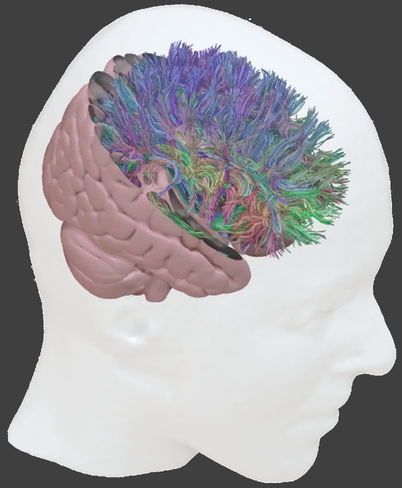
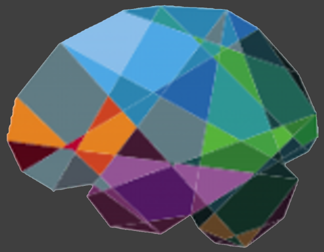
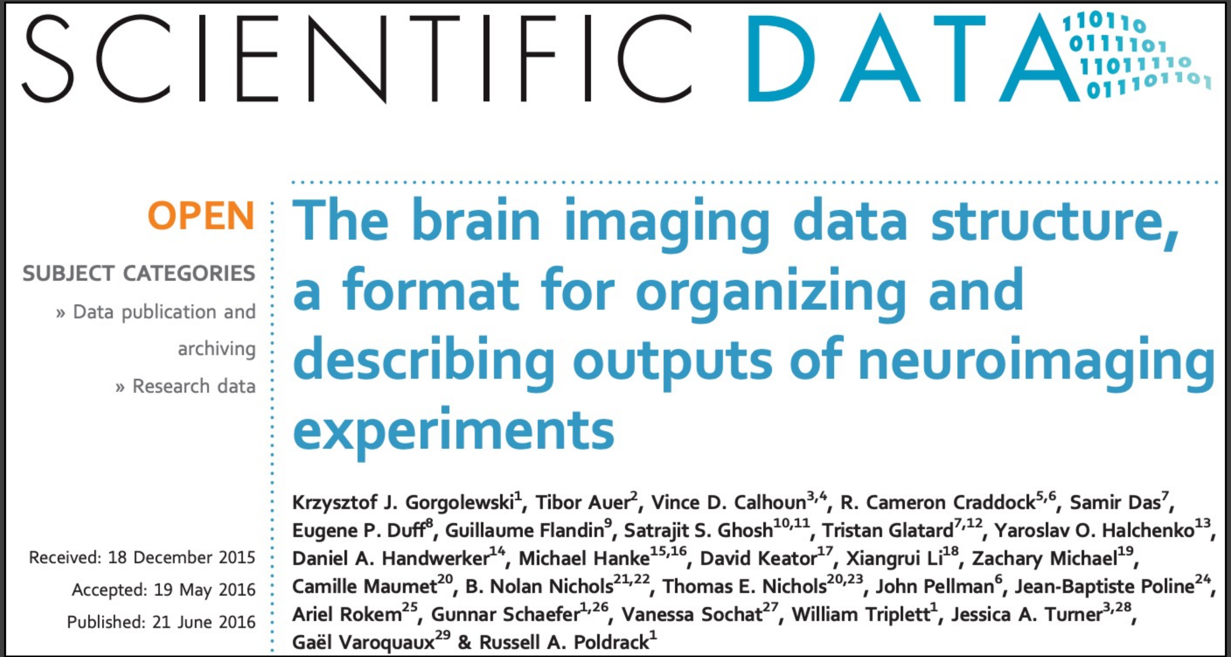
BIDS extension proposal computational models



Michael Schirner
Brain Simulation Section (PI: Petra Ritter)
Charité—Universitätsmedizin Berlin
Berlin, November 26th, 2021



Brain Imaging Data Structure



SCIENTIFIC DATA

110110
0111101
11011110
01110110

OPEN

SUBJECT CATEGORIES
» Data publication and archiving
» Research data

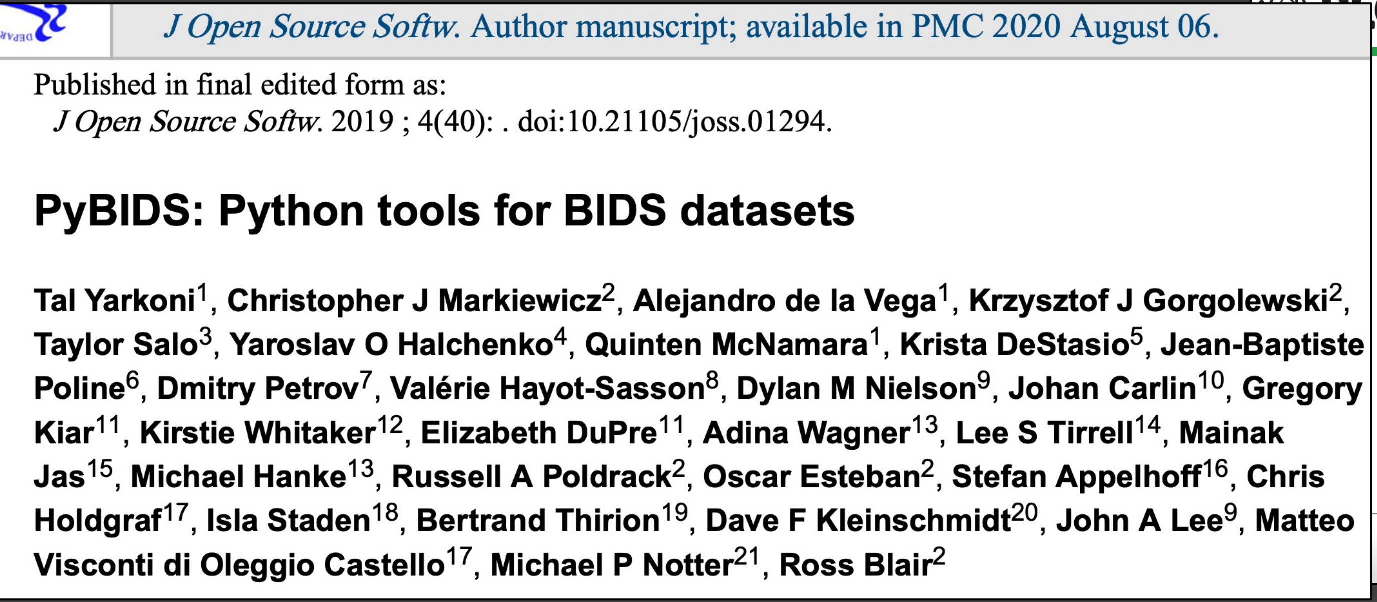
The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments

Krzysztof J. Gorgolewski[1], Tibor Auer[2], Vince D. Calhoun[3,4], R. Cameron Craddock[5,6], Samir Das[7], Eugene P. Duff[8], Guillaume Flandin[9], Satrajit S. Ghosh[10,11], Tristan Glatard[7,12], Yaroslav O. Halchenko[13], Daniel A. Handwerker[14], Michael Hanke[15,16], David Keator[17], Xiangrui Li[18], Zachary Michael[19], Camille Maumet[20], B. Nolan Nichols[21,22], Thomas E. Nichols[20,23], John Pellman[6], Jean-Baptiste Poline[24], Ariel Rokem[25], Gunnar Schaefer[1,26], Vanessa Sochat[27], William Triplett[1], Jessica A. Turner[3,28], Gaël Varoquaux[29] & Russell A. Poldrack[1]

Received: 18 December 2015
Accepted: 19 May 2016
Published: 21 June 2016

OpenNEURO

A free and open platform for sharing MRI, MEG, EEG, iEEG, and ECoG data

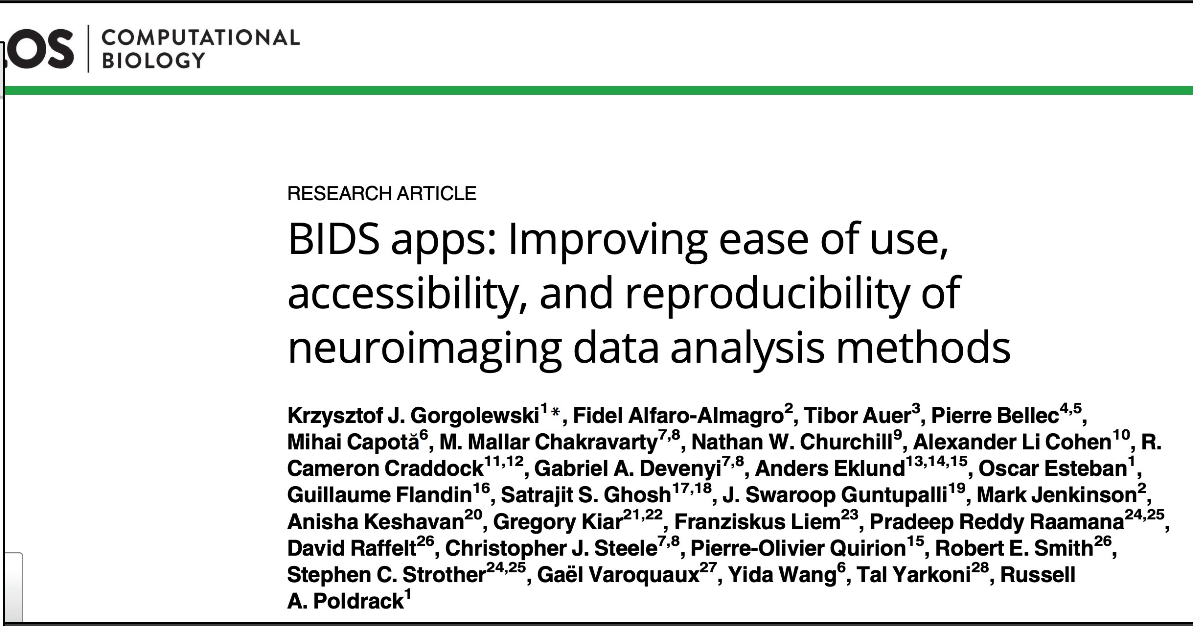


J Open Source Softw. Author manuscript; available in PMC 2020 August 06.

Published in final edited form as:
J Open Source Softw. 2019 ; 4(40): . doi:10.21105/joss.01294.

PyBIDS: Python tools for BIDS datasets

Tal Yarkoni[1], Christopher J Markiewicz[2], Alejandro de la Vega[1], Krzysztof J Gorgolewski[2], Taylor Salo[3], Yaroslav O Halchenko[4], Quinten McNamara[1], Krista DeStasio[5], Jean-Baptiste Poline[6], Dmitry Petrov[7], Valérie Hayot-Sasson[8], Dylan M Nielson[9], Johan Carlin[10], Gregory Kiar[11], Kirstie Whitaker[12], Elizabeth DuPre[11], Adina Wagner[13], Lee S Tirrell[14], Mainak Jas[15], Michael Hanke[13], Russell A Poldrack[2], Oscar Esteban[2], Stefan Appelhoff[16], Chris Holdgraf[17], Isla Staden[18], Bertrand Thirion[19], Dave F Kleinschmidt[20], John A Lee[9], Matteo Visconti di Oleggio Castello[17], Michael P Notter[21], Ross Blair[2]

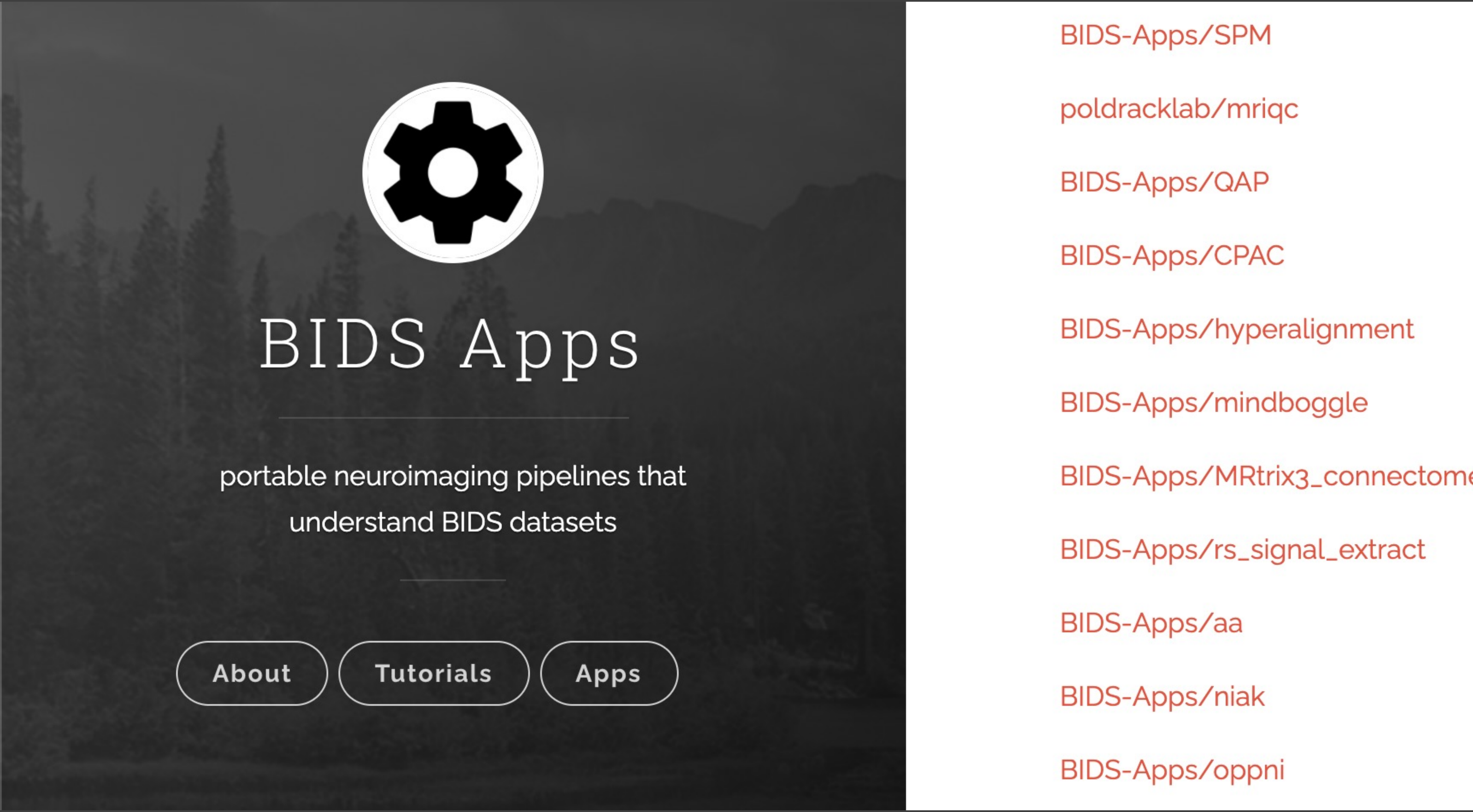


PLOS | COMPUTATIONAL BIOLOGY

RESEARCH ARTICLE

BIDS apps: Improving ease of use, accessibility, and reproducibility of neuroimaging data analysis methods

Krzysztof J. Gorgolewski[1]*, Fidel Alfaro-Almagro[2], Tibor Auer[3], Pierre Bellec[4,5], Mihai Capotă[6], M. Mallar Chakravarty[7,8], Nathan W. Churchill[9], Alexander Li Cohen[10], R. Cameron Craddock[11,12], Gabriel A. Devenyi[7,8], Anders Eklund[13,14,15], Oscar Esteban[1], Guillaume Flandin[16], Satrajit S. Ghosh[17,18], J. Swaroop Guntupalli[19], Mark Jenkinson[2], Anisha Keshavan[20], Gregory Kiar[21,22], Franziskus Liem[23], Pradeep Reddy Raamana[24,25], David Raffelt[26], Christopher J. Steele[7,8], Pierre-Olivier Quirion[15], Robert E. Smith[26], Stephen C. Strother[24,25], Gaël Varoquaux[27], Yida Wang[6], Tal Yarkoni[28], Russell A. Poldrack[1]



BIDS Apps

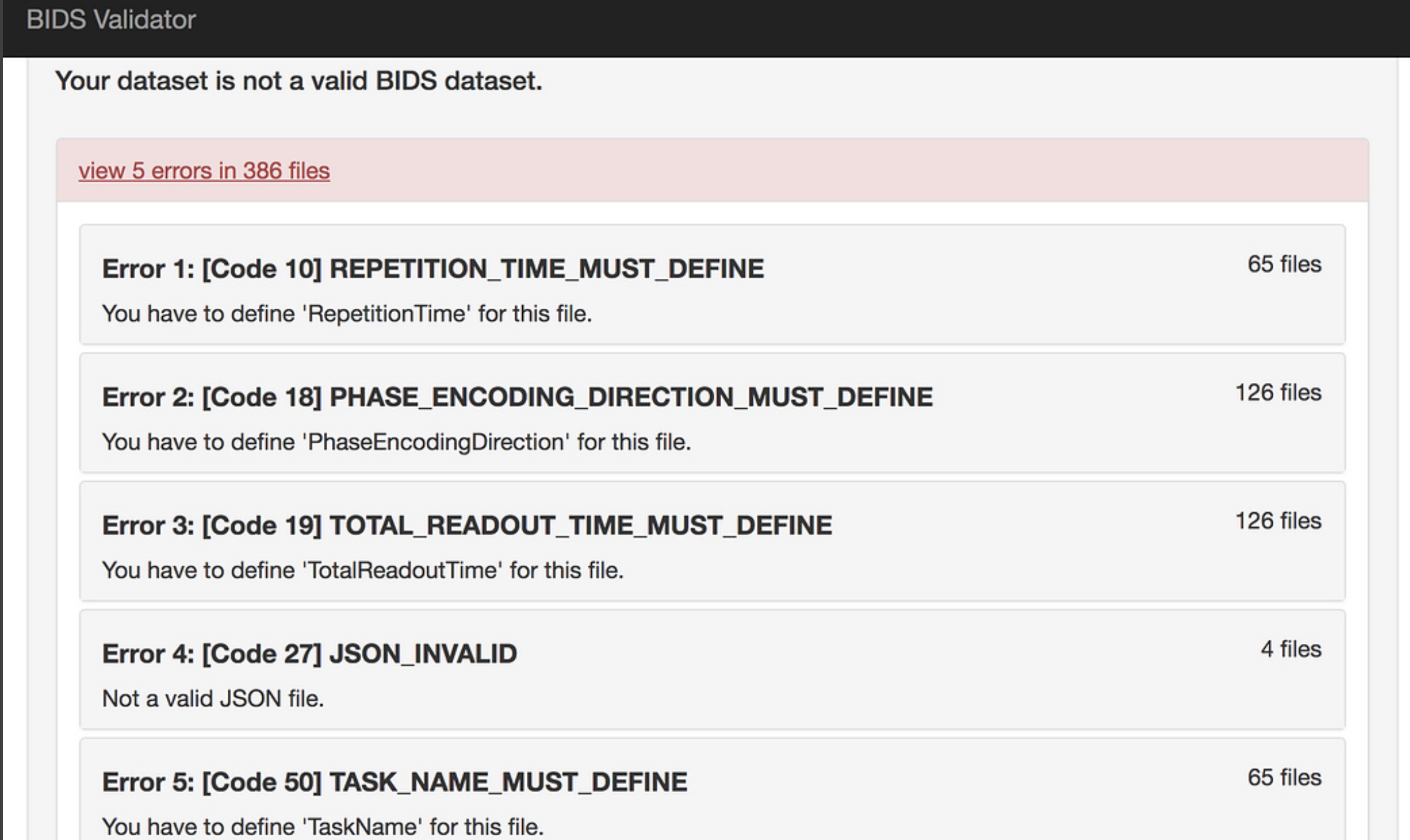
portable neuroimaging pipelines that understand BIDS datasets

- BIDS-Apps/SPM
- poldracklab/mriqc
- BIDS-Apps/QAP
- BIDS-Apps/CPAC
- BIDS-Apps/hyperalignment
- BIDS-Apps/mindboggle
- BIDS-Apps/MRtrix3_connectome
- BIDS-Apps/rs_signal_extract
- BIDS-Apps/aa
- BIDS-Apps/niak
- BIDS-Apps/oppni

Reasons for data standards

- Neuroimaging experiments produce **complicated data in many modalities and formats**
- **Lack of standards** leads to errors and wasted time
- **Reproducibility**: exact description of inputs, applied transformations and outputs needed
- **Robustness**: A good structure makes it easier to detect errors

BIDS validator



Your dataset is not a valid BIDS dataset.

view 5 errors in 386 files

Error 1: [Code 10] REPETITION_TIME_MUST_DEFINE 65 files

You have to define 'RepetitionTime' for this file.

Error 2: [Code 18] PHASE_ENCODING_DIRECTION_MUST_DEFINE 126 files

You have to define 'PhaseEncodingDirection' for this file.

Error 3: [Code 19] TOTAL_READOUT_TIME_MUST_DEFINE 126 files

You have to define 'TotalReadoutTime' for this file.

Error 4: [Code 27] JSON_INVALID 4 files

Not a valid JSON file.

Error 5: [Code 50] TASK_NAME_MUST_DEFINE 65 files

You have to define 'TaskName' for this file.

Scholarship & reproducibility

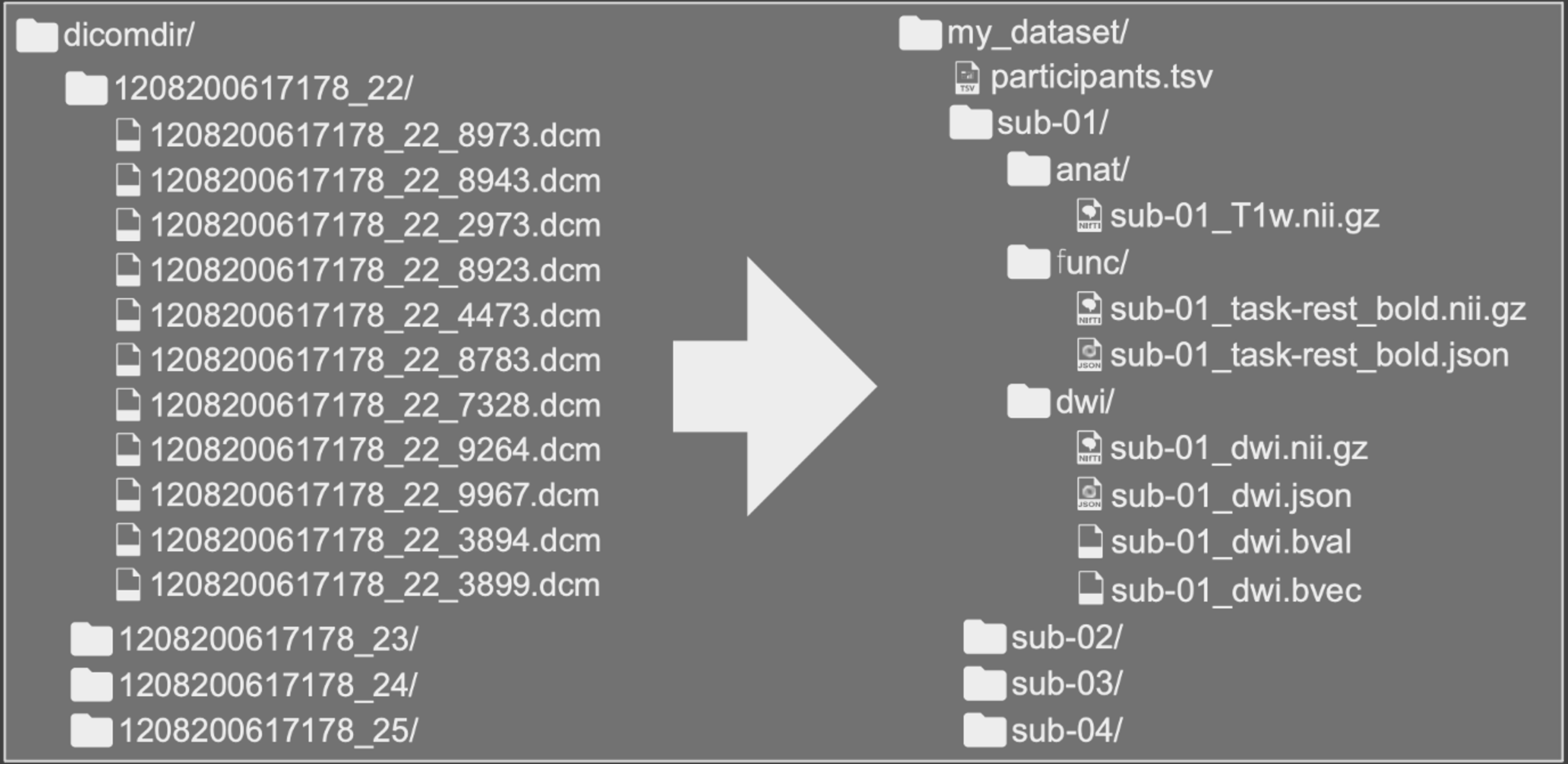
"An article about computational science...is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures."

Buckheit and Donoho (1995)

- Containerization enables only repeatability
- Reproducibility requires clear **specification of the model** such that it can be **replicated in a different environment and compared with other implementations**.

Poldrack,…,Ritter,… et al.,
2019, Comp Brain Behav

BIDS in a nutshell



```
dicomdir/
    1208200617178_22/
        1208200617178_22_8973.dcm
        1208200617178_22_8943.dcm
        1208200617178_22_2973.dcm
        1208200617178_22_8923.dcm
        1208200617178_22_4473.dcm
        1208200617178_22_8783.dcm
        1208200617178_22_7328.dcm
        1208200617178_22_9264.dcm
        1208200617178_22_9967.dcm
        1208200617178_22_3894.dcm
        1208200617178_22_3899.dcm
    1208200617178_23/
    1208200617178_24/
    1208200617178_25/
```

```
my_dataset/
    participants.tsv
    sub-01/
        anat/
            sub-01_T1w.nii.gz
        func/
            sub-01_task-rest_bold.nii.gz
            sub-01_task-rest_bold.json
        dwi/
            sub-01_dwi.nii.gz
            sub-01_dwi.json
            sub-01_dwi.bval
            sub-01_dwi.bvec
    sub-02/
    sub-03/
    sub-04/
```

MRI scanner output

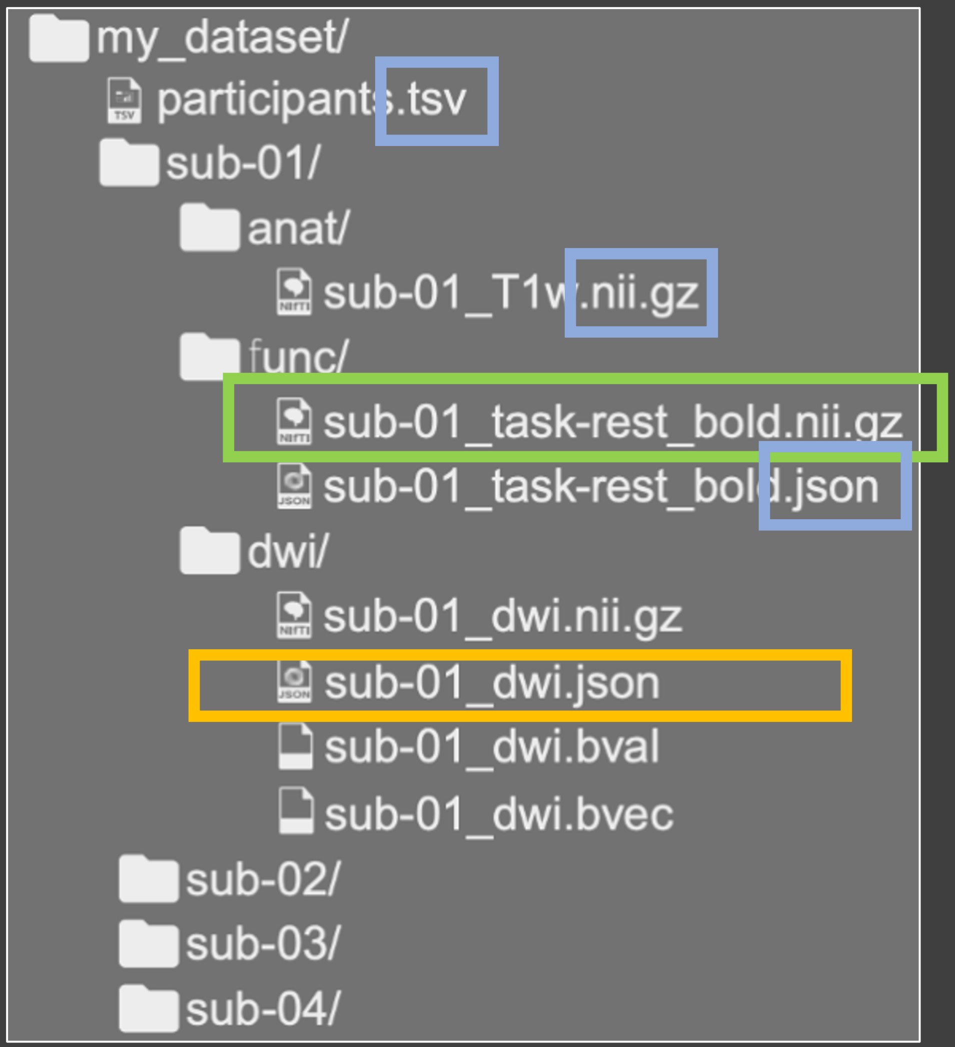
unorganized

BIDS

structured

BIDS in a nutshell

Directory structure



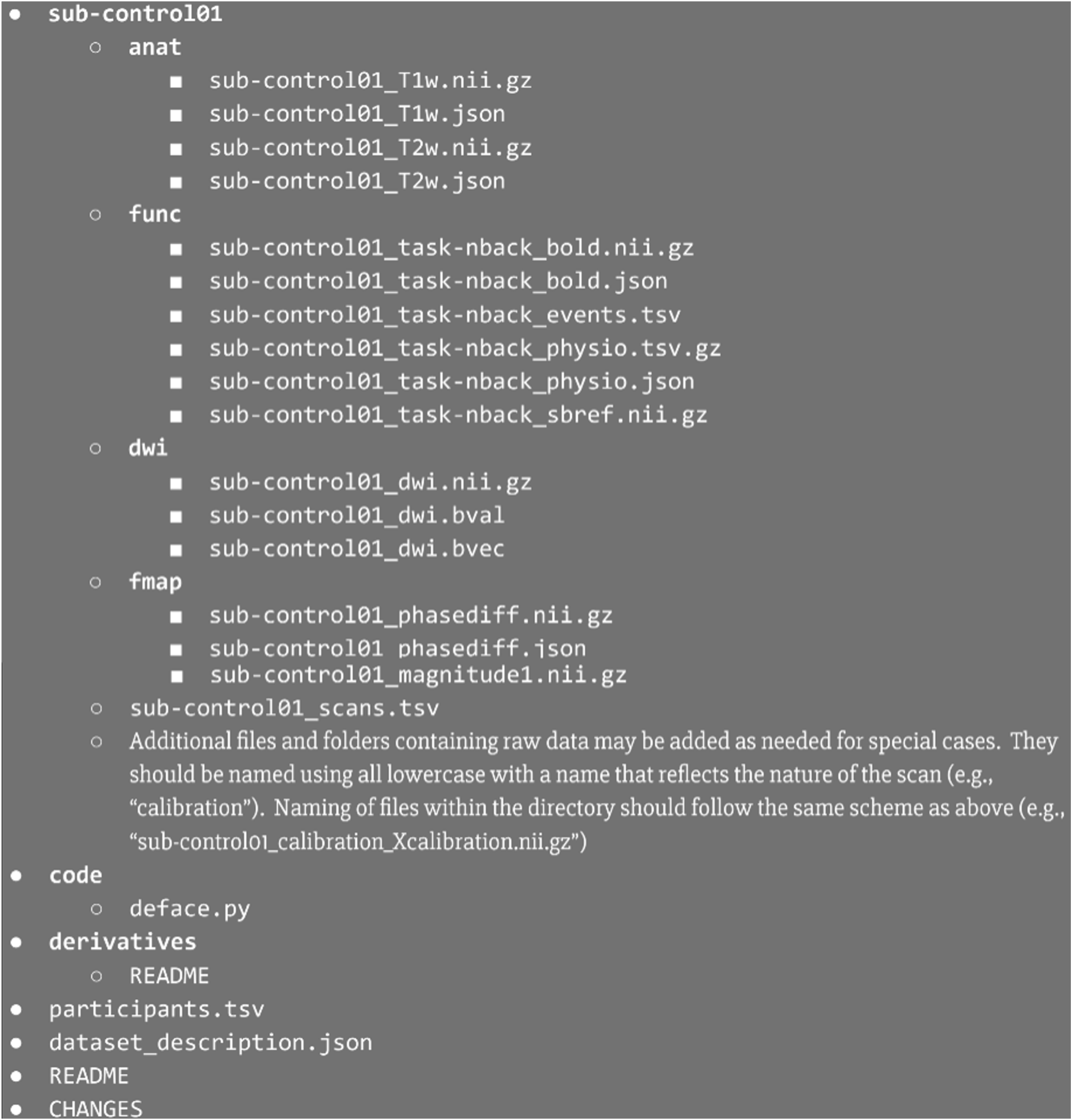
```
my_dataset/
    participants.tsv
    sub-01/
        anat/
            sub-01_T1w.nii.gz
        func/
            sub-01_task-rest_bold.nii.gz
            sub-01_task-rest_bold.json
        dwi/
            sub-01_dwi.nii.gz
            sub-01_dwi.json
            sub-01_dwi.bval
            sub-01_dwi.bvec
    sub-02/
    sub-03/
    sub-04/
```

Standardized data formats

Filename structure

JSON metadata sidecars

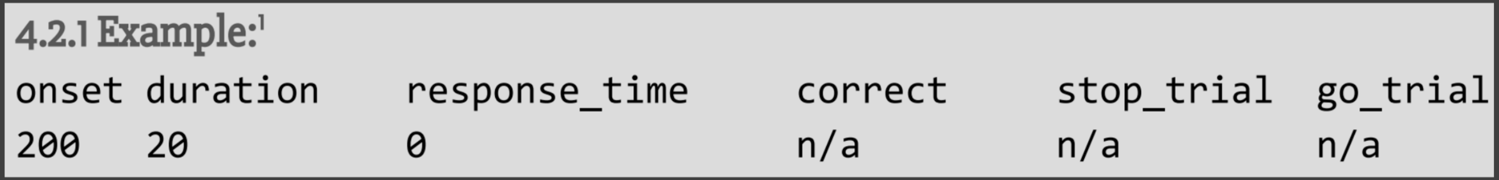
Multimodal example

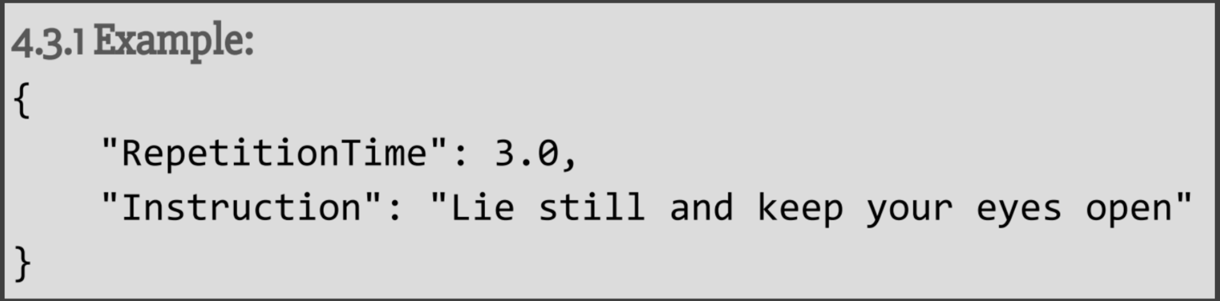


- **sub-control01**
  - **anat**
    - sub-control01_T1w.nii.gz
    - sub-control01_T1w.json
    - sub-control01_T2w.nii.gz
    - sub-control01_T2w.json
  - **func**
    - sub-control01_task-nback_bold.nii.gz
    - sub-control01_task-nback_bold.json
    - sub-control01_task-nback_events.tsv
    - sub-control01_task-nback_physio.tsv.gz
    - sub-control01_task-nback_physio.json
    - sub-control01_task-nback_sbref.nii.gz
  - **dwi**
    - sub-control01_dwi.nii.gz
    - sub-control01_dwi.bval
    - sub-control01_dwi.bvec
  - **fmap**
    - sub-control01_phasediff.nii.gz
    - sub-control01_phasediff.json
    - sub-control01_magnitude1.nii.gz
  - sub-control01_scans.tsv
  - Additional files and folders containing raw data may be added as needed for special cases. They should be named using all lowercase with a name that reflects the nature of the scan (e.g., "calibration"). Naming of files within the directory should follow the same scheme as above (e.g., "sub-control01_calibration_Xcalibration.nii.gz")
- **code**
  - deface.py
- **derivatives**
  - README
- participants.tsv
- dataset_description.json
- README
- CHANGES

File types in BIDS

Imaging files	All imaging data MUST be stored using the NIFTI file format.	Header + image cube

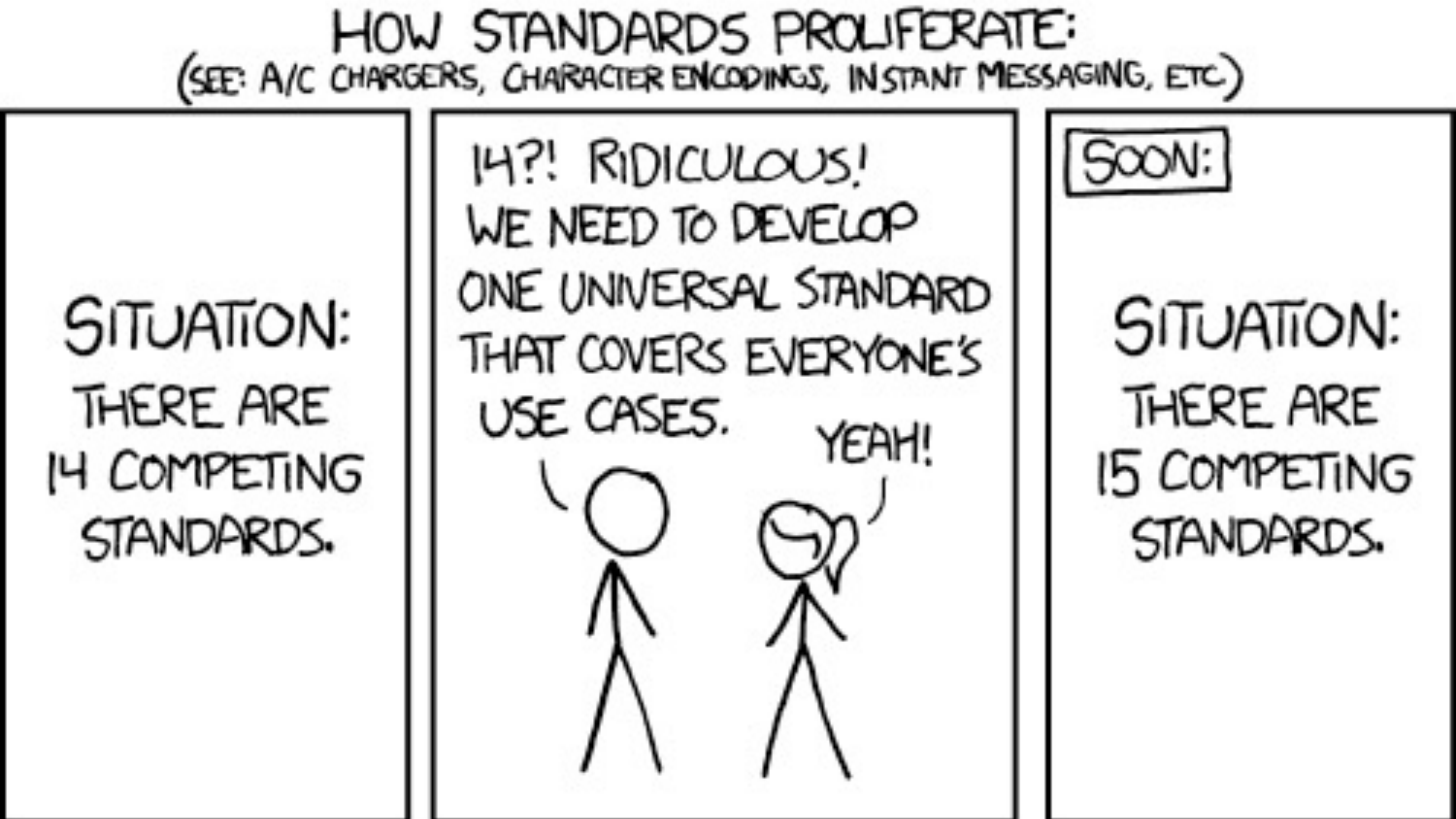
Tabular files	Tabular data MUST be saved as tab delimited values (.tsv) files
	Tabular files MAY be optionally accompanied by a simple data dictionary in a JSON format
		
		4.2.1 Example:[1]
		`onset duration response_time correct stop_trial go_trial`
		`200 20 0 n/a n/a n/a`

Key/value files (dictionaries)	JavaScript Object Notation (JSON) files MUST be used for storing key/value pairs.	
		4.3.1 Example:
		`{`
		`    "RepetitionTime": 3.0,`
		`    "Instruction": "Lie still and keep your eyes open"`
		`}`

BIDS comp. model extension Princeton meeting

Two broad classes of extensions would be needed:

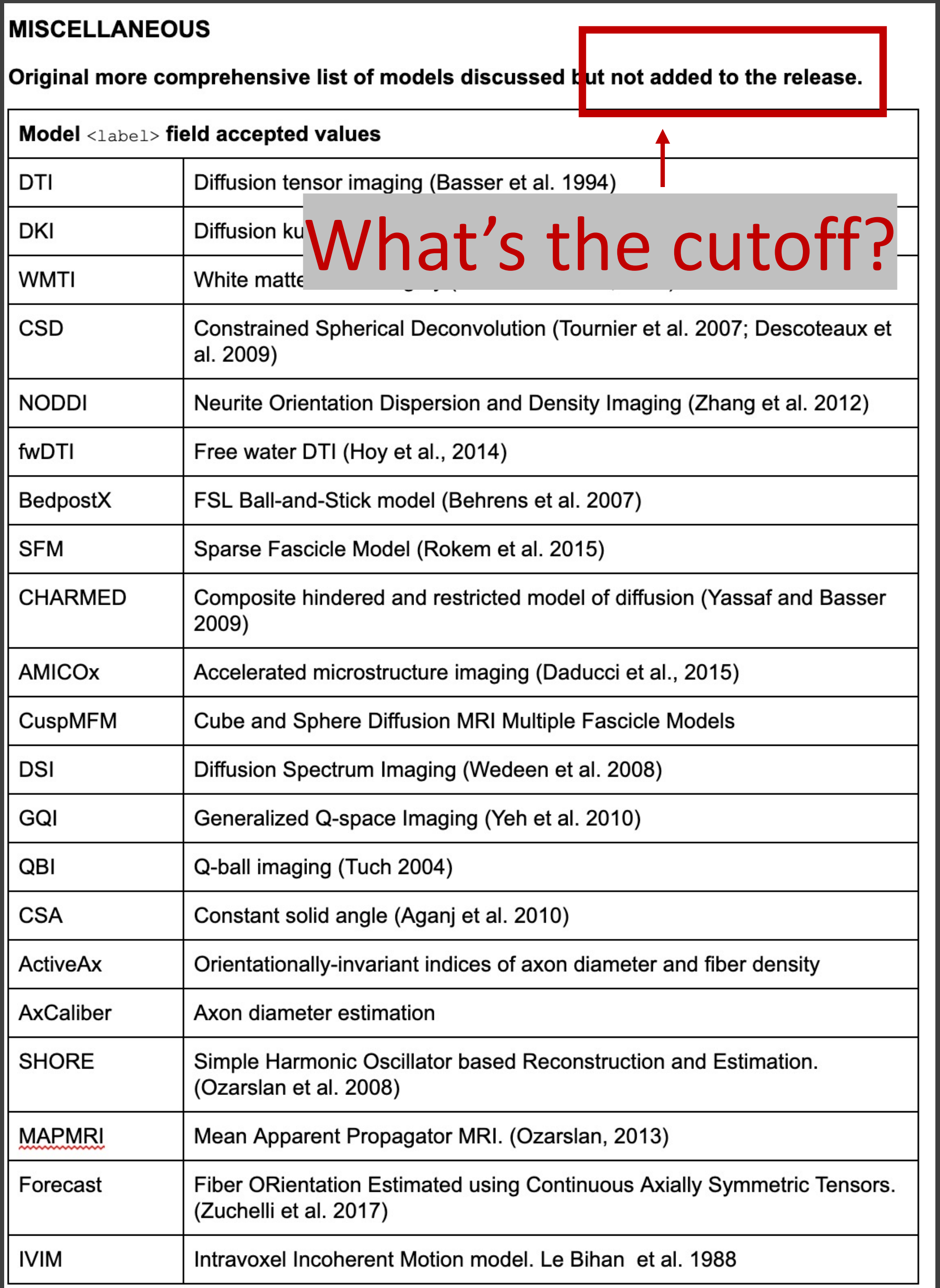
- Input and output data for various classes of models
- Model-specific language
    - Enormous potential advantages
        - ”automatic” implementation of the same model in different environments
        - Easier inspection and comparison with other models due to common syntax
    - Barriers
        - Balancing expressivity against simplicity: can a compact specification capture the full breadth of computational models?


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

```
sub-<participant_label>/[ses-<session_label>/]
    anat/
        sub-<participant_label>[_ses-
        <session_label>][_acq-<label>][_rec-
        <label>][_fa-<index>][_inv-<index>][_echo-
        <index>][_part-<phase|mag>][_run-
        <index>]_<sequence_label>.nii[.gz]
```

That's a very long filename!

That's a very long list of diffusion models!



MISCELLANEOUS

Original more comprehensive list of models discussed but not added to the release.

Model `<label>` **field accepted values**	
DTI	Diffusion tensor imaging (Basser et al. 1994)
DKI	Diffusion ku
WMTI	White matte
CSD	Constrained Spherical Deconvolution (Tournier et al. 2007; Descoteaux et al. 2009)
NODDI	Neurite Orientation Dispersion and Density Imaging (Zhang et al. 2012)
fwDTI	Free water DTI (Hoy et al., 2014)
BedpostX	FSL Ball-and-Stick model (Behrens et al. 2007)
SFM	Sparse Fascicle Model (Rokem et al. 2015)
CHARMED	Composite hindered and restricted model of diffusion (Yassaf and Basser 2009)
AMICOx	Accelerated microstructure imaging (Daducci et al., 2015)
CuspMFM	Cube and Sphere Diffusion MRI Multiple Fascicle Models
DSI	Diffusion Spectrum Imaging (Wedeen et al. 2008)
GQI	Generalized Q-space Imaging (Yeh et al. 2010)
QBI	Q-ball imaging (Tuch 2004)
CSA	Constant solid angle (Aganj et al. 2010)
ActiveAx	Orientationally-invariant indices of axon diameter and fiber density
AxCaliber	Axon diameter estimation
SHORE	Simple Harmonic Oscillator based Reconstruction and Estimation. (Ozarslan et al. 2008)
MAPMRI	Mean Apparent Propagator MRI. (Ozarslan, 2013)
Forecast	Fiber ORientation Estimated using Continuous Axially Symmetric Tensors. (Zuchelli et al. 2017)
IVIM	Intravoxel Incoherent Motion model. Le Bihan et al. 1988

What's the cutoff?

Desired features

- **avoiding over-specialization** to not end up with hundreds of file types, key-value pairs and “sub-standards”
  - **general applicability**: not only for TVB
- **”Built-in” support for**
  - **reproducibility**
    - explicit specification of the mathematical equations, the (physical) concepts, the particular software and implementations used for producing the result, including function definitions, algorithms, parameters and variable settings
  - **version control**
  - **provenance tracking**

Principles

- **Simple and generic data types and formats:** Tuning key-value pairs / metadata towards specific software products or frameworks is in contrast to the idea of having a generic standard.
  - It’s better if new software adapts to existing standards instead of creating new standards
  - The standard shouldn’t need to be actively modified everytime a new piece is added to the scientific framework
- **Short filenames**: computational models have many parameters. When files are distinguished based on long lists of characteristics, the defining characteristic will be buried in a swarm of key-value pairs, which makes visual parsing hard.
- **Domain-independent language (LEMS/NeuroML)** for expressing mathematical models of (physical) systems enables automatic high-performance code generation (exists in TVB)
- Make the data model (BIDS) agnostic of the metadata model (e.g. openMINDS), there is likely no “one-size-fits-all” solution

Suggested data types

Simple entities support many different use cases:

- spatial & temporal coordinate systems
- network graphs
- data vectors and matrices
  - time series
  - spatial objects
- mathematical equations and their physical interpretation
- computer code.

Suggested data types

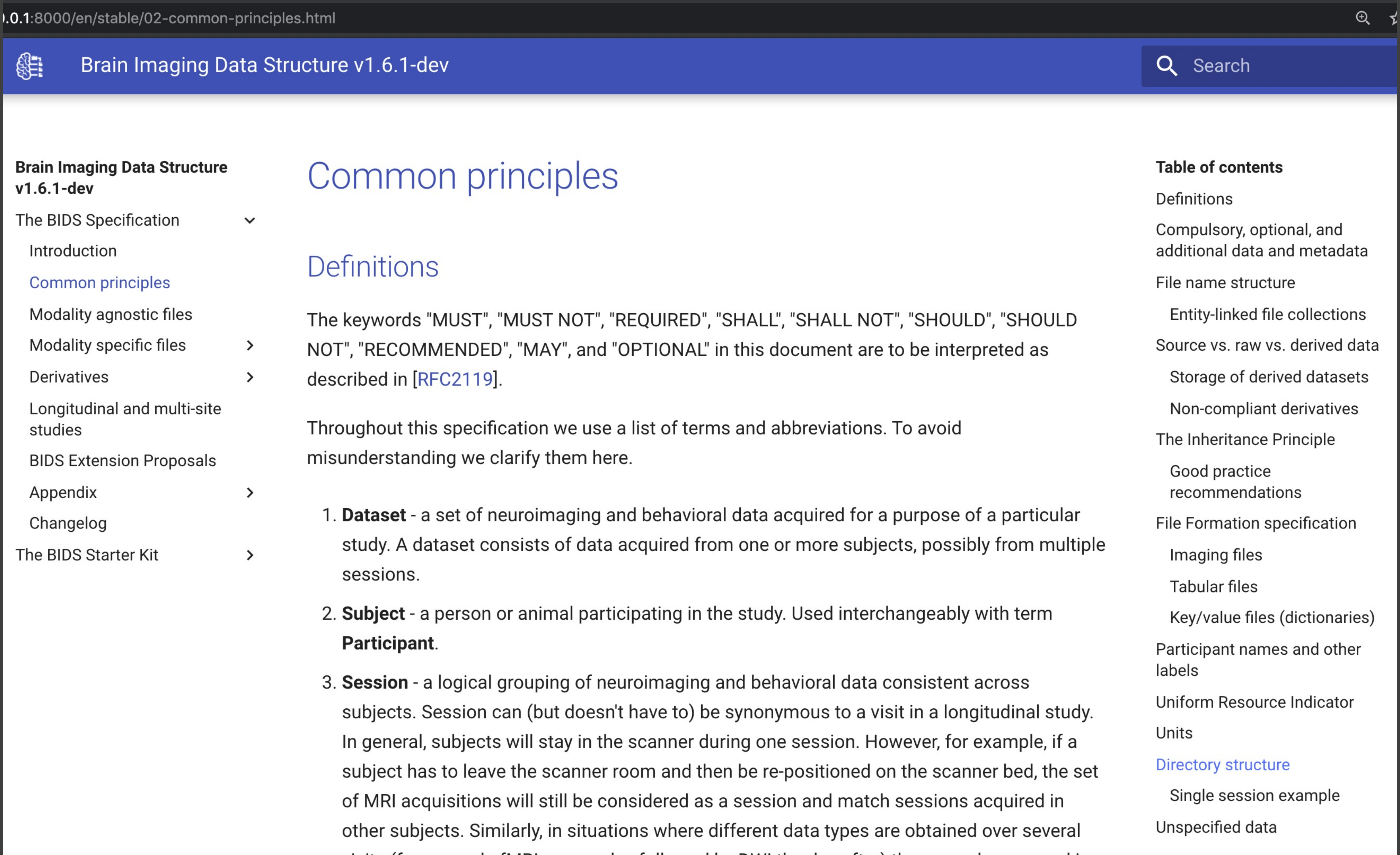
Simple entities support many different use cases:

TSV and JSON files

- spatial & temporal coordinate systems
- network graphs
- data vectors and matrices
  - time series
  - spatial objects

XML (LEMS/NeuroML)

- mathematical equations and their physical interpretation
- computer code.



Common principles

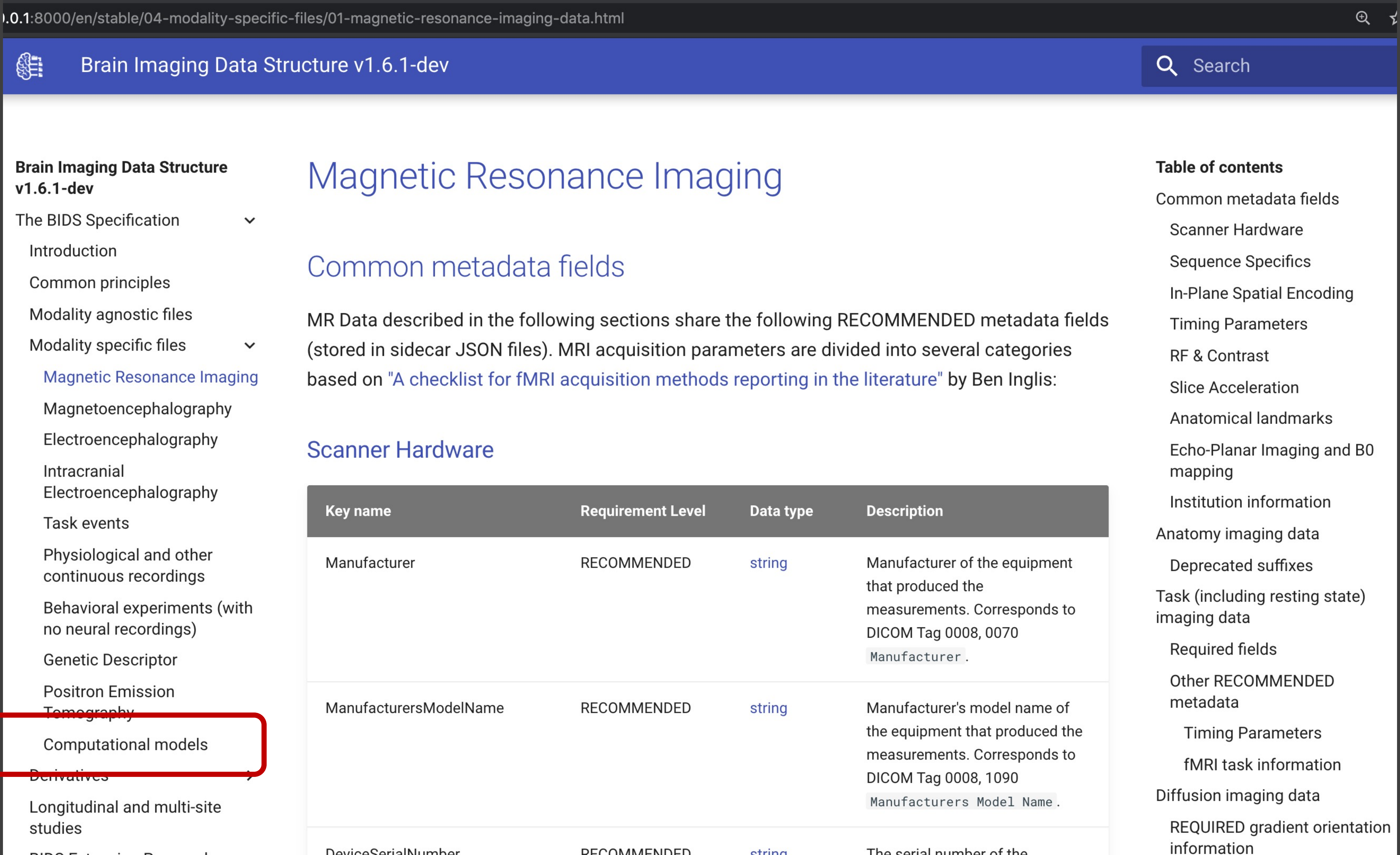
Definitions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Throughout this specification we use a list of terms and abbreviations. To avoid misunderstanding we clarify them here.

1. **Dataset** - a set of neuroimaging and behavioral data acquired for a purpose of a particular study. A dataset consists of data acquired from one or more subjects, possibly from multiple sessions.
2. **Subject** - a person or animal participating in the study. Used interchangeably with term **Participant**.
3. **Session** - a logical grouping of neuroimaging and behavioral data consistent across subjects. Session can (but doesn't have to) be synonymous to a visit in a longitudinal study. In general, subjects will stay in the scanner during one session. However, for example, if a subject has to leave the scanner room and then be re-positioned on the scanner bed, the set of MRI acquisitions will still be considered as a session and match sessions acquired in other subjects. Similarly, in situations where different data types are obtained over several

Table of contents



).0.1:8000/en/stable/04-modality-specific-files/01-magnetic-resonance-imaging-data.html

Brain Imaging Data Structure v1.6.1-dev

Search

Brain Imaging Data Structure v1.6.1-dev

The BIDS Specification

Introduction

Common principles

Modality agnostic files

Modality specific files

Magnetic Resonance Imaging

Magnetoencephalography

Electroencephalography

Intracranial Electroencephalography

Task events

Physiological and other continuous recordings

Behavioral experiments (with no neural recordings)

Genetic Descriptor

Positron Emission Tomography

Computational models

Derivatives

Longitudinal and multi-site studies

Magnetic Resonance Imaging

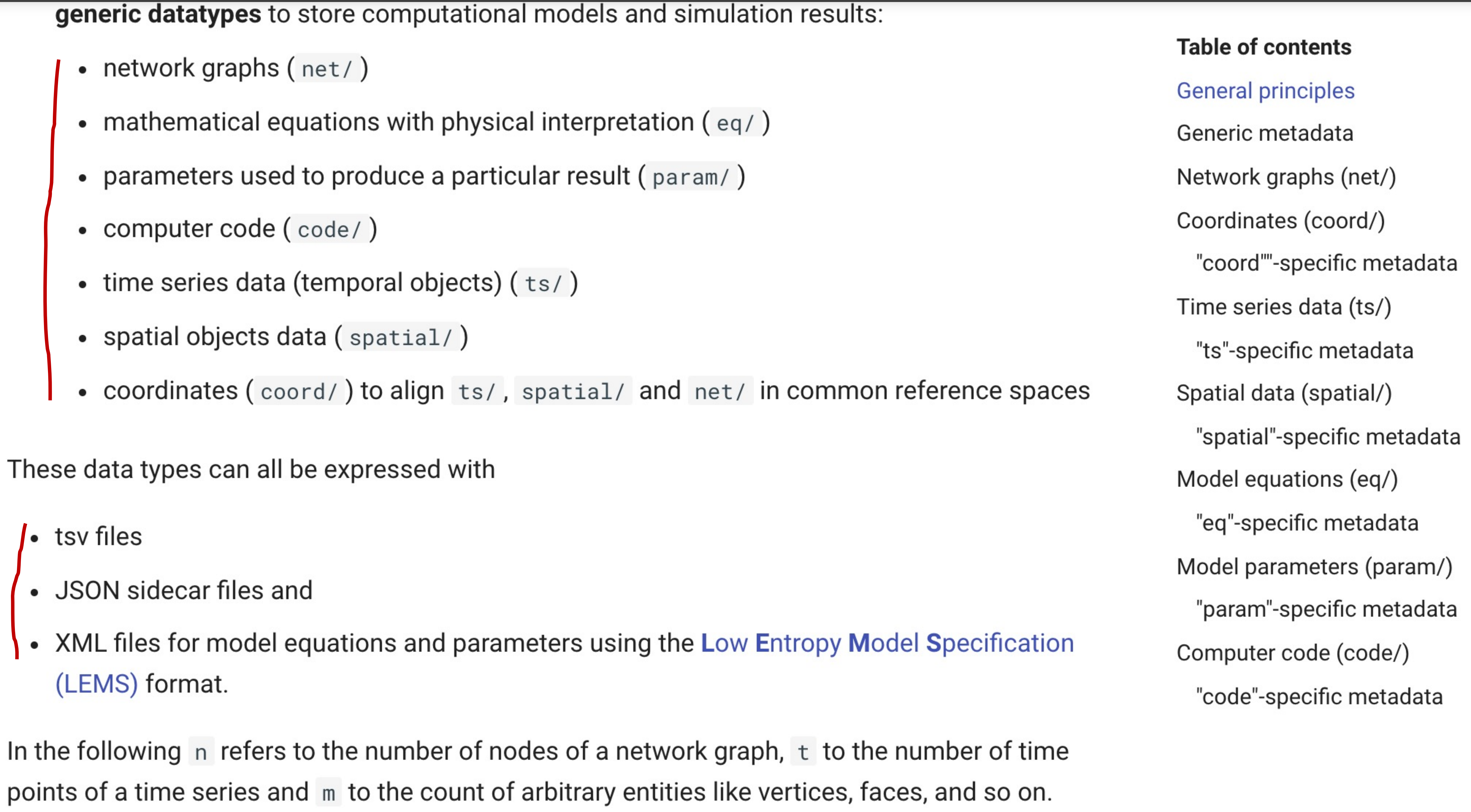
Common metadata fields

MR Data described in the following sections share the following RECOMMENDED metadata fields (stored in sidecar JSON files). MRI acquisition parameters are divided into several categories based on "A checklist for fMRI acquisition methods reporting in the literature" by Ben Inglis:

Scanner Hardware

Key name	Requirement Level	Data type	Description
Manufacturer	RECOMMENDED	string	Manufacturer of the equipment that produced the measurements. Corresponds to DICOM Tag 0008, 0070 `Manufacturer`.
ManufacturersModelName	RECOMMENDED	string	Manufacturer's model name of the equipment that produced the measurements. Corresponds to DICOM Tag 0008, 1090 `Manufacturers Model Name`.
DeviceSerialNumber	RECOMMENDED	string	The serial number of the

Table of contents



generic datatypes to store computational models and simulation results:

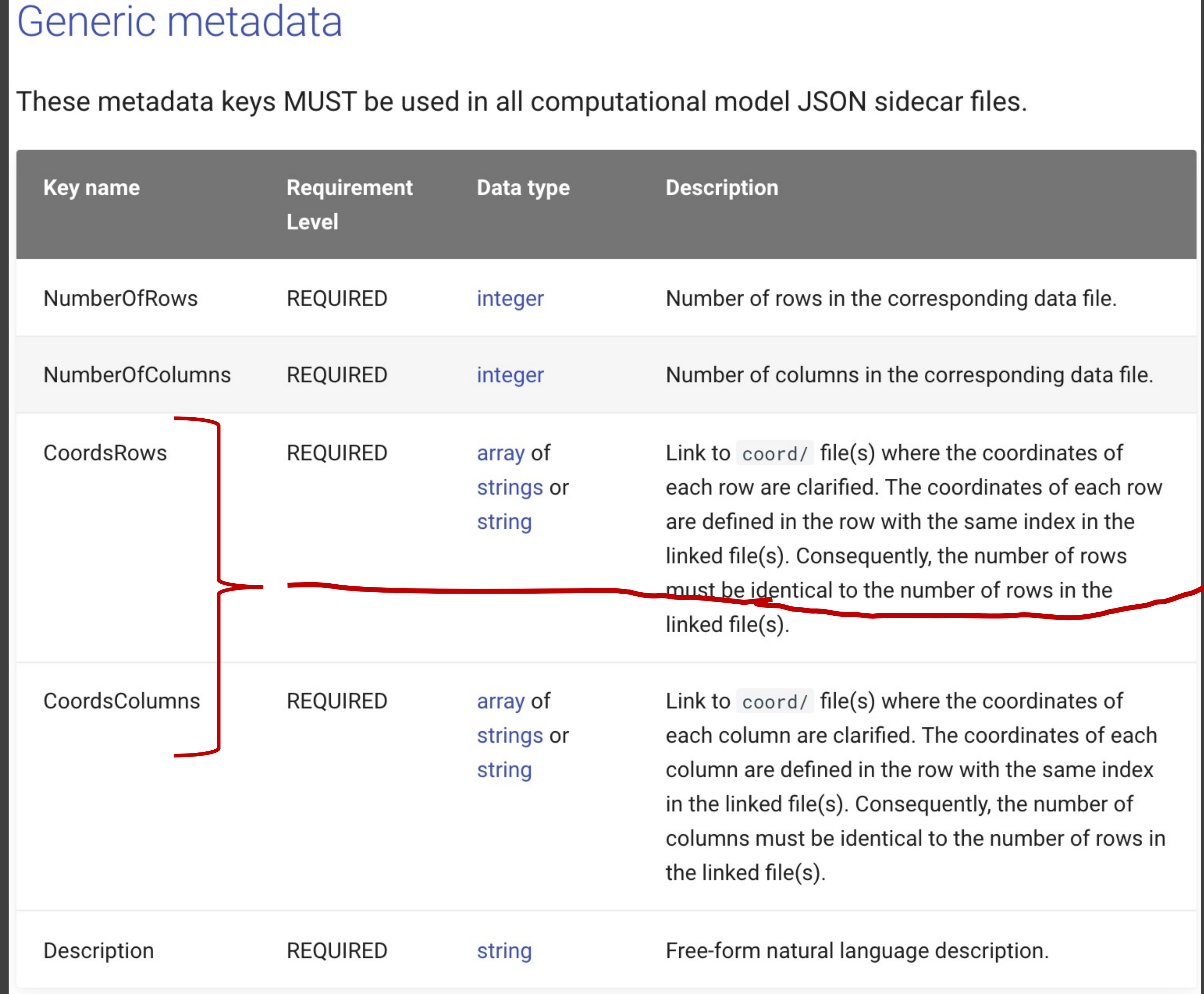
- network graphs (`net/`)
- mathematical equations with physical interpretation (`eq/`)
- parameters used to produce a particular result (`param/`)
- computer code (`code/`)
- time series data (temporal objects) (`ts/`)
- spatial objects data (`spatial/`)
- coordinates (`coord/`) to align `ts/`, `spatial/` and `net/` in common reference spaces

These data types can all be expressed with

- tsv files
- JSON sidecar files and
- XML files for model equations and parameters using the **L**ow **E**ntropy **M**odel **S**pecification (LEMS) format.

In the following `n` refers to the number of nodes of a network graph, `t` to the number of time points of a time series and `m` to the count of arbitrary entities like vertices, faces, and so on.

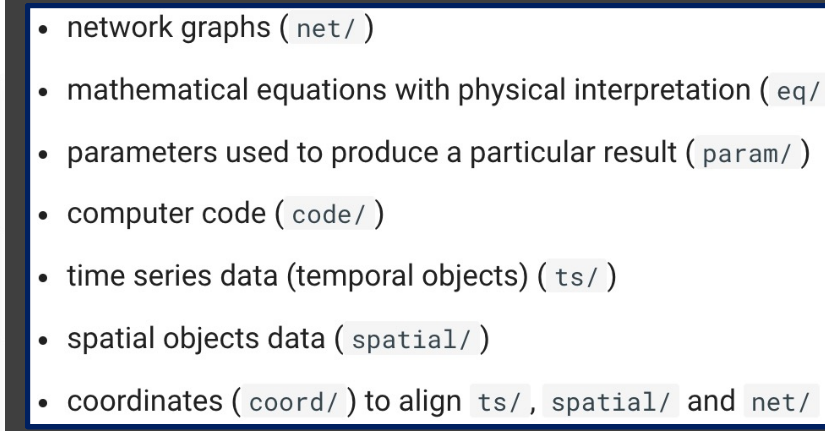
Table of contents



Generic metadata

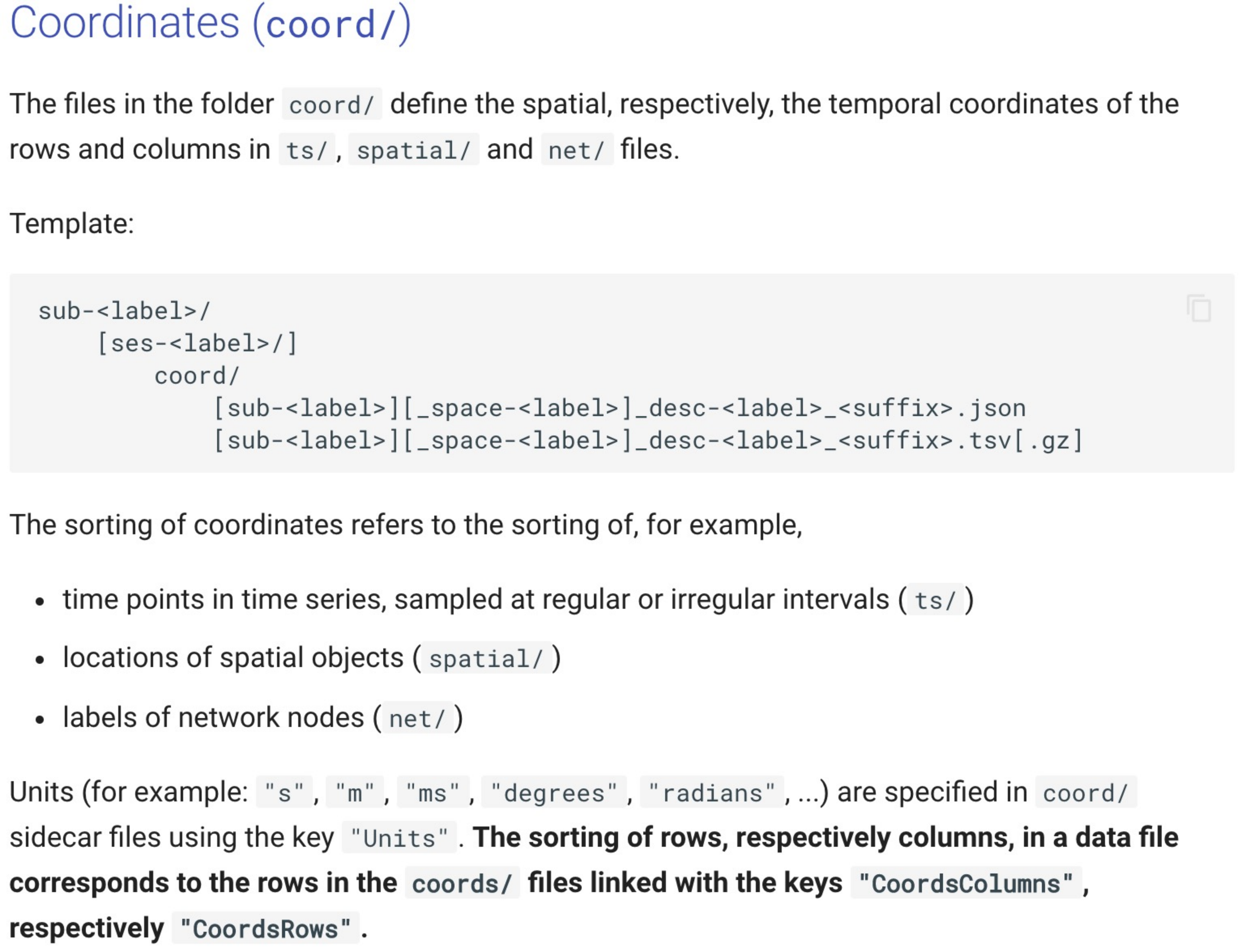
These metadata keys MUST be used in all computational model JSON sidecar files.

Key name	Requirement Level	Data type	Description
NumberOfRows	REQUIRED	integer	Number of rows in the corresponding data file.
NumberOfColumns	REQUIRED	integer	Number of columns in the corresponding data file.
CoordsRows	REQUIRED	array of strings or string	Link to `coord/` file(s) where the coordinates of each row are clarified. The coordinates of each row are defined in the row with the same index in the linked file(s). Consequently, the number of rows must be identical to the number of rows in the linked file(s).
CoordsColumns	REQUIRED	array of strings or string	Link to `coord/` file(s) where the coordinates of each column are clarified. The coordinates of each column are defined in the row with the same index in the linked file(s). Consequently, the number of columns must be identical to the number of rows in the linked file(s).
Description	REQUIRED	string	Free-form natural language description.



- network graphs (`net/`)
- mathematical equations with physical interpretation (`eq/`
- parameters used to produce a particular result (`param/`)
- computer code (`code/`)
- time series data (temporal objects) (`ts/`)
- spatial objects data (`spatial/`)
- coordinates (`coord/`) to align `ts/` , `spatial/` and `net/`

Coordinates (coord/)



Coordinates (coord/)

The files in the folder `coord/` define the spatial, respectively, the temporal coordinates of the rows and columns in `ts/`, `spatial/` and `net/` files.

Template:

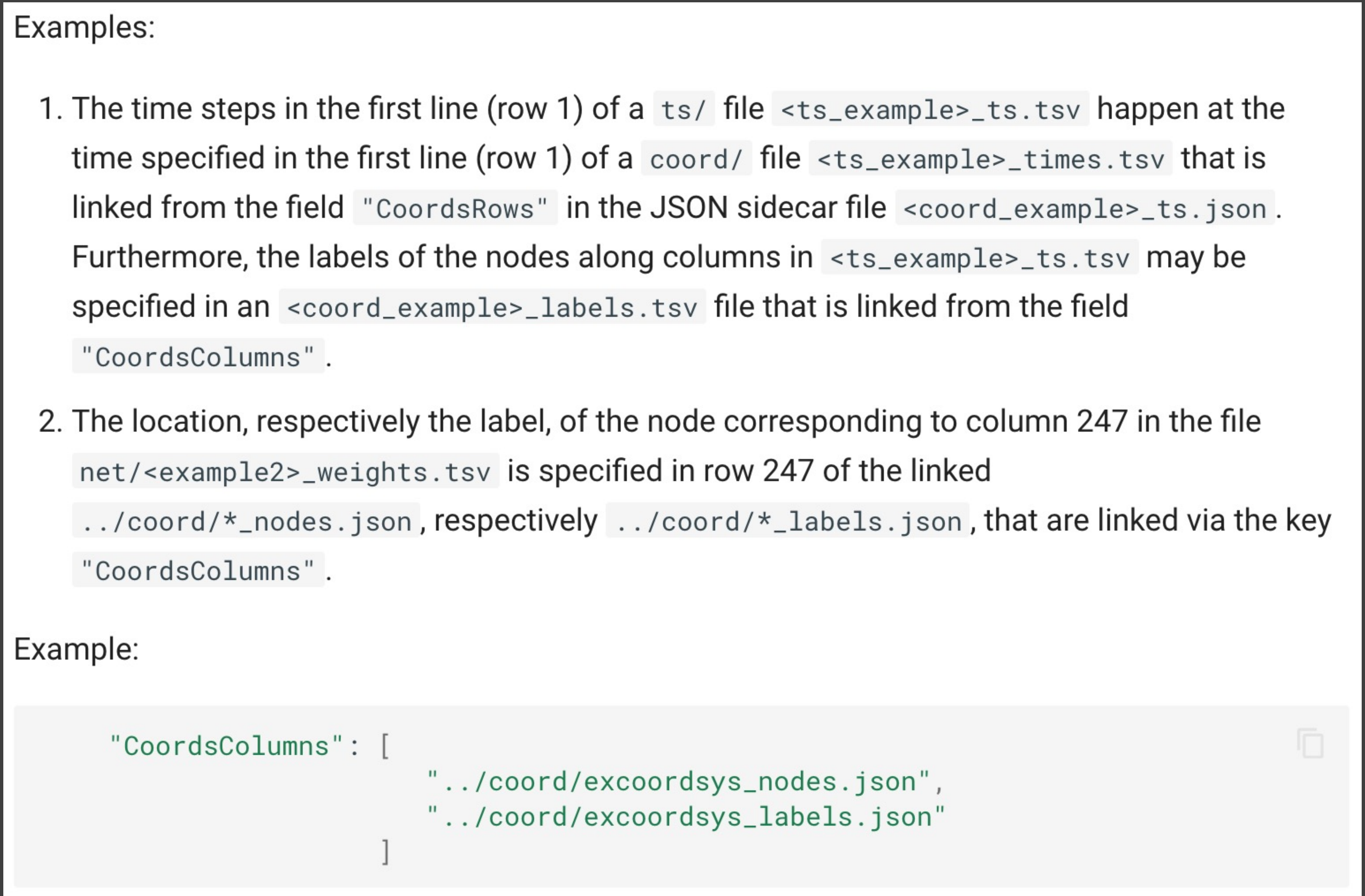
```
sub-<label>/
    [ses-<label>/]
        coord/
            [sub-<label>][_space-<label>]_desc-<label>_<suffix>.json
            [sub-<label>][_space-<label>]_desc-<label>_<suffix>.tsv[.gz]
```

The sorting of coordinates refers to the sorting of, for example,

- time points in time series, sampled at regular or irregular intervals (`ts/`)
- locations of spatial objects (`spatial/`)
- labels of network nodes (`net/`)

Units (for example: `"s"`, `"m"`, `"ms"`, `"degrees"`, `"radians"`, ...) are specified in `coord/` sidecar files using the key `"Units"`. **The sorting of rows, respectively columns, in a data file corresponds to the rows in the `coords/` files linked with the keys `"CoordsColumns"`, respectively `"CoordsRows"`.**

Coordinates (coord/)



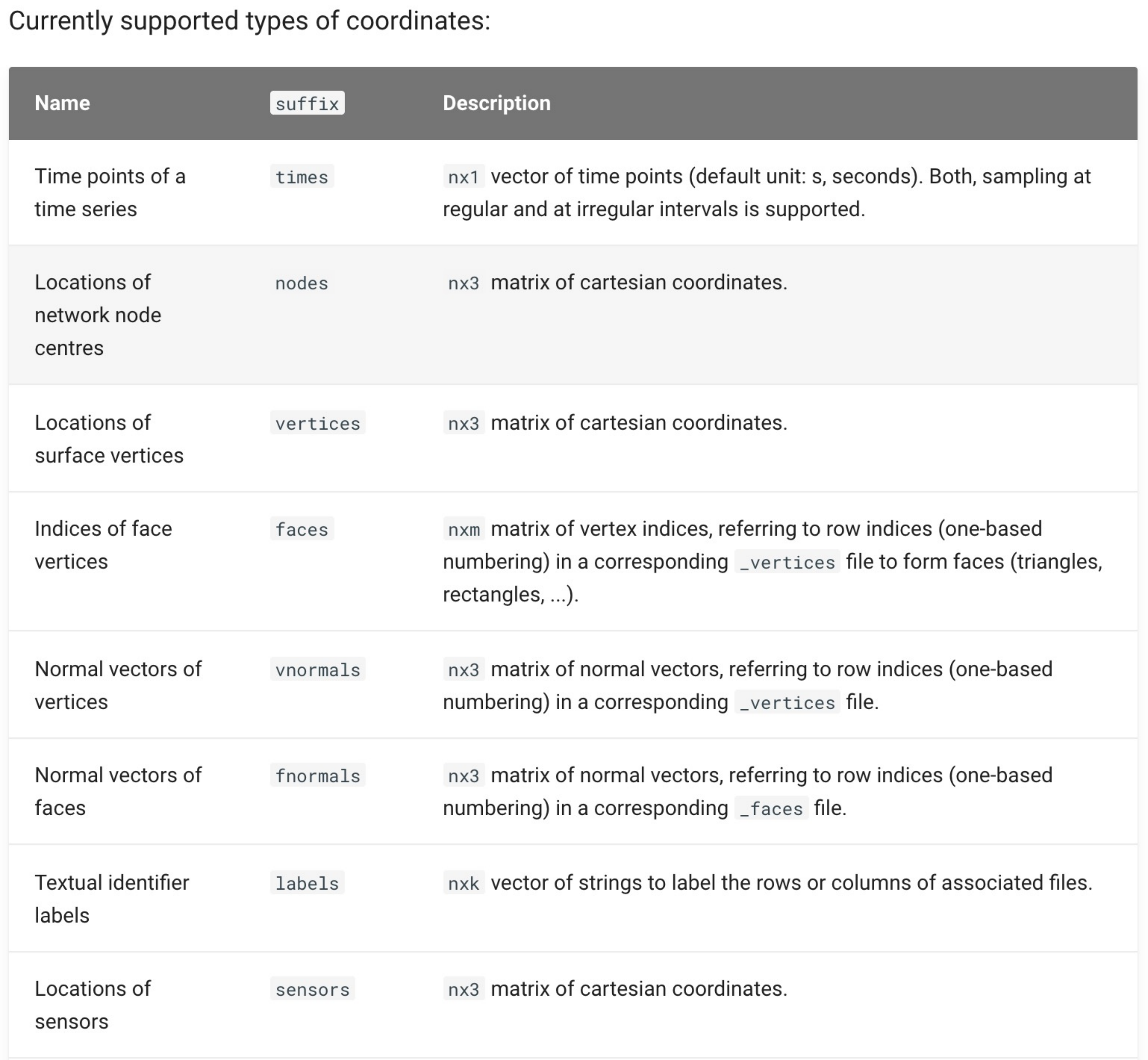
Examples:

1. The time steps in the first line (row 1) of a `ts/` file `<ts_example>_ts.tsv` happen at the time specified in the first line (row 1) of a `coord/` file `<ts_example>_times.tsv` that is linked from the field `"CoordsRows"` in the JSON sidecar file `<coord_example>_ts.json`. Furthermore, the labels of the nodes along columns in `<ts_example>_ts.tsv` may be specified in an `<coord_example>_labels.tsv` file that is linked from the field `"CoordsColumns"`.
2. The location, respectively the label, of the node corresponding to column 247 in the file `net/<example2>_weights.tsv` is specified in row 247 of the linked `../coord/*_nodes.json`, respectively `../coord/*_labels.json`, that are linked via the key `"CoordsColumns"`.

Example:

```
    "CoordsColumns": [
                        "../coord/excoordsys_nodes.json",
                        "../coord/excoordsys_labels.json"
                     ]
```

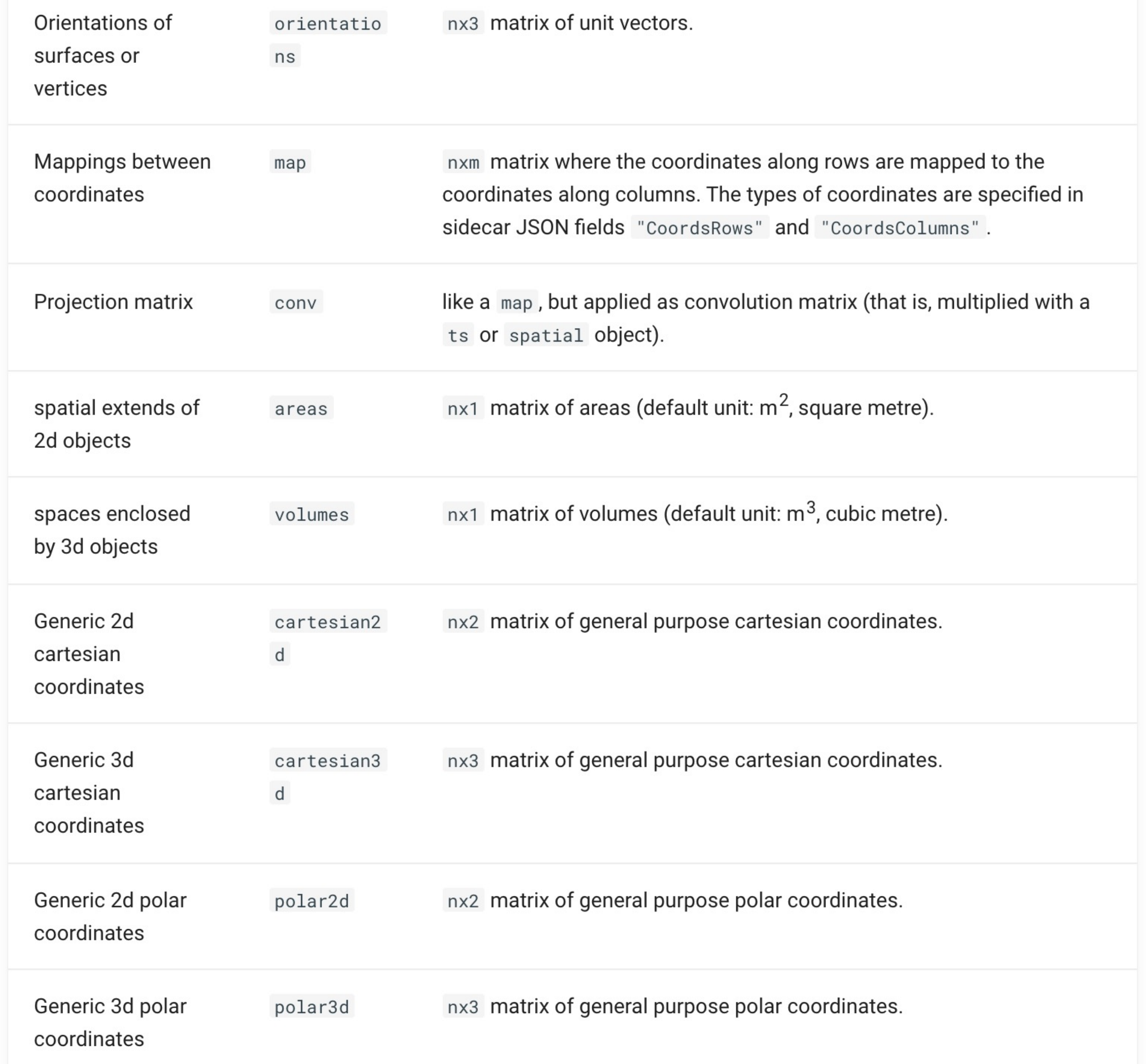
Coordinates (coord/)



Currently supported types of coordinates:

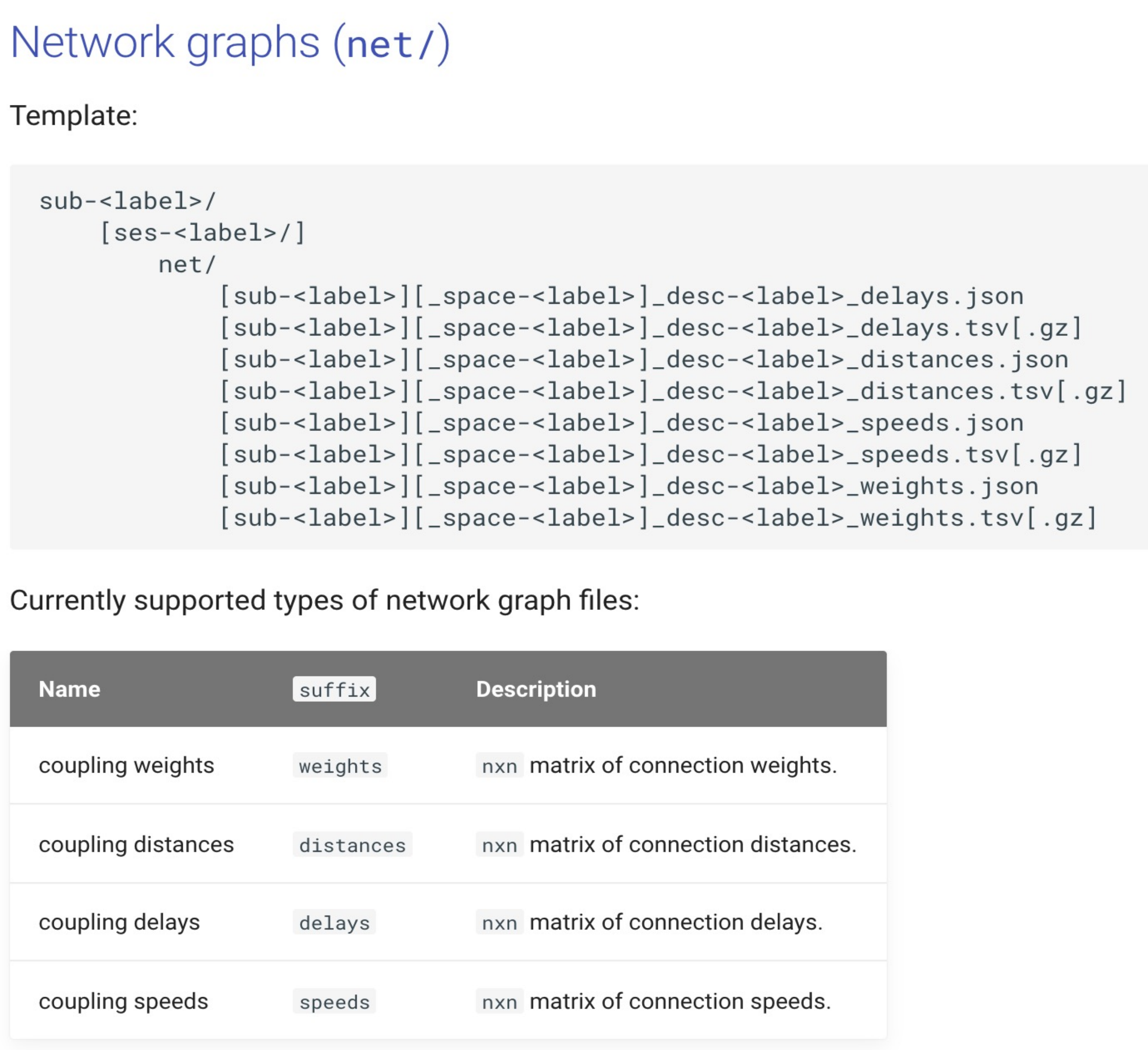
Name	`suffix`	Description
Time points of a time series	`times`	`nx1` vector of time points (default unit: s, seconds). Both, sampling at regular and at irregular intervals is supported.
Locations of network node centres	`nodes`	`nx3` matrix of cartesian coordinates.
Locations of surface vertices	`vertices`	`nx3` matrix of cartesian coordinates.
Indices of face vertices	`faces`	`nxm` matrix of vertex indices, referring to row indices (one-based numbering) in a corresponding `_vertices` file to form faces (triangles, rectangles, ...).
Normal vectors of vertices	`vnormals`	`nx3` matrix of normal vectors, referring to row indices (one-based numbering) in a corresponding `_vertices` file.
Normal vectors of faces	`fnormals`	`nx3` matrix of normal vectors, referring to row indices (one-based numbering) in a corresponding `_faces` file.
Textual identifier labels	`labels`	`nxk` vector of strings to label the rows or columns of associated files.
Locations of sensors	`sensors`	`nx3` matrix of cartesian coordinates.

Coordinates (coord/)



Orientations of surfaces or vertices	`orientations`	`nx3` matrix of unit vectors.
Mappings between coordinates	`map`	`nxm` matrix where the coordinates along rows are mapped to the coordinates along columns. The types of coordinates are specified in sidecar JSON fields `"CoordsRows"` and `"CoordsColumns"`.
Projection matrix	`conv`	like a `map`, but applied as convolution matrix (that is, multiplied with a `ts` or `spatial` object).
spatial extends of 2d objects	`areas`	`nx1` matrix of areas (default unit: m^2, square metre).
spaces enclosed by 3d objects	`volumes`	`nx1` matrix of volumes (default unit: m^3, cubic metre).
Generic 2d cartesian coordinates	`cartesian2d`	`nx2` matrix of general purpose cartesian coordinates.
Generic 3d cartesian coordinates	`cartesian3d`	`nx3` matrix of general purpose cartesian coordinates.
Generic 2d polar coordinates	`polar2d`	`nx2` matrix of general purpose polar coordinates.
Generic 3d polar coordinates	`polar3d`	`nx3` matrix of general purpose polar coordinates.

Network graphs (net/)



Network graphs (net/)

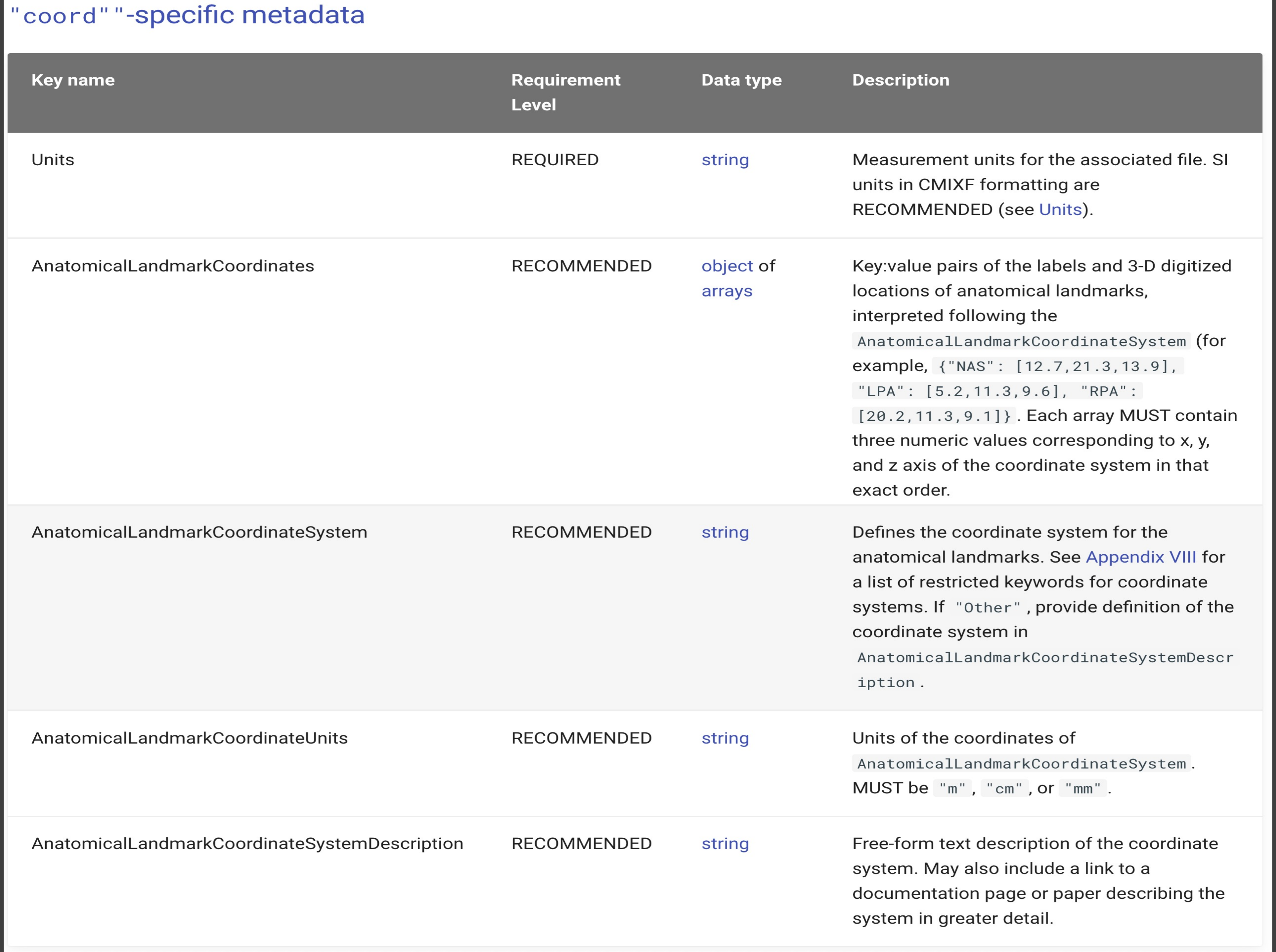
Template:

```
sub-<label>/
    [ses-<label>/]
        net/
            [sub-<label>][_space-<label>]_desc-<label>_delays.json
            [sub-<label>][_space-<label>]_desc-<label>_delays.tsv[.gz]
            [sub-<label>][_space-<label>]_desc-<label>_distances.json
            [sub-<label>][_space-<label>]_desc-<label>_distances.tsv[.gz]
            [sub-<label>][_space-<label>]_desc-<label>_speeds.json
            [sub-<label>][_space-<label>]_desc-<label>_speeds.tsv[.gz]
            [sub-<label>][_space-<label>]_desc-<label>_weights.json
            [sub-<label>][_space-<label>]_desc-<label>_weights.tsv[.gz]
```

Currently supported types of network graph files:

Name	suffix	Description
coupling weights	weights	nxn matrix of connection weights.
coupling distances	distances	nxn matrix of connection distances.
coupling delays	delays	nxn matrix of connection delays.
coupling speeds	speeds	nxn matrix of connection speeds.

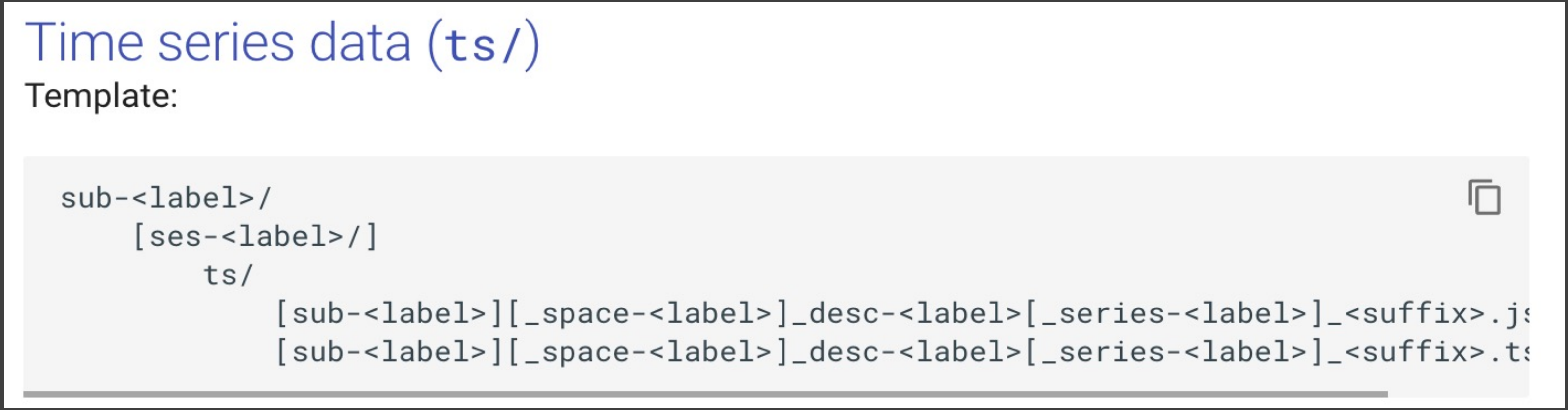
Coordinates (coord/)



"coord""-specific metadata

Key name	Requirement Level	Data type	Description
Units	REQUIRED	string	Measurement units for the associated file. SI units in CMIXF formatting are RECOMMENDED (see Units).
AnatomicalLandmarkCoordinates	RECOMMENDED	object of arrays	Key:value pairs of the labels and 3-D digitized locations of anatomical landmarks, interpreted following the `AnatomicalLandmarkCoordinateSystem` (for example, `{"NAS": [12.7,21.3,13.9], "LPA": [5.2,11.3,9.6], "RPA": [20.2,11.3,9.1]}`. Each array MUST contain three numeric values corresponding to x, y, and z axis of the coordinate system in that exact order.
AnatomicalLandmarkCoordinateSystem	RECOMMENDED	string	Defines the coordinate system for the anatomical landmarks. See Appendix VIII for a list of restricted keywords for coordinate systems. If `"Other"`, provide definition of the coordinate system in `AnatomicalLandmarkCoordinateSystemDescription`.
AnatomicalLandmarkCoordinateUnits	RECOMMENDED	string	Units of the coordinates of `AnatomicalLandmarkCoordinateSystem`. MUST be `"m"`, `"cm"`, or `"mm"`.
AnatomicalLandmarkCoordinateSystemDescription	RECOMMENDED	string	Free-form text description of the coordinate system. May also include a link to a documentation page or paper describing the system in greater detail.

Time series (ts/)

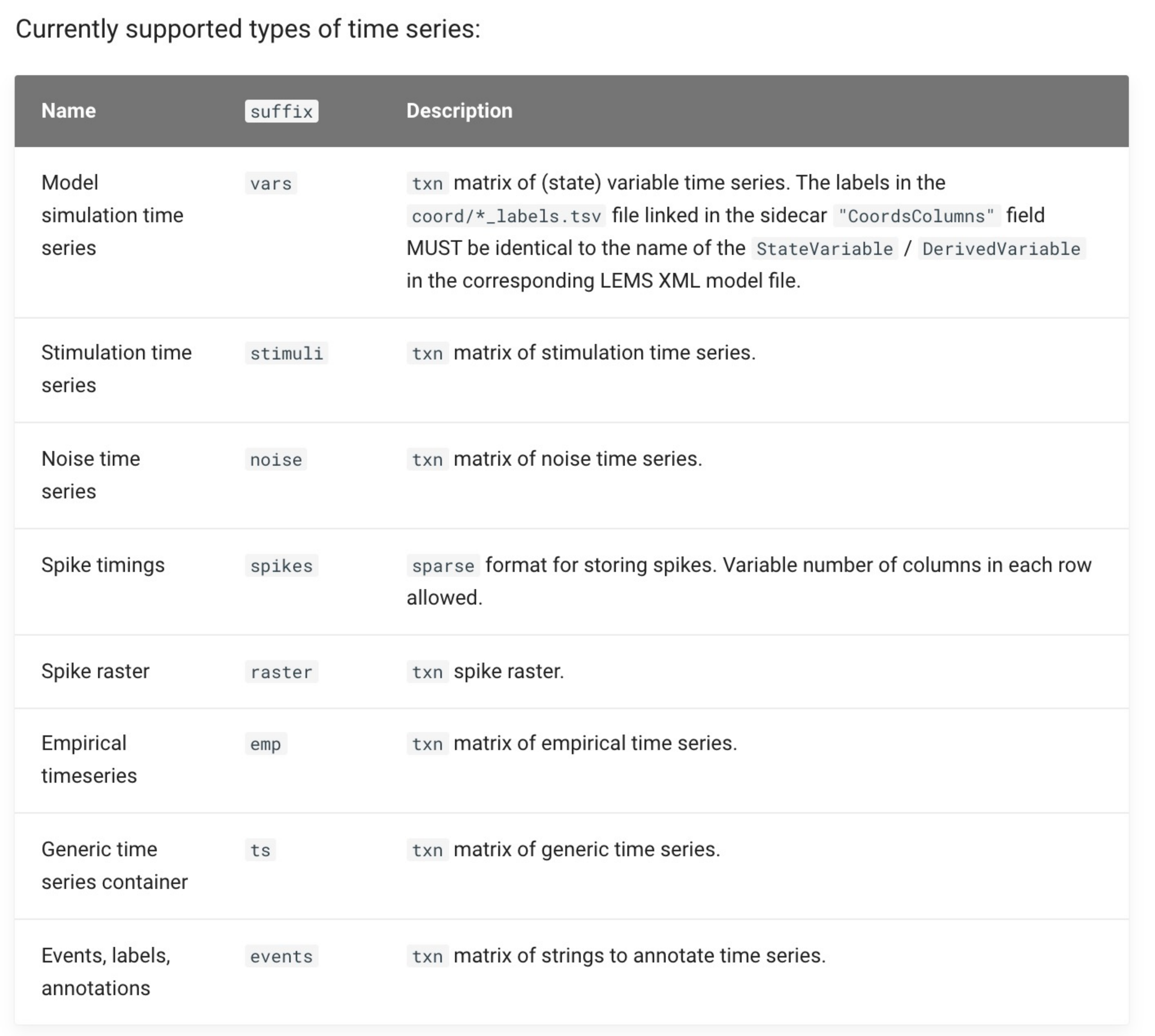


Time series data (ts/)

Template:

```
sub-<label>/
    [ses-<label>/]
        ts/
            [sub-<label>][_space-<label>]_desc-<label>[_series-<label>]_<suffix>.js
            [sub-<label>][_space-<label>]_desc-<label>[_series-<label>]_<suffix>.ts
```

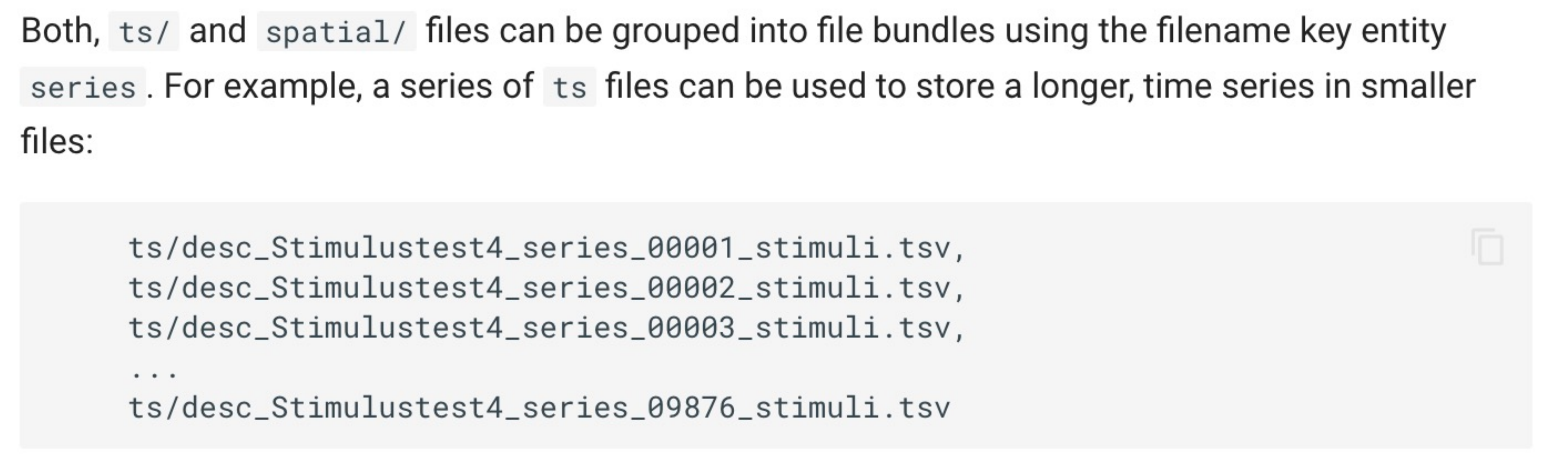
Time series (ts/)



Currently supported types of time series:

Name	`suffix`	Description
Model simulation time series	`vars`	`txn` matrix of (state) variable time series. The labels in the `coord/*_labels.tsv` file linked in the sidecar `"CoordsColumns"` field MUST be identical to the name of the `StateVariable` / `DerivedVariable` in the corresponding LEMS XML model file.
Stimulation time series	`stimuli`	`txn` matrix of stimulation time series.
Noise time series	`noise`	`txn` matrix of noise time series.
Spike timings	`spikes`	`sparse` format for storing spikes. Variable number of columns in each row allowed.
Spike raster	`raster`	`txn` spike raster.
Empirical timeseries	`emp`	`txn` matrix of empirical time series.
Generic time series container	`ts`	`txn` matrix of generic time series.
Events, labels, annotations	`events`	`txn` matrix of strings to annotate time series.

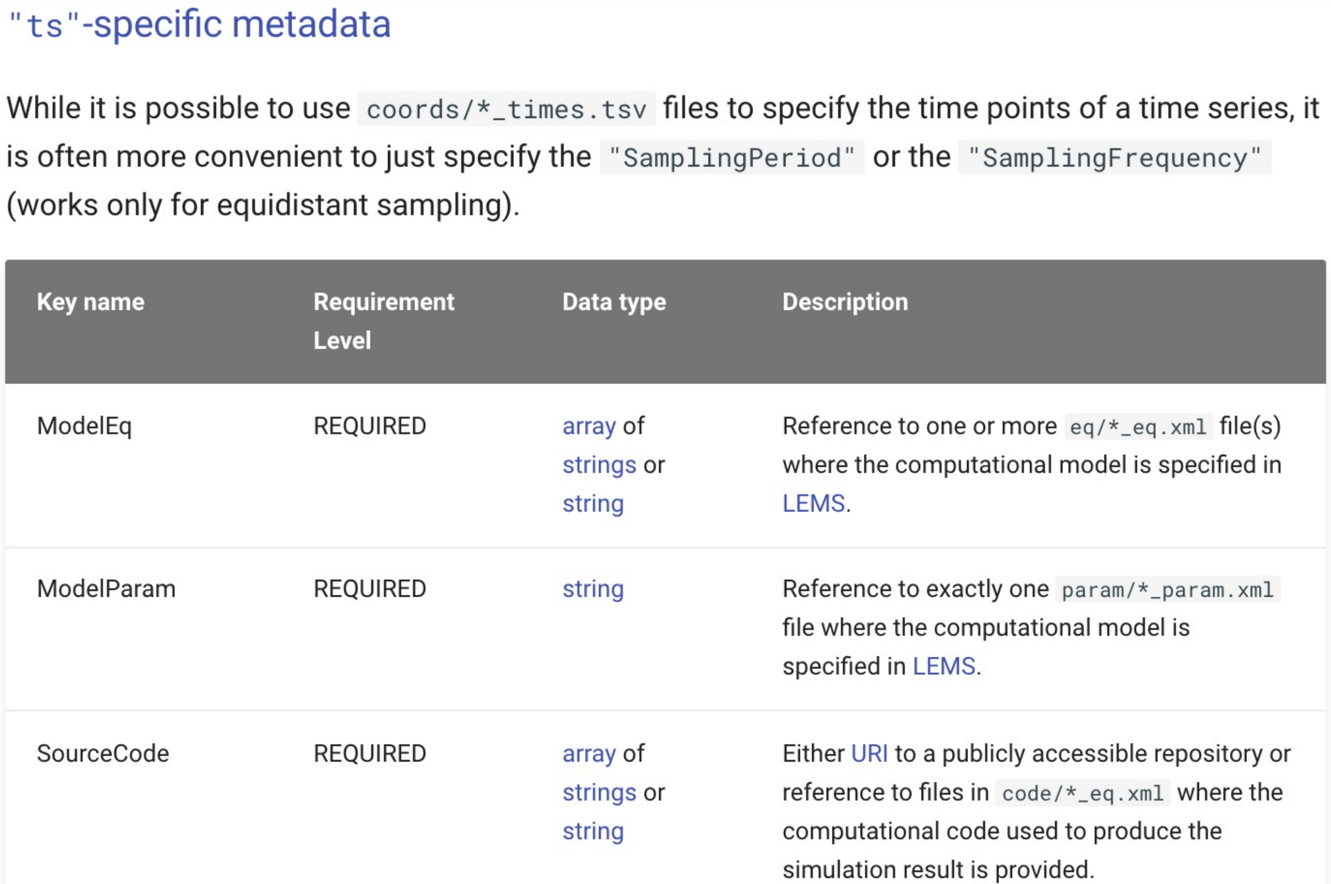
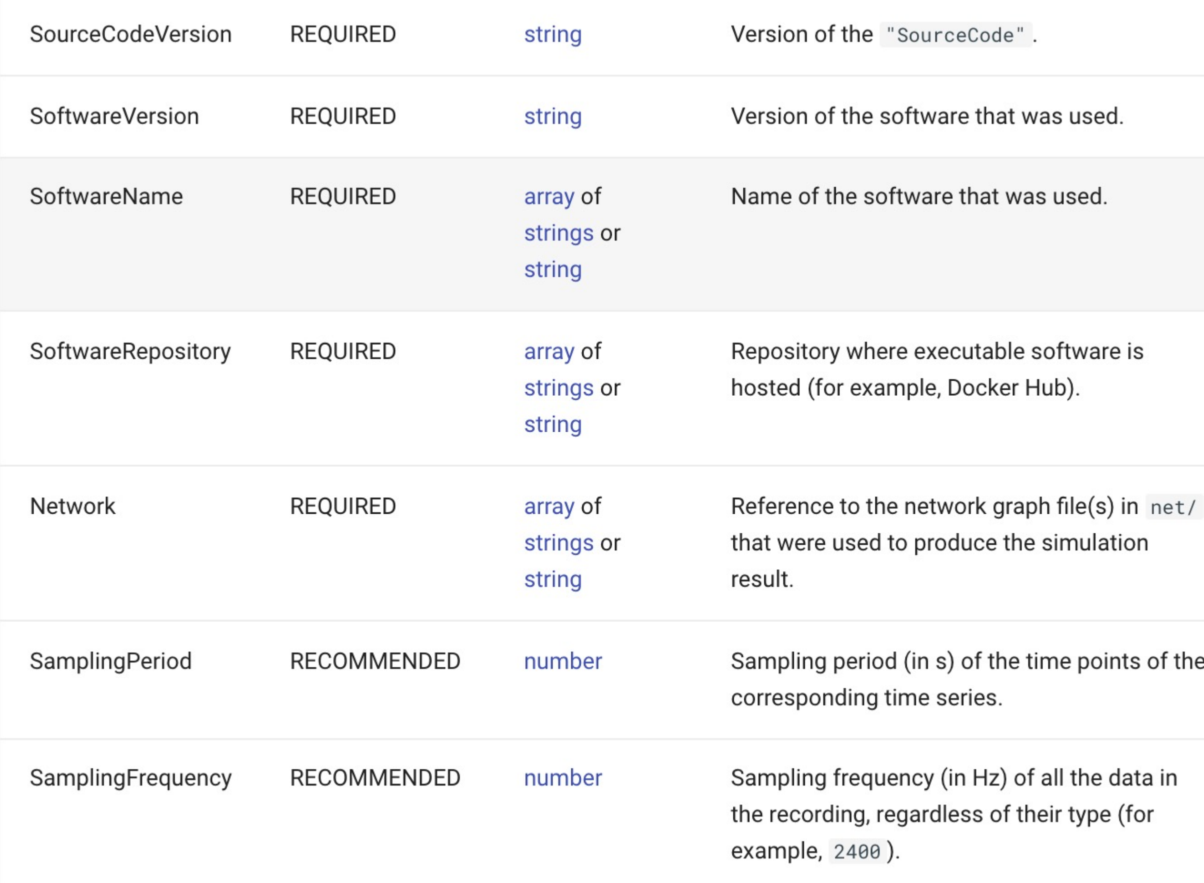
Time series (ts/)



Both, `ts/` and `spatial/` files can be grouped into file bundles using the filename key entity `series`. For example, a series of `ts` files can be used to store a longer, time series in smaller files:

```
ts/desc_Stimulustest4_series_00001_stimuli.tsv,
ts/desc_Stimulustest4_series_00002_stimuli.tsv,
ts/desc_Stimulustest4_series_00003_stimuli.tsv,
...
ts/desc_Stimulustest4_series_09876_stimuli.tsv
```

Time series (ts/)

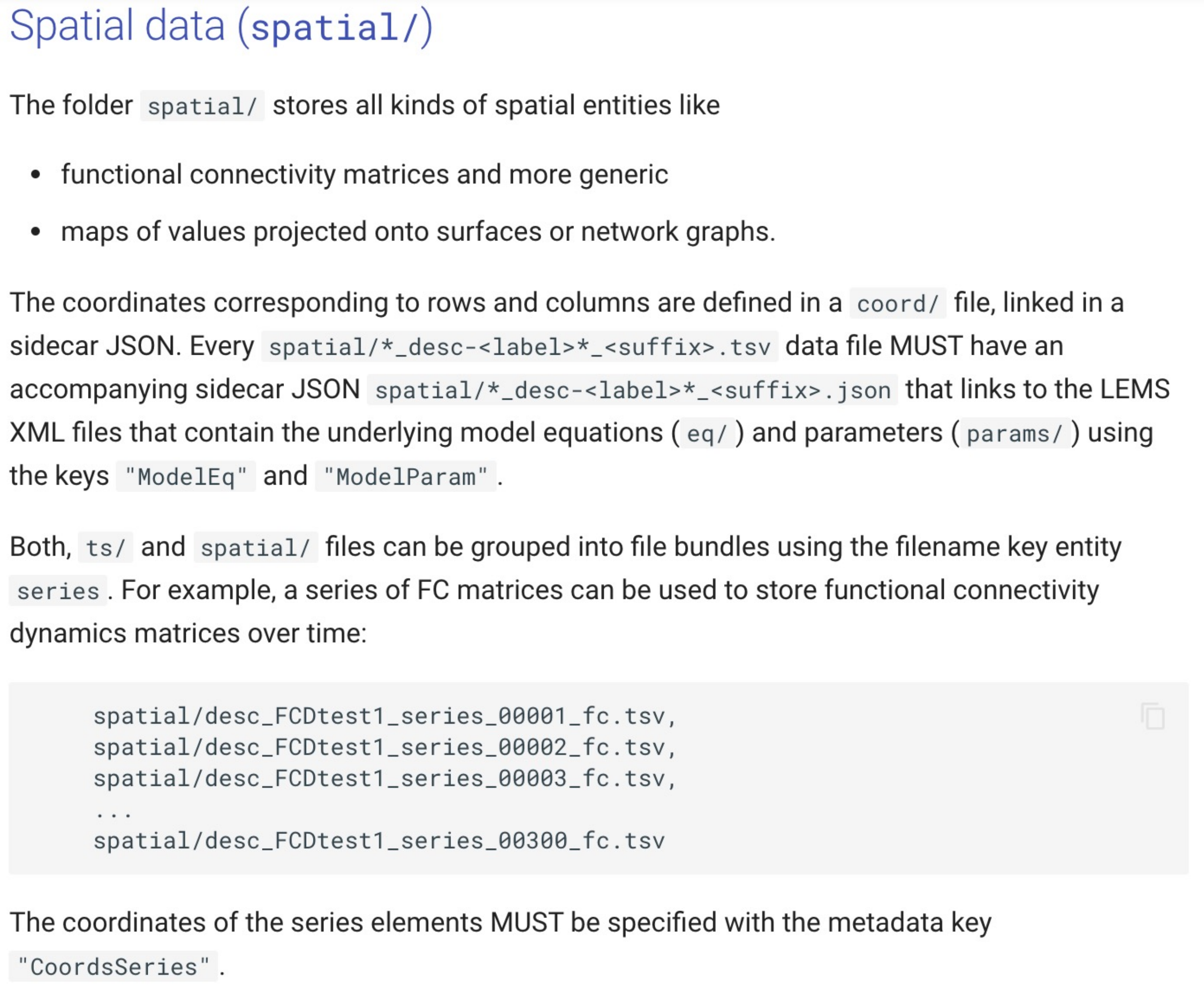



"ts"-specific metadata

While it is possible to use `coords/*_times.tsv` files to specify the time points of a time series, it is often more convenient to just specify the `"SamplingPeriod"` or the `"SamplingFrequency"` (works only for equidistant sampling).

Key name	Requirement Level	Data type	Description
ModelEq	REQUIRED	array of strings or string	Reference to one or more `eq/*_eq.xml` file(s) where the computational model is specified in LEMS.
ModelParam	REQUIRED	string	Reference to exactly one `param/*_param.xml` file where the computational model is specified in LEMS.
SourceCode	REQUIRED	array of strings or string	Either URI to a publicly accessible repository or reference to files in `code/*_eq.xml` where the computational code used to produce the simulation result is provided.
SourceCodeVersion	REQUIRED	string	Version of the `"SourceCode"`.
SoftwareVersion	REQUIRED	string	Version of the software that was used.
SoftwareName	REQUIRED	array of strings or string	Name of the software that was used.
SoftwareRepository	REQUIRED	array of strings or string	Repository where executable software is hosted (for example, Docker Hub).
Network	REQUIRED	array of strings or string	Reference to the network graph file(s) in `net/` that were used to produce the simulation result.
SamplingPeriod	RECOMMENDED	number	Sampling period (in s) of the time points of the corresponding time series.
SamplingFrequency	RECOMMENDED	number	Sampling frequency (in Hz) of all the data in the recording, regardless of their type (for example, `2400`).

Spatial data (spatial/)



Spatial data (`spatial/`)

The folder `spatial/` stores all kinds of spatial entities like

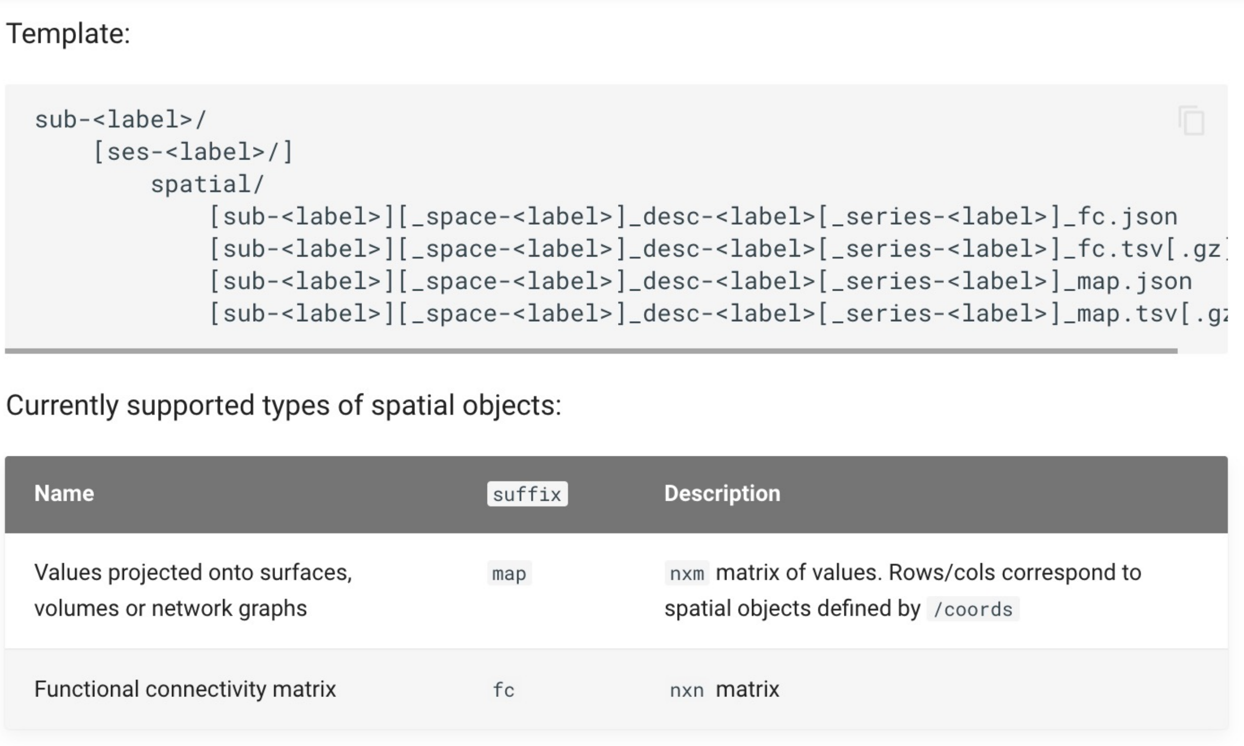
- functional connectivity matrices and more generic
- maps of values projected onto surfaces or network graphs.

The coordinates corresponding to rows and columns are defined in a `coord/` file, linked in a sidecar JSON. Every `spatial/*_desc-<label>*_<suffix>.tsv` data file MUST have an accompanying sidecar JSON `spatial/*_desc-<label>*_<suffix>.json` that links to the LEMS XML files that contain the underlying model equations (`eq/`) and parameters (`params/`) using the keys `"ModelEq"` and `"ModelParam"`.

Both, `ts/` and `spatial/` files can be grouped into file bundles using the filename key entity `series`. For example, a series of FC matrices can be used to store functional connectivity dynamics matrices over time:

```
    spatial/desc_FCDtest1_series_00001_fc.tsv,
    spatial/desc_FCDtest1_series_00002_fc.tsv,
    spatial/desc_FCDtest1_series_00003_fc.tsv,
    ...
    spatial/desc_FCDtest1_series_00300_fc.tsv
```

The coordinates of the series elements MUST be specified with the metadata key `"CoordsSeries"`.



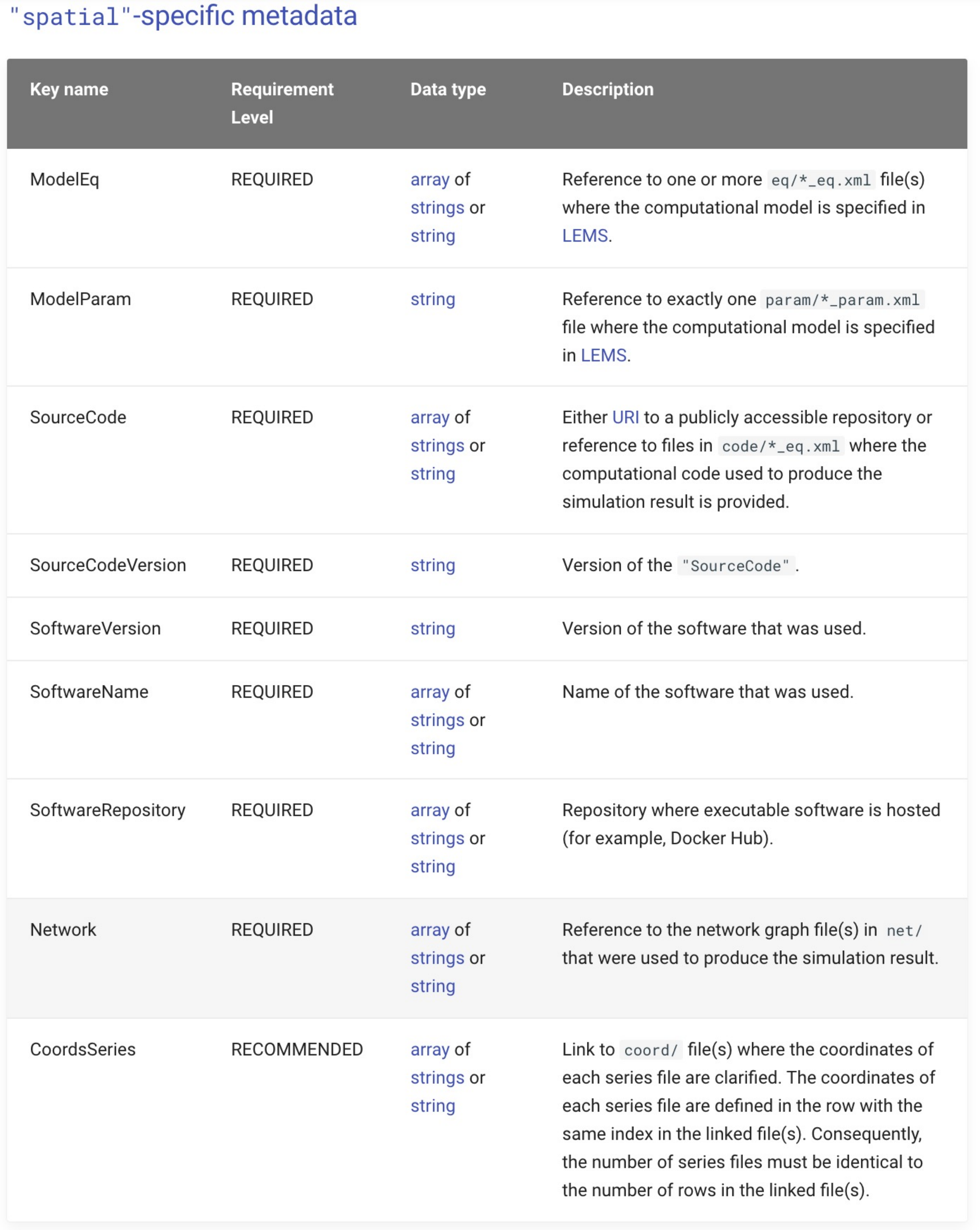
Template:

```
sub-<label>/
    [ses-<label>/]
        spatial/
            [sub-<label>][_space-<label>]_desc-<label>[_series-<label>]_fc.json
            [sub-<label>][_space-<label>]_desc-<label>[_series-<label>]_fc.tsv[.gz
            [sub-<label>][_space-<label>]_desc-<label>[_series-<label>]_map.json
            [sub-<label>][_space-<label>]_desc-<label>[_series-<label>]_map.tsv[.g
```

Currently supported types of spatial objects:

Name	`suffix`	Description
Values projected onto surfaces, volumes or network graphs	`map`	`nxm` matrix of values. Rows/cols correspond to spatial objects defined by `/coords`
Functional connectivity matrix	`fc`	`nxn` matrix

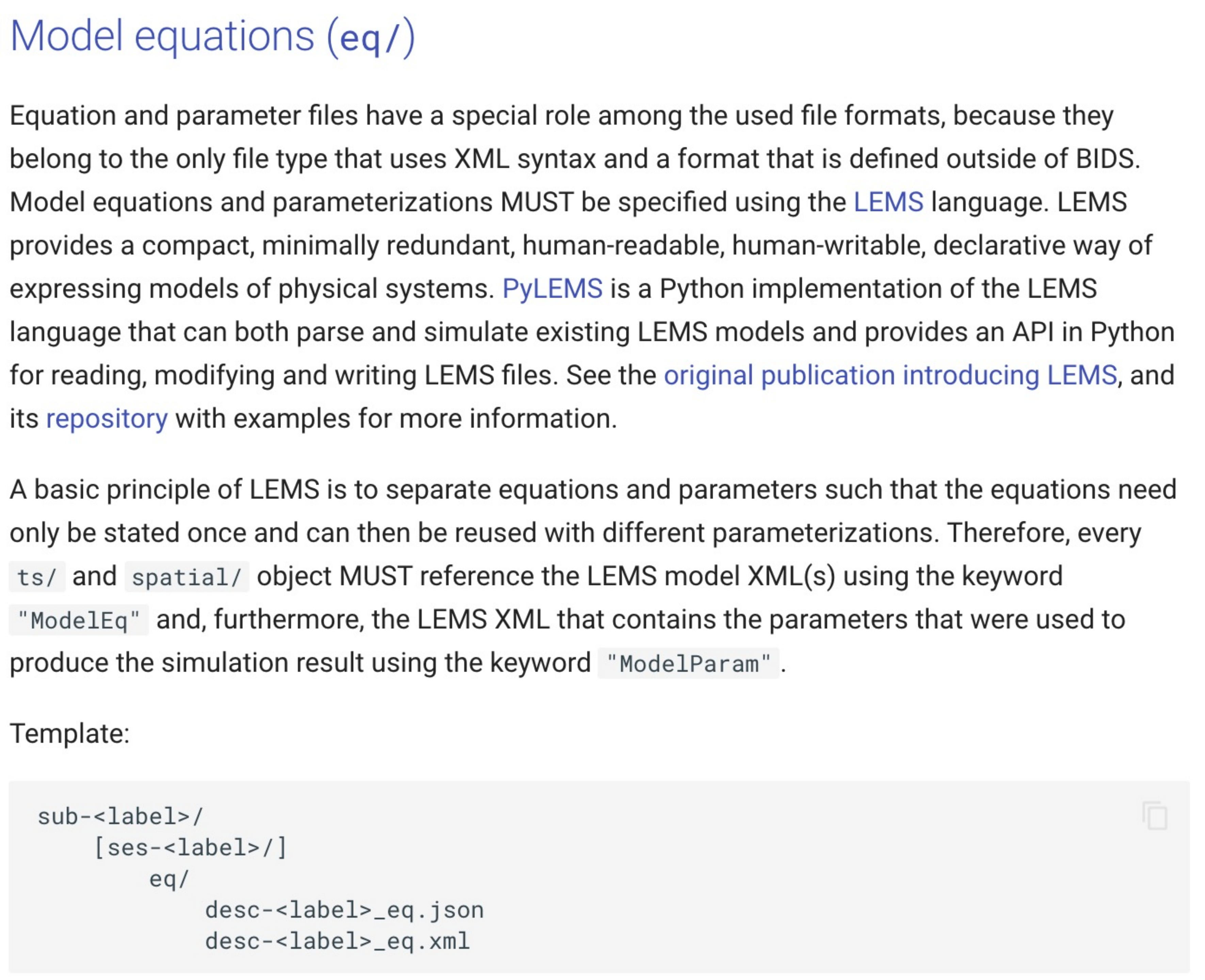
Spatial data (spatial/)



"spatial"-specific metadata

Key name	Requirement Level	Data type	Description
ModelEq	REQUIRED	array of strings or string	Reference to one or more `eq/*_eq.xml` file(s) where the computational model is specified in LEMS.
ModelParam	REQUIRED	string	Reference to exactly one `param/*_param.xml` file where the computational model is specified in LEMS.
SourceCode	REQUIRED	array of strings or string	Either URI to a publicly accessible repository or reference to files in `code/*_eq.xml` where the computational code used to produce the simulation result is provided.
SourceCodeVersion	REQUIRED	string	Version of the `"SourceCode"`.
SoftwareVersion	REQUIRED	string	Version of the software that was used.
SoftwareName	REQUIRED	array of strings or string	Name of the software that was used.
SoftwareRepository	REQUIRED	array of strings or string	Repository where executable software is hosted (for example, Docker Hub).
Network	REQUIRED	array of strings or string	Reference to the network graph file(s) in `net/` that were used to produce the simulation result.
CoordsSeries	RECOMMENDED	array of strings or string	Link to `coord/` file(s) where the coordinates of each series file are clarified. The coordinates of each series file are defined in the row with the same index in the linked file(s). Consequently, the number of series files must be identical to the number of rows in the linked file(s).

Model equations (eq/)



Model equations (eq/)

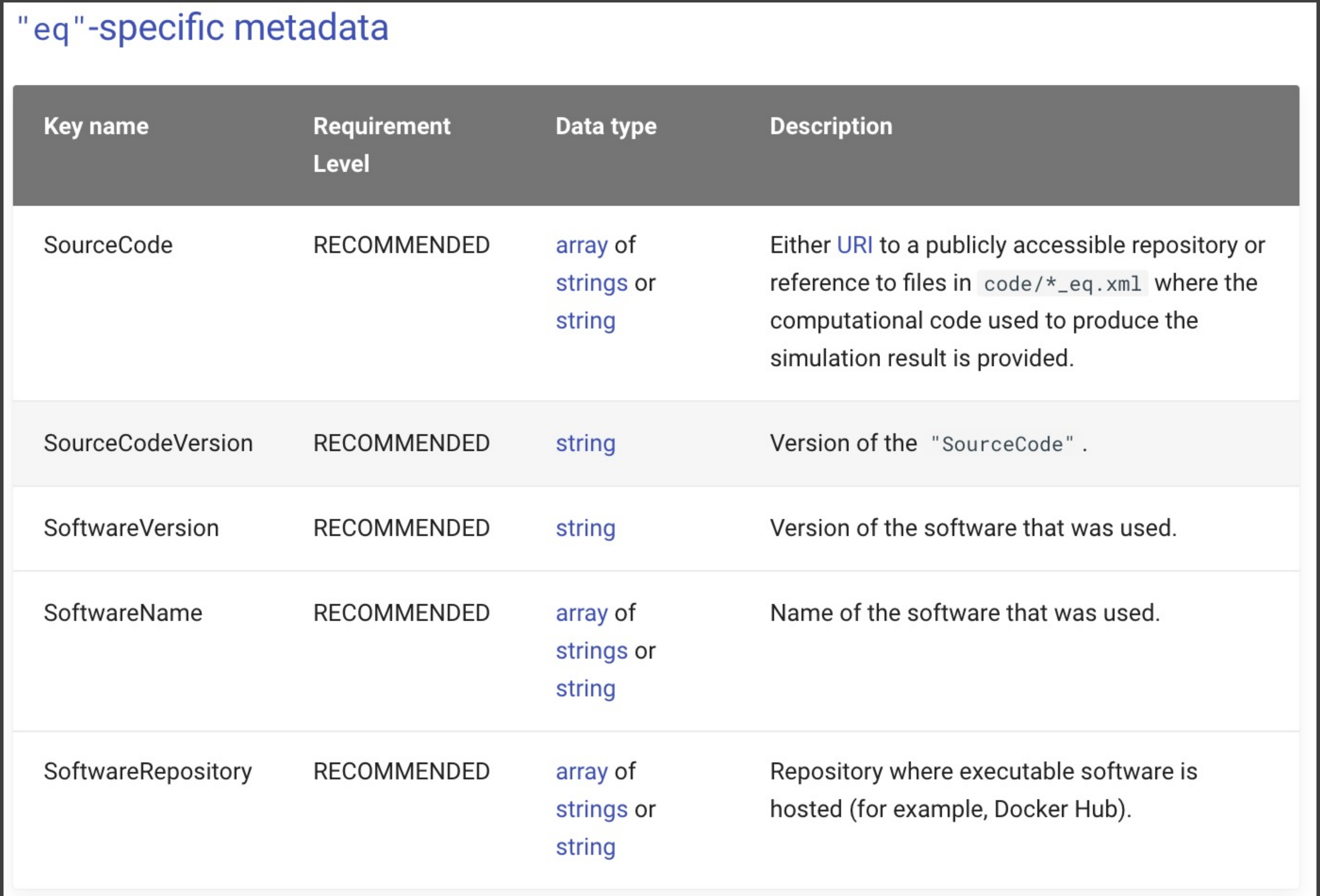
Equation and parameter files have a special role among the used file formats, because they belong to the only file type that uses XML syntax and a format that is defined outside of BIDS. Model equations and parameterizations MUST be specified using the LEMS language. LEMS provides a compact, minimally redundant, human-readable, human-writable, declarative way of expressing models of physical systems. PyLEMS is a Python implementation of the LEMS language that can both parse and simulate existing LEMS models and provides an API in Python for reading, modifying and writing LEMS files. See the original publication introducing LEMS, and its repository with examples for more information.

A basic principle of LEMS is to separate equations and parameters such that the equations need only be stated once and can then be reused with different parameterizations. Therefore, every `ts/` and `spatial/` object MUST reference the LEMS model XML(s) using the keyword `"ModelEq"` and, furthermore, the LEMS XML that contains the parameters that were used to produce the simulation result using the keyword `"ModelParam"`.

Template:

```
sub-<label>/
    [ses-<label>/]
        eq/
            desc-<label>_eq.json
            desc-<label>_eq.xml
```

Model equations (eq/)

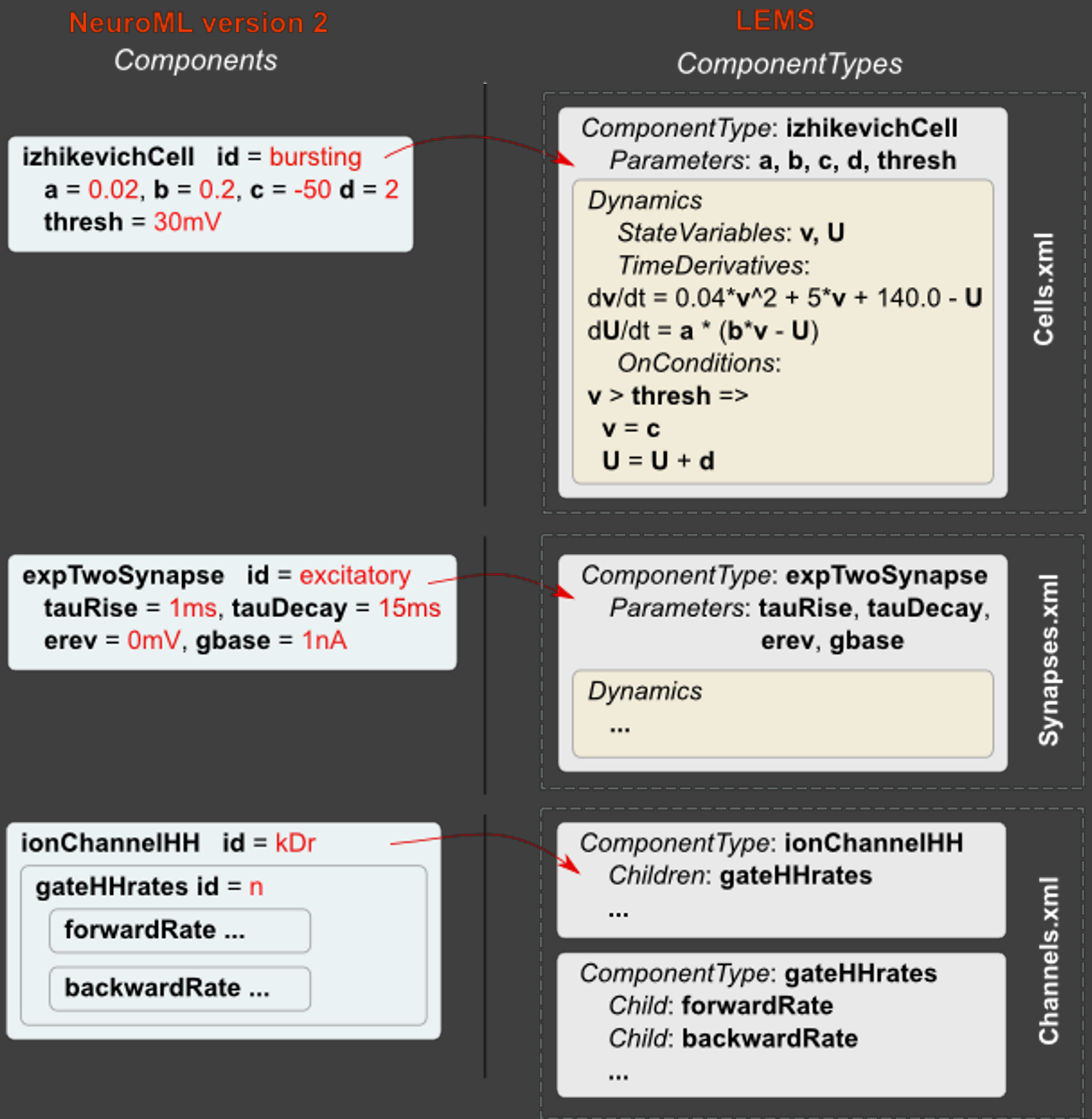


"eq"-specific metadata

Key name	Requirement Level	Data type	Description
SourceCode	RECOMMENDED	array of strings or string	Either URI to a publicly accessible repository or reference to files in `code/*_eq.xml` where the computational code used to produce the simulation result is provided.
SourceCodeVersion	RECOMMENDED	string	Version of the `"SourceCode"` .
SoftwareVersion	RECOMMENDED	string	Version of the software that was used.
SoftwareName	RECOMMENDED	array of strings or string	Name of the software that was used.
SoftwareRepository	RECOMMENDED	array of strings or string	Repository where executable software is hosted (for example, Docker Hub).

NeuroML/LEMS for specifying equations

- TVB-HPC (part of main TVB) automatically produces high-performance codes for CPUs (Numba) and GPUs (CUDA)
- based on 'LEMS' a domain-independent language for the declarative description of hierarchical mathematical models of physical entities in XML
- pyLEMS
  - simulator to run NeuroML2 models.
- libNeuroML API
  - Importer/Exporter: NeuroML Python object model



NeuroML version 2

Components

izhikevichCell id = bursting
a = 0.02, b = 0.2, c = -50 d = 2
thresh = 30mV

expTwoSynapse id = excitatory
tauRise = 1ms, tauDecay = 15ms
erev = 0mV, gbase = 1nA

ionChannelHH id = kDr
gateHHrates id = n
forwardRate ...
backwardRate ...

LEMS

ComponentTypes

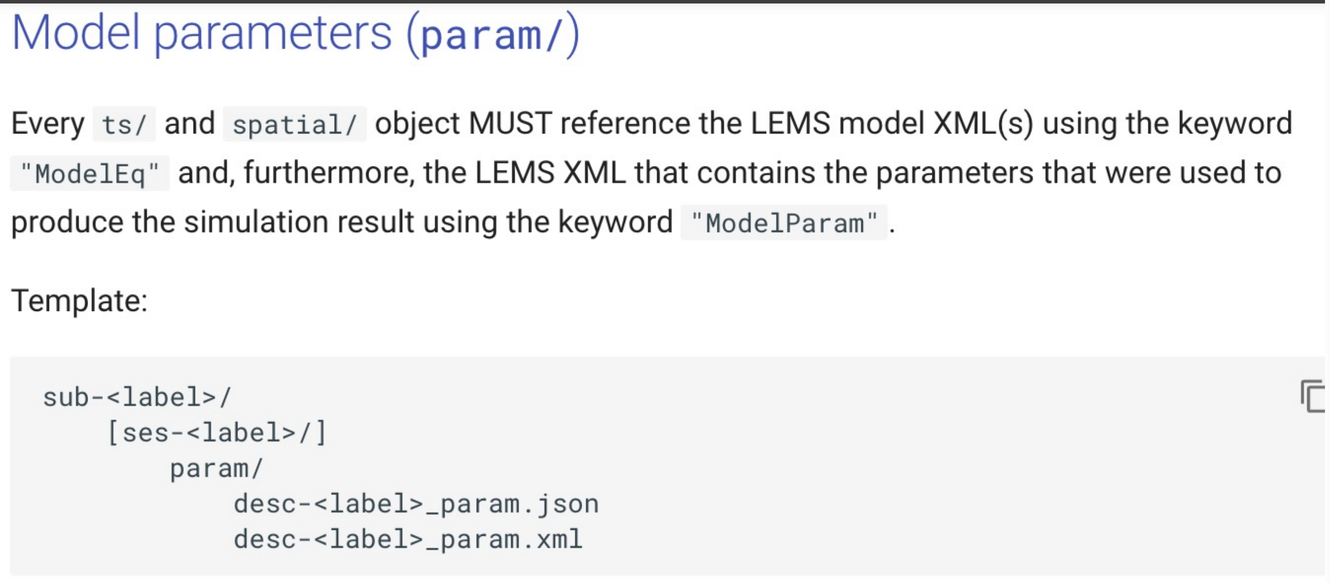
ComponentType: izhikevichCell
Parameters: a, b, c, d, thresh
Dynamics
StateVariables: v, U
TimeDerivatives:
dv/dt = 0.04*v^2 + 5*v + 140.0 - U
dU/dt = a * (b*v - U)
OnConditions:
v > thresh =>
v = c
U = U + d

ComponentType: expTwoSynapse
Parameters: tauRise, tauDecay, erev, gbase
Dynamics
...

ComponentType: ionChannelHH
Children: gateHHrates
...

ComponentType: gateHHrates
Child: forwardRate
Child: backwardRate
...

Model parameters (param/)

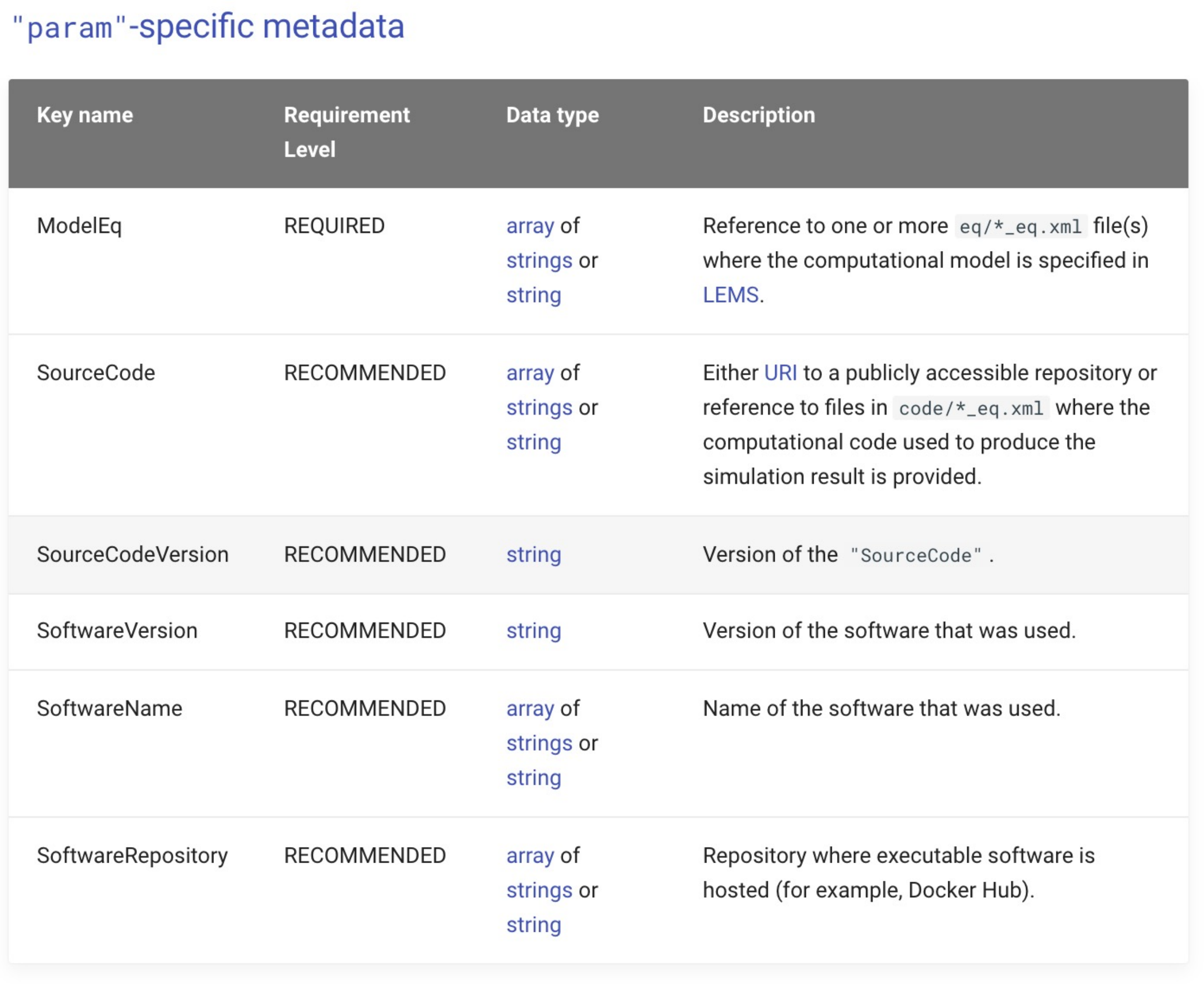


Model parameters (`param/`)

Every `ts/` and `spatial/` object MUST reference the LEMS model XML(s) using the keyword `"ModelEq"` and, furthermore, the LEMS XML that contains the parameters that were used to produce the simulation result using the keyword `"ModelParam"`.

Template:

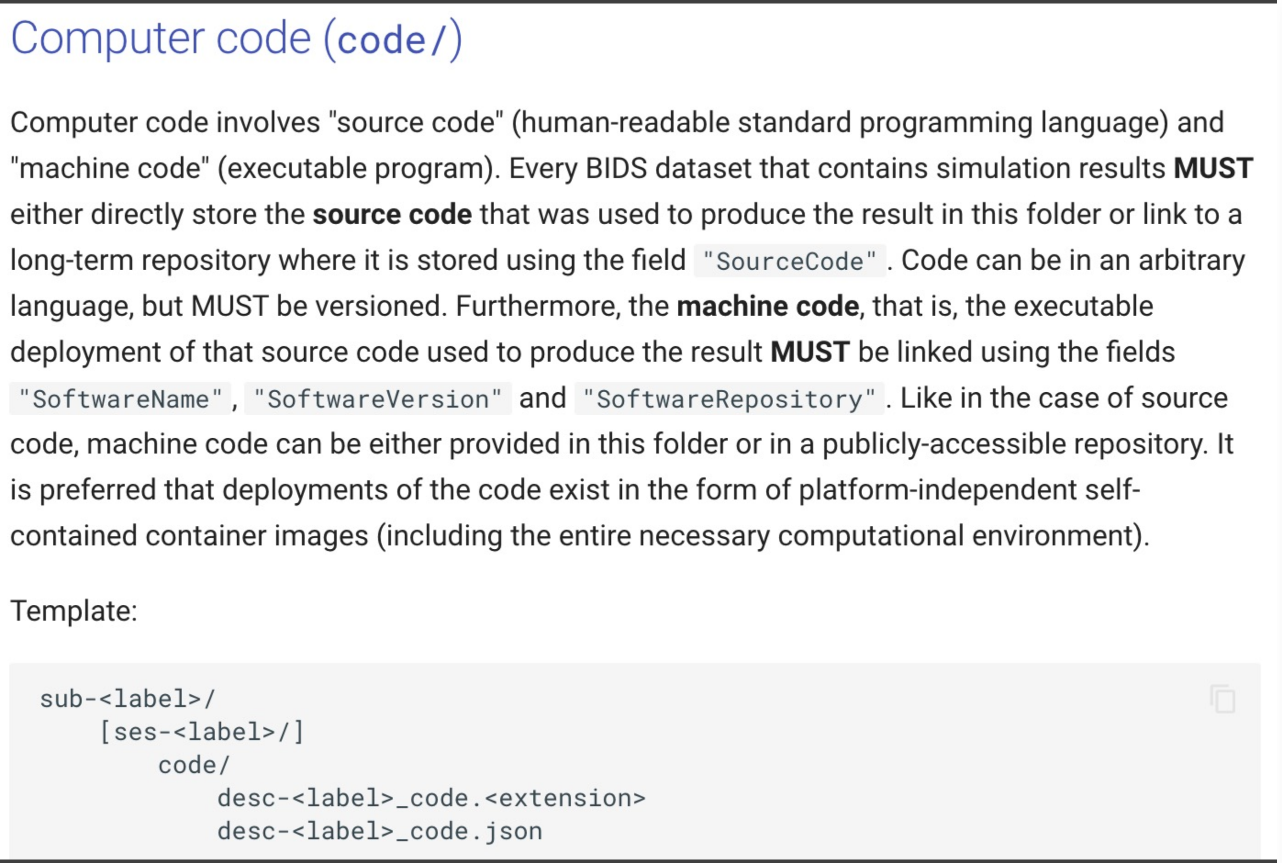
```
sub-<label>/
    [ses-<label>/]
        param/
            desc-<label>_param.json
            desc-<label>_param.xml
```



`"param"`-specific metadata

Key name	Requirement Level	Data type	Description
ModelEq	REQUIRED	array of strings or string	Reference to one or more `eq/*_eq.xml` file(s) where the computational model is specified in LEMS.
SourceCode	RECOMMENDED	array of strings or string	Either URI to a publicly accessible repository or reference to files in `code/*_eq.xml` where the computational code used to produce the simulation result is provided.
SourceCodeVersion	RECOMMENDED	string	Version of the `"SourceCode"`.
SoftwareVersion	RECOMMENDED	string	Version of the software that was used.
SoftwareName	RECOMMENDED	array of strings or string	Name of the software that was used.
SoftwareRepository	RECOMMENDED	array of strings or string	Repository where executable software is hosted (for example, Docker Hub).

Computer code (code/)

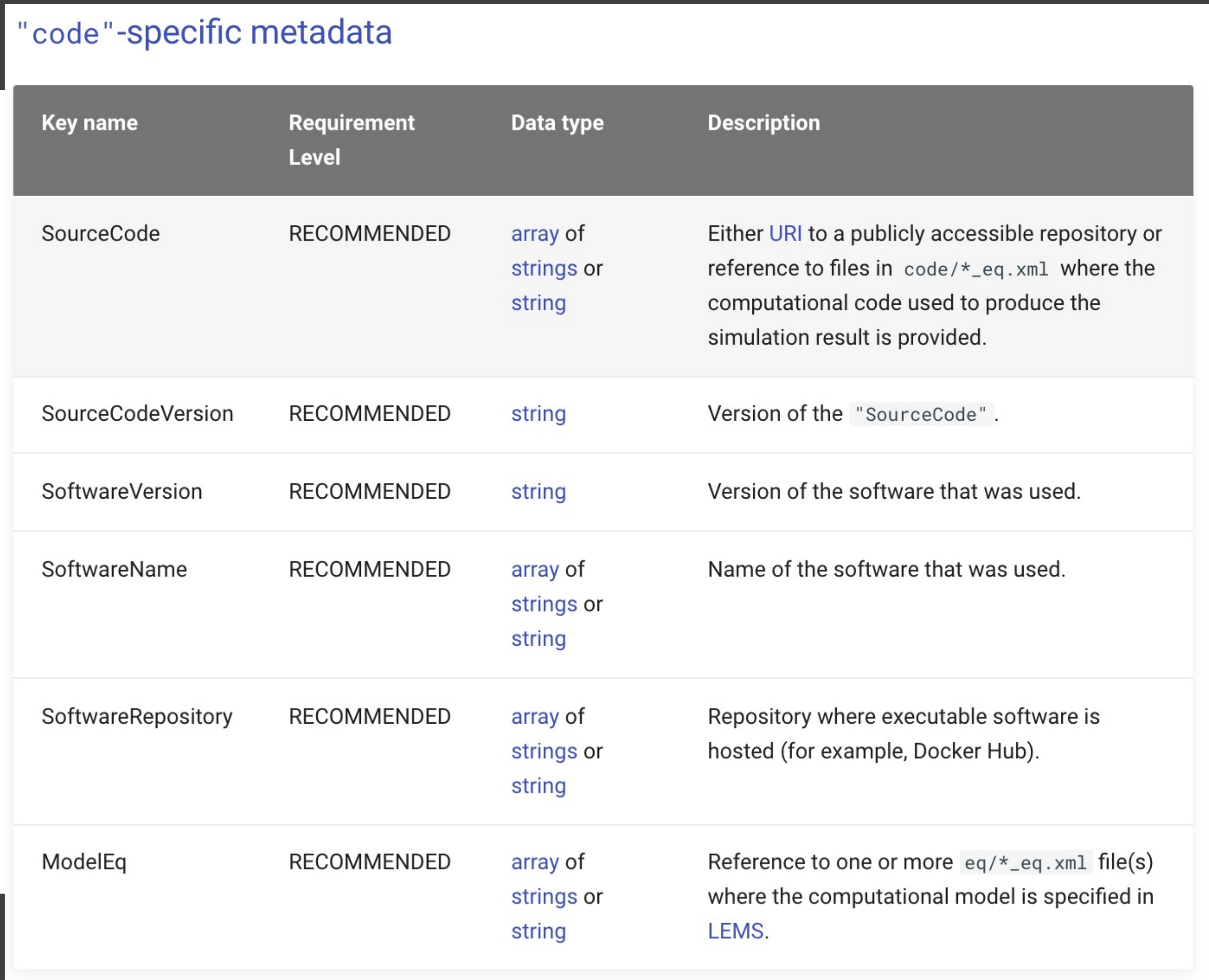


Computer code (`code/`)

Computer code involves "source code" (human-readable standard programming language) and "machine code" (executable program). Every BIDS dataset that contains simulation results **MUST** either directly store the **source code** that was used to produce the result in this folder or link to a long-term repository where it is stored using the field `"SourceCode"`. Code can be in an arbitrary language, but MUST be versioned. Furthermore, the **machine code**, that is, the executable deployment of that source code used to produce the result **MUST** be linked using the fields `"SoftwareName"`, `"SoftwareVersion"` and `"SoftwareRepository"`. Like in the case of source code, machine code can be either provided in this folder or in a publicly-accessible repository. It is preferred that deployments of the code exist in the form of platform-independent self-contained container images (including the entire necessary computational environment).

Template:

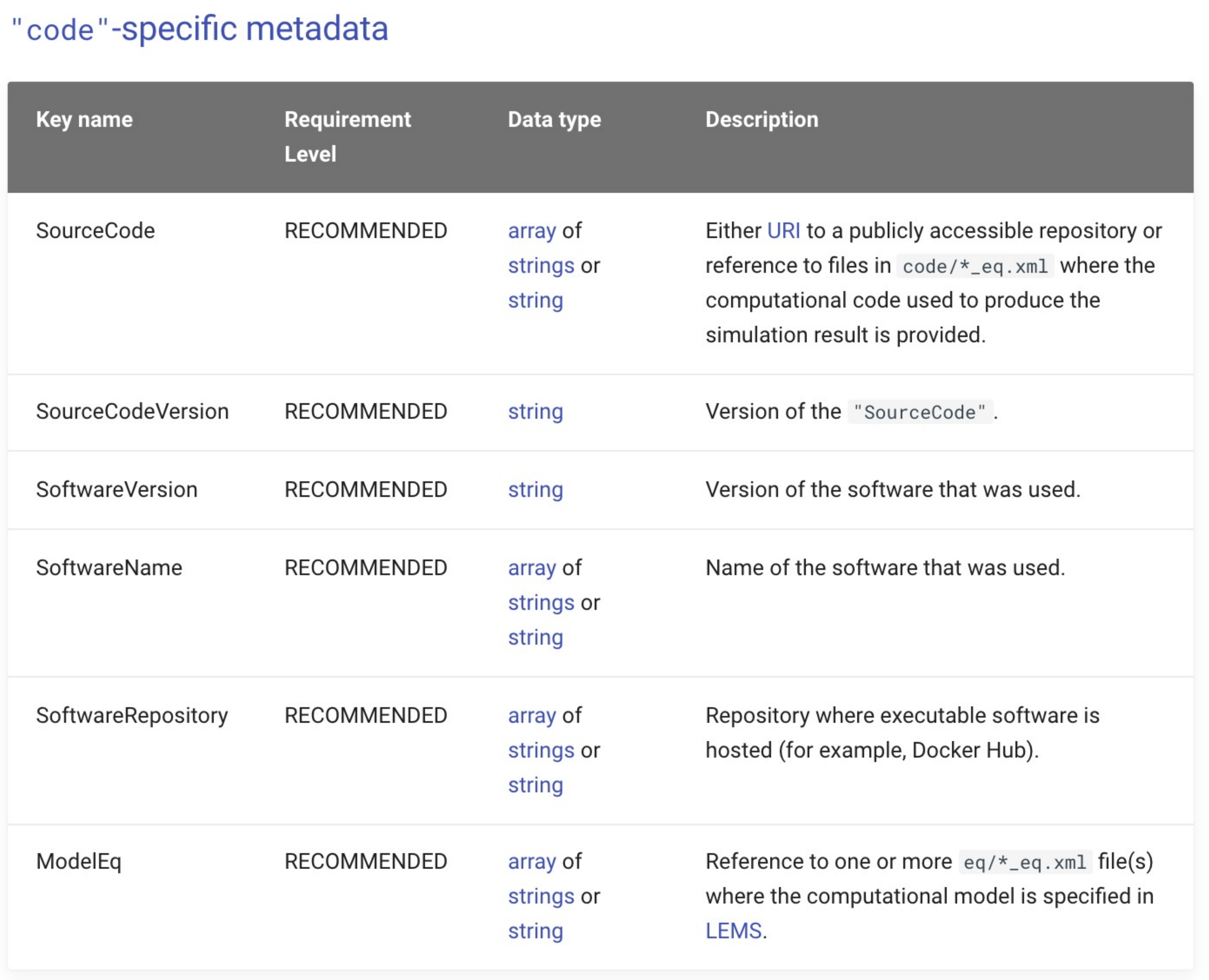
```
sub-<label>/
    [ses-<label>/]
        code/
            desc-<label>_code.<extension>
            desc-<label>_code.json
```



`"code"`-specific metadata

Key name	Requirement Level	Data type	Description
SourceCode	RECOMMENDED	array of strings or string	Either URI to a publicly accessible repository or reference to files in `code/*_eq.xml` where the computational code used to produce the simulation result is provided.
SourceCodeVersion	RECOMMENDED	string	Version of the `"SourceCode"`.
SoftwareVersion	RECOMMENDED	string	Version of the software that was used.
SoftwareName	RECOMMENDED	array of strings or string	Name of the software that was used.
SoftwareRepository	RECOMMENDED	array of strings or string	Repository where executable software is hosted (for example, Docker Hub).
ModelEq	RECOMMENDED	array of strings or string	Reference to one or more `eq/*_eq.xml` file(s) where the computational model is specified in LEMS.

Computer code (code/)



"code"-specific metadata

Key name	Requirement Level	Data type	Description
SourceCode	RECOMMENDED	array of strings or string	Either URI to a publicly accessible repository or reference to files in `code/*_eq.xml` where the computational code used to produce the simulation result is provided.
SourceCodeVersion	RECOMMENDED	string	Version of the `"SourceCode"`.
SoftwareVersion	RECOMMENDED	string	Version of the software that was used.
SoftwareName	RECOMMENDED	array of strings or string	Name of the software that was used.
SoftwareRepository	RECOMMENDED	array of strings or string	Repository where executable software is hosted (for example, Docker Hub).
ModelEq	RECOMMENDED	array of strings or string	Reference to one or more `eq/*_eq.xml` file(s) where the computational model is specified in LEMS.

Summary

- BIDS-like data model for computational model simulation results
- Based on simple and generic data types and exhaustive metadata annotation for reproducibility

THANK
YOU

Vision

- Future BIDS validator updates data set with proper ids and updates registry every time a change is applied
- helps data sharing, provenance and accountability tracking
    - every single file assumes an “identity” that is invariably associated with its metadata
    - rigorously associate data sets with desired metadata features
        - e.g. legal basis for sharing (and other agreements), “ownerships” and other roles with regard to data protection laws
- possible basis for a global indexing system to track the evolution of data sets with provable authenticity
- world-wide tracking of the evolution of annotated, verified and internally consistent data sets

Principles

- Instead of long lists of key-value pairs: short and concise filenames with an intuitive label
- unique IDs to distinguish files, and metadata in the sidecar JSON
- id could be a hash of the JSON sidecar and thereby also serve to increase confidence about data integrity, authenticity and validity (a checksum)
- cross-checking: data file name contains checksum of sidecar JSON and sidecar JSON contains checksum of data file content
- Future BIDS validator would be enabled to cross-validate integrity of metadata and data

Principles

- Unique identifiability and filename-content binding can be used to enforce rigorous provenance tracking
- Data transformation registry: every transformation involves updating checksums/ids that can be tracked in a registry
- rigorous structural validity is enforced
    - inconsistencies cannot go unnoticed
    - „enforces“ clean and reproducible workflows
- not necessary if every step is tracked (DataLad), but there are advantages if this is already inbuilt into the data format

Vision

- Future BIDS validator updates data set with proper ids and updates registry every time a change is applied
- May solve problems regarding worldwide data sharing, provenance and accountability tracking
- May be used to rigorously associate data sets with their legal basis for sharing (and other agreements), “ownerships” and other roles with regard to data protection laws
- Every single file assumes an “identity” that is invariably associated with its legal and other features
- Possible basis for global authoritative indexing system to track the evolution of neuroscience results with provable authenticity: world-wide recording of the evolution of annotated, verified and internally consistent data sets
- Could be further combined with encryption to have an all-in-one solution for data standardization, provenance tracking, data security and lawful data exchange.