brukerapi

Release 0.1.0

Tomas Psorn

Sep 07, 2023

TUTORIALS:

Bruker API documentation.

CHAPTER

ONE

INSTALLATION

1.1 Using pip

```
pip install brukerapi
```

1.2 From source

```
git clone https://github.com/isi-nmr/brukerapi-python.git
cd brukerapi-python
python setup.py build
python setup.py install
```

CHAPTER

TWO

HOW TO WORK WITH BRUKER STUDY?

```
from brukerapi.study import Study

study = Study('path_to_study')

#get list of scans (fid data sets) contained in the study
study.scans

#get list of recos (2dseq data sets) contained in the study
study.recos

#get data set from the study hierarchy
study.get_dataset(scan_id='2', reco_id='1')
```

Data set obtained from `Study` object are empty by default, to access its content, the data set needs to be loaded. Either using the load function.

```
dataset = study.get_dataset(scan_id='2', reco_id='1')

dataset.load()
dataset.data
dataset.get_value('VisuCoreSize')
```

Or using context manager.

```
with study.get_dataset(scan_id='2', reco_id='1') as dataset:
    dataset.data
    dataset.get_value('VisuCoreSize')
```

CHAPTER

THREE

HOW TO LOAD A 2DSEQ FILE?

The `Dataset` constructor accepts both a path to directory containing a fid file, or a path to the 2dseq file.

```
from brukerapi.dataset import Dataset

 dataset = Dataset('path_to_2dseq/')

 dataset = Dataset('path_to_2dseq/2dseq')
```

A *Dataset* object is primarily an interface to the data contained in the 2dseq file.

```
data = dataset.data
```

Data is typically and n-dimensional array, the physical meaning of individual dimensions is stored in `dim_type` property.

```
>> dataset.dim_type
>> ['spatial', 'spatial', 'FG_SLICE']
```

It is possible to directly access some of the most wanted measurement parameters.

```
>> dataset.TE
>> 3.0
>> dataset.TR
>> 15.0
>> dataset.flip_angle
>> 10.0
```

The visu_pars file is used to construct a 2dseq data set, it is possible to get value of any of hereby stored parameters.

```
>> dataset.get_value('VisuCoreSize')
>> [192 192]
>> dataset.get_value('VisuCoreDim')
>> 2
```

CHAPTER

FOUR

HOW TO LOAD A FID FILE?

The `Dataset` constructor accepts both a path to directory containing a fid file, or a path to the fid file.

```
from brukerapi.dataset import Dataset

 dataset = Dataset('path_to_fid/')

 dataset = Dataset('path_to_fid/fid')
```

A *Dataset* object is primarily an interface to the data contained in the fid file.

```
data = dataset.data
```

Data is typically and n-dimensional array, the physical meaning of individual dimensions is stored in `dim_type` property.

```
>> dataset.dim_type
>> ['kspace_encode_step_0', 'kspace_encode_step_1', 'slice', 'repetition', 'channel']
```

It is possible to directly access some of the most wanted measurement parameters.

```
>> dataset.TE
>> 3.0
>> dataset.TR
>> 15.0
>> dataset.flip_angle
>> 10.0
```

Both acqp and method files are used to construct a fid data set, it is possible to get value of any of hereby stored parameters.

```
>> dataset.get_value('PVM_Matrix')
>> [192 192]
>> dataset.get_value('ACQ_dim_desc')
>> ['Spatial' 'Spatial']
```

CHAPTER

FIVE

HOW TO FILTER BRUKER DIRECTORIES?

In all tutorials we use our publicly available ParaVision v6.0.1 dataset.

First we creeate a *Study* object and print its structure.

```
from brukerapi.study import Study

study = Study('20200612_094625_lego_phantom_3_1_2')

#print structure of directory
study.print()
```

```
20200612_094625_lego_phantom_3_1_2 [Study]
   └-- 1 [Experiment]
     └-- fid [Dataset]
     └-- acqp [JCAMPDX]
     └-- uxnmr.par [JCAMPDX]
     └-- AdjStatePerScan [JCAMPDX]
     └-- configscan [JCAMPDX]
     └-- pdata [Folder]
       └-- 1 [Processing]
         └-- procs [JCAMPDX]
         └-- reco [JCAMPDX]
         └-- 2dseq [Dataset]
         └-- id [JCAMPDX]
         └-- visu_pars [JCAMPDX]
     └-- visu_pars [JCAMPDX]
     └-- specpar [JCAMPDX]
     └-- method [JCAMPDX]
   └-- 2 [Experiment]
     └-- fid [Dataset]
     └-- acqp [JCAMPDX]
     └-- uxnmr.par [JCAMPDX]
     └-- AdjStatePerScan [JCAMPDX]
     └-- configscan [JCAMPDX]
     └-- pdata [Folder]
       └-- 1 [Processing]
         └-- procs [JCAMPDX]
         └-- reco [JCAMPDX]
         └-- 2dseq [Dataset]
         └-- id [JCAMPDX]
         └-- visu_pars [JCAMPDX]
```

(continues on next page)

(continued from previous page)

```
     └── visu_pars [JCAMPDX]
     └── specpar [JCAMPDX]
     └── method [JCAMPDX]
.
.
.
```

As we can see, there are all possible brukerapi object in the structure. Now we can use the *ParameterFilter* to get *Datasets* measured using the RARE pulse sequence only.

```
study.filter(parameter='PULPROG', operator='==', value='<RARE.ppg>')
study.filter(type=Dataset)
study.print()
```

The resulting folder structure matches our needs now:

```
20200612_094625_lego_phantom_3_1_2 [Study]
  └── 8 [Experiment]
    └── fid [Dataset]
  └── 47 [Experiment]
    └── fid [Dataset]
```

CHAPTER

SIX

COMMAND LINE INTERFACE (CLI)

Together with the API a *bruker* command line tool is installed. It can be used with the following sub-commands:

- *report*
- *split*
- *filter*

It is also possible to create command line *pipelines* using our api

Data used in examples on this page are freely available at Zenodo:

- https://doi.org/10.5281/zenodo.4048253

It is possible to run examples in this section by setting an environment variable *DATA_PATH* to contain a path to the downloaded dataset:

```
export DATA_PATH={path_to_20200612_094625_lego_phantom_3_1_2}
```

6.1 report

For each data set contained in a folder specified by the *-i* argument, the report sub-command saves properties individual data sets into a JSON, or a YAML file located in the dataset folder, or a folder, or file specified using the *-o* argument. It is also possible to only export properties defined by the *-p* argument.

Save properties of all data sets contained within the 20200612_094625_lego_phantom_3_1_2 folder:

```
bruker report -i 20200612_094625_lego_phantom_3_1_2/3/pdata/1/2dseq
```

The following is the content of the resulting *report.json* file located in the same folder as the *2dseq* file:

```
{
    "TE": 4,
    "dim_type": [
        "spatial",
        "spatial",
        "<FG_SLICE>"
    ],
    "encoded_dim": 2,
    "shape_final": [
        256,
        256,
        3
```

(continues on next page)

(continued from previous page)


```
    ],
    "is_single_slice": false,
    "shape_fg": [
        3
    ],
    "numpy_dtype": "int16",
    "shape_block": [
        256,
        256
    ],
    "pv_version": "6.0.1",
    "date": "<class 'datetime.datetime'>",
    "shape_frames": [
        3
    ],
    "offset": [
        0,
        0,
        0
    ],
    "TR": 100,
    "slope": [
        0.00176479209428005,
        0.00176479209428005,
        0.00176479209428005
    ],
    "shape_storage": [
        256,
        256,
        3
    ],
    "num_slice_packages": 3
}
```


The reporting sub-command can be used in one of five possible cases:

- **folder in-place**
- **folder to folder**
- **dataset in-place**
- **dataset to file**
- **dataset to folder**

Each of these cases is described in the examples section.

usage:

```
bruker report [-h] -i INPUT [-f {json,yml}] [-p PROPS [PROPS ...]]
```

arguments:

- **-h, –help** show help message and exit
- **-i –input** path to a Bruker data set, or a folder containing Bruker data sets

- **-o –output** path to a folder, or a file to report to
- **-f, –format** format of report files, one of {json,yml}, default value is **json**
- **-p, –props** list of properties to export, if undefined, all properties are exported

examples:

The following list of scenarios is avaliable for the report subcommand:

- **Folder in-place** – for every dataset within the folder (recursively) save its report file to the same folder where the dataset is located. It is possible to choose format using the *-f* argument.:

```
bruker report -i ${DATA_PATH}/20200612_094625_lego_phantom_3_1_2/ -f yml
```

- **Folder to folder** – for every dataset within the folder (recursively) save all report files to a folder specified by the *-o* argument. It is possible to choose format using the *-f* argument.:

```
bruker report -i ${DATA_PATH}/20200612_094625_lego_phantom_3_1_2/ -o ${DATA_PATH}/
↪tmp/ -f yml
```

- **Dataset in-place** – for a dataset within the folder (recursively) save all report files to a folder specified by the *-o* argument. It is possible to choose format using the *-f* argument.:

```
bruker report -i ${DATA_PATH}/20200612_094625_lego_phantom_3_1_2/3/pdata/1/2dseq  -
↪f yml
```

- **Dataset to file** – for every dataset within the folder (recursively) save all report files to a folder specified by the *-o* argument. It is possible to choose format using the *-f* argument.:

```
bruker report -i 20200612_094625_lego_phantom_3_1_2/3/pdata/1/2dseq -o ${DATA_PATH}/
↪tmp/report.json
```

- **Dataset to folder** – for every dataset within the folder (recursively) save all report files to a folder specified by the *-o* argument. It is possible to choose format using the *-f* argument.:

```
bruker report -i 20200612_094625_lego_phantom_3_1_2/ -o ${DATA_PATH}/tmp  -f yml
```

- Say we are only interested in *TE* and *TR* properties and we want to specify name of the report file, to achieve this, we can use the *-p* argument.:

```
bruker report -i 20200612_094625_lego_phantom_3_1_2/3/pdata/1/2dseq -p TE TR
```

6.2 split

Usage:

```
bruker report [-h] -i INPUT [-f {json,yml}] [-p PROPS [PROPS ...]]
```

arguments:

- **-h, –help** show help message and exit
- **-i –input** path to a Bruker data set, or a folder containing Bruker data sets
- **-f, –format** format of report files, one of {json,yml}
- **-p, –props** list of properties to export, if undefined, all properties are exported

Split by **slice package**:

```
bruker split -i 20200612_094625_lego_phantom_3_1_2/43/pdata/2/2dseq -s
```

Split by **`FG_ISA`**:

```
bruker split -i 20200612_094625_lego_phantom_3_1_2/43/pdata/2/2dseq -f FG_ISA
```

Split by **`FG_ECHO`**:

```
bruker split -i 20200612_094625_lego_phantom_3_1_2/43/pdata/2/2dseq -f FG_ECHO
```

6.3 filter

The *filter* sub-command provides an option to make various queries on folders containing Bruker data. It is possible to list all data sets measured with the same pulse sequence, data sets measured during the last month,etc.

List all data sets measured using the EPI pulse sequence:

```
bruker filter -i ${DATA_PATH}/20200612_094625_lego_phantom_3_1_2 -q "#PULPROG=='<EPI.ppg>
↪'"
```

6.4 pipelines

It is possible to assemble pipelines using the Bruker API and xargs. Let us see some examples:

Using filter and report subcommands to only report datasets measured by the MGE sequence:

```
bruker filter -i /home/tomas/data/20200612_094625_lego_phantom_3_1_2/ -q "#PULPROG=='
↪<MGE.ppg>'" | xargs -I {} bruker report -i {} -o /home/tomas/data/reports
```

CHAPTER

SEVEN

DATASET

class brukerapi.dataset.**Dataset**(*path*, ***state*)

Data set is created using one binary file {fid, 2dseq, rawdata, ser, 1r, 1i} and several JCAMP-DX files (method, acqp, visu_pars,...). The JCAMP-DX files necessary for a creation of a data set are denoted as **essential**. Each of the binary data files (fid, 2dseq,...) has slightly different data layout, i.e. the . The data in the binary files is stored Since the individual types of b some features We distinguish By he name of the binary file we determine the **type** of data set.

Main components of a data set:

- **parameters**:

Meta data essential for construction of schema and manipulation with the binary data file.

- **properties**

Derived from parameters.

- **schema:**
  - SchemaFid
  - Schema2dseq
  - SchemaSer
  - SchemaRawdata

An object encapsulating all functionality dependent on metadata. It provides method to reshape data.

- **data:**
  - numpy.ndarray

Array containing the data read from any of the supported binary files.

Example:

```
from bruker.dataset import Dataset

dataset = Dataset('path/2dseq')
```

__init__(*path*, ***state*)

Constructor of Dataset

Dataset can be constructed either by passing a path to one of the SUPPORTED binary files, or to a directory containing it. It is possible, to create an empty object using the load switch.

Parameters
path – **str** path to dataset

Raise

UnsuportedDatasetType
In case *Dataset.type* is not in SUPPORTED

Raise

IncompleteDataset
If any of the JCAMP-DX files, necessary to create a Dataset instance is missing

`__str__()`

String representation is a path to the data set.

`__call__(**kwargs)`

Call self as a function.

`load()`

Load parameters, properties, schema and data. In case, there is a traj file related to a fid file, traj is loaded as well.

`unload()`

Unload parameters, properties, schema and data. In case, there is a traj file related to a fid file, traj is unloaded as well.

`load_parameters()`

Load all parameters essential for reading of given dataset type. For instance, type *fid* data set loads acqp and method file, from parent directory in which the fid file is contained.

`add_parameter_file(file)`

Load additional jcamp-dx file and add it to Dataset parameter space. It is later available via getters, or using the dot notation. :param file_type: JCAMP-DX file to add to the data set. Must be located in the same folder, or the first proc subfolder.

Example:

```
from bruker.dataset import Dataset

dataset = Dataset('.../2dseq')
dataset.add_parameter_file('method')
dataset['PVM_DwDir'].value
```

`load_properties()`

Load properties from two default configuration files. First configuration file contains core properties - properties essential for data loading, second contains custom properties - to provide more information about given data set, such as the date of measurement, the echo time, etc.

Some properties depend on values of parameters from JCAMP-DX files which are not essential for creating the dataset. For instance, the date property of the fid dataset type is dependent on the AdjStatePerScan. Such JCAMP-DX file can be added using the *add_parameter_file* function, then the *load_properties* function can be called to reevaluate values of properties, so that the properties dependent on parameters stored in non-essential JCAMP-DX files are loaded.

Example:

```
from bruker.dataset import Dataset

dataset = Dataset('.../fid')
dataset.add_parameter_file('AdjStatePerScan')
```

(continues on next page)

(continued from previous page)

```
dataset.load_properties()
dataset.date
```

load_schema()

Load the schema for given data set.

load_data()

Load the data file. The data is first read from the binary file to a data vector. Then the data vector is deserialized into a data array. The process of deserialization is different for each data set type and is implemented in the individual subclasses of the brukerapi.schemas.Schema, i.e. brukerapi.schemas.SchemaFid, brukerapi.schemas.Schema2dseq, brukerapi.schemas.SchemaRawdata, brukerapi.schemas.SchemaSer.

If the object was created with random_access=True, the data is not read, instead it can be accessed using sub-arrays.

called in the class constructor.

unload_data()

Remove the data array from the data set.

write(*path*, ***kwargs*)

Write the Dataset instance to the disk. This consists of writing the binary data file {fid, rawdata, 2dseq, ser,...} and respective JCAMP-DX files {method, acqp, visu_pars, reco}.

Parameters

- **path** – *str* Path to one of the supported data set types.
- **kwargs** –

Returns

report(*path=None*, *props=None*, *verbose=None*)

Save properties to JSON, or YAML file.

if path is None then save report in-place as path / self.id + '.json' if path is a path path to a folder then save report to path / self.id + '.json' if path is a json, or yml file save report to path

Parameters

- **path** – *str* path to a resulting report file
- **names** – *list* names of properties to be exported

to_json(*path=None*, *props=None*)

Save properties to JSON file.

Parameters

- **path** – *str* path to a resulting report file
- **names** – *list* names of properties to be exported

to_yaml(*path=None*, *props=None*)

Save properties to YAML file.

Parameters

- **path** – *str* path to a resulting report file
- **names** – *list* names of properties to be exported

`to_dict`(*props=None*)

Export properties as dict.

Parameters

- **path** – *str* path to a resulting report file
- **names** – *list* names of properties to be exported

`__weakref__`

list of weak references to the object (if defined)

`property data`

Data array.

Type
numpy.ndarray

`property traj`

Trajectory array loaded from a *traj* file

Type
numpy.ndarray

`property dim`

number of dimensions of the data array

Type
int

`property shape`

shape of data array

Type
tuple

CHAPTER

EIGHT

FOLDERS

brukerapi.folders.**random**() → x in the interval [0, 1).

class brukerapi.folders.**Folder**(*path: str*, *parent: Folder = None*, *recursive: bool = True*, *dataset_index: list = ['fid', '2dseq', 'ser', 'rawdata']*, *dataset_state: dict = {'load': False, 'parameter_files': [], 'property_files': []}*)

A representation of a generic folder. It implements several functions to simplify the folder manipulation.

__init__(*path: str*, *parent: Folder = None*, *recursive: bool = True*, *dataset_index: list = ['fid', '2dseq', 'ser', 'rawdata']*, *dataset_state: dict = {'load': False, 'parameter_files': [], 'property_files': []}*)

The constructor for Folder class.

Parameters

- **path** – path to a folder
- **parent** – parent `Folder` object
- **recursive** – recursively create sub-folders
- **dataset_index** – only data sets listed here will be indexed

Returns

validate()

Validate whether the given path exists an leads to a folder. :return: :raises `NotADirectoryError`:

__str__() → str

Return str(self).

__getattr__(*name: str*)

Access individual files in folder. `Dataset` and `JCAMPDX` instances are not loaded, to access the data and parameters, to load the data, use context manager, or the *load()* function.

Example:

```
with folder.fid as fid
    data = fid.data
    te = fid.EffectiveTE
```

Parameters

name – Name of Dataset, JCAMPDX, or Folder

Returns

__getitem__(*name*)

Access individual files in folder, dict style. `Dataset` and `JCAMPDX` instances are not loaded, to access the data and parameters, to load the data, use context manager, or the *load()* function.

Example:

```
with folder['fid'] as fid
    data = fid.data
    te = fid.EffectiveTE
```

Parameters
name – Name of `Dataset`, `JCAMPDX`, or `Folder` object

Returns

query(*query*)

Query each dataset in the folder recursively.

Parameters
query –

Returns

get_dataset_list() → list

List of `Dataset` instances contained in folder

get_dataset_list_rec() → list

List of `Dataset` instances contained in folder

get_jcampdx_list() → list

List of `JCAMPDX` instances contained in folder

get_experiment_list() → list

List of `Experiment` instances contained in folder and its sub-folders

get_processing_list() → list

List of `Processing` instances contained in folder and its sub-folders

get_study_list() → list

List of `Study` instances contained in folder and its sub-folders

make_tree(*recursive: bool = True*) → list

Make a directory tree containing brukerapi objects only

Parameters

- **self** –
- **recursive** – explore all levels of hierarchy

Returns

static contains(*path: str*, *required: list*) → bool

Checks whether folder specified by path contains files listed in required.

Parameters

- **path** – path to a folder
- **required** – list of required files

Returns

print(*level=0*, *recursive=True*)

Print structure of the `Folder` instance.

Parameters

- **level** – level of hierarchy
- **recursive** – print recursively

Returns

clean(*node: Folder = None*) → Folder

Remove empty folders from the tree

Parameters
node –

Returns
tree without empty folders

__weakref__

list of weak references to the object (if defined)

class brukerapi.folders.**Study**(*path: str*, *parent: Folder = None*, *recursive: bool = True*, *dataset_index: list = ['fid', '2dseq', 'ser', 'rawdata']*, *dataset_state: dict = {'load': False, 'parameter_files': [], 'property_files': []}*)

Representation of the Bruker Study folder. The folder contains a subject info and a number of experiment folders.

Tutorial *How to work with Bruker study?*

__init__(*path: str*, *parent: Folder = None*, *recursive: bool = True*, *dataset_index: list = ['fid', '2dseq', 'ser', 'rawdata']*, *dataset_state: dict = {'load': False, 'parameter_files': [], 'property_files': []}*)

The constructor for Study class.

Parameters

- **path** – path to a folder
- **parent** – parent `Folder` object
- **recursive** – recursively create sub-folders

Returns

validate()

Validate whether the given path exists an leads to a `Study` folder.

Raises
`NotStudyFolder`: if the path does not lead to folder, or the folder does not contain a subject file

get_dataset(*exp_id: str = None*, *proc_id: str = None*) → Dataset

Get a `Dataset` from the study folder. Fid data set is returned if *exp_id* is specified, 2dseq data set is returned if *exp_id* and *proc_id* are specified.

Parameters

- **exp_id** – name of the experiment folder
- **proc_id** – name of the processing folder

Returns
fid, or 2dseq `Dataset`

class brukerapi.folders.**Experiment**(*path: str*, *parent: Folder = None*, *recursive: bool = True*, *dataset_index: list = ['fid', 'ser', 'rawdata']*, *dataset_state: dict = {'load': False, 'parameter_files': [], 'property_files': []}*)

Representation of the Bruker Experiment folder. The folder can contain *fid*, *ser* a *rawdata.SUBTYPE* data sets. It can contain multiple `Processing` instances.

__init__(*path: str*, *parent: Folder = None*, *recursive: bool = True*, *dataset_index: list = ['fid', 'ser', 'rawdata']*, *dataset_state: dict = {'load': False, 'parameter_files': [], 'property_files': []}*)

The constructor for Experiment class.

Parameters

- **path** – path to a folder
- **parent** – parent `Folder` object
- **recursive** – recursively create sub-folders

Returns

validate()

Validate whether the given path exists an leads to a `Experiment` folder.

Raises
`NotExperimentFolder`: if the path does not lead to folder, or the folder does not contain an acqp file

class brukerapi.folders.**Processing**(*path*, *parent=None*, *recursive=True*, *dataset_index=['2dseq', '1r', '1i']*, *dataset_state: dict = {'load': False, 'parameter_files': [], 'property_files': []}*)

__init__(*path*, *parent=None*, *recursive=True*, *dataset_index=['2dseq', '1r', '1i']*, *dataset_state: dict = {'load': False, 'parameter_files': [], 'property_files': []}*)

The constructor for Processing class.

Parameters

- **path** – path to a folder
- **parent** – parent `Folder` object
- **recursive** – recursively create sub-folders

Returns

validate()

Validate whether the given path exists an leads to a `Processing` folder.

Raises
`NotProcessingFolder`: if the path does not lead to folder, or the folder does not contain an *visu_pars* file

CHAPTER

NINE

DATA FOR TESTING

9.1 PV 6.0.1 study

Folder: 20200612_094625_lego_phantom_3_1_2

This study contains datasets measured all standard pulse sequences provided in PV6.0.1.

9.2 PV 5.1 study

Folder: 0.2H2

This study contains datasets measured all standard pulse sequences provided in PV5.1.

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

B

C

E

F

G

M

P

Q

R

S

V