# BioMed Sample Sheets Documentation

## Release 0.1

**Manuel Holtgrewe**

**Apr 04, 2017**

# Overview

**Note: Project Status**

While the cancer and germline TSV sheets, the JSON sheets, and the shortcut modules are useable, this project is still in "pre-release" status. We hope that the API will remain fairly stable but it can (and probably) will change in the future. The main aim of these early releases is for allowing to get feedback.

This document describes the current draft of the biomedical sample sheets.

The project consists of

- a JSON-based data format for the description of biomedical sample sheets,

- pre-defined data types to be used in the sample sheets,

- a Python program for the validation thereof,

- a Python program for conversion between JSON and TSV-based sample sheets (for easier viewing and editing),

- a Python library for access of the JSON sheets, and

- a Python library for simplified access of the JSON sheets for the important cases of matched tumor/normal studies and rare disease studies with pedigree relationships.

# Open Questions

- Keep or drop globally unique PKs from the file names? Random pks make everything very hard to use as files, add complexity to shortcut schemas. Sequential pks at least keep a temporal order.

## Introduction

A commonly recurring theme in the analysis of biomedical data is requiring the description of this data in machine-readable form. Two important areas where such a requirement arises are:

1. storing meta data for the results of an experiment/assay and

2. processing biomedical data sets in an automated and reproducible ways using some workflow engine.

This document describes a proposal for structuring the data, i.e., a data schema that allows for representing most important use cases that occurred for the Core Unit Bioinformatics (CUBI) at the Berlin Institute of Health (BIH). Further, this document proposes certain shortcuts/simplifications that make this data schema more easier to navigate and use for a handful of important use cases.

Originally, this is an adaption of the data schema used by Sven Nahnsen's group in Tubingen.

## High-Level Overview

The figure below gives an overview of the schema. Each box represents a table while the connections and labels specify the relation counts between them. The "core" schema is shown in yellow while the NGS-specific part is shown in blue.

**BioEntity** describes biological entities (or specimens or donors). This might be a human patient, a mouse, a cell culture, etc. **BioSample** describes one biological sample taken from the biological entity. This might be a blood sample, saliva swab, part of a primary tumor biopsy, part of a metathesis biopsy etc. **TestSample** describes a further processing of a BioSample, e.g., extracted DNA, RNA, etc. from a tumor biopsy. This data schema is taken from the Qbic tool by Sven Nahnsen's group.
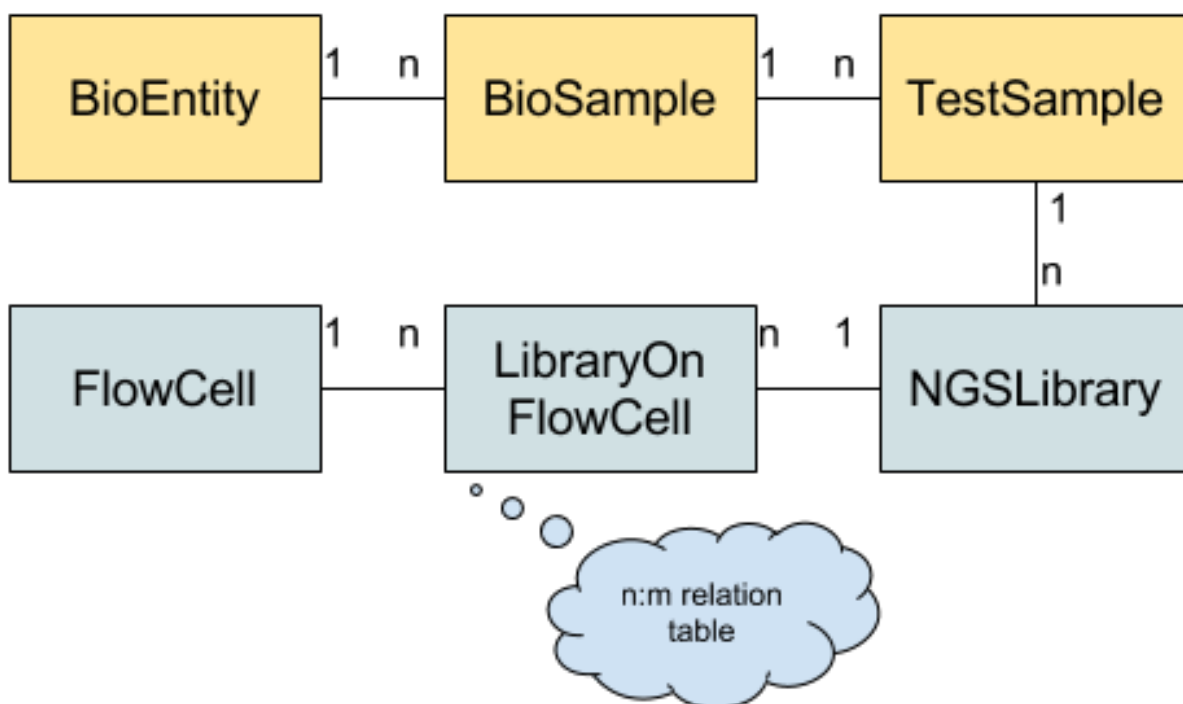
Fig. 1.1: Tables used in the BioMed Sheet data schema.

This data schema has the nice property that it models most of the relevant parts of many studies which are later analyzed by Bioinformaticians. For example, BioEntity could be cancer patients, BioSample describes the health samples and tumor biopsies taken from them (multiple samples can be modeled, e.g., for several saliva swabs at different time points, multiple biopsies, e.g., the primary tumor before regression, primary tumor after regression, and several metastases). TestSample then describes the result of the preparation of the sample, e.g., multiple DNA or RNA extractions can be modeled. Being able to describe multiple BioSamples and TestSamples has the advantage that certain batch and test- and sample-specific effects can be later considered based on the available meta data.

On the other hand, there are certain things that are not directly modeled, e.g., the link and order between tumors before and after regression is not modeled in the primary data schema. However, arbitrary meta data can be attached to each record and then be interpreted by a particular workflow.

Thus, the proposed data schema is useful enough such that generic software libraries can be written that can then be used for modeling many important specific cases and adjusted to the environment of its users.

For the NGS specialization, NGSLibrary represents on library prepared for sequencing (e.g. WES, WGS, RNA library). The various steps in the wet lab protocols are not modeled explicitly but each library record could be annotated by the user if he needs this information in a downstream analysis. FlowCell represents one flow cell run on a specific machine while LibraryOnFlowCell allows for assigning one library to multiple lanes on a given flow cell.

Using this schema, possible flow cell–specific or lane-specific biases can later be incorporated into an analysis. In terms of mapping to files, each LibraryOnFlowCell then corresponds to one compressed FASTQ file in the case of single-end reads and two compressed FASTQ files in the case of paired reads (at least for bcl2fastq v2, it might correspond to a multiple when using v1).

## Sample Sheets vs. Database Schemas

One important point here is the relation and separation between sample sheets and database schemas.

The aim of the sample sheets is to provide a useful computer-readable description of an experiment for processing the data with Bioinformatics workflows or further downstream analyses (e.g. in R or using Jupyter notebooks). This document describes a concept for such schemas and an implementation using JSON for the file format.

Of course, sample sheets could also be stored in a RDBMS (e.g., Postgres), with a suitable database schema. Often such databases would be part of a LIMS system. A sample sheet could serve as a poor Bioinformatician's LIMS system, if only some meta information for samples and their relation is to be tracked. However, the more complex part of such a LIMS system or even a hospital information management system are mostly not of direct relevance to the Bioinformatics analysis.

Rather, the data from a laboratory or hospital IT system can be used to generate biomedical sample sheets. BioMed sample sheets further allow linking back from the bio entities, bio samples, test samples, ngs libraries to such systems by storing lists of URLs for each of the previous entities.

# Table Fields

## Core Table Fields

This section describes the common table fields. Generally, the pk field is an integer primary key that is to be automatically generated (i.e. autoincrement in RDBMS). The field secondary_id is an identifier assigned by the "data owner" (e.g., the collaboration partner). This identifier has to be unique within a given project but can be ambiguous globally.

A possible best practice is to enforce the secondary_id to only consist of alphanumeric characters and underscores. Then, they should be constructed as (none of the `<Field>` values should contain a hyphen itself):

```
<BioEntity>-<BioSample>-<TestSample>-<NGSLibrary>
```

(of course only up to "BioSample" for BioSamples etc.).

Examples are:

- BioEntity secondary ids: 2355, BIH-234
- **BioSample secondary ids:**
    - 2355-B1 (first blood sample from patient 2355)
    - BIH_234-N1 (first normal sample from patient BIH-234)
    - BIH_234-T2 (second tumor sample from patient BIH-234)
- **TestSample secondary ids:**
    - 2355-B1-DNA1 (first DNA extraction from first blood sample)
    - BIH_234-T1-RNA1 (first RNA extraction from first tumor sample)
    - BIH_234-T2-DNA2 (second DNA extraction from second tumor sample)

Generally, the following are "core fields".

## BioEntity

- pk: integer
- secondary_id: string

## BioSample

- pk: integer
- bio_entity: fk to BioEntity.pk
- secondary_id: string

## TestSample

- pk: integer
- bio_sample: fk to BioSample.pk
- secondary_id: string

## NGSLibrary

- pk: integer
- test_sample: fk to TestSample.pk
- secondary_id: string

### FlowCell

- pk: integer

- machine_name: string

- flowcell_name: string

### NGSLibraryOnFlowCell

- pk: integer

- ngs_library: fk to NGSLibrary.pk

- flowcell: fk to FlowCell

- lane: int

## Common Table Fields

For many major use cases, the following table fields are useful additions to get a list of "common fields".

For all tables, adding a list of strings with external IDs (e.g., called "external_ids") is recommendable. This way, external resources can be linked out to. A recommendation is to use URLs for giving reads an unambiguous prefix. These URLs can be pseudo URLs or real entry points in remote REST APIs. Further, each record has a meta_data field for structured data in JSON format.

### BioEntity

- affected: boolean, optional field for specifying the "affected" state in rare disease studies

- sex: {'male', 'female', 'unknown'}, optional field for person's sex in germline studies

- father: fk to BioEntity.pk, optional fields for linking to father

- mother: fk to BioEntity.pk, optional fields for linking to mother

### BioSample

- cell_type: string with controlled vocabulary, optional field for specifying cell type

### TestSample

- extraction_type: controlled vocabulary with extraction type, e.g. {'DNA', 'RNA'} or a superset thereof; optional field for describing extracted data

### NGSLibrary

- library_kind: controlled vocabulary with library preparation type, e.g., {'WES', 'WGS', 'RNA-seq', 'other'} or a superset thereof; required field for describing library type

- kit: controlled vocabulary describing kit and version used for targeted sequencing, or RNA amplifcation method

**NGSLIbraryOnFC**

- adapter_name: string, optional field describing name of used adapter barcode(s)

- adapter_seq: string, optional field giving sequence of used adapter barcode(s)

# JSON Sample Sheets

## Rough Structure

Overall, a JSON sample sheet looks as follows.

The sheet is described as a JSON object and is given an ID, a title, and a description.

```
{
    "id": "https://omics.cubi.bihealth.org/experiments/33",
    "title": "Tumor/Normal Study Example",
    "description": "Example biomedical sheet for standard tumor/normal study",
```

This is followed by a section describing the optional additional fields for each of the objects.

```
    "extraInfoDefs": {
```

The extra fields can be described in each schema, e.g., as in the following example referring to the NCBI organism taxonomy.

```
        "ncbiTaxon": {
            "docs": "Reference to NCBI taxonomy",
            "key": "taxon",
            "type": "string",
            "pattern": "^NCBITaxon_[1-9][0-9]*$"
        }
    },
```

Or by refering to the built-in standard fields bundled with the `biomedsheets` module.

```
    "bioSample": {
        "uberonCellSource": { "$ref": "resource://biomedsheets/data/std_fields.
→json#/extraInfoDefs/template/uberonCellSource" }
    },
```

There can be field definitions for each data type.

```
        "testSample": {},
        "ngsLibrary": {},
        "msProteinPool": {}
    },
```

Then, the bio entities are given. They are stored in a JSON object/map. The attribute name/key is the secondary ID that has to be unique within the project. Each BioEntity must have a primary key, can have some extra IDs and additional information (as described in `extraInfoDefs` above).

```
        "BIH_001": {
            "pk": "123001",
            "extraIds": [
```

```
            "http://cancer-registry.hospital.de/PAT12345",
            "http://virtual-cuts.pathology.hospital.de/SMPL000021"
        ],
        "extraInfo": {
            "ncbiTaxon": "NCBITaxon_9606"
        },
        "bioSamples": {
```

Then, each BioEntity can have a number of BioSamples. Note that the secondary id is given without the prefix of the secondary ID of the containing BioSample. The BioSample must have a global ID `pk`, can have extra infos attached (and, of course extra IDs, omitted here).

```
            "N1": {
                "pk": "234001",
                "extraInfo": {
                    "uberonCellSource": "UBERON:0000178"
                },
```

Recursively, each BioSample can have a number of TestSamples which can have a number of NGSLibrary's and MSProteinPool's.

```
            "testSamples": {
                "DNA1": {
                    "pk": "345001",
                    "extraInfo": { "extractionType": "DNA" },
                    "ngsLibraries": {
                        "WES1": {
                            "pk": "567001",
                            "extraInfo": { "libraryType": "WES" }
                        }
                    }
                }
            }
        }
    }
}
```

## Sheet Validation

Validation of sample sheets has four steps:

1. the sheet must be valid JSON,

2. expansion of JSON pointers `{ "$ref": "<URL>" }` is performed,

3. the sheet must conform to the JSON schema bundled with `biomedsheets` (in the future it will be versionised at some URL),

4. additional validation based on `extraInfoDefs` is performed.

Steps 1 and 3 can be performed with standard tools or libraries. Step 2 is relatively easy and the `biomedsheets` module ships with code for performing this easily (the functionality is available as a Python program as well). Step 4 is not implemented yet.

On the one hand, custom fields allow for the definition of arbitrary "simple" values. Currently, it is possible to have boolean, numbers, strings, enums and lists of the atomic types. On the other hand, using JSON pointers, centrally

defined field types can be used. This allows for easy sharing of data types and easier computat

# Examples

## Sheet Examples

Below is an example JSON file with a cancer sample sheet. Note that the sheet only contains one donor with two bio samples (normal sample `N1` and primary tumor `T1`).

```json
{
    "identifier": "file://examples/example_cancer_matched.tsv",
    "title": "Cancer Sample Sheet",
    "description": "Sample Sheet constructed from cancer matched samples compact TSV␣
→file",
    "extraInfoDefs": {
        "bioEntity": {
            "ncbiTaxon": {
                "docs": "Reference to NCBI taxonomy",
                "key": "taxon",
                "type": "string",
                "pattern": "^NCBITaxon_[1-9][0-9]*$"
            }
        },
        "bioSample": {
            "isTumor": {
                "docs": "Boolean flag for distinguishing tumor/normal samples",
                "key": "isTumor",
                "type": "boolean"
            }
        },
        "testSample": {
            "extractionType": {
                "docs": "Describes extracted",
                "key": "extractionType",
                "type": "enum",
                "choices": [
                    "DNA",
                    "RNA",
                    "other"
                ]
            }
        },
        "ngsLibrary": {
            "seqPlatform": {
                "docs": "Sequencing platform used",
                "key": "kitName",
                "type": "enum",
                "choices": [
                    "Illumina",
                    "PacBio",
                    "other"
                ]
            },
            "libraryType": {
                "docs": "Rough classificiation of the library type",
                "key": "libraryType",
```

```json
                    "type": "enum",
                    "choices": [
                        "Panel-seq",
                        "WES",
                        "WGS",
                        "mRNA-seq",
                        "tRNA-seq",
                        "other"
                    ]
                },
                "folderName": {
                    "docs": "Name of folder with FASTQ files",
                    "key": "folderName",
                    "type": "string"
                }
            }
        },
    "bioEntities": {
        "P001": {
            "pk": 1,
            "extraInfo": {
                "ncbiTaxon": "NCBITaxon_9606"
            },
            "bioSamples": {
                "N1": {
                    "pk": 2,
                    "extraInfo": {
                        "isTumor": false
                    },
                    "testSamples": {
                        "DNA1": {
                            "pk": 3,
                            "extraInfo": {
                                "extractionType": "DNA"
                            },
                            "ngsLibraries": {
                                "WES1": {
                                    "pk": 4,
                                    "extraInfo": {
                                        "seqPlatform": "Illumina",
                                        "folderName": "P001-N1-DNA1-WES1",
                                        "libraryType": "WES"
                                    }
                                }
                            }
                        }
                    }
                },
                "T1": {
                    "pk": 5,
                    "extraInfo": {
                        "isTumor": true
                    },
                    "testSamples": {
                        "DNA1": {
                            "pk": 6,
                            "extraInfo": {
                                "extractionType": "DNA"
```

```
                                },
                                "ngsLibraries": {
                                    "WES1": {
                                        "pk": 7,
                                        "extraInfo": {
                                            "seqPlatform": "Illumina",
                                            "folderName": "P001-T1-DNA1-WES1",
                                            "libraryType": "WES"
                                        }
                                    }
                                }
                            },
                            "RNA1": {
                                "pk": 8,
                                "extraInfo": {
                                    "extractionType": "RNA"
                                },
                                "ngsLibraries": {
                                    "mRNA_seq1": {
                                        "pk": 9,
                                        "extraInfo": {
                                            "seqPlatform": "Illumina",
                                            "folderName": "P001-T1-RNA1-mRNAseq1",
                                            "libraryType": "mRNA_seq"
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

## Code Examples

The following Python program uses the `biomedsheets` module for loading the JSON sheet from above. It them prints the names of the donors and the names of the NGS libraries for the tumor/normal pairs.

```python
# -*- coding: utf-8 -*-
"""Demonstrate shortcuts for cancer sample sheet
"""

import collections
import os

from biomedsheets import io, ref_resolver, shortcuts


def load_sheet():
    """Return ``Sheet`` instance for the cancer example"""
    path = os.path.join(os.path.abspath(
        os.path.dirname(__file__)), 'example_cancer_matched.json')
    sheet_json = io.json_loads_ordered(open(path, 'rt').read())
    resolver = ref_resolver.RefResolver(dict_class=collections.OrderedDict)
```

```python
    return io.SheetBuilder(
        resolver.resolve('file://' + path, sheet_json)).run()


def main():
    """Main program entry point"""
    cancer_cases = shortcuts.CancerCaseSheet(load_sheet())
    print('Donors\n')
    for donor in cancer_cases.donors:
        print('  {}'.format(donor.name))
    print('\nLibraries of all tumor/normal pairs\n')
    for pair in cancer_cases.all_sample_pairs:
        print('  {}'.format(pair.donor.name))
        print('    normal DNA: {}'.format(pair.normal_sample.dna_ngs_library.name))
        if pair.normal_sample.rna_ngs_library:
            print('    normal RNA: {}'.format(pair.normal_sample.rna_ngs_library.
→name))
        print('    tumor DNA:  {}'.format(pair.tumor_sample.dna_ngs_library.name))
        if pair.tumor_sample.rna_ngs_library:
            print('    tumor RNA:  {}'.format(pair.tumor_sample.rna_ngs_library.name))


if __name__ == '__main__':
    main()
```

## Output

The output of the program is as follows:

```
Donors

P001-000001

Libraries of all tumor/normal pairs

  P001-000001
    normal DNA: P001-N1-DNA1-WES1-000004
    tumor DNA:  P001-T1-DNA1-WES1-000007
    tumor RNA:  P001-T1-RNA1-mRNA_seq1-000009
```

# Frequently Asked Questions

## Why JSON?

While YAML is easier to process by human beings, JSON has better tool support and is used more widely. In particular, JSON schema is widely accepted as are JSON pointers, and RDBMS such as Postgres have good support for JSON fields as well.

Further, JSON is valid YAML, so YAML parsers can be used for reading the JSON files and the resulting structures can then be validated by JSON validators.

### Why only secondary ID parts in JSON?

Here, the idea is to increase readability a bit and remove redundancy. The prefix is implicitly defined by the secondary ids on the path to the root.

## Workflow Assumptions

One aim for the biomedical sheets is to drive workflow engines. Currently, the main aim is supporting Snakemake but the described data structures and file formats are straightforward enough so they can be used by any engine. Further, the schemas and workflows described below focus on high-throughput sequencing data. Adaption to proteomics, metabolomics, etc. should be

One central assumption is that the overall workflow is modularized and each module deals with one "well-defined" processing step with "homogeneous data". While this is not clearly defined, some examples are:

- A module that aligns HTS data from FASTQ files, supporting both RNA/DNA sequencing data and paired/single read data. Depending on the annotated test sample extraction type, read length, and paired/single mode, the aligner is chosen (BWA-MEM, BWA-ALN, STAR). The alignments are post-processed for masking duplicates, converted to BAM, sorted, indexed, and basic statistics are computed. The module is smart enough to create appropriate read groups in the BAM file for each pair of/single FASTQ files for correcting for lane bias.

- A module that takes HTS alignments in BAM data and performs "simple" variant calling, i.e., each sample is considered independently. The resulting VCF file is appropriately postprocessed (normalize indels, sort, bgzip, index). The module can also interpret donor-based pedigree information from the sample sheet and call variants for all samples from a family independently. If the pedigree mode is used then there is the restriction that there must be only one sample from each person in the family. Based on configuration, one or multiple of a set of supported variant callers can be used. A "cancer" mode performs variant calling of all samples of one donor together.

- A module that takes HTS alignments in BAM data and performs "somatic" variant calling. Each donor must have exactly one sample flagged as "is not cancer" and one or more samples flagged as "is cancer". For each cancer sample, the module performs paired somatic variant calling with one or more tools from a supported list. Only one library for each the control and the cancer samples is supported. The result is a VCF file for each tool and each cancer sample with the somatic variants. The VCF is appropriately preformatted.

Such typical modules are much easier to write and use if certain assumptions about the sample sheets hold. Generally, the aim is to define a small core set of information that is required for the proper processing in workflows. Schema-specific settings are described in the following chapters. Core information is described here (somewhat redundant with the chapter *High-Level Overview*).

## Core Fields

- **BioEntity**
    - pk
    - secondaryId
- **BioSample**
    - pk
    - secondaryId
- **TestSample**
    - pk

- – secondaryId

- – extraction_type

- **AssaySample**

    - – pk

    - – secondaryId

## HTS Core Fields

The following fields are required by all workflow steps that process HTS data.

- **TestSample**

    - – extractionType – extracted sample, currently one of {`DNA`, `RNA`, `other`}

- **AssaySample**

    - – libraryType – library type, currently one of {`WES`, `WGS`, `Panel_seq`, `mRNA_seq`, `total_RNA_seq`, `other`}

# Entity Names

The term **name** is used for strings/tokens that represent a bio entity, bio sample, test sample, NGS library, etc. uniquely as part of file names. Further, they are used where sample names or similar are required in file contents, e.g., in VCF or BAM files. There are many possible strategies for this.

One important point is that the name must be unique, ideally globally. For this reason, it is recommended to include the primary key in the name, such that read alignment files can be named `{sample_name}.bam`, for example, and the file name is then unique throughout all systems.

## Primary Key as Name Parts

The recommendation is to construct sample names etc. as `{secondary_id}-{pk}` where `{secondary_id}` is the **full** secondary id (e.g., `PATIENT-T1-DNA1-WES1`) and `pk` is the integer primary key from the database.

Example file names:

- `bwa.PATIENT-T1-DNA1-WES-0000001.bam`

- `mutect.PATIENT-T1-DNA1-WES-0000001.vcf.gz`

## Secondary IDs as Names

If no primary key has been assigned yet, a possible alternative strategy is to only use the secondary id. This allows stable file names even if no stable primary key can be assigned.

---

**Note:** This is the recommendation for cubi_pipeline until we have a good id assignment system available.

---

Example file names:

- `bwa.PATIENT-T1-DNA1-WES.bam`

- `mutect.PATIENT-T1-DNA1-WES.vcf.gz`

# Matched Tumor Samples

A relatively simple schema for the analysis of matched tumor/normal samples from cancer studies. The assumed
setting is as follows.

- Each bio entity is a patient/donor.

- Each donor gives one normal (bio) sample (e.g., blood or saliva) and at least one (bio) sample from the cancer
  (e.g., primary tumor or metastesis).

- For each tumor and non-tumor sample, there is at least one DNA HTS library sequenced.

- For each tumor sample, there can be RNA HTS libraries.

- Only the first seen DNA/RNA library is considered for each sample (the "primary one")

---

**Note:** The requirement of one DNA HTS library for each sample and RNA only for tumor can be dropped in the
future.

---

## Matched Tumor Fields

The following fields must be present for matched tumor sample sheets.

- **BioSample**

  - isTumor – a boolean defining whether the sample was taken from tumor cells

## Matched Tumor TSV Schema

Additionally, there is an alternative to defining schemas in JSON format for matched tumor sample sheets. Instead, a
TSV-based schema can be used.

Optionally, the schema can contain meta data, starting with `[Metadata]` INI-style section header (the data section
has to start with `[Data]`).

```
[Metadata]
schema          cancer_matched
schema_version  v1
title           Example matched cancer tumor/normal study
description     The study has two patients, P001 has one tumor sample, P002 has two

[Data]
```

The `schema` and `schema_version` lines are optional.

If the file does not start with an INI-style section header, it starts with tab-separated column names. An example is
shown below:

```
patientName sampleName  isTumor     libraryType     folderName
P001        N1          N           WES             P001-N1-DNA1-WES1
P001        T1          Y           WES             P001-T1-DNA1-WES1
P001        T1          Y           mRNA-seq        P001-T1-RNA1-mRNAseq1
P002        N1          N           WES             P001-N1-DNA1-WES1
P002        T1          Y           WES             P001-T1-DNA1-WES1
P002        T1          Y           WES             P001-T1-RNA1-RNAseq1
```

| | | | | |
|---|---|---|---|---|
| P002 | T2 | Y | WES | P001-T2-DNA1-WES1 |
| P002 | T2 | Y | mRNA-seq | P001-T2-RNA1-mRNAseq1 |

They are as follows:

1. `patientName` – name of the patient, used for identifying the patient in the sample sheet.

2. `sampleName` – name of the sample, used for identifying the sample for the patient in the sample sheet (the combination of patient and sample must be unique in the sheet).

3. `isTumor` – a flag identifying a sample as being from tumor, one of {`Y`, `N`, `1`, `0`}

4. `extractionType` – a valid extraction type as in the JSON schema

5. `libraryType` – a valid libraryType, as in the JSON schema

6. `folderName` – a folder name to search the library's FASTQ files for. A list of base folders to search for the folder names is given in the configuration, so no full path is given here.

Note that the name of the `TestSample` and and `NGSLibrary` entities are missing, they will be auto-generated based on the `extractionType` and `libraryType`.

Optionally, the following fields can be added:

- `seqPlatform` can be one of `Illumina` and `PacBio`, default is `Illumina`

# Germline Variants Samples

A relatively simple schema with the essential information for genetic germline variant samples with support for pedigree information. The assumed setting is as follows.

- Each bio entity is a patient/person/donor.

- Each donor gives one normal (bio) sample (e.g., blood or saliva) and each sample has at one DNA library.

- The pedigree has to be described in a way such that all members who donated a sample are connected.

- Within each pedigree, the flag for "affected" specifies the same genetic disease (i.e., in a polydactily family, only members affected with this phenotype are flagged as affected).

---

**Note:** The requirement of one DNA HTS library for each donor can be dropped in the future.

---

## Germline Fields

The following fields must be present for matched cancer sample sheets.

- **BioSample**

    - fatherPk – pk value of the father

    - motherPk – pk value of the mother

    - sex – sex of the patient

    - affected – "is affected" flag

## Germline TSV Schema

Additionally, there is an alternative to defining schemas in JSON format for germline variants sample sheets. Instead, a TSV-based schema can be used.

Optionally, the schema can contain meta data, starting with `[Metadata]` INI-style section header (the data section has to start with `[Data]`).

```
[Metadata]
schema          germline_variants
schema_version  v1
title           Example germline study
description     The [Data] section is similar to a PEDigree file


[Data]
```

The `schema` and `schema_version` lines are optional.

If the file does not start with an INI-style section header, it starts with tab-separated column names. An example is shown below:

```
patientName fatherName  motherName  sex affected  libraryType folderName  hpoTerms
12_345      12_346      12_347      1   2          WGS         12-345      HP:0009946,
→HP:0009899
12_348      12_346      12_347      1   1          WGS         12-348      .
12_346      0           0           1   1          .           .           .
12_347      0           0           2   1          WGS         12-347      .
```

They are as follows:

1. `patientName` – name of the patient, used for identifying the patient in the sample sheet.

2. `fatherName` – name of the patient's father, dot (`.`) for founder

3. `mother` – name of the patient's mother, dot (`.`) for founder

4. `sex` – flag for sex, one of `M`: male, `F`: female, `.`: unknown/missing, or `0`, `1`, `2`, as in PED

5. `affected` – flag for being affected, one of `Y`: yes, `N`: no, `.`: unknown/missing, or `0`, `1`, `2`, as in PED

6. `folderName` – a folder name to search the library's FASTQ files for, `.` if not sequenced A list of base folders to search for the folder names is given in the configuration, so no full path is given here.

7. `hpoTerms` – a comma-separated list of HPO terms

8. `libraryType` – a rough classification of the library type {`WGS`, `WES`, `Panel-seq`}, `.` if not sequenced or unknown

9. `kitName`/`kitType` – type/name of the kit used (free text/controlled vocabulary), `.` if not sequenced or unknown; optional. This column can be left out.

10. `kitVersion` – version of the kit used, `.` if not sequenced or unknown; optional. This column can be left out.

Note that the name of `BioSample`, `TestSample`, and `NGSLibrary` entities are missing, they will be auto-generated based on the `extractionType` and `libraryType`.

Optionally, the following fields can be added:

- `seqPlatform` can be one of `Illumina` and `PacBio`, default is `Illumina`

# Models

## biomedsheets.models.Sheet

## biomedsheets.models.BioEntity

## biomedsheets.models.BioSample

## biomedsheets.models.TestSample

## biomedsheets.models.NGSLibrary

## biomedsheets.models.CrawlMixin

# Shortcuts

**Contents**

**biomedsheets.shortcuts.TestSampleChildShortcut**

**biomedsheets.shortcuts.NGSLibraryShortcut**

**biomedsheets.shortcuts.TestSampleShortcut**

**biomedsheets.shortcuts.GermlineCaseSheet**

**biomedsheets.shortcuts.CancerCaseSheet**

**biomedsheets.shortcuts.CancerDonor**

**biomedsheets.shortcuts.CancerMatchedSamplePair**

**biomedsheets.shortcuts.CancerBioSample**

**biomedsheets.shortcuts.SampleSheet**

# Input/Output

Contents

**biomedsheets.io.SheetSchema**

**biomedsheets.io.SheetBuilder**

# TSV Input/Output

Contents

**biomedsheets.io_tsv.DELIM**

**biomedsheets.io_tsv.LIBRARY_TYPES**

**biomedsheets.io_tsv.LIBRARY_TO_EXTRACTION**

**biomedsheets.io_tsv.EXTRACTION_TYPE_DNA**

**biomedsheets.io_tsv.EXTRACTION_TYPE_RNA**

**biomedsheets.io_tsv.EXTRACTION_TYPES**

**biomedsheets.io_tsv.KEY_TITLE**

**biomedsheets.io_tsv.KEY_DESCRIPTION**

**biomedsheets.io_tsv.PLATFORM_ILLUMINA**

**biomedsheets.io_tsv.PLATFORM_PACBIO**

**biomedsheets.io_tsv.PLATFORM_OTHER**

**biomedsheets.io_tsv.PLATFORM_DEFAULT**

**biomedsheets.io_tsv.PLATFORM_NAMES**

**biomedsheets.io_tsv.BOOL_VALUES**

**biomedsheets.io_tsv.NCBI_TAXON_HUMAN**

**biomedsheets.io_tsv.CANCER_DEFAULT_TITLE**

**biomedsheets.io_tsv.CANCER_DEFAULT_DESCRIPTION**

**biomedsheets.io_tsv.CANCER_TSV_HEADER**

**biomedsheets.io_tsv.CANCER_EXTRA_INFO_DEFS**

**biomedsheets.io_tsv.GERMLINE_DEFAULT_TITLE**

**biomedsheets.io_tsv.GERMLINE_DEFAULT_DESCRIPTION**

**biomedsheets.io_tsv.GERMLINE_TSV_HEADER**

**biomedsheets.io_tsv.GERMLINE_EXTRA_INFO_DEFS**

**biomedsheets.io_tsv.SEX_VALUES**

**biomedsheets.io_tsv.AFFECTED_VALUES**

**biomedsheets.io_tsv.SheetIOException**

**biomedsheets.io_tsv.TSVSheetException**

**biomedsheets.io_tsv.std_field**