

---

# **ibllib Documentation**

**International Brain Laboratory**

**Feb 14, 2020**



---

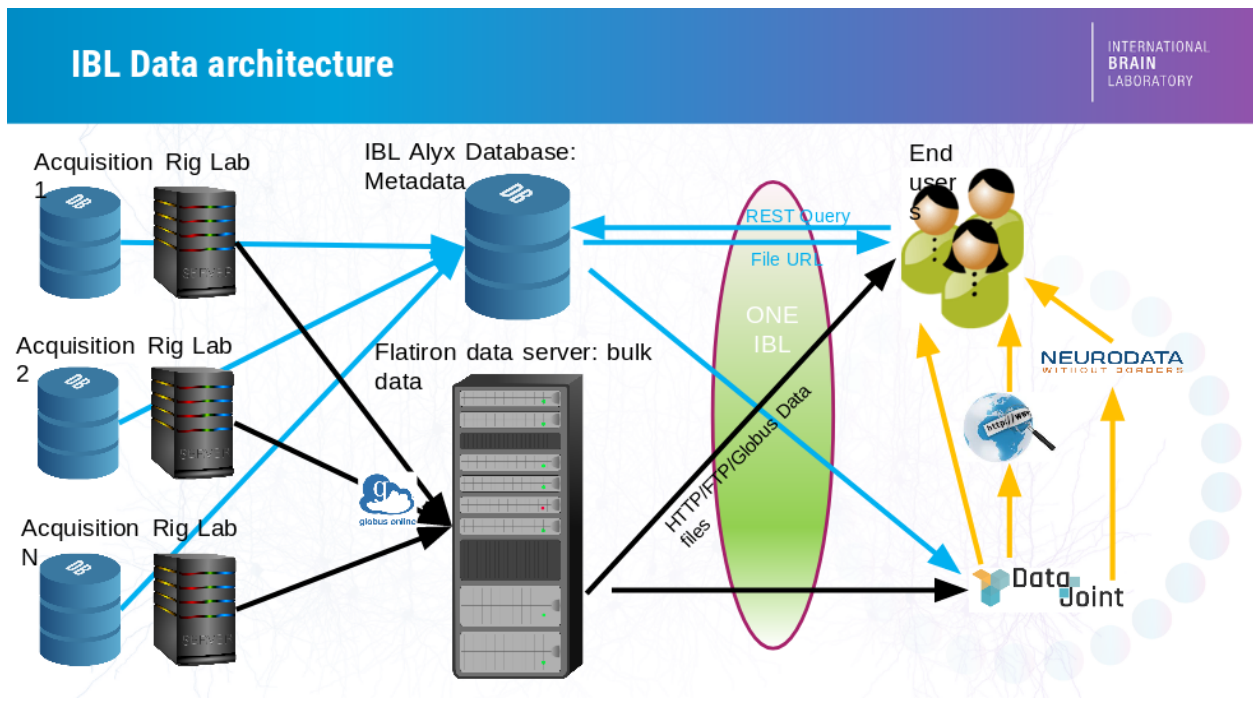
## Contents

---

<b>1 Overview</b>	<b>1</b>
<b>2 Python Installation and Tutorial</b>	<b>3</b>
<b>3 Matlab Installation and Tutorial</b>	<b>11</b>
<b>4 Reference Information on ALF and ONE</b>	<b>19</b>
<b>5 Docstrings Reference</b>	<b>25</b>
<b>Python Module Index</b>	<b>55</b>
<b>Index</b>	<b>57</b>



### 1.1 IBL data structure



data structure

- **Alyx**: database that stores meta-data in a relational manner. Lives on the cloud.
- **Datajoint**: processing Pipeline for IBL neuroscience data.
- **FlatIron**: storage of bulky raw data time-series. The Alyx database points to the files on this server. It is accessible through HTTP, FTP and Globus.

- **ONE**: set of normalized functions to access the IBL data. It queries the Alyx database and downloads data files from the FLaTIron.

### 1.1.1 Getting started:

Ipython notebook [here](#)

## 2.1 IBLLIB Python Installation Guide

### 2.1.1 Python-specific Dependencies

Python-specific dependency : **Python 3.6 or higher**.

#### Install Anaconda/Miniconda onto your machine

Download and install the Anaconda python distribution from here (choosing the right OS): <https://www.anaconda.com/download/#download> *Note* : Download the latest version.

### 2.1.2 Initialisation

Before you begin, make sure you have installed ibllib properly on your system as per the previous instructions. Make sure your computer is connected to an IBL accredited network.

The following steps will indicate how to :

1. setup a virtual environment using Anaconda
2. install requirements and packages
3. define ONE connection settings
4. test for the installation All of that can be done using shell terminal command lines.

#### Environment and ibllib setup

##### Using virtualenv

In a shell terminal, type the following commands:

```
cd ibllib/  
virtualenv iblenv --python=python3.7  
source ./venv/bin/activate  
pip install ibllib
```

### Using Anaconda (recommended for Windows users)

In a shell terminal, type the following commands:

```
cd ibllib/  
conda env create --name ibllib -f ibllib_conda.yaml  
pip install ibllib
```

### Troubleshooting environment issues

Our recipes have been tested for creating environments from scratch for Mac/Linux and Windows. Solving package dependencies can be challenging especially when trying to setup a scientific environment on top of an older environment. If you experience any problem it is recommended to start from a blank environment, and or update to the latest conda version.

### Instantiate One class: Define connection settings

The first step is to instantiate the **One class**: behind the scenes, the constructor connects to the IBL cloud database and gets credentials.

The connections settings are defined in a JSON parameter file (named `.one_params`).

- In Linux, the file is in `~/ .one_params`.
- In Windows, the file is in the Roaming App directory `C:\Users\CurrentUser\AppData\Roaming\ .one_params`.
- In Mac OS, the file is in the user directory `/Users/CurrentUser/ .one_params`.

In case of doubt, type the command `io.getappdir` in a Matlab prompt, from `pathlib` import `Path`; `print(Path.home())` in Python. It will return the directory of the JSON `.one_params` file.

**Note: The JSON `.one_params` file is uniquely stored, and shared across Matlab and Python.**

There are two ways to define the connection settings.

#### 1. The `setup()` static method

In a Python terminal, type:

```
from oneibl.one import ONE  
ONE.setup() # For first time use, need to define connection credentials
```

**Note:**

- you can access a python terminal from a shell terminal by typing the command `python` in the `ibllibenv` virtual environment) For an `ipython` console, `pip install ipython` in the environment.
- To run with *Spyder*, link the Python Interpreter to the virtual environment you created.



- To run with *Jupyter*, type in the command `jupyter notebook` with your virtual environment activated as above (`pip install jupyter` will install `jupyter` if it's not already available).

You will be asked to enter the following information:

```
ALYX_LOGIN                # Input your IBL user name
ALYX_PWD                  # Input your IBL password
ALYX_URL:                 # Should be automatically set as: https://
↪alyx.internationalbrainlab.org - press ENTER
CACHE_DIR:               # Local repository, can ammend or press_
↪ENTER
FTP_DATA_SERVER:         # Should be automatically set as: ftp://ibl.
↪flatironinstitute.org - press ENTER
FTP_DATA_SERVER_LOGIN:   # Should be automatically set as: iblftp - press ENTER
FTP_DATA_SERVER_PWD      # Request Password for FTP from Olivier
HTTP_DATA_SERVER:        # Should be automatically set as: http://ibl.
↪flatironinstitute.org - press ENTER
HTTP_DATA_SERVER_LOGIN:  # Should be automatically set as: iblmember - press ENTER
HTTP_DATA_SERVER_PWD     # Request Password for HTTP from IBL admins
```

**Note:** using one .setup changes the JSON *.one\_params* file. Again, the file is shared across Python and Matlab platforms.

## 2. Edit the JSON *.one\_params* file manually

**Note:** In Mac OS/Linux, use the command `nano` in a terminal.

Once the connections settings are defined, there is no need to setup the class `One` again if willing to connect with the credentials saved in the JSON *.one\_params* file.

The tutorial in the next section will show you how to change credentials withouth changing the JSON file (useful for seldom connection with different credentials).

## 2.2 IBLLIB Python Developer installation guide

### 1. Install Git onto your machine

- **Linux** : Git comes with Linux.
- **Mac OS** : Download and install Git from here: <https://git-scm.com/download/mac> .
- **Windows** : Download and install Git from here: <https://github.com/git-for-windows/git/releases/> .

### 2. Activate/install your target Python 3.6 or greater environment (latest always recommended). For example on Linux:

**Linux:**

```
cd ibllib/
virtualenv iblenv --python=python3.8
source ./venv/bin/activate
```

**Windows:**

```
conda create -n iblenv python=3.8
conda activate iblenv
```

### Mac OS:

```
conda create -n iblenv python=3.8 anaconda
source activate iblenv
```

#### 1. Clone the ibllib repository and install ibllib in place:

```
git clone https://github.com/int-brain-lab/ibllib.git
cd ibllib
pip install -r requirements.txt
pip install -e .
```

#### 1. Run tests

- In a shell terminal, in the ibllib folder with the proper environment activated:
  - **Linux and Mac OS:** `source run_tests`
  - **Windows:** call `run_tests.bat` Console output should end with OK

## 2.3 ONE Tutorial Python

Before you begin, make sure you have installed ibllib properly on your system as per the previous instructions. [Here](#) is a shorter introduction to the One module in Ipython notebook format.

### 2.3.1 Create ONE object

Once the One class is instantiated and setup, we can create an **one object** (here labelled `one`).

#### With default connection settings

Type in python prompt:

```
from oneibl.one import ONE
one = ONE() # need to instantiate the class to have the API. You will have to write_
↳these lines of code everytime you re-open python.
```

Reminder: connection parameters inserted via `one.setup()` will modify the JSON `.one_params` file [see here](#)

#### With different connection settings for single time use

For this tutorial we will be connecting to a **test database** with a **test user**. As these credentials will be used for this tutorial only, we do not want to change the base parameters of the JSON `.one_params` file.

```
from oneibl.one import ONE
one = ONE(username='test_user',
          password='TapetesBloc18',
          base_url='https://test.alyx.internationalbrainlab.org')
```

### 2.3.2 Find an experiment

Each experiment is identified by a unique string known as the “experiment ID” (EID). To find an EID, use the `one.search` command.

The following example shows how to find the experiment(s) performed by the user `olivier`, on the 24 Aug 2018:

```
from ibllib.misc import pprint
eid = one.search(users='olivier', date_range=['2018-08-24', '2018-08-24'])
pprint(eid)
```

returns

```
[
    "cf264653-2deb-44cb-aa84-89b82507028a"
]
```

For full information dictionary about the session:

```
eid, ses_info= one.search(users='olivier', date_range=['2018-08-24', '2018-08-24'],
↪ details=True)
```

The searchable fields are listed with the following method:

```
one.search_terms()

# Example search keywords:
['dataset_types',
 'date_range',
 'lab',
 'location',
 'number',
 'performance_gte',
 'performance_lte',
 'subject',
 'task_protocol',
 'users']
```

### 2.3.3 List method

Once you know the EID, you can list all the datasets for the experiment using the `list` command:

```
one.list(eid)
```

returns

```
['_ibl_lickPiezo.raw', '_ibl_lickPiezo.timestamps', '_ibl_wheel.position', 'channels.
↪ brainLocation', 'channels.probe', 'channels.rawRow', 'channels.site', 'channels.
↪ sitePositions', 'clusters._phy_annotation', 'clusters.depths', 'clusters.peakChannel
↪', 'clusters.probes', 'clusters.templateWaveforms', 'clusters.waveformDuration',
↪ 'eye.area', 'eye.blink', 'eye.timestamps', 'eye.xyPos', 'licks.times', 'probes.
↪ description', 'probes.insertion', 'probes.rawFilename', 'probes.sitePositions',
↪ 'spikes.amps', 'spikes.clusters', 'spikes.depths', 'spikes.times', 'spontaneous.
↪ intervals', 'unknown']
```

For more detailed datasets info, this will return a dataclass with `dataset_type`, `data_url` and `dataset_id` fields amongst others:

```
d = one.list(eid, details=True)
print(d)
```

To get the full contextual information about the session, get all fields:

```
one.list(eid, 'all')
```

To navigate the database, it may be useful to get the range of possible keywords values to search for sessions. For example to print a list of the dataset-types, users and subjects in the command window:

```
one.list(None, 'dataset-types')
one.list(None, 'users')
one.list(None, 'subjects')
one.list(None, 'labs')
```

### 2.3.4 Load method

#### General Use

To load data for a given EID, use the `one.load` command:

```
dataset_types = ['clusters.templateWaveforms', 'clusters.probes', 'clusters.depths']
eid = 'cf264653-2deb-44cb-aa84-89b82507028a'
wf, pr, d = one.load(eid, dataset_types=dataset_types)
```

Depending on the use case, it may be handier to wrap the arrays in a dataclass (a structure for Matlab users) so that a bit of context is included with the array. This would be useful when concatenated information for datasets belonging to several sessions.

```
my_data = one.load(eid, dataset_types=dataset_types, dclass_output=True)
from ibllib.misc import pprint
print(my_data.local_path)
pprint(my_data.dataset_type)
```

```
[
  "/home/owinter/Downloads/FlatIronCache/clusters.templateWaveforms.2291afac-1d42-
↪4021-a07c-c5539865f42c.npy",
  "/home/owinter/Downloads/FlatIronCache/clusters.probes.66567f54-a5f4-45d1-a9e6-
↪b103ece86339.npy",
  "/home/owinter/Downloads/FlatIronCache/clusters.depths.a26662b5-ff9c-4f15-a8cf-
↪5e9c9e85690f.npy"
]
[
  "clusters.templateWaveforms",
  "clusters.probes",
  "clusters.depths"
]
```

The dataclass contains the following keys, each of which contains a list of 3 items corresponding the the 3 queried datasets

- `data` (*numpy.array*): the numpy array
- `dataset_id` (*str*): the UUID of the dataset in Alyx
- `local_path` (*str*): the local full path of the file

- `dataset_type` (*str*): as per Alyx table
- `url` (*str*): the link on the FlatIron server
- `eid` (*str*): the session UUID in Alyx

It is also possible to query all datasets attached to a given session, in which case the output has to be a dictionary:

```
eid = one.search(subjects='flowers')
my_data = one.load(eid[0])
pprint(my_data.dataset_type)
```

### Specific cases

If a dataset type queried doesn't exist or is not on the FlatIron server, an empty list is returned. This allows to keep the proper order of output arguments

```
eid = 'cf264653-2deb-44cb-aa84-89b82507028a'
dataset_types = ['clusters.probes', 'thisDataset.IveJustMadeUp', 'clusters.depths']
t, empty, cl = one.load(eid, dataset_types=dataset_types)
```

Returns an empty list for *cr* so that *t* and *cl* still get assigned the corresponding datasets values.

## 2.3.5 Search method

The search methods allows to query the database to filter the list of UUIDs according to the following fields:

- `dataset_types`
- `users`
- `subject`
- `date_range`

### One-to-one matches: subjects

This is the simplest case that queries EEIDs (sessions) associated with a subject. There can only be one subject per session.

```
eid = one.search(subject='flowers')
pprint(eid)
```

The list of search terms can be queried through:

```
ONE.search_terms()
```

Here is the simple implementation of the filter, where we query for the EEIDs (sessions) co-owned by all of the following users: olivier and test\_user (case-sensitive).

```
eid = one.search(users=['test_user', 'olivier'])
```

To get all context information about the returned sessions, use the flag `details`:

```
eid, session_details= one.search(users=['test_user', 'olivier'], details=True)
```

The following would get all of the dataset for which olivier is an owner or a co-owner:

```
eid = one.search(users=['olivier'])  
pprint(eid)
```

It is also possible to filter sessions using a date-range:

```
eid = one.search(users='olivier', date_range=['2018-08-24', '2018-08-24'])
```

### 3.1 IBLLIB Matlab Installation Guide

#### 3.1.1 Matlab-specific Dependencies

Matlab-specific dependency : **Matlab R2016b or higher**.

#### Install Matlab onto your machine

Download and install the latest Matlab version from here: <https://www.mathworks.com/downloads/>

#### 3.1.2 Initialisation

Clone or download the repository here: <https://github.com/int-brain-lab/ibllib-matlab>

Launch Matlab. Set the Matlab path, add with subfolders the full `.\ibllib-matlab` directory.

#### Instantiate One class: Define connection settings

The first step is to instantiate the **One class**: behind the scenes, the constructor connects to the IBL cloud database and gets credentials.

The connections settings are defined in a JSON parameter file (named `.one_params`).

- In Linux, the file is in `~/ .one_params`.
- In Windows, the file is in the Roaming App directory `C:\Users\olivier\AppData\Roaming\ .one_params`.
- In Mac OS, the file is in the user directory `/Users/olivier/ .one_params`.

In case of doubt, type the command `io.getappdir` in a Matlab prompt. It will return the directory of the JSON `.one_params` file.

There are two manners to define the connection settings.

### 1. The `setup()` static method in Matlab

In a Matlab prompt, write:

```
One.setup
```

You will be asked to enter the following information:

```
ALYX_LOGIN           % Input your IBL user name
ALYX_PWD             % Input your IBL password
ALYX_URL:           % Should be automatically set as: https://
↳alyx.internationalbrainlab.org - press ENTER
CACHE_DIR:         % Local repository, can ammend or press_
↳ENTER
FTP_DATA_SERVER:    % Should be automatically set as: ftp://ibl.
↳flatironinstitute.org - press ENTER
FTP_DATA_SERVER_LOGIN: % Should be automatically set as: iblftp - press ENTER
FTP_DATA_SERVER_PWD % Request Password for FTP from Olivier
HTTP_DATA_SERVER:   % Should be automatically set as: http://ibl.
↳flatironinstitute.org - press ENTER
HTTP_DATA_SERVER_LOGIN: % Should be automatically set as: iblmember - press ENTER
HTTP_DATA_SERVER_PWD % Request Password for HTTP from Olivier
```

The path to the `.one_params` file is displayed in the Matlab prompt as `ans`.

**Note:** using `One.setup` changes the JSON `.one_params` file. Also note that the file is shared across Python and Matlab platforms.

### 2. Edit the JSON `.one_params`

**Note:** In Mac OS or Linux, use the command `nano` in a terminal.

Once the connections settings are defined, there is no need to setup the class `One` again if willing to connect with the credentials saved in the JSON `.one_params` file.

The tutorial in the next section will show you how to change credentials without changing the JSON file (useful for a temporary connection with different credentials).

### Run tests

Once the `One` class is instantiated with your IBL credentials, run the suite of Unit tests to check the installation:

```
RunTestsIBL('All')
```

If you see any Failure message, please report on GitHub or contact data admins (Olivier / Niccolo). If not, you are ready for the tutorial - go to next section !



## 3.2 ONE Tutorial Matlab

Before you begin, make sure you have installed ibllib properly on your system as per the previous instructions.

### 3.2.1 Create ONE object

Once the One class is instantiated and setup, we can create an **one object** (here labelled `one`).

#### With default connection settings

Type in Matlab prompt:

```
one = One(); % this line of code will be the first line to write everytime you re-
↳open Matlab
```

**Reminder:** connection parameters inserted via `one.setup` will modify the JSON `.one_params` file see [installation notes here](#)

#### With different connection settings for single time use

For this tutorial we will be connecting to a **test database** with a **test user**. As these credentials will be used for this tutorial only, we do not want to change the base parameters of the JSON `.one_params` file.

To change the credentials without changing the JSON `.one_params` file, type:

```
one = One(      'alyx_login', 'test_user', ...
               'alyx_pwd', 'TapetesBloc18', ...
               'alyx_url', 'https://test.alyx.internationalbrainlab.org');
```

You now have created an `one` object. This object has several fields, the following are of interest for analysis. Type `help` to retrieve the documentation on each field:

```
help one.list
help one.load
help one.search
```

### 3.2.2 Find an experiment

Each experiment is identified by a unique string known as the “experiment ID” (EID). To find an EID, use the `one.search` command.

The following example shows how to find the experiment(s) performed by the user `olivier`, on the 24 Aug 2018:

```
[eid, ses] = one.search('users', {'olivier'}, 'date_range', datenum([2018 8 24 ; 2018_
↳8 24]));
```

returns

```
eid =
    'cf264653-2deb-44cb-aa84-89b82507028a'
```

The searchable fields are listed by calling the `search` method with no parameters:

```
one.search

% Example search keywords:
    'dataset_types'
    'date_range'
    'labs'
    'subjects'
    'users'
```

### 3.2.3 List method

Once you know the EID, you can list all the datasets for the experiment using the list command:

```
one.list(eid)
```

returns

```
ans =

29x1 cell array

    {'_ibl_lickPiezo.raw'          }
    {'_ibl_lickPiezo.timestamps'  }
    {'_ibl_wheel.position'        }
    {'_ibl_wheel.timestamps'     }
    {'channels.brainLocation'     }
    {'channels.probe'            }
    {'channels.rawRow'           }
    {'channels.site'             }
    {'channels.sitePositions'    }
    {'clusters._phy_annotation'   }
    {'clusters.depths'           }
    {'clusters.peakChannel'      }
    {'clusters.probes'           }
    {'clusters.templateWaveforms'}
    {'clusters.waveformDuration' }
    {'eye.area'                  }
    {'eye.blink'                 }
    {'eye.timestamps'            }
    {'eye.xyPos'                 }
    {'licks.times'               }
    {'probes.description'        }
    {'probes.insertion'          }
    {'probes.rawFilename'        }
    {'probes.sitePositions'      }
    {'spikes.amps'               }
    {'spikes.clusters'           }
    {'spikes.depths'             }
    {'spikes.times'              }
    {'spontaneous.intervals'     }
```

For more detailed datasets info, this command will return a dataclass with `dataset_type`, `data_url` and `dataset_id` fields amongst others:

```
[dtypes details] = one.list(eid);
details =
```

(continues on next page)

(continued from previous page)

```

struct with fields:
    id: {29x1 cell}
    name: {29x1 cell}
    dataset_type: {29x1 cell}
    data_url: {29x1 cell}
    data_format: {29x1 cell}
    url: {29x1 cell}
    eid: {29x1 cell}

```

To get the full contextual information about the session, get all fields:

```
ses_info = one.list(eid, 'keyword', 'all')
```

To navigate further the database, it may be useful to get the range of possible keywords values to search for sessions. For example to print a list of the dataset-types, users and subjects in the command window:

```

one.list([], 'keyword', 'labs')
one.list([], 'keyword', 'datasets')
one.list([], 'keyword', 'users')
one.list([], 'keyword', 'subjects')

```

### 3.2.4 Load method

#### General Use

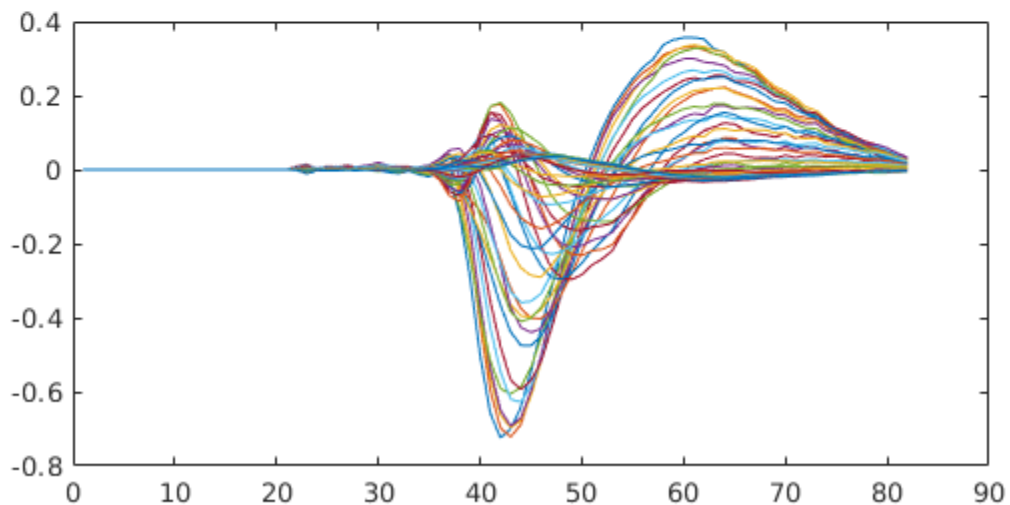
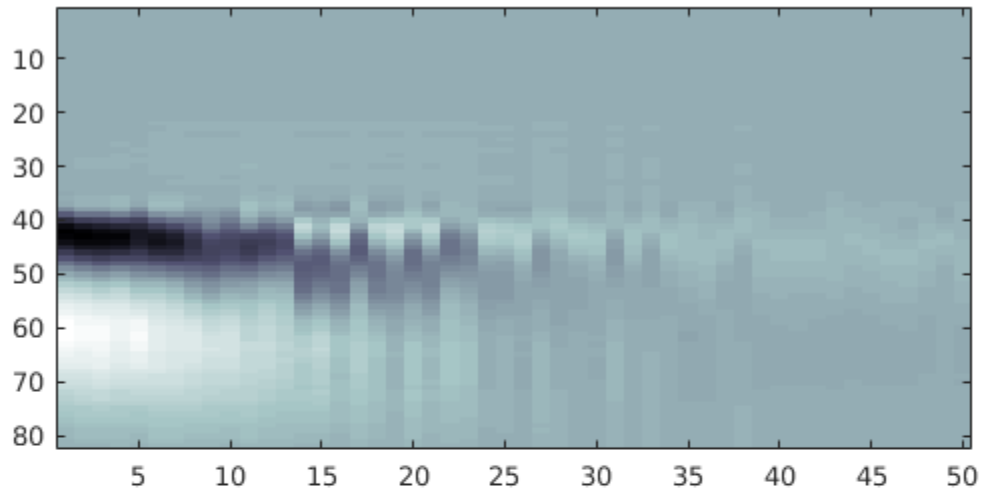
To load data for a given EID, use the `One.load` command:

```

dataset_types = {'clusters.templateWaveforms', 'clusters.probes', 'clusters.depths'};
eid = 'cf264653-2deb-44cb-aa84-89b82507028a';
[wf, pr, d] = one.load(eid, 'data', dataset_types);

figure,
imagesc(squeeze(wf(1, :, :)), 'Parent', subplot(2,1,1)), colormap('bone')
plot(subplot(2,1,2), squeeze(wf(1, :, :)))

```



Alyx

data structure

Depending on the use case, it may be handier to wrap the arrays in a dataclass (a structure for Matlab users) so that a bit of context is included with the array. This would be useful when concatenated information for datasets belonging to several sessions.

```
D = one.load(eid, 'data', dataset_types, 'dclass_output', true);  
  
disp(D.local_path)  
disp(D.dataset_type)  
disp('dimensions: ')  
disp(cellfun(@(x) length(x), D.data))
```

will print in the command window the following:

```

    '/home/owinter/Downloads/FlatIron/mainenlab/Subjects/clns0730/2018-08-24/1/
↳clusters.templateWaveforms.5e7f6ede-2618-41d2-95ee-ce144ea12851.npy'
    '/home/owinter/Downloads/FlatIron/mainenlab/Subjects/clns0730/2018-08-24/1/
↳clusters.probes.c41dd877-d511-42cb-90a3-01bb19297117.npy'
    '/home/owinter/Downloads/FlatIron/mainenlab/Subjects/clns0730/2018-08-24/1/
↳clusters.depths.42507ace-1ced-4358-a5ef-c4ddd8b7071c.npy'

    'clusters.templateWaveforms'
    'clusters.probes'
    'clusters.depths'

dimensions:
    1109
    1109
    1109

```

The dataclass contains the following keys, each of which contains a list of 3 items corresponding to the 3 queried datasets

- `data` cell (\*array\*): the array
- `dataset_id` cell (\*str\*): the UUID of the dataset in Alyx
- `local_path` cell (\*str\*): the local full path of the file
- `dataset_type` cell (\*str\*): as per Alyx table
- `url` cell (\*str\*): the link on the FlatIron server
- `eid` cell (\*str\*): the session UUID in Alyx

It is also possible to query all datasets attached to a given session, in which case the output has to be a table/structure as seen above:

```

[eid, ses_info ]= one.search('subjects', 'flowers');
D = one.load(eid);

```

In this case it will return a table with 5 datasets.

### Specific cases

If a dataset type queried doesn't exist or is not on the FlatIron server, an empty list is returned. This allows to keep the proper order of output arguments

```

eid = 'cf264653-2deb-44cb-aa84-89b82507028a';
dataset_types = {'clusters.probes', 'thisDataset.IveJustMadeUp', 'clusters.depths'};
[t, empty, cl ] = one.load(eid, 'data', dataset_types)
isempty(empty) % true !

```

Returns an empty array for `empty` so that `t` and `cl` still get assigned the corresponding datasets values.

### 3.2.5 Search method

The search methods allows to query the database to filter the list of UUIDs according to the following fields:

- `dataset_types`
- `users`

- subject
- date\_range

### One-to-one matches: subjects

This is the simplest case that queries EEIDs (sessions) associated with a subject. There can only be one subject per session.

```
eid = one.search('subject', 'flowers');
```

Another use case is to query EEIDs associated with a dataset type.

```
[eids,ses] = one.search('data', 'channels.brainLocation')
```

Here is the simple implementation of the filter, where we query for the EEIDs (sessions) co-owned by all of the following users: olivier and niccolo (case-sensitive).

```
eid = one.search('users',{'test_user', 'olivier'});
```

The following would get all of the dataset for which olivier is an owner or a co-owner:

```
[eid, ses_info] = one.search('users', 'olivier');
```

Note that unlike the first example, here we used an optional second output argument , to get all context information about the returned sessions.

It is also possible to filter sessions using a date-range:

```
eid = one.search('users','olivier', 'date_range', ['2018-08-24'; '2018-08-24'])
```

In the example above we used strings to specify the date however it is a safer practice to cast the range as a 2 elements datenum vector.

```
drange = datenum(['2018-08-24'; '2018-08-24'])  
eid = one.search('users','olivier', 'date_range', drange)
```

---

## Reference Information on ALF and ONE

---

### 4.1 Reference

#### 4.1.1 Open Neurophysiology Environment

Neurophysiology needs data standardization. A scientist should be able to analyze data collected from multiple labs using a single analysis program, without spending untold hours figuring out new file formats. Substantial efforts have recently been put into developing neurodata file standards, with the most successful being the [Neurodata Without Borders](#) (NWB) format. The NWB format has a comprehensive and careful design that allows one to store all data and metadata pertaining to a neurophysiology experiment. Nevertheless, its comprehensive design also means a steep learning curve for potential users, which has limited its adoption.

#### How the open neurophysiology environment works

Here we provide a solution to this problem: a set of four simple functions, that allow users to access and search data from multiple sources. To adopt the standard, data providers can use any format they like - all they need to do is implement these functions to fetch the data from their server and load it into Python or MATLAB. Users can then analyze this data with the same exact code as data from any other provider. The learning curve will be simple, and the loader functions would be enough for around 90% of common use cases.

By a *data provider* we mean an organization that hosts a set of neurophysiology data on an internet server (for example, the [International Brain Lab](#)). The Open Neurophysiology Environment (ONE) provides a way for scientists to analyze data from multiple data providers using the same analysis code. There is no need for the scientist to explicitly download data files or understand their format - this is all handled seamlessly by the ONE framework. The ONE protocol can also be used to access a scientist's own experiments stored on their personal computer, but we do not describe this use-case here.

When a user wants to analyze data released by a provider, they first import that provider's loader functions. In python, to analyze IBL data, they would type

```
from oneibl.one import ONE
```

Because it is up to data providers to maintain the loader functions, all a user needs to do to work with data from a specific provider is import their loader module. To analyze Allen data, they could instead type `import one_allen`. After that, all other analysis code will be the same, regardless of which provider's data they are analyzing.

Every experiment a data provider releases is identified by an *experiment ID* (eID) – a small token that uniquely identifies a particular experiment. It is up to the data provider to specify the format of their eIDs.

### Loading data

If a user already knows the eID of an experiment they are interested in, they can load data for the experiment using a command like:

```
st, sc, cbl = ONE.load(eID, dataset_types=['spikes.times', 'spikes.clusters',
↳ 'clusters.brain_location'])
```

This command will download three datasets containing the times and cluster assignments of all spikes recorded in that experiment, together with an estimate of the brain location of each cluster. (In practice, the data will be cached on the user's local machine so it can be loaded repeatedly with only one download.)

Many neural data signals are time series, and synchronizing these signals is often challenging. ONE will provide a function to interpolate any required timeseries to an evenly or unevenly-sampled timescale of the user's choice. For example the command:

```
hxy, hth, t = ONE.loadTS(eID, dataset_types=['headTracking.xyPos', 'lfp.raw'], sample_
↳ rate=1000)
```

would interpolate head position and lfp to a common 1000 Hz sampling rate. The sample times are returned as `t`.

### Searching for experiments

Finally, a user needs to be able to search the data released by a provider, to obtain the eIDs of experiments they want to analyze. To do so they would run a command like:

```
eIDs = ONE.search(lab='CortexLabUCL', subject='hercules', dataset_types=['spikes.times
↳ ', 'spikes.clusters', 'headTracking.xyPos'])
eIDs, eInfo = ONE.search(details=True, lab='CortexLabUCL', subject='hercules',
↳ dataset_types=['spikes.times', 'spikes.clusters', 'headTracking.xyPos'])
```

This would find the eIDs for all experiments collected in the specified lab for the specified experimental subject, for which all of the required data is present. There will be more metadata options to refine the search (e.g. dates, genotypes, experimenter), and additional metadata on each matching experiment is returned in `eInfo`. However, the existence of dataset types is normally enough to find the data you want. For example, if you want to analyze electrophysiology in a particular behavior task, the experiments you want are exactly those with datasets describing the ephys recordings and that task's parameters.

### Standardization

The key to ONE's standardization is the concept of a “standard dataset type”. When a user requests one of these (such as `'spikes.times'`), they are guaranteed that each data provider will return them the same information, organized in the same way - in this case, the times of all extracellularly recorded spikes, measured in seconds relative to experiment start, and returned as a 1-dimensional column vector. It is guaranteed that any dataset types of the form `*.times` or `*.*_times` will be measured in seconds relative to experiment start. Furthermore, all dataset types differing only in their last word (e.g. `spikes.times` and `spikes.clusters`) will have the same number of rows, describing multiple attributes of the same objects. Finally, words matching across dataset types encode references: for



example, `spikes.clusters` is guaranteed to contain integers, and to find the brain location of each of these one looks to the corresponding row of `clusters.brain_location`, counting from 0.

Not all data can be standardized, since each project will do unique experiments. Data providers can therefore add their own project-specific dataset types. The list of standard dataset types will be maintained centrally, and will start small but increase over time as the community converges on good ways to standardize more information. It is therefore important to distinguish dataset types agreed as universal standards from types specific to individual projects. To achieve this, names beginning with an underscore are guaranteed never to be standard. It is recommended that nonstandard names identify the group that produces them: for example the dataset types `_ibl_trials.stimulusContrast` and `clusters._ibl_task_modulation` could contain information specific to the IBL project.

## Versioning and subcollections

Data are often released in multiple versions. Most users will want always to have the latest version, but sometimes a user working will want to continue working with a historical version even after it has been updated, to maintain consistency with previous work. To enable versioning, the ONE functions will have accept an optional argument of the form `version='v1'`, which will ensure this specific version is loaded. If the argument is not passed, the latest version will be loaded.

Sometimes the data will contain multiple measurements of the same type, for example if recordings are made with multiple recording probes simultaneously. In these cases, the dataset types for probe 0 will have names like `probe00/spikes.times`, `probe00/spikes.clusters`, and `probe00/clusters.brain_location`; data for probe 1 will be `probe01/spikes.times`, etc. Encoding of references works within a subcollection: i.e. the entries of `probe00/spikes.clusters` point to the rows of `probe00/clusters.brain_location`, starting from 0, and independently of any datasets starting with `probe01/`.

## For data sharers

Data standards are only adopted when they are easy to use, for both providers and users of data. For users, the three ONE functions will be simple to learn, and will cover most common use cases.

For providers, a key advantage of this framework is its low barrier to entry. To share data with ONE, providers do not need to run and maintain a backend server, just to upload their data to a website. We provide a “ONE light” implementation of the ONE loader functions that searches, downloads and caches files from a web server. This will allow producers who do not have in-house computational staff two simple paths to achieve ONE compatibility. The first is to place the data in a directory, using a standard file-naming convention described below, in standard formats including `numpy`, `csv`, `json`, and `tiff`. Next, the user runs a program in this directory, which uploads the files to a website or to `figshare`. Users can then access this data using ONE light. An example of ONE light data is [here](#). The ONE light code is [here](#).

### 4.1.2 ALF

ALF stands for “ALyx Files”. It not a format but rather a format-neutral file-naming convention. ALF is how IBL organizes files that will be loaded via the ONE protocol.

In ALF, the measurements in an experiment are represented by collections of files in a directory. Each filename has three parts, for example `spikes.times.npy` or `spikes.clusters.npy`. The first two correspond to the ONE dataset type, and we refer to them as the *object* and the *attribute*. The third part of the filename, the *extension*, specifies what physical format the file is in - we primarily use `.npy` and `.tsv` but you could use any format, for example video or `json`.

Each file contains information about particular attribute of the object. For example `spikes.times.npy` indicates the times of spikes and `spikes.clusters.npy` indicates their cluster assignments. You could have another file `spikes.amplitudes.npy` to convey their amplitudes. The important thing is that every file describing an object

has the same number of rows (i.e. the 1st dimension of an npy file, number of frames in a video file, etc). You can therefore think of the files for an object as together defining a table, with column headings given by the attribute in the file names, and values given by the file contents.

ALF objects can represent anything. But three types of object are special:

### Event series

If there is a file with attribute `times`, (i.e. filename `obj.times.ext`), it indicates that this object is an event series. The `times` file contains a numerical array containing times of the events in seconds, relative to a universal timescale common to all files. Other attributes of the events are stored in different files. If you want to represent times relative to another timescale, do this by appending to `timescale` after an underscore (e.g. `spikes.times_ephysClock.npy`). By convention, any other file with attribute that ends in `_times` is understood to be a time in universal seconds; for example `trials.reward_times.npy`. An attribute ending with `_times_timescale` is by convention a time in that timescale.

### Interval series

If there is a file with attribute `intervals`, (i.e. filename `tones.intervals.npy`), it should have two columns, indicating the start and end times of each interval relative to the universal timescale. Again, other attributes of the events can be stored in different files (e.g. `tones.frequencies.npy`). Event times relative to other timescales can be represented by a file with attribute `intervals_timescale`. Again, any other attributes of the form `trials.cue_intervals.npy` are by convention measured in universal seconds.

### Continuous timeseries

If there is a file with attribute `timestamps`, it indicates the object is a continuous timeseries. The `timestamps` file represents information required to synchronize the timeseries to the universal timebase, even if they were unevenly sampled. There are 2 possibilities:

- The `timestamps` file contains a single column containing time of every sample within the timescale.
- The `timestamps` file contains two rows. Each row of the `timestamps` file represents a synchronization point, with the first column giving the sample number (counting from 0), and the second column giving the corresponding time in universal seconds. The times corresponding to all samples are then found by linear interpolation. Note that the `timestamps` file is an exception to the rule that all files representing a continuous timeseries object must have one row per sample, as it will often have substantially less. Note that an evenly-sampled recording should have just two timestamps, giving the times of the first and last sample.

### File types

ALF can deal with any sort of file, as long as it has a concept of a number of rows (or primary dimension). The type of file is recognized by its extension. Preferred choices:

`.npy`: numpy array file. This is recommended over flat binary since datatype and shape is stored in the file. If you want to name the columns, use a metadata file. If you have an array of 3 or more dimensions, the first dimension counts as the number of rows. To access npy files from MATLAB use [this](#).

`.tsv`: tab-delimited text file. This is recommended over comma-separated files since text fields often have commas in. All rows should have the same number of columns. The first row contains tab-separated names for each column.

`.bin`: flat binary file. It's better to use `.npy` for storing binary data but some recording systems save in flat binary. Rather than convert them, you can ALFize a flat binary file by adding a metadata file, which specifies the number of

columns (as the size of the “columns” array) and the binary datatype as a top-level key “dtype”, using numpy naming conventions.

## Relations

Encoding of relations between objects can be achieved by a simplified relational model. If the attribute name of one file matches the object name of a second, then the first file is guaranteed to contain integers referring to the rows of the second. For example, `spikes.clusters.npy` would contain integer references to the rows of `clusters.brain_location.json` and `clusters.probes.npy`; and `clusters.probes.npy` would contain integer references to `probes.insertion.json`. Be careful of plurals (`clusters.probe.npy` would not correspond to `probes.insertion.json`) and remember we count arrays starting from 0.

## Longer file names

A proposed extension to ALF would allow encoding of additional information in filenames with more than 3 parts. In this proposal, file names could have as many parts as you like: `object.attribute.x1.x2. ... .xN.extension`. The extra name parts play no formal role in the ALF conventions, but can serve several additional purposes. For example, if you want unique file names to make archiving and backups easier, they could contain a unique string, for example a Dataset ID from Alyx, or the md5 hash of the file. Extra parts could be used to encode the subject name if you are worried about accidentally moving files between directories. The filenames might get long; however the important information in the filename is in the first two parts, which users can tab-autocomplete when typing them at the command line; also, because the extension is last, they can also double-click the file to open it with a default application such as a movie viewer.

Finally, if there are multiple files with the same object, attribute, and extensions but different extra parts, these should be treated as files to be concatenated, for example to allow multiple-part tif files as produced by `scnimage` to be encoded in ALF. The concatenation would happen in hierarchical lexicographical order: i.e. by lexicographic order of `x1`, then `x2` if `x1` is equal, etc.

## Metadata

Sometimes you will want to provide metadata on the columns or rows of a data file. For example, `clusters.ccf_location.tsv` could be a 4-column tab-delimited text file in which the first 3 columns contain xyz coordinates of a cluster and the 4th contains its inferred brain location as text. In this case, an additional JSON file `clusters.ccf_location.metadata.json` can provide information about the columns and rows. The metadata file can contain anything, but if it has a top-level key “columns”, that should be an array of size the number of columns, and if it has a top-level key “rows” that should be an array of size the number of rows. If the entries in the columns and rows arrays have a key “name” that defines a name for the column or row; a key “unit” defines a unit of measurement. You can add anything else you want.

Note that in ALF you should not have generally two data files with the same object and attribute: if you have `tones.frequencies.npy`, you can’t also have `tones.frequencies.tsv`. Metadata files are an exception to this: if you have `tones.frequencies.npy`, you can also have `tones.frequencies.metadata.json`.

## Versioning and subcollections

Versioning and subcollections are achieved with subdirectories. If your main ALF directory contains a subfolder `v1`, any files in there are assumed to be version `v1`. The root directory is assumed to contain the most recent files.

Subcollections are also achieved with subdirectories: a subfolder `probe00` of the main ALF directory contains the files specific to probe 0; you can also have a subcollection within a version such as `v1/probe00`.



## 5.1 Python Modules Docstrings

### 5.1.1 alf module

---

<code>alf.io</code>	Generic ALF I/O module.
<code>alf.folders</code>	

---

#### **alf.io**

Generic ALF I/O module. Provides support for time-series reading and interpolation as per the specifications For a full overview of the scope of the format, see: [https://ibllib.readthedocs.io/en/develop/04\\_reference.html#alf](https://ibllib.readthedocs.io/en/develop/04_reference.html#alf)

**class** `alf.io.AlfBunch` (\*args, \*\*kwargs)

**append** (*b*, *inplace=False*)

Appends one bunch to another, key by key :param bunch: :return: Bunch

**to\_df** ()

Attempts to returns a pandas.DataFrame if all elements are arrays of the same length Returns the original bunch if it can't

`alf.io.check_dimensions` (*dico*)

Test for consistency of dimensions as per ALF specs in a dictionary. Raises a Value Error.

Alf broadcasting rules: only accepts consistent dimensions for a given axis a dimension is consistent with another if it's empty, 1, or equal to the other arrays dims [a, 1], [1, b] and [a, b] are all consistent, [c, 1] is not

**Parameters** *dico* – dictionary containing data

**Returns** status 0 for consistent dimensions, 1 for inconsistent dimensions

`alf.io.dataframe` (*adict*)

Converts an Bunch conforming to size conventions into a pandas Dataframe For 2-D arrays, stops at 10 columns

per attribute :return: pandas Dataframe

`alf.io.exists` (*alfpath*, *object*, *attributes=None*, *glob='.\*'*)

Test if ALF object and optionally specific attributes exist in the given path :param *alfpath*: str or `pathlib.Path` of the folder to look into :param *object*: str ALF object name :param *attributes*: list or list of strings for wanted attributes :param *glob*: (“.\*”) glob pattern to look for files or list of parts as per ALF specifications :return: Bool. For multiple attributes, returns True only if all attributes are found

`alf.io.get_session_path` (*path: Union[str, pathlib.Path]*) → `pathlib.Path`

Returns the session path from any filepath if the date/number pattern is found

`alf.io.is_session_path` (*path\_object*)

**Checks if the syntax corresponds to a session path. Note that there is no physical check** about existence nor contents

**Parameters** *path\_object* –

**Returns**

`alf.io.is_uuid_string` (*string: str*) → bool

Bool test for uuid version 4

`alf.io.load_file_content` (*fil*)

Returns content of files. Designed for very generic file formats: so far supported contents are *json*, *numpy*, *csv*, *tsv*, *ssv*, *jsonable*

**Parameters** *fil* – file to read

:return:array/json/pandas dataframe depending on format

`alf.io.load_object` (*alfpath*, *object=None*, *glob='.\*'*, *short\_keys=False*)

Reads all files (ie. attributes) sharing the same object. For example, if the file provided to the function is *spikes.times*, the function will load *spikes.time*, *spikes.clusters*, *spikes.depths*, *spike.amps* in a dictionary whose keys will be *time*, *clusters*, *depths*, *amps* Full Reference here: <https://github.com/cortex-lab/ALF> Simplified example: `_namespace_object.attribute.part1.part2.extension`

**Parameters**

- **alfpath** – any alf file pertaining to the object OR directory containing files
- **object** – if a directory is provided, need to specify the name of object to load
- **glob** – a file filter string like one used in glob: “.amps.” for example
- **short\_keys** – by default, the output dictionary keys will be compounds of attributes and any eventual parts separated by a dot. Use True to shorten the keys to the bare attribute.

**Returns** a dictionary of all attributes pertaining to the object

example: `spikes = ibllib.io.alf.load_object('/path/to/my/alffolder/', 'spikes')`

`alf.io.read_ts` (*filename*)

Load time-series from ALF format t, d = `alf.read_ts(filename)`

`alf.io.remove_uuid_file` (*file\_path*, *dry=False*)

Renames a file without the UUID and returns the new `pathlib.Path` object

`alf.io.remove_uuid_recursive` (*folder*, *dry=False*)

Within a folder, recursive renaming of all files to remove UUID

`alf.io.save_metadata` (*file\_alf*, *dico*)

Writes a meta data file matching a current alf file object. For example given an alf file *clusters.ccf\_location.ssv* this will write a dictionary in json format in *clusters.ccf\_location.metadata.json* Reserved keywords:

- **columns**: column names for binary tables.
- **row**: row names for binary tables.
- **unit**

#### Parameters

- **file\_alf** – full path to the alf object
- **dico** – dictionary containing meta-data.

**Returns** None

`alf.io.save_object_npy(alfpath, dico, object, parts=)`

Saves a dictionary in alf format using object as object name and dictionary keys as attribute names. Dimensions have to be consistent. Reference here: <https://github.com/cortex-lab/ALF> Simplified example: `_namespace_object.attribute.part1.part2.extension`

#### Parameters

- **alfpath** – path of the folder to save data to
- **dico** – dictionary to save to npy
- **object** – name of the object to save
- **parts** – extra parts to the ALF name

**Returns** List of written files

example: `ibllib.io.alf.save_object_npy('/path/to/my/alffolder/', spikes, 'spikes')`

## alf.folders

`alf.folders.find_sessions(folder: Union[str, pathlib.Path]) → List[str]`

Returns all sessions found in all subfolders of a main data folder

`alf.folders.find_subject_folders(folder: Union[str, pathlib.Path]) → List[pathlib.Path]`

Returns all subject folders found from a main data folder

`alf.folders.find_subject_names(folder: Union[str, pathlib.Path]) → List[str]`

Returns all subject names found from a main data folder

`alf.folders.next_num_folder(session_date_folder: str) → str`

Return the next number for a session given a session\_date\_folder

`alf.folders.remove_empty_folders(folder: Union[str, pathlib.Path]) → None`

Will iteratively remove any children empty folders

`alf.folders.session_name(path: Union[str, pathlib.Path]) → str`

Returns the session name (subject/date/number) string for any filepath using session\_path

`alf.folders.session_path(path: Union[str, pathlib.Path]) → str`

Returns the session path from any filepath if the date/number pattern is found

`alf.folders.subjects_data_folder(folder: pathlib.Path, rglob: bool = False) → pathlib.Path`

Given a root\_data\_folder will try to find a 'Subjects' data folder. If Subjects folder is passed will return it directly.

## 5.1.2 ibllib module

<code>ibllib.dsp.fourier</code>	Low-level functions to work in frequency domain for n-dim arrays
<code>ibllib.dsp.utils</code>	Window generator, front detections, rms
<code>ibllib.ephys.ephysqc</code>	Quality control of raw Neuropixel electrophysiology data.
<code>ibllib.io.extractors.biased_trials</code>	
<code>ibllib.io.extractors.biased_wheel</code>	
<code>ibllib.io.extractors.ephys_fpga</code>	
<code>ibllib.io.extractors.ephys_trials</code>	
<code>ibllib.io.extractors.training_audio</code>	
<code>ibllib.io.extractors.training_trials</code>	<b>ALF extractors</b> are a collection of functions that extract alf files from the PyBpod rig raw data.
<code>ibllib.io.extractors.training_wheel</code>	Training wheel extractor from Pybpod output.
<code>ibllib.io.flags</code>	
<code>ibllib.io.params</code>	
<code>ibllib.io.raw_data_loaders</code>	Raw Data Loader functions for PyBpod rig
<code>ibllib.io.spikeglx</code>	
<code>ibllib.misc.misc</code>	
<code>ibllib.misc.version</code>	
<code>ibllib.pipes.experimental_data</code>	Entry point to system commands for IBL behaviour pipeline.
<code>ibllib.pipes.extract_session</code>	Find task name Check if extractors for specific task exist Extract data OR return error to user saying that the task has no extractors
<code>ibllib.pipes.purge_rig_data</code>	Purge data from RIG - Find all files by rglob - Find all sessions of the found files - Check Alyx if corresponding datasetTypes have been registered as existing sessions and files on Flatiron - Delete local raw file if found on Flatiron
<code>ibllib.pipes.transfer_rig_data</code>	
<code>ibllib.graphic</code>	
<code>ibllib.plots</code>	

### ibllib.dsp.fourier

Low-level functions to work in frequency domain for n-dim arrays

`ibllib.dsp.fourier.bp` (*ts*, *si*, *b*, *axis=None*)

Band-pass filter in frequency domain

#### Parameters

- **ts** – time serie
- **si** – sampling interval in seconds
- **b** – cutout frequencies: 4 elements vector or list
- **axis** – axis along which to perform reduction (last axis by default)

**Returns** filtered time serie

`ibllib.dsp.fourier.fexpand` (*x*, *ns=1*, *axis=None*)

Reconstructs full spectrum from positive frequencies Works on the last dimension (contiguous in c-stored array)



**Parameters**

- **x** – numpy.ndarray
- **axis** – axis along which to perform reduction (last axis by default)

**Returns** numpy.ndarray

`ibllib.dsp.fourier.freduce(x, axis=None)`

Reduces a spectrum to positive frequencies only Works on the last dimension (contiguous in c-stored array)

**Parameters**

- **x** – numpy.ndarray
- **axis** – axis along which to perform reduction (last axis by default)

**Returns** numpy.ndarray

`ibllib.dsp.fourier.fscale(ns, si=1, one_sided=False)`

numpy.fft.fftfreq returns Nyquist as a negative frequency so we propose this instead

**Parameters**

- **ns** – number of samples
- **si** – sampling interval in seconds
- **one\_sided** – if True, returns only positive frequencies

**Returns** fscale: numpy vector containing frequencies in Hertz

`ibllib.dsp.fourier.hp(ts, si, b, axis=None)`

High-pass filter in frequency domain

**Parameters**

- **ts** – time serie
- **si** – sampling interval in seconds
- **b** – cutout frequencies: 2 elements vector or list
- **axis** – axis along which to perform reduction (last axis by default)

**Returns** filtered time serie

`ibllib.dsp.fourier.lp(ts, si, b, axis=None)`

Low-pass filter in frequency domain

**Parameters**

- **ts** – time serie
- **si** – sampling interval in seconds
- **b** – cutout frequencies: 2 elements vector or list
- **axis** – axis along which to perform reduction (last axis by default)

**Returns** filtered time serie

**ibllib.dsp.utils**

Window generator, front detections, rms

**class** `ibllib.dsp.utils.WindowGenerator` (*ns, nswin, overlap*)

`wg = WindowGenerator(ns, nswin, overlap)`

Provide sliding windows indices generator for signal processing applications. For straightforward spectrogram / periodogram implementation, prefer scipy methods !

Example of implementations in `test_dsp.py`.

**firstlast**

Generator that yields first and last index of windows

**Returns** tuple of [first\_index, last\_index] of the window

**print\_progress** ()

Prints progress using a terminal progress bar

**slice**

Generator that yields slices of windows

**Returns** a slice of the window

**slice\_array** (*sig, axis=-1*)

Provided an array or sliceable object, generator that yields slices corresponding to windows. Especially useful when working on memmpaps

**Parameters**

- **sig** – array
- **axis** – (optional, -1) dimension along which to provide the slice

**Returns** array slice Generator

**tyscale** (*fs*)

Returns the time scale associated with Window slicing (middle of window) :param fs: sampling frequency (Hz) :return: time axis scale

`ibllib.dsp.utils.falls` (*x, axis=-1, step=-1*)

Detects Falling edges of a voltage signal, returns indices

**Parameters**

- **x** – array on which to compute RMS
- **axis** – (optional, -1) negative value
- **step** – (optional, -1) value of the step to detect

**Returns** numpy array

`ibllib.dsp.utils.fronts` (*x, axis=-1, step=1*)

Detects Rising and Falling edges of a voltage signal, returns indices and

**Parameters**

- **x** – array on which to compute RMS
- **axis** – (optional, -1) negative value
- **step** – (optional, -1) value of the step to detect

**Returns** numpy array of indices, numpy array of rises (1) and falls (-1)

`ibllib.dsp.utils.rises` (*x, axis=-1, step=1*)

Detect Rising edges of a voltage signal, returns indices

**Parameters**

- **x** – array on which to compute RMS
- **axis** – (optional, -1)
- **step** – (optional, 1) amplitude of the step to detect

**Returns** numpy array

`ibllib.dsp.utils.rms(x, axis=-1)`  
Root mean square of array along axis

**Parameters**

- **x** – array on which to compute RMS
- **axis** – (optional, -1)

**Returns** numpy array

## ibllib.ephys.ephysqc

Quality control of raw Neuropixel electrophysiology data.

`ibllib.ephys.ephysqc.amplitude_cutoff(amplitudes, num_histogram_bins=500, histogram_smoothing_value=3)`

Calculate approximate fraction of spikes missing from a distribution of amplitudes

Assumes the amplitude histogram is symmetric (not valid in the presence of drift)

Inspired by metric described in Hill et al. (2011) J Neurosci 31: 8699-8705

**amplitudes** [numpy.ndarray] Array of amplitudes (don't need to be in physical units)

**fraction\_missing** [float] Fraction of missing spikes (0-0.5) If more than 50% of spikes are missing, an accurate estimate isn't possible

`ibllib.ephys.ephysqc.extract_rmsmap(fbin, out_folder=None, force=False)`

Wrapper for rmsmap that outputs `_ibl_ephysRmsMap` and `_ibl_ephysSpectra` ALF files

**Parameters**

- **fbin** – binary file in spike glx format (will look for attached metadata)
- **out\_folder** – folder in which to store output ALF files. Default uses the folder in which the `fbin` file lives.
- **force** – do not re-extract if all ALF files already exist
- **label** – string or list of strings that will be appended to the filename before extension

**Returns** None

`ibllib.ephys.ephysqc.isi_violations(spike_train, min_time, max_time, isi_threshold, min_isi=0)`

Calculate ISI violations for a spike train.

Based on metric described in Hill et al. (2011) J Neurosci 31: 8699-8705

modified by Dan Denman from cortex-lab/sortingQuality GitHub by Nick Steinmetz

`spike_train` : array of spike times `min_time` : minimum time for potential spikes `max_time` : maximum time for potential spikes `isi_threshold` : threshold for isi violation `min_isi` : threshold for duplicate spikes

**fpRate** [rate of contaminating spikes as a fraction of overall rate] A perfect unit has a `fpRate` = 0 A unit with some contamination has a `fpRate` < 0.5 A unit with lots of contamination has a `fpRate` > 1.0

num\_violations : total number of violations

ibllib.ephys.ephysqc.qc\_fpga\_task (fpga\_trials, alf\_trials)

:fpga\_task is the dictionary output of ibllib.io.extractors.ephys\_fpga.extract\_behaviour\_sync : bpod\_trials is the dictionary output of ibllib.io.extractors.ephys\_trials.extract\_all : alf\_trials is the ALF \_ibl\_trials object after extraction (alf.io.load\_object) :return: qc\_session, qc\_trials, True means QC passes while False indicates a failure

ibllib.ephys.ephysqc.rmsmap (fbin)

Computes RMS map in time domain and spectra for each channel of Neuropixel probe

**Parameters** fbin (*str* or *pathlib.Path*) – binary file in spike glx format (will look for attached metatdata)

**Returns** a dictionary with amplitudes in channeltime space, channelfrequency space, time and frequency scales

ibllib.ephys.ephysqc.spike\_sorting\_metrics (spike\_times, spike\_clusters, spike\_amplitudes, params={'drift\_metrics\_interval\_s': 51, 'drift\_metrics\_min\_spikes\_per\_interval': 10, 'isi\_threshold': 0.0015, 'max\_spikes\_for\_nn': 10000, 'max\_spikes\_for\_unit': 500, 'min\_isi': 0.000166, 'n\_neighbors': 4, 'n\_silhouette': 10000, 'num\_channels\_to\_compare': 13, 'presence\_bin\_length\_secs': 20, 'quality\_metrics\_output\_file': 'metrics.csv'}, epochs=None)

Spike sorting QC metrics

ibllib.ephys.ephysqc.validate\_ttl\_test (ses\_path, display=False)

For a mock session on the Ephys Choice world task, check the sync channels for all device properly connected and perform a synchronization if dual probes to check that all channels are recorded properly :param ses\_path: session path :param display: show the probe synchronization plot if several probes :return: True if tests pass, errors otherwise

### ibllib.io.extractors.biased\_trials

ibllib.io.extractors.biased\_trials.get\_contrastLR (session\_path, save=False, data=False, settings=False)

Get left and right contrasts from raw datafile. Optionally, saves \_ibl\_trials.contrastLeft.npy and \_ibl\_trials.contrastRight.npy to alf folder.

Uses signed\_contrast to create left and right contrast vectors.

**Parameters**

- **session\_path** (*str*) – absolute path of session folder
- **save** (*bool*, *optional*) – whether to save the corresponding alf file to the alf folder, defaults to False

**Returns** numpy.ndarray

**Return type** dtype('float64')

### ibllib.io.extractors.biased\_wheel

**ibllib.io.extractors.ephys\_fpga**

`ibllib.io.extractors.ephys_fpga.align_with_bpod(session_path)`

Reads in trials.intervals ALF dataset from bpod and fpga. Asserts consistency between datasets and compute the median time difference

**Parameters** `session_path` –

**Returns** dt: median time difference of trial start times (fpga - bpod)

`ibllib.io.extractors.ephys_fpga.extract_all(session_path, save=False, tmax=None)`

**For the IBL ephys task, reads ephys binary file and extract:**

- sync
- wheel
- behaviour
- video time stamps

**Parameters**

- `session_path` – '/path/to/subject/yyyy-mm-dd/001'
- `save` – Bool, defaults to False
- `version` – bpod version, defaults to None

**Returns** None

`ibllib.io.extractors.ephys_fpga.extract_behaviour_sync(sync, output_path=None, save=False, chmap=None, display=False, tmax=inf)`

Extract wheel positions and times from sync fronts dictionary

**Parameters**

- `sync` – dictionary 'times', 'polarities' of fronts detected on sync trace for all 16 chans
- `output_path` – where to save the data
- `save` – True/False
- `chmap` – dictionary containing channel index. Default to constant. `chmap = {'bpod': 7, 'frame2ttl': 12, 'audio': 15}`
- `display` – bool or matplotlib axes: show the full session sync pulses display

defaults to False :return: trials dictionary

`ibllib.io.extractors.ephys_fpga.extract_camera_sync(sync, output_path=None, save=False, chmap=None)`

Extract camera timestamps from the sync matrix

**Parameters**

- `sync` – dictionary 'times', 'polarities' of fronts detected on sync trace
- `output_path` – where to save the data
- `save` – True/False
- `chmap` – dictionary containing channel indices. Default to constant.

**Returns** dictionary containing camera timestamps

`ibllib.io.extractors.ephys_fpga.extract_sync` (*session\_path*, *save=False*, *force=False*,  
*ephys\_files=None*)

Reads ephys binary file (s) and extract sync within the binary file folder Assumes ephys data is within a *raw\_ephys\_data* folder

### Parameters

- **session\_path** – '/path/to/subject/yyyy-mm-dd/001'
- **save** – Bool, defaults to False
- **force** – Bool on re-extraction, forces overwrite instead of loading existing sync files

**Returns** list of sync dictionaries

`ibllib.io.extractors.ephys_fpga.extract_wheel_sync` (*sync*, *output\_path=None*,  
*save=False*, *chmap=None*)

Extract wheel positions and times from sync fronts dictionary for all 16 chans

### Parameters

- **sync** – dictionary 'times', 'polarities' of fronts detected on sync trace
- **output\_path** – where to save the data
- **save** – True/False
- **chmap** – dictionary containing channel indices. Default to constant. `chmap = {'rotary_encoder_0': 13, 'rotary_encoder_1': 14}`

**Returns** dictionary containing wheel data, 'wheel\_ts', 're\_ts'

`ibllib.io.extractors.ephys_fpga.get_ibl_sync_map` (*ef*, *version*)

Gets default channel map for the version/binary file type combination :param ef: `ibllib.io.spikeglx.glob_ephys_file` dictionary with field 'ap' or 'nidq' :return: channel map dictionary

## ibllib.io.extractors.ephys\_trials

`ibllib.io.extractors.ephys_trials.extract_all` (*session\_path*, *save=False*, *data=False*,  
*output\_path=None*)

Extract all behaviour data from Bpod within the specified folder. The timing information from FPGA is extracted in `ephys_fpga()`

### Parameters

- **session\_path** (*str* or *pathlib.Path*) – folder containing sessions
- **save** – bool
- **data** – raw Bpod data dictionary

**Returns** dictionary of trial related vectors (one row per trial)

## ibllib.io.extractors.training\_audio

`ibllib.io.extractors.training_audio.welchogram` (*fs*, *wav*, *nswin=262144*, *overlap=131072.0*, *nperseg=512*)

Computes a spectrogram on a very large audio file.

### Parameters

- **fs** – sampling frequency (Hz)
- **wav** – wav signal (vector or memmap)

- **nswin** – n samples of the sliding window
- **overlap** – n samples of the overlap between windows
- **nperseg** – n samples for the computation of the spectrogram

**Returns** tscale, fscale, downsampled\_spectrogram

### ibllib.io.extractors.training\_trials

**ALF extractors** are a collection of functions that extract alf files from the PyBpod rig raw data.

Each DatasetType in the IBL pipeline should have one extractor function.

`ibllib.io.extractors.training_trials.check_alf_folder` (*session\_path*)

Check if alf folder exists, creates it if it doesn't.

**Parameters** `session_path` (*str*) – absolute path of session folder

`ibllib.io.extractors.training_trials.get_camera_timestamps` (*session\_path*,  
*data=False*,  
*save=False*, *set-*  
*tings=False*)

Get the camera timestamps from the Bpod

The camera events are logged only during the events not in between, so the times need to be interpolated

#### Parameters

- **session\_path** – Absolute path of session folder
- **save** – bool, optional

**Returns** `numpy.ndarray`

`ibllib.io.extractors.training_trials.get_choice` (*session\_path*, *save=False*, *data=False*,  
*settings=False*)

Get the subject's choice in every trial. **Optional:** saves `_ibl_trials.choice.npy` to alf folder.

Uses `signed_contrast` and `trial_correct`. -1 is a CCW turn (towards the left) +1 is a CW turn (towards the right) 0 is a no\_go trial If a trial is correct the choice of the animal was the inverse of the sign of the position.

```
>>> choice[t] = -np.sign(position[t]) if trial_correct[t]
```

#### Parameters

- **session\_path** (*str*) – absolute path of session folder
- **save** (*bool*, *optional*) – whether to save the corresponding alf file to the alf folder, defaults to False

**Returns** `numpy.ndarray`

**Return type** `dtype('int64')`

`ibllib.io.extractors.training_trials.get_contrastLR` (*session\_path*, *save=False*,  
*data=False*, *settings=False*)

Get left and right contrasts from raw datafile. Optionally, saves `_ibl_trials.contrastLeft.npy` and `_ibl_trials.contrastRight.npy` to alf folder.

Uses `signed_contrast` to create left and right contrast vectors.

#### Parameters

- **session\_path** (*str*) – absolute path of session folder
- **save** (*bool, optional*) – whether to save the corresponding alf file to the alf folder, defaults to False

**Returns** numpy.ndarray

**Return type** dtype('float64')

`ibllib.io.extractors.training_trials.get_feedbackType` (*session\_path, save=False, data=False, settings=False*)

Get the feedback that was delivered to subject. **Optional:** saves `_ibl_trials.feedbackType.npy`

Checks in raw datafile for error and reward state. Will raise an error if more than one of the mutually exclusive states have been triggered.

Sets feedbackType to -1 if error state was triggered (applies to no-go) Sets feedbackType to +1 if reward state was triggered

**Parameters**

- **session\_path** (*str*) – absolute path of session folder
- **save** (*bool, optional*) – whether to save the corresponding alf file to the alf folder, defaults to False

**Returns** numpy.ndarray

**Return type** dtype('int64')

`ibllib.io.extractors.training_trials.get_feedback_times` (*session\_path, save=False, data=False, settings=False*)

Get the times the water or error tone was delivered to the animal. **Optional:** saves `_ibl_trials.feedback_times.npy`

Gets reward and error state init times vectors, checks if the intersection of nans is empty, then merges the 2 vectors.

**Parameters**

- **session\_path** (*str*) – Absolute path of session folder
- **save** – whether to save the corresponding alf file to the alf folder, defaults to False
- **save** – bool, optional

**Returns** numpy.ndarray

**Return type** dtype('float64')

`ibllib.io.extractors.training_trials.get_goCueOnset_times` (*session\_path, save=False, data=False, settings=False*)

Get trigger times of goCue from state machine.

Current software solution for triggering sounds uses PyBpod soft codes. Delays can be in the order of 10's of ms. This is the time when the command to play the sound was executed. To measure accurate time, either getting the sound onset from the future microphone OR the new xonar soundcard and setup developed by Sanworks guarantees a set latency (in testing).

**Parameters**

- **session\_path** (*str*) – Absolute path of session folder



- **save** – bool, optional

**Returns** numpy.ndarray

**Return type** dtype('float64')

```
ibllib.io.extractors.training_trials.get_goCueTrigger_times(session_path,
                                                           save=False,
                                                           data=False,      set-
                                                           tings=False)
```

Get trigger times of goCue from state machine.

Current software solution for triggering sounds uses PyBpod soft codes. Delays can be in the order of 10's of ms. This is the time when the command to play the sound was executed. To measure accurate time, either getting the sound onset from xonar soundcard sync pulse (latencies may vary).

#### Parameters

- **session\_path** (*str*) – Absolute path of session folder
- **save** – whether to save the corresponding alf file to the alf folder, defaults to False
- **save** – bool, optional

**Returns** numpy.ndarray

**Return type** dtype('float64')

```
ibllib.io.extractors.training_trials.get_intervals(session_path,      save=False,
                                                  data=False, settings=False)
```

Trial start to trial end. Trial end includes 1 or 2 seconds after feedback, (depending on the feedback) and 0.5 seconds of iti. **Optional:** saves `_ibl_trials.intervals.npy`

Uses the corrected Trial start and Trial end timestamp values from PyBpod.

#### Parameters

- **session\_path** (*str*) – Absolute path of session folder
- **save** – whether to save the corresponding alf file to the alf folder, defaults to False
- **save** – bool, optional

**Returns** 2D numpy.ndarray (col0 = start, col1 = end)

**Return type** dtype('float64')

```
ibllib.io.extractors.training_trials.get_iti_duration(session_path,  save=False,
                                                       data=False, settings=False)
```

Calculate duration of iti from state timestamps. **Optional:** saves `_ibl_trials.iti_duration.npy`

Uses Trial end timestamp and `get_response_times` to calculate iti.

#### Parameters

- **session\_path** (*str*) – Absolute path of session folder
- **save** – whether to save the corresponding alf file to the alf folder, defaults to False
- **save** – bool, optional

**Returns** numpy.ndarray

**Return type** dtype('float64')

```
ibllib.io.extractors.training_trials.get_repNum(session_path, save=False, data=False,
                                                settings=False)
```

Count the consecutive repeated trials. **Optional:** saves `_ibl_trials.repNum.npy` to alf folder.

Creates `trial_repeated` from `trial['contrast']['type'] == 'RepeatContrast'`

```
>>> trial_repeated = [0, 1, 1, 0, 1, 0, 1, 1, 1, 0]
>>> repNum =         [0, 1, 2, 0, 1, 0, 1, 2, 3, 0]
```

#### Parameters

- **session\_path** (*str*) – absolute path of session folder
- **save** (*bool, optional*) – whether to save the corresponding alf file to the alf folder, defaults to False

**Returns** `numpy.ndarray`

**Return type** `dtype('int64')`

`ibllib.io.extractors.training_trials.get_response_times` (*session\_path*, *save=False*,  
*data=False*, *settings=False*)

Time (in absolute seconds from session start) when a response was recorded. **Optional:** saves `_ibl_trials.response_times.npy`

Uses the timestamp of the end of the `closed_loop` state.

#### Parameters

- **session\_path** (*str*) – Absolute path of session folder
- **save** – whether to save the corresponding alf file to the alf folder, defaults to False
- **save** – bool, optional

**Returns** `numpy.ndarray`

**Return type** `dtype('float64')`

`ibllib.io.extractors.training_trials.get_rewardVolume` (*session\_path*, *save=False*,  
*data=False*, *settings=False*)

Load reward volume delivered for each trial. **Optional:** saves `_ibl_trials.rewardVolume.npy`

Uses `reward_current` to accumulate the amount of

#### Parameters

- **session\_path** (*str*) – Absolute path of session folder
- **save** – whether to save the corresponding alf file to the alf folder, defaults to False
- **save** – bool, optional

**Returns** `numpy.ndarray`

**Return type** `dtype('int64')`

`ibllib.io.extractors.training_trials.get_stimOn_times` (*session\_path*, *save=False*,  
*data=False*, *settings=False*)

Find the time of the statemachine command to turn on the stim (state `stim_on` start or `rotary_encoder_event2`)  
Find the next frame change from the photodiode after that TS. Screen is not displaying anything until then.  
(Frame changes are in `BNC1High` and `BNC1Low`)

`ibllib.io.extractors.training_trials.get_stimOn_times_ge5` (*session\_path*,  
*data=False*)

Find first and last `stim_sync` pulse of the trial. `stimOn_times` should be the first after the `stim_on` state. (Stim updates are in `BNC1High` and `BNC1Low` - frame2TTL device) Check that all trials have frame changes. Find

length of stim\_on\_state [start, stop]. If either check fails the HW device failed to detect the stim\_sync square change Substitute that trial's missing or incorrect value with a NaN. return stimOn\_times

```
ibllib.io.extractors.training_trials.get_stimOn_times_lt5(session_path,
                                                         data=False)
```

Find the time of the statemachine command to turn on the stim (state stim\_on start or rotary\_encoder\_event2)  
Find the next frame change from the photodiode after that TS. Screen is not displaying anything until then.  
(Frame changes are in BNC1High and BNC1Low)

### ibllib.io.extractors.training\_wheel

Training wheel extractor from Pybpod output.

```
ibllib.io.extractors.training_wheel.check_alf_folder(session_path)
```

Check if alf folder exists, creates it if it doesn't.

**Parameters** `session_path` (*str*) – absolute path of session folder

```
ibllib.io.extractors.training_wheel.get_velocity(session_path, save=False,
                                                  data_wheel=None)
```

Compute velocity from non-uniformly acquired positions and timestamps. **Optional:** save  
\_ibl\_trials.velocity.npy

Uses signed\_contrast to create left and right contrast vectors.

#### Parameters

- **session\_path** (*str*) – absolute path of session folder
- **save** (*bool, optional*) – whether to save the corresponding alf file to the alf folder, defaults to False

**Returns** numpy.ndarray

**Return type** dtype('float64')

```
ibllib.io.extractors.training_wheel.get_wheel_data(session_path, bp_data=None,
                                                    save=False, display=False)
```

**Get wheel data from raw files and converts positions into radians mathematical convention** (anti-clockwise = +) and timestamps into seconds relative to Bpod clock.

**Optional:** saves \_ibl\_wheel.times.npy and \_ibl\_wheel.position.npy

**Times:** Gets Rotary Encoder timestamps (us) for each position and converts to times. Synchronize with Bpod and outputs

**Positions:** Radians mathematical convention

#### Parameters

- **session\_path** (*str*) – absolute path of session folder
- **data** (*dict, optional*) – dictionary containing the contents pybpod jsonable file read with raw.load\_data
- **save** (*bool, optional*) – whether to save the corresponding alf file to the alf folder, defaults to False

**Returns** Numpy structured array.

**Return type** numpy.ndarray

### ibllib.io.flags

`ibllib.io.flags.excise_flag_file` (*fname*, *removed\_files=None*)

Remove one or several specific files if they figure within the file If no file is left, deletes the flag.

**Parameters** `fname` (*str* or *pathlib.Path*) – full file path of the flag file

**Returns** None

`ibllib.io.flags.read_flag_file` (*fname*)

Flag files are \*.flag files within a session folder used to schedule some jobs If they are empty, should return True

**Parameters** `fname` (*str* or *pathlib.Path*) – full file path of the flag file

**Returns** None

`ibllib.io.flags.write_flag_file` (*fname*, *file\_list: list = None*, *clobber=False*)

Flag files are \*.flag files within a session folder used to schedule some jobs Each line references to a file to extract or register

**Parameters**

- `fname` (*str* or *pathlib.Path*) – full file path of the flag file
- `file_list` (*list*) – None or list of relative paths to write in the file
- `clobber` (*bool*, *optional*) – (False) overwrites the flag file if any

**Returns** None

### ibllib.io.params

`ibllib.io.params.getfile` (*str\_params*)

**Returns full path of the param file per system convention:** linux/mac: ~/str\_params, Windows: APPDATA folder

**Parameters** `str_params` – string that identifies parm file

**Returns** string of full path

`ibllib.io.params.read` (*str\_params*, *default=None*)

Reads in and parse Json parameter file into dictionary

**Parameters**

- `str_params` – path to text json file
- `default` – default values for missing parameters

**Returns** named tuple containing parameters

`ibllib.io.params.write` (*str\_params*, *par*)

Write a parameter file in Json format

**Parameters**

- `str_params` – path to text json file
- `par` – dictionary containing parameters values

**Returns** None

## ibllib.io.raw\_data\_loaders

Raw Data Loader functions for PyBpod rig

Module contains one loader function per raw datafile

`ibllib.io.raw_data_loaders.get_port_events` (*events: dict, name: str = ''*) → list  
Return all timestamps from bpod raw data trial['behavior\_data']['Events timestamps'] that match name

`ibllib.io.raw_data_loaders.load_ambient_sensor` (*session\_path*)  
Load Ambient Sensor data from session.

Probably could be extracted to DatasetTypes: `_ibl_trials.temperature_C`, `_ibl_trials.airPressure_mb`, `_ibl_trials.relativeHumidity` Returns a list of dicts one dict per trial. dict keys are: `dict_keys(['Temperature_C', 'AirPressure_mb', 'RelativeHumidity'])`

**Parameters** `session_path` (*str*) – Absolute path of session folder

**Returns** list of dicts

**Return type** list

`ibllib.io.raw_data_loaders.load_bpod` (*session\_path*)  
Load both settings and data from bpod (.json and .jsonable)

**Parameters** `session_path` – Absolute path of session folder

**Returns** dict settings and list of dicts data

`ibllib.io.raw_data_loaders.load_data` (*session\_path, time='absolute'*)  
Load PyBpod data files (.jsonable).

Bpod timestamps are in microseconds ( $\mu$ s) PyBpod timestamps are in seconds (s)

**Parameters** `session_path` (*str*) – Absolute path of session folder

**Returns** A list of len ntrials each trial being a dictionary

**Return type** list of dicts

`ibllib.io.raw_data_loaders.load_encoder_events` (*session\_path, settings=False*)  
Load Rotary Encoder (RE) events raw data file.

Assumes that a folder called “raw\_behavior\_data” exists in folder.

Events number correspond to following bpod states: 1: correct / hide\_stim 2: stim\_on 3: closed\_loop 4: freeze\_error / freeze\_correct

```
>>> data.columns
>>> ['re_ts',      # Rotary Encoder Timestamp (ms)  'numpy.int64'
    'sm_ev',      # State Machine Event           'numpy.int64'
    'bns_ts']    # Bonsai Timestamp (int)         'pandas.Timestamp'
    # pd.to_datetime(data.bns_ts) to work in datetimes
```

**Parameters** `session_path` (*[type]*) – [description]

**Returns** dataframe w/ 3 cols and (ntrials \* 3) lines

**Return type** Pandas.DataFrame

`ibllib.io.raw_data_loaders.load_encoder_positions` (*session\_path, settings=False*)  
Load Rotary Encoder (RE) positions from raw data file within a session path.

Assumes that a folder called “raw\_behavior\_data” exists in folder. Positions are RE ticks [-512, 512] == [-180°, 180°] 0 == trial stim init position Positive nums are rightwards movements (mouse) or RE CW (mouse)

Variable line number, depends on movements.

**Raw datafile Columns:** Position, RE timestamp, RE Position, Bonsai Timestamp

Position is always equal to ‘Position’ so this column was dropped.

```
>>> data.columns
>>> ['re_ts',      # Rotary Encoder Timestamp (ms)      'numpy.int64'
     're_pos',    # Rotary Encoder position (ticks)    'numpy.int64'
     'bns_ts']   # Bonsai Timestamp                    'pandas.Timestamp'
     # pd.to_datetime(data.bns_ts) to work in datetimes
```

**Parameters** `session_path` (*str*) – Absolute path of session folder

**Returns** dataframe w/ 3 cols and N positions

**Return type** Pandas.DataFrame

`ibllib.io.raw_data_loaders.load_encoder_trial_info` (*session\_path*)

Load Rotary Encoder trial info from raw data file.

Assumes that a folder called “raw\_behavior\_data” exists in folder.

NOTE: Last trial probably inexistent data (Trial info is sent on trial start and data is only saved on trial exit...) max(trialnum) should be N+1 if N is the amount of trial data saved.

Raw datafile Columns:

```
>>> data.columns
>>> ['trial_num',      # Trial Number                    'numpy.int64'
     'stim_pos_init', # Initial position of visual stim 'numpy.int64'
     'stim_contrast', # Contrast of visual stimulus          'numpy.float64'
     'stim_freq',    # Frequency of gabor patch              'numpy.float64'
     'stim_angle',   # Angle of Gabor 0 = Vertical            'numpy.float64'
     'stim_gain',    # Wheel gain (mm/° of stim)             'numpy.float64'
     'stim_sigma',   # Size of patch                          'numpy.float64'
     'stim_phase',   # Phase of gabor                          'numpy.float64'
     'bns_ts' ]     # Bonsai Timestamp                    'pandas.Timestamp'
     # pd.to_datetime(data.bns_ts) to work in datetimes
```

**Parameters** `session_path` (*str*) – Absolute path of session folder

**Returns** dataframe w/ 9 cols and ntrials lines

**Return type** Pandas.DataFrame

`ibllib.io.raw_data_loaders.load_mic` (*session\_path*)

Load Microphone wav file to np.array of len nSamples

**Parameters** `session_path` (*str*) – Absolute path of session folder

**Returns** An array of values of the sound waveform

**Return type** numpy.array

`ibllib.io.raw_data_loaders.load_settings` (*session\_path*)

Load PyBpod Settings files (.json).

[description]

**Parameters** `session_path` (*str*) – Absolute path of session folder

**Returns** Settings dictionary

**Return type** dict

`ibllib.io.raw_data_loaders.sync_trials_robust` (*t0*, *t1*, *diff\_threshold=0.001*,  
*drift\_threshold\_ppm=200*, *max\_shift=5*,  
*return\_index=False*)

Attempts to find matching timestamps in 2 time-series that have an offset, are drifting, and are most likely incomplete: sizes don't have to match, some pulses may be missing in any serie. Only works with irregular time series as it relies on the derivative to match sync. :param t0: :param t1: :param diff\_threshold: :param drift\_threshold\_ppm: (150) :param max\_shift: (200) :param return\_index (False) :return:

`ibllib.io.raw_data_loaders.trial_times_to_times` (*raw\_trial*)

Parse and convert all trial timestamps to “absolute” time. Float64 seconds from session start.

0—BpodStart—TrialStart0—TrialEnd0—TrialStart1—TrialEnd1...0—ts0—ts1— tsN...absTS = tsN + TrialStartN - BpodStart

Bpod timestamps are in microseconds ( $\mu$ s) PyBpod timestamps are in seconds (s)

**Parameters** `raw_trial` (*dict*) – raw trial data

**Returns** trial data with modified timestamps

**Return type** dict

## ibllib.io.spikeglx

**class** `ibllib.io.spikeglx.Reader` (*sglx\_file*)

Class for SpikeGLX reading purposes Some format description was found looking at the Matlab SDK here <https://github.com/billkarsh/SpikeGLX/blob/master/MATLAB-SDK/DemoReadSGLXData.m>

**compress\_file** (*keep\_original=True*, *\*\*kwargs*)

Compresses :param keep\_original: defaults True. If False, the original uncompressed file is deleted and the current spikeglx.Reader object is modified in place

**Parameters** `kwargs` –

**Returns** `pathlib.Path` of the compressed \*.cbin file

**decompress\_file** (*keep\_original=True*, *\*\*kwargs*)

Decompresses a mtscomp file :param keep\_original: defaults True. If False, the original compressed file is deleted

and the current spikeglx.Reader object is modified in place

**Returns** `pathlib.Path` of the decompressed \*.bin file

**fs**

**Returns** sampling frequency (Hz)

**nc**

**Returns** number of channels

**ns**

**Returns** number of samples

**read** (*nset=slice(0, 10000, None)*, *cset=slice(None, None, None)*, *sync=True*)

Read from slices or indexes :param slice\_n: slice or sample indices :param slice\_c: slice or channel indices  
:return: float32 array

**read\_samples** (*first\_sample=0*, *last\_sample=10000*, *channels=None*)

reads all channels from first\_sample to last\_sample, following numpy slicing convention  
sglx.read\_samples(first=0, last=100) would be equivalent to slicing the array D D[:,0:100] where  
the last axis represent time and the first channels.

**param first\_sample** first sample to be read, python slice-wise

**param last\_sample** last sample to be read, python slice-wise

**param channels** slice or numpy array of indices

**return** numpy array of int16

**read\_sync** (*\_slice=slice(0, 10000, None)*, *threshold=1.2*)

Reads all sync trace. Convert analog to digital with selected threshold and append to array :param \_slice:  
samples slice :param threshold: (V) threshold for front detection, defaults to 1.2 V :return: int8 array

**read\_sync\_analog** (*\_slice=slice(0, 10000, None)*)

Reads only the analog sync traces at specified samples using slicing syntax >>> sync\_samples =  
sr.read\_sync\_analog(slice(0,10000))

**read\_sync\_digital** (*\_slice=slice(0, 10000, None)*)

Reads only the digital sync trace at specified samples using slicing syntax >>> sync\_samples =  
sr.read\_sync\_digital(slice(0,10000))

## type

**Returns** ap, lf or nidq. Useful to index dictionaries

**verify\_hash** ()

Computes SHA-1 hash and returns True if it matches metadata, False otherwise :return: boolean

ibllib.io.spikeglx.**get\_hardware\_config** (*config\_file*)

Reads the neuropixel\_wirings.json file containing sync mapping and parameters :param config\_file: folder or  
json file :return: dictionary or None

ibllib.io.spikeglx.**glob\_ephys\_files** (*session\_path*, *suffix='.meta'*, *recursive=True*,  
*bin\_exists=True*)

From an arbitrary folder (usually session folder) gets the ap and lf files and labels Associated to the subfolders  
where they are the expected folder tree is: |— 3A | |— imec0 | | |— sync\_testing\_g0\_t0.imec0.ap.bin  
| | |— sync\_testing\_g0\_t0.imec0.lf.bin | |— imec1 | |— sync\_testing\_g0\_t0.imec1.ap.bin | |—  
sync\_testing\_g0\_t0.imec1.lf.bin |— 3B

|— sync\_testing\_g0\_t0.nidq.bin |— imec0 | |— sync\_testing\_g0\_t0.imec0.ap.bin | |—  
sync\_testing\_g0\_t0.imec0.lf.bin |— imec1

|— sync\_testing\_g0\_t0.imec1.ap.bin |— sync\_testing\_g0\_t0.imec1.lf.bin

## Parameters

- **bin\_exists** –
- **suffix** –
- **recursive** –
- **session\_path** – folder, string or pathlib.Path
- **glob\_pattern** – pattern to look recursively for (defaults to `*.ap.*bin`)

**Returns** a list of dictionaries with keys ‘ap’: apfile, ‘lf’: lf file and ‘label’



`ibllib.io.spikeglx.read(sglx_file, first_sample=0, last_sample=10000)`

Function to read from a spikeglx binary file without instantiating the class. Gets the meta-data as well.

```
>>> ibllib.io.spikeglx.read('/path/to/file.bin', first_sample=0, last_sample=1000)
```

#### Parameters

- **sglx\_file** – full path the the binary file to read
- **first\_sample** – first sample to be read, python slice-wise
- **last\_sample** – last sample to be read, python slice-wise

**Returns** Data array, sync trace, meta-data

`ibllib.io.spikeglx.read_meta_data(md_file)`

Reads the spkike glx metadata file and parse in a dictionary Agnostic: does not make any assumption on the keys/content, it just parses key=values

**Parameters** **md\_file** – last sample to be read, python slice-wise

**Returns** Data array, sync trace, meta-data

`ibllib.io.spikeglx.split_sync(sync_tr)`

The synchronization channelx are stored as single bits, this will split the int16 original channel into 16 single bits channels

**Parameters** **sync\_tr** – numpy vector: samples of synchronisation trace

**Returns** int8 numpy array of 16 channels, 1 column per sync trace

### ibllib.misc.misc

`ibllib.misc.misc.print_progress(iteration, total, prefix="", suffix="", decimals=1, length=100, fill='')`

Call in a loop to create terminal progress bar

#### Parameters

- **iteration** – Required : current iteration (Int)
- **total** – Required : total iterations (Int)
- **prefix** – Optional : prefix string (Str)
- **suffix** – Optional: suffix string (Str)
- **decimals** – Optional: positive number of decimals in percent complete (Int)
- **length** – Optional: character length of bar (Int)
- **fill** – Optional: bar fill character (Str)

**Returns** None

### ibllib.misc.version

`ibllib.misc.version.eq(v1, v2)`

check if v1 == v2

#### Parameters

- **v1** (*str*) – version string, in the form “1.23.3”
- **v2** (*str*) – version string, in the form “1.23.3”

**Returns** bool

`ibllib.misc.version.ge(v1, v2)`  
check if  $v1 \geq v2$

**Parameters**

- **v1** (*str*) – version string, in the form “1.23.3”
- **v2** (*str*) – version string, in the form “1.23.3”

**Returns** bool

`ibllib.misc.version.gt(v1, v2)`  
check if  $v1 > v2$

**Parameters**

- **v1** (*str*) – version string, in the form “1.23.3”
- **v2** (*str*) – version string, in the form “1.23.3”

**Returns** bool

`ibllib.misc.version.le(v1, v2)`  
check if  $v1 \leq v2$

**Parameters**

- **v1** (*str*) – version string, in the form “1.23.3”
- **v2** (*str*) – version string, in the form “1.23.3”

**Returns** bool

`ibllib.misc.version.lt(v1, v2)`  
check if  $v1 < v2$

**Parameters**

- **v1** (*str*) – version string, in the form “1.23.3”
- **v2** (*str*) – version string, in the form “1.23.3”

**Returns** bool

## **ibllib.pipes.experimental\_data**

Entry point to system commands for IBL behaviour pipeline. Each function below corresponds to a command-line tool.

`ibllib.pipes.experimental_data.compress_ephys` (*root\_data\_folder*, *dry=False*,  
*max\_sessions=5*)

Compress ephys files looking for *compress\_ephys.flag* within the probes folder Original bin file will be removed The registration flag created contains targeted file names at the root of the session

`ibllib.pipes.experimental_data.extract` (*root\_data\_folder*, *dry=False*)  
Extracts behaviour only

`ibllib.pipes.experimental_data.extract_ephys` (*root\_data\_folder*, *dry=False*,  
*max\_sessions=10*)

Extracts ephys session only

`ibllib.pipes.experimental_data.raw_ephys_qc` (*root\_data\_folder*, *dry=False*,  
*max\_sessions=10*, *force=False*)

Computes raw electrophysiology QC

`ibllib.pipes.experimental_data.sync_merge_ephys` (*root\_data\_folder*, *dry=False*)

Post spike-sorting processing: - synchronization of probes - ks2 to ALF conversion for each probes in alf/probeXX folder - computes spike sorting QC - creates probes object in alf folder To start the job for a session, all electrophysiology ap files from session need to be associated with a *sync\_merge\_ephys.flag* file  
Outputs individual probes

### ibllib.pipes.extract\_session

Find task name Check if extractors for specific task exist Extract data OR return error to user saying that the task has no extractors

`ibllib.pipes.extract_session.get_session_extractor_type` (*session\_path*)

From a session path, loads the settings file, finds the task and checks if extractors exist task names examples:  
:param session\_path: :return: bool

`ibllib.pipes.extract_session.get_task_extractor_type` (*task\_name*)

Splits the task name according to naming convention: - ignores everything  
\_iblrig\_tasks\_biasedChoiceWorld3.7.0 returns “biased” \_iblrig\_tasks\_trainingChoiceWorld3.6.0 returns “training”  
:param task\_name: :return:

### ibllib.pipes.purge\_rig\_data

Purge data from RIG - Find all files by rglob - Find all sessions of the found files - Check Alyx if corresponding datasetTypes have been registered as existing sessions and files on Flatiron - Delete local raw file if found on Flatiron

### ibllib.pipes.transfer\_rig\_data

### ibllib.graphic

`ibllib.graphic.login` (*title='Enter Credentials'*, *default\_username=None*, *default\_passwd=None*,  
*add\_fields=None*)

Dialog box prompting for username and password.

#### Parameters

- **title** – Window title
- **default\_username** – default field for username
- **default\_passwd** – default field for password
- **add\_fields** – list of new fields to be added to the dialog

#### Returns

`ibllib.graphic.strinput` (*title*, *prompt*, *default='COM'*, *nullable=False*)

Example: >>> strinput(“RIG CONFIG”, “Insert RE com port:”, default=“COM”)

### ibllib.plots

`ibllib.plots.squares` (*tscale*, *polarity*, *ax=None*, *yrange=[-1, 1]*, *\*\*kwargs*)

Matplotlib display of rising and falling fronts in a square-wave pattern

**Parameters**

- **tscale** – time of indices of fronts
- **polarity** – polarity of front (1: rising, -1:falling)
- **ax** – matplotlib axes object

**Returns** None

`ibllib.plots.traces(w, **kwargs)`

Matplotlib display of traces

**Parameters**

- **w** – 2D array (numpy array dimension nsamples, ntraces)
- **fs** – sampling frequency
- **gain** – display gain
- **ax** – matplotlib axes object

**Returns** None

`ibllib.plots.vertical_lines(x, ymin=0, ymax=1, ax=None, **kwargs)`

From a x vector, draw separate vertical lines at each x location ranging from ymin to ymax

**Parameters**

- **x** – numpy array vector of x values where to display lines
- **ymin** – lower end of the lines (scalar)
- **ymax** – higher end of the lines (scalar)
- **ax** – (optional) matplotlib axis instance

**Returns** None

`ibllib.plots.wiggle(w, fs=1, gain=0.71, color='k', ax=None, fill=True, linewidth=0.5, t0=0, **kwargs)`

Matplotlib display of wiggle traces

**Parameters**

- **w** – 2D array (numpy array dimension nsamples, ntraces)
- **fs** – sampling frequency
- **gain** – display gain
- **color** – ('k') color of traces
- **ax** – (None) matplotlib axes object
- **fill** – (True) fill variable area above 0
- **t0** –  
(0) timestamp of the first sample

**Returns** None

### 5.1.3 oneibl module

---

`oneibl.one`

---

`oneibl.registration`

---

`oneibl.webclient`

---

## oneibl.one

**class** `oneibl.one.ONE` (*username=None, password=None, base\_url=None, silent=False*)

**static keywords** ()

Returns possible keywords to be used in the `one.list` method

**Returns** a tuple containing possible search terms:

**Return type** tuple

**list** (*eid=None, keyword='dataset-type', details=False*)

From a Session ID, queries Alyx database for datasets-types related to a session.

**Parameters**

- **eid** (*str or list of strings*) – Experiment ID, for IBL this is the UUID String of the Session as per Alyx database. Example: '698361f6-b7d0-447d-a25d-42afdef7a0da' If None, returns the set of possible values. Only for the following keys: ('users', 'dataset-types', subjects')
- **keyword** (*str*) – The attribute to be listed.
- **details** (*bool*) – returns a second argument with a full dictionary to provide context

**Returns** list of strings, plus list of dictionaries if details option selected

**Return type** list, list

for a list of keywords, use the methods `one.search_terms()`

**load** (*eid, dataset\_types=None, dclass\_output=False, dry\_run=False, cache\_dir=None, download\_only=False, clobber=False, offline=False, keep\_uuid=False*)

From a Session ID and dataset types, queries Alyx database, downloads the data from Globus, and loads into numpy array.

**Parameters**

- **eid** (*str*) – Experiment ID, for IBL this is the UUID of the Session as per Alyx database. Could be a full Alyx URL: '<http://localhost:8000/sessions/698361f6-b7d0-447d-a25d-42afdef7a0da>' or only the UUID: '698361f6-b7d0-447d-a25d-42afdef7a0da'. Can also be a list of the above for multiple eids.
- **dataset\_types** (*list*) – [None]: Alyx dataset types to be returned.
- **dclass\_output** (*bool* If None or an empty dataset\_type is specified, the output will be a dictionary by default.) – [False]: forces the output as dataclass to provide context.
- **cache\_dir** (*str*) – temporarily overrides the cache\_dir from the parameter file
- **download\_only** (*bool*) – do not attempt to load data in memory, just download the files
- **clobber** (*bool*) – force downloading even if files exists locally
- **keep\_uuid** (*bool*) – keeps the UUID at the end of the filename (defaults to False)

**Returns** List of numpy arrays matching the size of `dataset_types` parameter, OR a dataclass containing arrays and context data.

**Return type** list, dict, dataclass `SessionDataInfo`

**load\_dataset** (*eid*, *dataset\_type*, *\*\*kwargs*)

Load a single dataset from a Session ID and a dataset type.

**Parameters**

- **eid** (*str*) – Experiment ID, for IBL this is the UUID of the Session as per Alyx database. Could be a full Alyx URL: `'http://localhost:8000/sessions/698361f6-b7d0-447d-a25d-42afdef7a0da'` or only the UUID: `'698361f6-b7d0-447d-a25d-42afdef7a0da'`.
- **dataset\_type** – Alyx dataset type to be returned.

**Returns** A numpy array.

**Return type** numpy array

**load\_object** (*eid*, *obj*, *\*\*kwargs*)

Load all attributes of an ALF object from a Session ID and an object name.

**Parameters**

- **eid** (*str*) – Experiment ID, for IBL this is the UUID of the Session as per Alyx database. Could be a full Alyx URL: `'http://localhost:8000/sessions/698361f6-b7d0-447d-a25d-42afdef7a0da'` or only the UUID: `'698361f6-b7d0-447d-a25d-42afdef7a0da'`.
- **obj** (*str*) – Alyx object to load.

**Returns** A dictionary-like structure with one key per dataset type for the requested object, and a NumPy array per value.

**Return type** `AlfBunch` instance

**search** (*details=False*, *limit=None*, *\*\*kwargs*)

**Applies a filter to the sessions (eid) table and returns a list of json dictionaries** corresponding to sessions.

For a list of search terms, use the methods

```
>>> one.search_terms()
```

**Parameters**

- **details** (*bool*) – default `False`, returns also the session details as per the REST response
- **limit** (*int List of possible search terms*) – default `None`, limits results (if pagination enabled on server)
- **dataset\_types** (*list of str*) – list of `dataset_types`
- **users** (*list or str*) – a list of users
- **subjects** (*list or str*) – a list of subjects nickname
- **lab** (*list or str*) – a str or list of lab names
- **date\_range** (*list*) – list of 2 strings or list of 2 dates that define the range
- **number** (*str or int*) – session number

**Returns** list of eids, if details is True, also returns a list of json dictionaries, each entry corresponding to a matching session

**Return type** list, list

**static search\_terms** ()

Returns possible search terms to be used in the one.search method.

**Returns** a tuple containing possible search terms:

**Return type** tuple

**static setup** ()

Interactive command tool that populates parameter file for ONE IBL.

## oneibl.registration

**class** oneibl.registration.**RegistrationClient** (*one=None*)

Object that keeps the ONE instance and provides method to create sessions and register data.

**create\_sessions** (*root\_data\_folder*, *dry=False*)

Create sessions looking recursively for flag files

### Parameters

- **root\_data\_folder** – folder to look for create\_me.flag
- **dry** – bool. Dry run if True

**Returns** None

**register\_sync** (*root\_data\_folder*, *dry=False*)

Register sessions looking recursively for flag files

### Parameters

- **root\_data\_folder** – folder to look for register\_me.flag
- **dry** – bool. Dry run if True

**Returns**

## oneibl.webclient

**class** oneibl.webclient.**AlyxClient** (*\*\*kwargs*)

Class that implements simple GET/POST wrappers for the Alyx REST API <http://alyx.readthedocs.io/en/latest/api.html>

**authenticate** (*username=""*, *password=""*, *base\_url=""*)

Gets a security token from the Alyx REST API to create requests headers. Credentials are in the params\_secret\_template.py file

### Parameters

- **username** (*str*) – Alyx database user
- **password** (*str*) – Alyx database password
- **base\_url** (*str*) – Alyx server address, including port and protocol

**delete** (*rest\_query*)

Sends a DELETE request to the Alyx server. Will raise an exception on any status\_code other than 200, 201.

**Parameters** `rest_query` (*str*) – examples: `‘/weighings/c617562d-c107-432e-a8ee-682c17f9e698’` `‘https://test.alyx.internationalbrainlab.org/weighings/c617562d-c107-432e-a8ee-682c17f9e698’`.

**Returns** (dict/list) json interpreted dictionary from response

**get** (*rest\_query*)

Sends a GET request to the Alyx server. Will raise an exception on any `status_code` other than 200, 201. For the dictionary contents and list of endpoints, refer to: <https://alyx.internationalbrainlab.org/docs>

**Parameters** `rest_query` (*str*) – example: `‘/sessions?user=Hamish’`.

**Returns** (dict/list) json interpreted dictionary from response

**patch** (*rest\_query*, *data=None*)

Sends a PATCH request to the Alyx server. For the dictionary contents, refer to: <https://alyx.internationalbrainlab.org/docs>

**Parameters**

- `rest_query` (*str*) – (required) the endpoint as full or relative URL
- `data` (*None*, *dict* or *str*) – json encoded string or dictionary

**Returns** response object

**post** (*rest\_query*, *data=None*)

Sends a POST request to the Alyx server. For the dictionary contents, refer to: <https://alyx.internationalbrainlab.org/docs>

**Parameters**

- `rest_query` (*str*) – (required) the endpoint as full or relative URL
- `data` (*None*, *dict* or *str*) – dictionary or json encoded string

**Returns** response object

**put** (*rest\_query*, *data=None*)

Sends a PUT request to the Alyx server. For the dictionary contents, refer to: <https://alyx.internationalbrainlab.org/docs>

**Parameters**

- `rest_query` (*str*) – (required) the endpoint as full or relative URL
- `data` (*None*, *dict* or *str*) – dictionary or json encoded string

**Returns** response object

**rest** (*url=None*, *action=None*, *id=None*, *data=None*, *\*\*kwargs*)

`alyx_client.rest()`: lists endpoints `alyx_client.rest(endpoint)`: lists actions for endpoint  
`alyx_client.rest(endpoint, action)`: lists fields and URL

Example with a rest endpoint with all actions

```
>>> alyx.client.rest('subjects', 'list')
alyx.client.rest('subjects', 'list', field_filter1='filterval')
alyx.client.rest('subjects', 'create', data=sub_dict)
alyx.client.rest('subjects', 'read', id='nickname')
alyx.client.rest('subjects', 'update', id='nickname', data=sub_dict)
alyx.client.rest('subjects', 'partial_update', id='nickname', data=sub_
↩ict)
alyx.client.rest('subjects', 'delete', id='nickname')
```



**Parameters**

- **url** – endpoint name
- **action** – ‘list’, ‘create’, ‘read’, ‘update’, ‘partial\_update’, ‘delete’
- **id** – lookup string for actions ‘read’, ‘update’, ‘partial\_update’, and ‘delete’
- **data** – data dictionary for actions ‘update’, ‘partial\_update’ and ‘create’
- **\*\*kwargs** – filter as per the REST documentation

**Returns** list of queried dicts (‘list’) or dict (other actions)

`oneibl.webclient.dataset_record_to_url(dataset_record)`

Extracts a list of files urls from a list of dataset queries.

**Parameters** `dataset_record` (*list*) – dataset Json from a rest request.

**Returns** (*list*) a list of strings representing files urls corresponding to the datasets records

`oneibl.webclient.file_record_to_url(file_records, urls=[])`

Translate a Json dictionary to an usable http url for downloading files.

**Parameters**

- **file\_records** (*dict*) – json containing a ‘data\_url’ field
- **urls** (*list*) – a list of strings containing previous data\_urls on which new urls will be appended

**Returns** `urls`: (*list*) a list of strings representing full data urls

`oneibl.webclient.http_download_file(full_link_to_file, *, clobber=False, offline=False, username="", password="", cache_dir="")`

**Parameters**

- **full\_link\_to\_file** (*str*) – http link to the file.
- **clobber** (*bool*) – [False] If True, force overwrite the existing file.
- **username** (*str*) – [‘’] authentication for password protected file server.
- **password** (*str*) – [‘’] authentication for password protected file server.
- **cache\_dir** (*str*) – [‘’] directory in which files are cached; defaults to user’s Download directory.

**Returns** (*str*) a list of the local full path of the downloaded files.

`oneibl.webclient.http_download_file_list(links_to_file_list, **kwargs)`

Downloads a list of files from the flat Iron from a list of links. Same options behaviour as `http_download_file`

**Parameters** `links_to_file_list` (*list*) – list of http links to files.

**Returns** (*list*) a list of the local full path of the downloaded files.

- `genindex`



## a

alf.folders, 27  
alf.io, 25

## i

ibllib.dsp.fourier, 28  
ibllib.dsp.utils, 29  
ibllib.ephys.ephysqc, 31  
ibllib.graphic, 47  
ibllib.io.extractors.biased\_trials, 32  
ibllib.io.extractors.biased\_wheel, 32  
ibllib.io.extractors.ephys\_fpga, 33  
ibllib.io.extractors.ephys\_trials, 34  
ibllib.io.extractors.training\_audio, 34  
ibllib.io.extractors.training\_trials,  
35  
ibllib.io.extractors.training\_wheel, 39  
ibllib.io.flags, 40  
ibllib.io.params, 40  
ibllib.io.raw\_data\_loaders, 41  
ibllib.io.spikeglx, 43  
ibllib.misc.misc, 45  
ibllib.misc.version, 45  
ibllib.pipes.experimental\_data, 46  
ibllib.pipes.extract\_session, 47  
ibllib.pipes.purge\_rig\_data, 47  
ibllib.pipes.transfer\_rig\_data, 47  
ibllib.plots, 47

## O

oneibl.one, 49  
oneibl.registration, 51  
oneibl.webclient, 51



## A

alf.folders (module), 27  
 alf.io (module), 25  
 AlfBunch (class in alf.io), 25  
 align\_with\_bpod() (in module  
 ibllib.io.extractors.ephys\_fpga), 33  
 AlyxClient (class in oneibl.webclient), 51  
 amplitude\_cutoff() (in module  
 ibllib.ephys.ephysqc), 31  
 append() (alf.io.AlfBunch method), 25  
 authenticate() (oneibl.webclient.AlyxClient  
 method), 51

## B

bp() (in module ibllib.dsp.fourier), 28

## C

check\_alf\_folder() (in module  
 ibllib.io.extractors.training\_trials), 35  
 check\_alf\_folder() (in module  
 ibllib.io.extractors.training\_wheel), 39  
 check\_dimensions() (in module alf.io), 25  
 compress\_ephys() (in module  
 ibllib.pipes.experimental\_data), 46  
 compress\_file() (ibllib.io.spikeglx.Reader method),  
 43  
 create\_sessions() (oneibl.registration.RegistrationClient  
 method), 51

## D

dataframe() (in module alf.io), 25  
 dataset\_record\_to\_url() (in module  
 oneibl.webclient), 53  
 decompress\_file() (ibllib.io.spikeglx.Reader  
 method), 43  
 delete() (oneibl.webclient.AlyxClient method), 51

## E

eq() (in module ibllib.misc.version), 45

excise\_flag\_file() (in module ibllib.io.flags), 40  
 exists() (in module alf.io), 26  
 extract() (in module ibllib.pipes.experimental\_data),  
 46  
 extract\_all() (in module  
 ibllib.io.extractors.ephys\_fpga), 33  
 extract\_all() (in module  
 ibllib.io.extractors.ephys\_trials), 34  
 extract\_behaviour\_sync() (in module  
 ibllib.io.extractors.ephys\_fpga), 33  
 extract\_camera\_sync() (in module  
 ibllib.io.extractors.ephys\_fpga), 33  
 extract\_ephys() (in module  
 ibllib.pipes.experimental\_data), 46  
 extract\_rmsmap() (in module ibllib.ephys.ephysqc),  
 31  
 extract\_sync() (in module  
 ibllib.io.extractors.ephys\_fpga), 33  
 extract\_wheel\_sync() (in module  
 ibllib.io.extractors.ephys\_fpga), 34

## F

falls() (in module ibllib.dsp.utils), 30  
 fexpand() (in module ibllib.dsp.fourier), 28  
 file\_record\_to\_url() (in module  
 oneibl.webclient), 53  
 find\_sessions() (in module alf.folders), 27  
 find\_subject\_folders() (in module alf.folders),  
 27  
 find\_subject\_names() (in module alf.folders), 27  
 firstlast (ibllib.dsp.utils.WindowGenerator at-  
 tribute), 30  
 freduce() (in module ibllib.dsp.fourier), 29  
 fronts() (in module ibllib.dsp.utils), 30  
 fs (ibllib.io.spikeglx.Reader attribute), 43  
 fscale() (in module ibllib.dsp.fourier), 29

## G

ge() (in module ibllib.misc.version), 46

get () (*oneibl.webclient.AlyxClient* method), 52  
 get\_camera\_timestamps () (in module *ibllib.io.extractors.training\_trials*), 35  
 get\_choice () (in module *ibllib.io.extractors.training\_trials*), 35  
 get\_contrastLR () (in module *ibllib.io.extractors.biased\_trials*), 32  
 get\_contrastLR () (in module *ibllib.io.extractors.training\_trials*), 35  
 get\_feedback\_times () (in module *ibllib.io.extractors.training\_trials*), 36  
 get\_feedbackType () (in module *ibllib.io.extractors.training\_trials*), 36  
 get\_goCueOnset\_times () (in module *ibllib.io.extractors.training\_trials*), 36  
 get\_goCueTrigger\_times () (in module *ibllib.io.extractors.training\_trials*), 37  
 get\_hardware\_config () (in module *ibllib.io.spikeglx*), 44  
 get\_ibl\_sync\_map () (in module *ibllib.io.extractors.ephys\_fpga*), 34  
 get\_intervals () (in module *ibllib.io.extractors.training\_trials*), 37  
 get\_iti\_duration () (in module *ibllib.io.extractors.training\_trials*), 37  
 get\_port\_events () (in module *ibllib.io.raw\_data\_loaders*), 41  
 get\_repNum () (in module *ibllib.io.extractors.training\_trials*), 37  
 get\_response\_times () (in module *ibllib.io.extractors.training\_trials*), 38  
 get\_rewardVolume () (in module *ibllib.io.extractors.training\_trials*), 38  
 get\_session\_extractor\_type () (in module *ibllib.pipes.extract\_session*), 47  
 get\_session\_path () (in module *alf.io*), 26  
 get\_stimOn\_times () (in module *ibllib.io.extractors.training\_trials*), 38  
 get\_stimOn\_times\_ge5 () (in module *ibllib.io.extractors.training\_trials*), 38  
 get\_stimOn\_times\_lt5 () (in module *ibllib.io.extractors.training\_trials*), 39  
 get\_task\_extractor\_type () (in module *ibllib.pipes.extract\_session*), 47  
 get\_velocity () (in module *ibllib.io.extractors.training\_wheel*), 39  
 get\_wheel\_data () (in module *ibllib.io.extractors.training\_wheel*), 39  
 getfile () (in module *ibllib.io.params*), 40  
 glob\_ephys\_files () (in module *ibllib.io.spikeglx*), 44  
 gt () (in module *ibllib.misc.version*), 46

## H

hp () (in module *ibllib.dsp.fourier*), 29  
 http\_download\_file () (in module *oneibl.webclient*), 53  
 http\_download\_file\_list () (in module *oneibl.webclient*), 53

## I

*ibllib.dsp.fourier* (module), 28  
*ibllib.dsp.utils* (module), 29  
*ibllib.ephys.ephysqc* (module), 31  
*ibllib.graphic* (module), 47  
*ibllib.io.extractors.biased\_trials* (module), 32  
*ibllib.io.extractors.biased\_wheel* (module), 32  
*ibllib.io.extractors.ephys\_fpga* (module), 33  
*ibllib.io.extractors.ephys\_trials* (module), 34  
*ibllib.io.extractors.training\_audio* (module), 34  
*ibllib.io.extractors.training\_trials* (module), 35  
*ibllib.io.extractors.training\_wheel* (module), 39  
*ibllib.io.flags* (module), 40  
*ibllib.io.params* (module), 40  
*ibllib.io.raw\_data\_loaders* (module), 41  
*ibllib.io.spikeglx* (module), 43  
*ibllib.misc.misc* (module), 45  
*ibllib.misc.version* (module), 45  
*ibllib.pipes.experimental\_data* (module), 46  
*ibllib.pipes.extract\_session* (module), 47  
*ibllib.pipes.purge\_rig\_data* (module), 47  
*ibllib.pipes.transfer\_rig\_data* (module), 47  
*ibllib.plots* (module), 47  
*is\_session\_path* () (in module *alf.io*), 26  
*is\_uuid\_string* () (in module *alf.io*), 26  
*isi\_violations* () (in module *ibllib.ephys.ephysqc*), 31

## K

keywords () (*oneibl.one.ONE* static method), 49

## L

le () (in module *ibllib.misc.version*), 46  
 list () (*oneibl.one.ONE* method), 49  
 load () (*oneibl.one.ONE* method), 49  
 load\_ambient\_sensor () (in module *ibllib.io.raw\_data\_loaders*), 41

- load\_bpod() (in module *ibllib.io.raw\_data\_loaders*), 41
- load\_data() (in module *ibllib.io.raw\_data\_loaders*), 41
- load\_dataset() (*oneibl.one.ONE* method), 50
- load\_encoder\_events() (in module *ibllib.io.raw\_data\_loaders*), 41
- load\_encoder\_positions() (in module *ibllib.io.raw\_data\_loaders*), 41
- load\_encoder\_trial\_info() (in module *ibllib.io.raw\_data\_loaders*), 42
- load\_file\_content() (in module *alf.io*), 26
- load\_mic() (in module *ibllib.io.raw\_data\_loaders*), 42
- load\_object() (in module *alf.io*), 26
- load\_object() (*oneibl.one.ONE* method), 50
- load\_settings() (in module *ibllib.io.raw\_data\_loaders*), 42
- login() (in module *ibllib.graphic*), 47
- lp() (in module *ibllib.dsp.fourier*), 29
- lt() (in module *ibllib.misc.version*), 46
- ## N
- nc (*ibllib.io.spikeglx.Reader* attribute), 43
- next\_num\_folder() (in module *alf.folders*), 27
- ns (*ibllib.io.spikeglx.Reader* attribute), 43
- ## O
- ONE (class in *oneibl.one*), 49
- oneibl.one (module), 49
- oneibl.registration (module), 51
- oneibl.webclient (module), 51
- ## P
- patch() (*oneibl.webclient.AlyxClient* method), 52
- post() (*oneibl.webclient.AlyxClient* method), 52
- print\_progress() (*ibllib.dsp.utils.WindowGenerator* method), 30
- print\_progress() (in module *ibllib.misc.misc*), 45
- put() (*oneibl.webclient.AlyxClient* method), 52
- ## Q
- qc\_fpga\_task() (in module *ibllib.ephys.ephysqc*), 32
- ## R
- raw\_ephys\_qc() (in module *ibllib.pipes.experimental\_data*), 46
- read() (*ibllib.io.spikeglx.Reader* method), 43
- read() (in module *ibllib.io.params*), 40
- read() (in module *ibllib.io.spikeglx*), 45
- read\_flag\_file() (in module *ibllib.io.flags*), 40
- read\_meta\_data() (in module *ibllib.io.spikeglx*), 45
- read\_samples() (*ibllib.io.spikeglx.Reader* method), 44
- read\_sync() (*ibllib.io.spikeglx.Reader* method), 44
- read\_sync\_analog() (*ibllib.io.spikeglx.Reader* method), 44
- read\_sync\_digital() (*ibllib.io.spikeglx.Reader* method), 44
- read\_ts() (in module *alf.io*), 26
- Reader (class in *ibllib.io.spikeglx*), 43
- register\_sync() (*oneibl.registration.RegistrationClient* method), 51
- RegistrationClient (class in *oneibl.registration*), 51
- remove\_empty\_folders() (in module *alf.folders*), 27
- remove\_uuid\_file() (in module *alf.io*), 26
- remove\_uuid\_recursive() (in module *alf.io*), 26
- rest() (*oneibl.webclient.AlyxClient* method), 52
- rises() (in module *ibllib.dsp.utils*), 30
- rms() (in module *ibllib.dsp.utils*), 31
- rmsmap() (in module *ibllib.ephys.ephysqc*), 32
- ## S
- save\_metadata() (in module *alf.io*), 26
- save\_object\_npy() (in module *alf.io*), 27
- search() (*oneibl.one.ONE* method), 50
- search\_terms() (*oneibl.one.ONE* static method), 51
- session\_name() (in module *alf.folders*), 27
- session\_path() (in module *alf.folders*), 27
- setup() (*oneibl.one.ONE* static method), 51
- slice (*ibllib.dsp.utils.WindowGenerator* attribute), 30
- slice\_array() (*ibllib.dsp.utils.WindowGenerator* method), 30
- spike\_sorting\_metrics() (in module *ibllib.ephys.ephysqc*), 32
- split\_sync() (in module *ibllib.io.spikeglx*), 45
- squares() (in module *ibllib.plots*), 47
- strinput() (in module *ibllib.graphic*), 47
- subjects\_data\_folder() (in module *alf.folders*), 27
- sync\_merge\_ephys() (in module *ibllib.pipes.experimental\_data*), 47
- sync\_trials\_robust() (in module *ibllib.io.raw\_data\_loaders*), 43
- ## T
- to\_df() (*alf.io.AlfBunch* method), 25
- traces() (in module *ibllib.plots*), 48
- trial\_times\_to\_times() (in module *ibllib.io.raw\_data\_loaders*), 43
- tscale() (*ibllib.dsp.utils.WindowGenerator* method), 30
- type (*ibllib.io.spikeglx.Reader* attribute), 44
- ## V
- validate\_ttl\_test() (in module

*ibllib.ephys.ephysqc*), 32  
verify\_hash() (*ibllib.io.spikeglx.Reader* method), 44  
vertical\_lines() (*in module ibllib.plots*), 48

## W

welchogram() (*in module  
ibllib.io.extractors.training\_audio*), 34  
wiggle() (*in module ibllib.plots*), 48  
WindowGenerator (*class in ibllib.dsp.utils*), 29  
write() (*in module ibllib.io.params*), 40  
write\_flag\_file() (*in module ibllib.io.flags*), 40