the ASCII header and convert it into a Matlab structure.

EXAMPLE 1: modify existing metadata
NB: a very basic example (NIFTI file with extended header, no JSON file associated), not the most common usage of set_metadata, but should help getting familiar with handling the metadata...

```
% select nii file with extended header
niifile = spm_select(1,'^*.nii');
% read the extended header
hdr = get_metadata(niifile);
```

```
% define json format (since working with NIFTI file, we can choose
% to limit set_metadata to updating the extended header without
% caring about updating the JSON file associated).
json = struct('extended',true,'separate',false,'overwrite',true);

% modify the header of the existing nii file
modif_hdr = hdr{1};
modif_hdr.history.procstep.descrip = 'original file with modified header';
modif_hdr.history.procstep.version = 'v0.1.2';
modif_hdr.history.procstep.procpar = [];
modif_hdr.history.input(1).filename = niifile;
modif_hdr.history.input(1).history = hdr{1}.history;
modif_hdr.history.output.imtype = hdr{1}.history.output.imtype;
modif_hdr.history.output.units = hdr{1}.history.output.units;

% write the modified metadata
set_metadata(niifile, modif_hdr, json);
```

EXAMPLE 2: save a newly computed R1 map with extended header
NB: for the example, we simply read an existing NIFTI volume and copy it into another NIFTI file to which we add an extended header.

```
% Select nifti file to copy
P = spm_select(1,'^*.nii');
V = spm_vol(P);
Y = spm_read_vols(V);

% initialize new nifti object
N = nifti;
% fill up the fields and data section of the nifti object as usual...
N.dat  = V(1).private.dat;
N.dat  = file_array('R1map.nii',V.dim,V.dt,0,V.pinfo(1),V.pinfo(2));
N.mat  = V.mat;
N.mat0 = V.mat;
N.mat_intent  = 'Scanner';
N.mat0_intent = 'Scanner';
N.descrip = 'test creating a new NIFTI file with extended JSON header';
create(N);
% write the data
N.dat(:,:,:) = Y;

% create a new structure for the extended header
outhdr = struct('history', struct('procstep',[],'input',[],'output',[]));
% NB: since it is no longer an original image from the scanner, the field
% 'acqpar' is dropped. Input files can be retrieved with their full
% acquisition parameters specifications from the history.input field.
outhdr.history.procstep.descrip = 'map creation';
outhdr.history.procstep.version = 'map_creation_v3.2';
% insert job parameters here:
outhdr.history.procstep.procpar = struct('par1',[1 2],'par2','test');
outhdr.history.input{1}.filename = V.fname;
inhdr = get_metadata(V.fname);
if ~isempty(inhdr)
    outhdr.history.input{1}.history = inhdr{1}.history;
else
    outhdr.history.input{1}.history = 'No history available';
end
outhdr.history.output.imtype = 'R1 map';
```

```
outhdr.history.output.units = '1/sec';

% Write the metadata as both extended NIFTI header and separate JSON file
% NB: if no json structure provided, set_metadata will check for existing
% JSON and NIFTI files and update the existing ones.
json = struct('extended',true,'separate',true,'overwrite',true);
set_metadata('R1map.nii', outhdr, json);
```

7. How to retrieve specific parameters from the JSON metadata

This section deals mainly with the *acqpar* subfield of the JSON metadata, i.e. the parameters retrieved from the DICOM header, but the tools implemented to search the metadata structure can be used more generally to search any Matlab structure.

Since the extended header is a structure, searching for a specific parameter is often equivalent to searching for the corresponding field name. For that reason, the function `find_field_name` has been implemented to recursively search for a specific field name (exact or partial match, case sensitive or not). The `get_metadata_val` script makes use of `find_field_name` to sort out the parameters and return the desired value. More refined versions of `get_metadata_val` will be developed with time to deal with hardware/software/sequences variations. However the front end of `get_metadata_val` should stay unchanged for backwards compatibility.

Below are listed a series of common parameters and how they can be retrieved from the JSON metadata. Note that these examples may need to be adapted according to the version of the MRI hardware, software and sequences. Other arbitrary parameters can easily be searched using the same `get_metadata_val` script. See the examples at the end of this section.

Retrieve the whole header (type `help get_metadata` for details on the syntax and input files)

```
hdr = get_metadata('my_extended_nifti_file.nii');
```

Excitation flip angle [deg]

```
fa = get_metadata_val(hdr{1},'FlipAngle');
```

Repetition time [ms]

```
TR = get_metadata_val(hdr{1},'RepetitionTime');
```

Echo time(s) [ms] – and array of TEs if multi-echo sequence

```
TE = get_metadata_val(hdr{1},'EchoTime');
```

Magnetization transfer (MT) pulse = 1/0 according to pulse switched on/off

```
MT = get_metadata_val(hdr{1},'MT');
```

Field strength B0 [T]

```
B0 = get_metadata_val(hdr{1},'FieldStrength');
```

Center frequency [Hz]

```
freq = get_metadata_val(hdr{1},'Frequency');
```

Protocol Name
```
ProtName = get_metadata_val(hdr{1},'ProtocolName');
```

Sequence Name
```
SeqName = get_metadata_val(hdr{1},'SequenceName');
```

RF spoiling phase increment (warning: sequence dependent, see comments in the code)
```
RFSpoilIncr = get_metadata_val(hdr{1},'RFSpoilingPhaseIncrement');
```

Bandwidth per pixel (in the RO direction) in Hz/Px
```
BWPPRO = get_metadata_val(hdr{1},'BandwidthPerPixelRO');
```

Bandwidth per pixel (in the PE direction) in Hz/Px
```
BWPPPE = get_metadata_val(hdr{1},'BandwidthPerPixelPE');
```

PAT acceleration (structure containing all parameters)
```
PAT = get_metadata_val(hdr{1},'PATparameters');
```

PAT acceleration factor in PE direction (in-plane)
```
RPE = get_metadata_val(hdr{1},'AccelFactorPE');
```

PAT acceleration factor in 3D direction (through-slice direction for 3D acquisitions)
```
R3D = get_metadata_val(hdr{1},'AccelFactor3D');
```

All WIP parameters (Siemens ASCII field: returns a structure with two fields alFree and adFree containing arrays of long and double values respectively)
```
WIP = get_metadata_val(hdr{1},'WipParameters');
```

EPI parameters (including distortion correction)

- Short and long echo times from dual-echo field mapping:
```
TE = get_metadata_val(hdr_gre_field_mapping{1},'EchoTime');
```
- EPI phase encoding direction (ROW/COL)
```
PEDir = get_metadata_val(hdr_epi{1},'PhaseEncodingDirection');
```
- EPI phase encoding direction positive (e.g. A>>P is positive (1), P>>A is negative (-1))
```
PEDirPos =get_metadata_val(hdr_epi{1},'PhaseEncodingDirectionSign');
```
- Total EPI readout duration (warning: tricky for home-made sequences with uncompleted headers)
```
epiROdur = get_metadata_val(hdr{1},'epiReadoutDuration');
```
- EPI echo spacing (warning: tricky for home-made sequences with uncompleted headers)
```
epiEchoSpacing = get_metadata_val(hdr{1},'EchoSpacing');
```

Parameters for processing B1 maps (al_b1mapping)

- Nominal FA values (betas)
```
beta = get_metadata_val(hdr{1},'B1mapNominalFAValues');
```
- Mixing time (TM)
```
TM = get_metadata_val(hdr{1},'B1mapMixingTime');
```

DWI parameters

This is specific to Siemens DICOM metadata. In that case, the diffusion parameters can be retrieved following two distinct ways.

1. Reading all the directions and b-values at once from CSASeriesHeaderInfo subfield

NB: the current implementation assumes that Free diffusion mode is used, with a user-defined set of diffusion directions.

- Diffusion directions: list of directions as defined in the *.dvs file (normalised or not)

```
DiffDir = get_metadata_val(hdr{1},'AllDiffusionDirections');
```

- b-values: list of b-values as defined in the Diff tab of the UI

```
bVal = get_metadata_val(hdr{1},'AllBValues');
```

2. Reading the individual directions and b-values for each image from CSAImageHeaderInfo subfield

NB: this implementation is expected to work more generally. I've noticed that the signs of the y and z components of the direction vector are inverted, does this ring a bell with anybody why this is so? Or has anybody noticed they needed to invert signs in their processing? (it actually rings a bell with me as something fellows have mentioned to me, but I'm not processing much DWI myself). The respective orientation of patient/image/scanner axes is probably relevant too :(… Help from DWI expert required here ;)!

- Diffusion direction: normalised vector

```
DiffDir = get_metadata_val(hdr{1},'DiffusionDirection');
```

- b-values: corresponding b-value

```
bVal = get_metadata_val(hdr{1},'BValue');
```

There is, to my knowledge, no way to retrieve the δ, Δ and G values describing the timing and amplitude of the diffusion gradients :(...

General examples searching for arbitrary parameters

The recursive search of the field names of the Matlab structure follows simple pattern matching rules. Any pattern, partial or full, case sensitive or not, can be used. One can either use the `get_metadata_val` script with any desired pattern (see EXAMPLES 1 & 2 below), or use the `find_field_name` script with more specific matching rules (see EXAMPLES 3 & 4). In the former case, the search is *not case sensitive* and will retrieve any field that *contains* the pattern. In the latter case, one can specify whether to be case sensitive or not, and whether to allow partial matches or not.

In EXAMPLES 1 & 2, we search for any field containing "diff" because we want to identify potential diffusion parameters. The search is performed on a DWI image and on a t1-mprage image.

EXAMPLE 1 – search in an (extended) NIFTI file containing a t1-mprage image:

```
[parVal, parFieldNam] = get_metadata_val('t1mprage.nii','diff');
% the output is:
parVal =
    ulMode: 1
parFieldNam =
    acqpar.CSASeriesHeaderInfo.MrPhoenixProtocol.sDiffusion
```

A single field name is found containing "diff" and its value is a structure, with a single field `ulMode` and value `1`.

EXAMPLE 2 – search an (extended) NIFTI file containing a DWI image:

```
[parVal, parFieldNam] = get_metadata_val('dwi.nii','diff');
% the output is:
parVal = {
    [3x1 double],
    'DIRECTIONAL',
    'GAAXkVNRAAD\/\/\/…',
    [1x1 struct],
    [2],
    [72],
    [1x1 struct],
    [72],
    [71x1 struct]
}
parFieldNam = {
  acqpar.CSAImageHeaderInfo.DiffusionGradientDirection
  acqpar.CSAImageHeaderInfo.DiffusionDirectionality
  acqpar.CSAImageHeaderInfo.MRDiffusion
  acqpar.CSASeriesHeaderInfo.MrPhoenixProtocol.sDiffusion
  acqpar.CSASeriesHeaderInfo.MrPhoenixProtocol.sDiffusion.lDiffWeightings
  acqpar.CSASeriesHeaderInfo.MrPhoenixProtocol.sDiffusion.lDiffDirections
  acqpar.CSASeriesHeaderInfo.MrPhoenixProtocol.sDiffusion.sFreeDiffusionData
  acqpar.CSASeriesHeaderInfo.MrPhoenixProtocol.sDiffusion.sFreeDiffusionData.lDiffDirections
  acqpar.CSASeriesHeaderInfo.MrPhoenixProtocol.sDiffusion.sFreeDiffusionData.asDiffDirVector
}
```

In the following examples, we search the same DWI metadata as above using `find_field_name`. Type `help find_field_name` in Matlab to know more about `find_field_name`'s syntax.

EXAMPLE 3 – search for "Diff" (case sensitive and partial match):

```
hdr = get_metadata('dwi.nii');
[nFieldFound, fieldList] = find_field_name(hdr{1},'Diff','caseSens',
'sensitive','matchType','partial');
% the output is:
nFieldFound =
     9
fieldList =
    'acqpar'  'CSAImageHeaderInfo'   'DiffusionGradient…'           []                  []                []
    'acqpar'  'CSAImageHeaderInfo'   'DiffusionDirectio…'           []                  []                []
    'acqpar'  'CSAImageHeaderInfo'   'MRDiffusion'                  []                  []                []
    'acqpar'  'CSASeriesHeaderInfo'  'MrPhoenixProtocol'    'sDiffusion'                []                []
    'acqpar'  'CSASeriesHeaderInfo'  'MrPhoenixProtocol'    'sDiffusion'  'lDiffWeightings'               []
    'acqpar'  'CSASeriesHeaderInfo'  'MrPhoenixProtocol'    'sDiffusion'  'lDiffDirections'               []
    'acqpar'  'CSASeriesHeaderInfo'  'MrPhoenixProtocol'    'sDiffusion'  'sFreeDiffusionData'            []
    'acqpar'  'CSASeriesHeaderInfo'  'MrPhoenixProtocol'    'sDiffusion'  'sFreeDiffusionData'  'lDiffDirections'
    'acqpar'  'CSASeriesHeaderInfo'  'MrPhoenixProtocol'    'sDiffusion'  'sFreeDiffusionData'  'asDiffDirVector'
```

EXAMPLE 4 – search for "sfreediffusiondata" (case insensitive and exact match):

```
hdr = get_metadata('dwi.nii');
[nFieldFound, fieldList] =
     find_field_name(hdr{1},'sfreediffusiondata','caseSens',
     'insensitive','matchType','exact');
% the output is:
nFieldFound =
     1
fieldList =
    'acqpar'    'CSASeriesHeaderInfo'    'MrPhoenixProtocol'    'sDiffusion'    'sFreeDiffusionData'
```