Official Documentation of OrganaSys

Introduction:

This guide introduces the fundamentals of working with OrganaSys' NeuroPlatform. The first section covers the essentials:

- How to stimulate organoids
- Collecting data from the platform
- Interpret neural spikes for closed loop experiments

The second section highlights many utilities that may help support your research, whether it's for configuring parameters to improving your data analytics.

Content Table:

1. Working with the NeuroPlatform:

- Core System Functions & Organoid Stimulation
- Data Management: Access and Extraction of results
- Collection Metadata & Review
- OrganaSys' Recommendation for Reliable Use of the NeuroPlatform

2. Utilities:

- Cross Correlograms Analysis
- Neural Spike Classification
- Real-Time Data Recording
- Parameter Loading & Scanning for Stimulation

1.1 - Core System Functions & Organoid Stimulation

Basic Concepts:

Here we introduce the fundamental pieces of the system, and the ideas behind how it works. Don't worry if nothing is clear, we will walk through later sections using the Python API in addition to worked examples.

System Specification:

Hardware:

- Sampling rate is 30 kHz
- Filters Applied: 250 - 5000 Hz
- 4 MEAs each with an array of 4x8 electrodes
    - *Standard layout currently in use*
    - *8 electrodes per neurosphere*
    - *Other layouts under testing permitting multiple connected neurospheres*

Wetware:

- Neurospheres are organoids derived from human neural stem cells
    - *less than 1mm in diameter*
    - *Contains neurons and glial cells*
    - *2 to 12 weeks lifespan on MEAs ranges*

- Growth mediums including Neurobasal+ & Brain Phys
- 1 to 4 neurospheres placed per MEA site

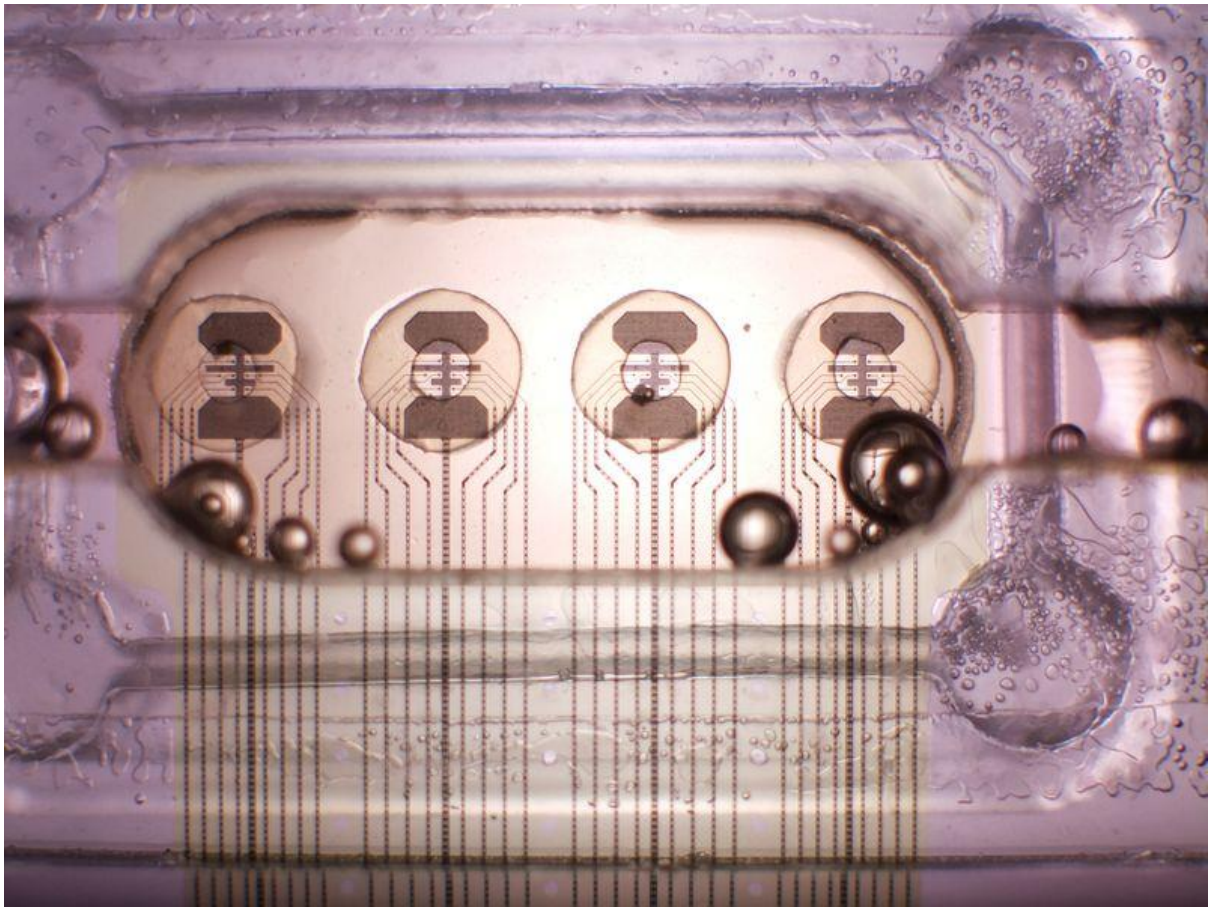


Exhibit 1 *- 4x8 layout MEA with neurospheres*

System Components:

There are 4 main parts that make up OrganaSys:

- **Spike Database *(DB)*:** Captures spike events from electrodes whilst continuously running. The spike event registers when the electrode voltage exceeds 6 times the standard deviation of background noise.

- **Intan Software:** Interfaces with MEAs, allowing configuration and data acquisition. It's accessed via Python in order to control it.

- **Intan Controller:** The physical hardware connected to the MEAs that carries out the recording and the stimulation commands.

- **Trigger Generator:** Sends trigger signals to the Intan Controller. Users define Stimulus Parameters or Stimparam, linking them to the triggers and once triggered, the Intan system delivers the corresponding stimulation.

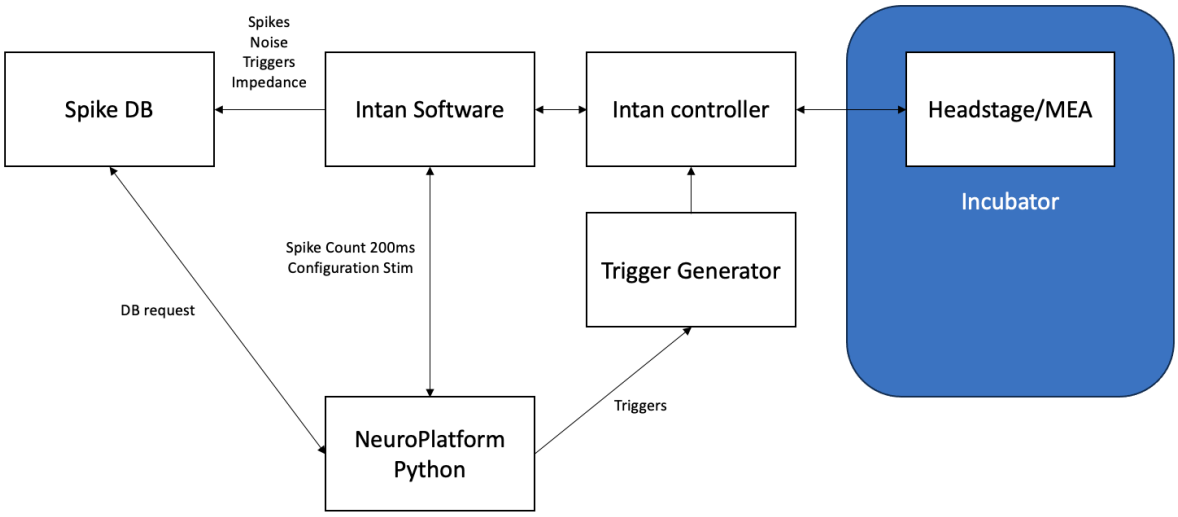


Exhibit 2 *- The NeuroPlatform System*

Core API Concepts:

Key abstractions and how they work together:

- **StimParam:** A structure defining stimulation: duration, amplitude, frequency, electrode index and more. It's linked to a trigger so when the trigger is sent, the Intan Controller carries it out associated with that trigger on the designated electrode.

- **Index:** Refers to which electrode to stimulate.

- **Trigger Key:** Identifies which trigger number between 0 and 15 is used. Multiple Stimparams may share the same trigger key but must have distinct electrode indices.

CAUTION: *Using the same index in two different StimParams can overwrite the previous one and only use the last one.*

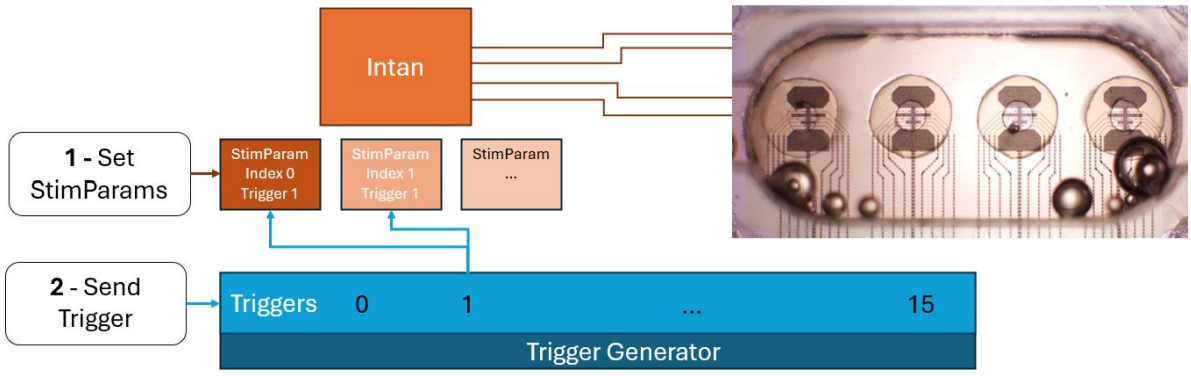


Exhibit 3 *- The relationship between StimParams and triggers. When a StimParam is assigned to a specific electrode index, it sends the corresponding trigger prompts to the Intan Controller, applying that StimParam to the designated electrode. Each index can only hold 1 StimParam but a single trigger may reference multiple indices.*

API Usage:

Creating Experiments: Using tokens and 'Experiments' can help users interact with the system. It's used to check that no other experiment is running while others use the system, avoiding conflicts.

Imports:

```
import numpy as np
import time
from datetime import datetime
from neuroplatform import StimParam, IntanSoftware, Trigger, StimPolarity, Experiment
```

Experiment & Token:

```
token = "9T5KLS6T7x"   # We provide you with a token for the experiment
exp = Experiment(token)
print(f"Electrodes: {exp.electrodes}")  # Electrodes that you can use")
```

To be able to see the live view of the electrode, go to the Live View page whilst checking the 'Absolute Index' function to get the correct electrode number.

TIP: *Always remember to stop the experiment once you've started it with* [exp.stop()] *. We also highly recommend to use* [try... finally] *to ensure that the experiment is forced stopped.* (edited)

```
try:
    exp.start()  # Your experiment code here


finally:
    exp.stop()   # This ensures a proper shutdown of the experiment
```

Stimulation Parameters:

With Python, you can control the Itan Software to set the stimulation parameters. The elements within the stimulation is shown below:

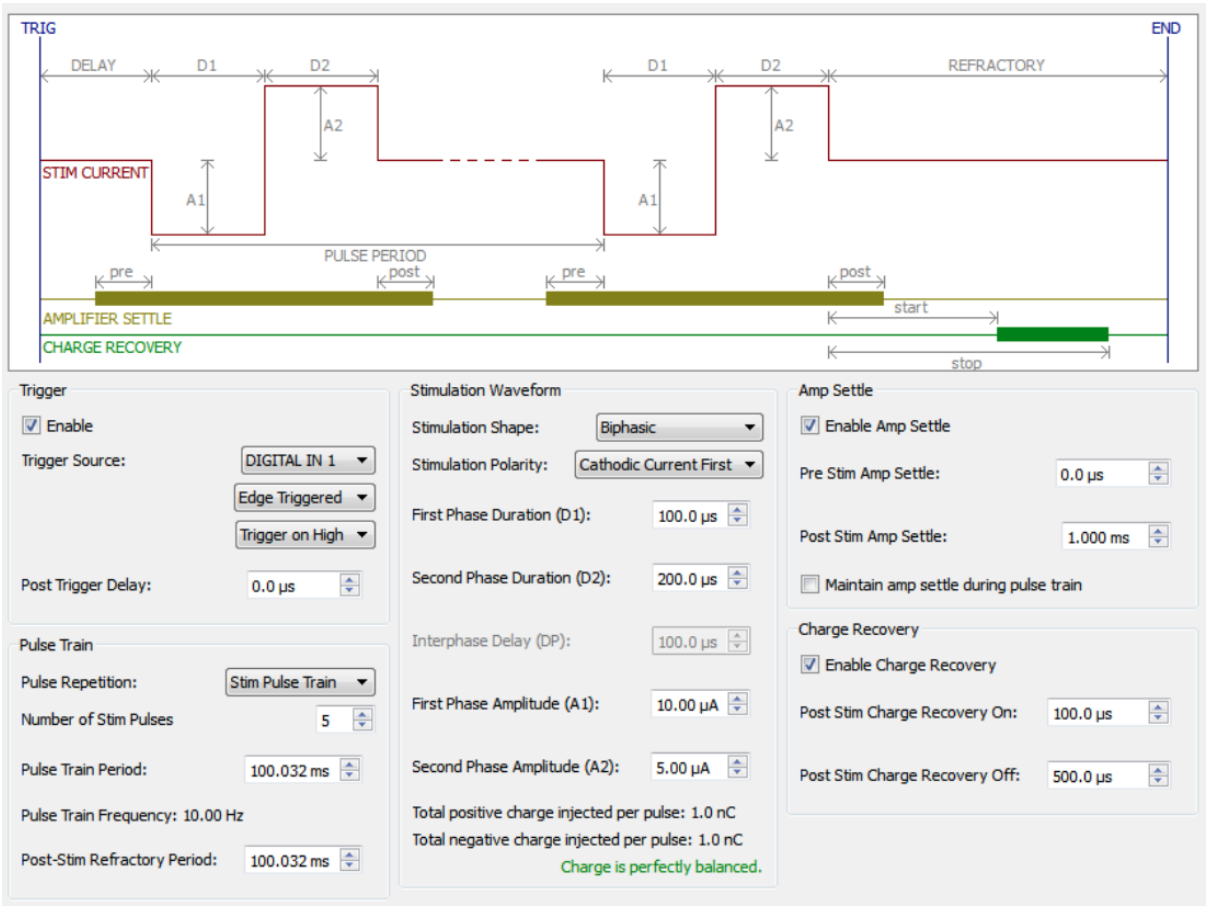


Exhibit 4 - Stimulation Parameters

It's defined by many values such as the examples in the list below:

Name	Description	Default Value
enable	Enable stimulation	True
index	Electrode index [0-127]	0
trigger_key	Trigger key [0-15]	0
polarity	Polarity of the stimulation	NegativeFirst
phase_duration1	D1 [us]	100.0
phase_amplitude1	A1 [uA]	1.0
phase_duration2	D2 [us]	100
phase_amplitude2	A2 [uA]	1.0
----------	----------	----------
stim_shape	Stimulation Shape	Biphasic
interphase_delay	Interphase delay [us]	0.0
trigger_delay	Post trigger delay [us]	0
nb_pulse	Number of pulses	0
pulse_train_period	Pulse Train Period [us]	10000
----------	----------	----------
post_stim_ref_period	Post-Stimulation Refractory Period [us]	1000.0
enable_amp_settle	Enable amplitude settling	True
pre_stim_amp_settle	Pre-stimulation amplitude settling [us]	0.0
post_stim_amp_settle	Post-stimulation amplitude settling [us]	1000.0
enable_charge_recovery	Enable charge recovery	True
post_charge_recovery_on	Post charge recovery on [us]	0.0
post_charge_recovery_off	Post charge recovery off [us]	100.0

Configuring stimulation parameters:

The StimParaLoader tool can be used to organise and preview your stimulation settings. It's divided into 3 main groups:

1. **Core Parameters -**

These define the essentials aspects of stimulation. You'll need to adjust them for each experiment done.

- When possible, keep the charge balanced between phases, which helps to extend the lifespan of the organoid and the electrodes.
- Polarity strongly influences the responses. Recommendation to test and confirm which polarity works best for your specific setup.

2. Experiment Specific Options -

Additional parameters can be tuned depending on your goals.

- **[nb_pulse]** is helpful when designing burst stimulations.
- **[pulse_train_period]** controls the timing between each bursts.

3. Fixed Parameters -

Settings that shouldn't be changed. If unsure about the importance for your experiences, reach out to us for guidance before modifying them.

Example:

```
stim_param1 = StimParam()
stim_param1.enable = True   # Enable the stimulation
stim_param1.trigger_key = 0   # Trigger key to be used
stim_param1.index = 0   Index of the stimulation
stim_param1.polarity = StimPolarity.PostiveFirst
stim_param1.phase_duration2 = 100
stim_param1.phase_duration1 = 100
stim_param1.phase_amplitude2 = 1
stim_param1.phase_amplitude1 = 1
stim_param1.display_attributes()
```

```
stim_param2 = StimParam()
stim_param2.index = exp.electrodes[
    8 # Use your provided electrodes to choose the site of stimulation
]


stim_param2.enable = True
stim_param2.trigger_key = (
    10 # The trigger key will be used to send the stimulation
)
stim_param2.polarity = StimPolarity.NegativeFirst
stim_param2.display_attributes()
```

TIP: *Using Python to time your stimulation will always be less specific than using the pulse train settings. It can't be changed quickly during an experiment so users must choose between flexible spike trains with less specific timing i.e. Python or precise timing with less flexibility i.e. Pulse Training settings.*

When creating a parameter, keep in mind:

- Index: Selects the electrode to be stimulated, each of them can only be assigned to 1 parameter as multiple parameters will overwrite the previous one.
- Trigger Key: Defines the trigger signal sent to the Intan Controller, which supports up to 16 triggers all numbered 0-15.
- Enable: Indicates whether the parameter is active. If so, the Intan Software records the parameter linked to the corresponding trigger.

Connecting the Intan and Trigger Generator:

This can be done by doing the following:

```
inta = IntanSoftware()
params = [stim_param1, stim_param2]   # Create a list of stimulation parameters

intan.send_stimparam(
    params
)   # Send the stimulation parameters to the Intan Software. Takes 10s.

intan.close()   # Close the connection to the Intan Software
```

Bare in mind, it will take 10 seconds for the parameters to be completely sent due to the Intan Software checking the parameters and sending it to the Intan Controller, then to the MEA.

Connecting to the Trigger Generator and sending triggers can be done with the following below, taking trigger 3 as an example:

```
trigger_gen = Trigger()
trigger_array = np.zeros(
    16, dtype=np.uint8
)   # All triggers set to 0

trigger_array[3] = 1   # This will send a trigger to the stimulation with trigger key = 3

trigger_gen.close()   # Close the trigger generator
```

Setting all corresponding indices to 1 helps to enable several triggers at the same time. This should be the final result.

```
intan = IntanSoftware()   # Connects to the Intan IntanSoftware
trigger_gen = Trigger()   # Connects to the trigger generator

stim_params = [stim_param1, stim_param2]

try:
    if exp.start():   # Signals the start of an experiment
    intan.send_stimparam(stimparams)
    start_exp = datatime.utcnow()
    print(start_exp)

    trigger0 = np.zeros(16, dtype-np.uint8)
    trigger0[0] = (
        1   # Enable stimulation on trigger 0, bounded to stim_param1
    )

    for i in range(10):
        # Stimulation on electrode 10
        trigger_gen.send(
            trigger0
        )   # Sends 10x the StimParam on electrode 0

        time.sleep(1)

        ####################

        trigger0and10 = np.zeros(16, dtype-np.uint8)
        trigger0and10[0] = 1
        trigger0and10[10] = 10

        for i in range(10):
            # Stimulation on both electrodes
            trigger_gen.send(
                trigger0and10
            )
            time.sleep(1)

        stop_exp = datatime.utcnow()
        print(stop_exp)

        # Disable all stimparams
        for stim in stim_params:
            stim.eables = False
        intan.send_stimparam(stim_params)

finally:
    # Disconnect to the trigger generator
    trigger_gen.close()
    # Disconnect to the Intan IntanSoftware
    intan.close()
    # Signal the end of an experiment
    exp.stop()
```

1.2 - Data Management: Access & Extraction of Results

This section shows how to retrieve the generated data via recording the electrical activity of the organoids from the database itself.

Database Connection:

To access experimental data, create a database object and use its methods to retrieve the information needed. Each time organoids are changed on the MEA, a new fs ID is generated i.e. [“fs300”], which is accessible through your experiment token specifically through the [exp_name] attribute. Secure the token associated with the experiment is a highly recommended due to future data analysis.

```
from dateutil import parser
from datetime import timedelta

from neuroplatform import Database, Experiment

db = Database()
exp = Experiment("Your token here")
FS_NAME = (
    exp.exp_name
)   # FS ID for the MEA. Changes everytime we change organoids
```

Functions of the Database:

A spike event is always logged whenever the voltage from an electrode crosses a threshold of 6 times the standard deviation of the noise. Using the amplitude and the channel number using [get_spike_event] can help to give a list of spike events within a certain timestamp. Highly recommend starting with 5 minutes and increasing/decreasing the number as needed.

```
start = parser.parse("2025-04-05T00:00:00.000Z")
stop = start + timedelta(minutes=5)

spike_event_df = db.get_spike_event(start, stop, FS_NAME)
```

Using [get_raw_spike_event] helps to record each spike event over a 3ms frame. The output is a DataFrame containing the timestamp of the event, electrode number and the amplitude values for the 3ms signal segment.

```
spike_per_min_df = db.get_spike_count(start, stop, FS_NAME)
```

As we're using the sampling rate of 30kHz, each spike events will have a total of 90 samples.

```
from tqdm import tqdm

df_spike_event = db.get_spike_event(start, start + timedelta(seconds=1), FS_NAME)

raws = []
for i, row in tqdm(df_spike_event.iterrows(), total=df_spike_event.shape[0]):
    t = row["Time"]
    t1 = t - timedelta(milliseconds=1)
    t2 = t + timedelta(milliseconds=2)
    raws.append(db.get_raw_spike(t1, t2, row["channel"]))
```

Here's the example plot:

```
import numpy as np
from lets_plot import (
    ggplot,
    ggtitle,
    geom_line
    aes,
    ggsize,
    gggrid,
)


def plot_raw_channel(channel: int):
    df_chan = df_spike_event[df_spike_event["channel"] == channel]
    # Start with an employ plot
    p = ggplot()
    p += ggtitle(f"Channel {channel}")

    for i, row in df_chan.iterrows():
        df_raw = raws[i]
        if df_raw.shape[0] == 90:
            df_raw["Time [ms]"] = np.linspace(-1, 2, 90)
            p += geom_line(aes(x="Time [ms]", y="Amplitude"), data=df_raw)

    return p


channels = df_spike_event["channel"].unique()
list_plot = []
for i, chan in enumerate(channels):
        list_plot.append(plot_raw_channel(chan))

nbCol = 3
 bRow = len(list_plot) // nbCol
 if len(list_plot) % nbCol != 0:
     nbRow += -1
gggrid(list_plot, ncol=nbCol) + ggsize(800, 300 * nbRow)
```

Users can obtain the triggers used using [get_triggers] to return a DataFrame with the trigger number, timestamp and the current status which corresponds to the times where the trigger was triggered.

```
triggers_df = db.get_all_triggers(start, stop)
```

When measuring the impedance using the Intan Controller, you get the recorded values using [get_impedance] to return the same as the [get_triggers] but replacing the status with the impedance values.

```
df_impedance = db.get_impedance(start, stop)
```

Miscellaneous Measurements:

When we provide access to several environmental measurements from the incubator hosting the MEAs, including:

- Temperature
- Humidity
- Pressure
- CO_2 Levels
- O_2 Levels

```
temp_df = dbtemperature(start, stop)
humidity_df = db.humidity(start, stop)
pressure_df = db.pressure(start, stop)
co2_df = db.co2(start, stop)
o2_df = db.o2(start, stop)
```

- Pump Flow Rate (1 for each MEA, total of 4)

```
port = 3   # port is the MEA number from 1 to 4
flow_df = db.flowrate(start, stop, port)
```

- Door Open/Close Events
  - Incubator 0 hosts the MEAs
  - Incubator 1 hosts the organoid in culture

```
door_df = db.door(start, stop)
```

Accessing Experiment Metadata:

These following commands below can help you get information about your experiments.

```
from neuroplatform import elab

# Create the object elab()
metadata = elab()
metadata.check_server()   # Should print "The server works !" if right

# Choose the experiment you want to study: For example, we use fs001
fs_name = "fs001"
```

1. Starting date of your experiment:

```
starting_date = metadata.get_start_date(fs_name)
print(starting_date)
```

2. Getting the cell batch:

```
cell_batch = metadata.get_cell_batch(fs_name)
print(cell_batch)
```

3. Getting the medium used on the MEA keeping the organoids alive:

```
medium = metadata.get_medium(fs_name)
print(medium)
```

4. Getting the MEA *i.e. the name*:

```
mea = metadata.get_mea(fs_name)
print(mea)
```

5. Getting the pump *i.e. type, flux and flux unit*:

```
pump = metadata.get_pump(fs_name)
print(pump)
```

6. Getting the version of MEA used for the experiment:

```
protocols = metadata.get_protocols(fs_name)
print(protocols)
```

7. Getting the date where the organoids were removed from the MEA:

```
death_date = metadata.get_death_date(fs_name)
print(death_date)
```

8. Getting the name of the camera associated with the MEA:

```
camera = metadata.get_camera(fs_name)
print(camera)
```

9. Getting the name of the incubator and the index:

```
incubator = metadat.get_incubator(fs_name)
print(incubator)
```

1.3 - Collection Metadata & Review

We explain how to use the Intan system to measure the spike counts whilst supporting closed-loop experiments. Stimulations need to be triggered based on organoid activity; these functions can help to detect responses and act accordingly. Spike counting performs over the 200ms window once a trigger has been sent with the counts being stored in a buffer covering all 128 electrodes.

Recognising number of spikes using Intan:

The IntanSoftware provides these methods to be used for closed-loop experiments:

- **[set_count]** : set which triggers start the counting process.
  - Input is the list of trigger IDs, starting from 0 to 15

- Must send the trigger in question to start the counting process.
- **[read_count]** helps to retrieve the buffer storing the number of spikes, not counting them individually.
  - The buffer is a 128-length array containing the number of spikes for each electrode
  - Select the relevant channels to your experiment.

Code Example:

```
from neuroplatform import Experiment, StimParam, IntanSoftware, Trigger
from datetime import datetime, UTC
import datetime
import numpy as np
```

```
token = "..."   # Experiment token
exp = Experiment(token)
print(f"Electrodes: {exp.electrodes}")   # Electrodes that can be used with the token
```

```
stim_param1 = StimParam()
stim_param1.index = exp.electrodes[0]   # The first electrodes
stim_param1.trigger_key = 0   # The first trigger
stim_param1.phase_amplitude1 = 2.0
stim_param1.phase_amplitude2 = 2.0
stim_param1.display_attributes()
```

```
intan = IntanSoftware()
trigger_gen = Trigger()
intan.set_count(
    [0, 1]
)   # Set the Intan to record the number of spikes whenever sending trigger 0 or 1

stim_params = [stim_param1]

try:
   if exp.start():   # Signals the start of the experiment
       # Send stim parameters to the Intan
       intan.send)stimparam(stim_params)
       start_exp = datetime.now(UTC)

       # Create the array of triggers, set desired triggers to 1
       trigger0 = np.zeros(16, dtype=np.uint8)
       trigger0[0] = 1
       trigger1 = np.zeros(16, dtype=np.uint8)
       trigger1 [1] = 1

       for i in range(10):
           # Stimulation on electrode 0
           trigger_gen.send(trigger0)   # Triggers both stim and the read. Wait 200ms to read for a close loop
           time.sleep(0.2)
           count_spikes = intan.read_count()   # np.array(128)
           # Get number of spikes of electrode 22 for the read
           total_spikes_stim = count_spikes[22]
           ###
           # Send trigger 1 - no stimulation
           trigger_gen.send(trigger1)
           time.sleep(0.2)
           count_spikes = intan.read_count()   # np.array(128)
           total_spikes_read_only = count_spikes [22]

           print(f"Spikes with stimulation: {total_spikes_stim}")
           print(f"Spikes without stimulation: {total_spikes_read_only}")

           time.sleep(1)

       stop_exp = datetime.now(UTC)

       # Disable all parameters
       for stim in stim_params:
           stim.enable = False
           intan.send_stimparam(stim_params)

finally:
    trigger_gen.close()
    intan.close()
    exp.stop()
```

1.4 - OrganaSys' Recommendation for Reliable Use of the NeuroPlatform

1. Biological Context & Preparation

Factor	Why it matters	Solution	Data to Record
Organoid Age	Neural activity shifts significantly with age	Determine age from event history; compare responses at multiple ages	Organoid age per each session
Initial Bursting Phase	Early activity can be unstable	Waiting several hours before recording, which confirms visually that bursts have been stabilised	Presence of bursts
Natural Decline over Time	Activity may fade independently of experiments	Track long term changes	Activity trend across the time
Organoid Token	Biological variability requires proper tracking	Save all organoid tokens to link experiments with different conditions	Token ID for each organoid used

2. Data Quality & Reliability

Factor	Why it matters	Solution	Data to Record
Raw Signal & Spike Shape Review	Real spikes differ from noise	Visualise raw signals and/or overly spikes. Should be expecting consistent patterns	Spike overlays showing reproducible shapes
Detecting Stimulation Artifacts	Fast responses aren't biological	Exclude or check spikes within first	Artifact free spike dataset

	< 10ms	10ms after stimulation	
Identical Spikes Across all Channels	Could indicate technical artifacts	Comapre spatial and temporal spread across electrodes	Activity across all electrodes
Electrode Performance Check	Some channels may not provide reliable data	Exclude all inactive electrodes before analysis	List of functional & faulty electrodes

3. Experimental Design & Replication

Factor	Why it matters	Solution	Data to Record
Replication on the same organoid	Ensures activity is not time dependent	Repeat trials with consistent spacing	Data from repeated trials
Interval Between Sessions	Neural fatigue or plasticity may affect responses	Allow recovery periods between stimulations	Responsiveness post recovery
Replication Across Organoids	Results must generalise across biological samples	Test on multiple organoids	Results comparison aross organoids
Use of Negative Controls	Confirms stimulation caused observed effects	Always include a baseline	Baseline activity levels

4. Platform Use & Parameter Settings

Factor	Why it matters	Solution	Data to Record
Login / Logout Procedures	Prevents session conflicts	Use [try / finally] for safe closure	Confirmation of proper session handling
Setting [stim_params] & Triggers	Hard to verify afterwards	Double check Param Loader, confirm electrodes, amplitude and duration	Electrode IDs and stimulation parameters
Parameter Tracking & Verification	Strong stimulations yield clearer results	Keep a log, perform grid searches across amplitudes and electrodes before the final run	Responsiveness under different settings
Spontaneous vs Evoked Activity	Helps isolate stimulation effects	Subtract baseline spikes from post-response window	Adjusted spike count

2.1 - Cross Correlograms Analysis

To start the process, run the following command:

```
pip install git+https://github.com/OrganaSys-np/np-utils.git#egg=np_utils[CCM]
```

Cross correlograms are tools used to examine the timing relationships between two sets of event occurrences. They're highly valuable in the study of neural spike data, where they help reveal how different channels interact. For example, by identifying synchronised firing between neurons or uncovering more complex patterns of co-activation across sites, such as oscillatory activity.

Setting Parameters:

To compute cross correlograms, began by starting the Cross Correlogram class with your chosen time window and bin size. Time window being the total duration of the cross correlogram in ms and bin size being the width of each bin in ms. An optional task is to normalise the parameter to 'True' to normalise the cross correlogram by square rooting the product of the number of events in each data frame.

Although it's recommended to use bin sizes that are about 1/10 smaller than the time window, the most important factor for capturing the temporal structure of the data is choosing an appropriate time window.

Key Considerations:

- Select a time window that matches the timescale of the phenomena you're working on.
- When including stimulation data, be careful as it can introduce artifacts into the cross correlogram.
- The amount of data required depends on the activity of the electrodes and the phenomena being studied but typically ranges from minutes to hours.

An example of the initialisation is below:

```
from np_utils.cross_corr import CrossCorrelogram

cc = CrossCorrelogram(
    deltat=50, bin_size=1, normalise=True
)   # deltat and bin_size in ms
```

Computing and putting in the cross-correlograms:

#1: Single Cross Correlogram

To compute a correlogram between two channels, you can use the method [compute_from_db].

Parameters:

- **[fs_id (str)]:** Experiment name associated with your token.
- **[channel_from (int)]:** ID of the first channel, ranging 0-127.
- **[channel_to (int)]:** ID of the second channel.
- **[start_time (pd.Timestamp)]:** Start time for the analysis.
- **[end_time (pd.Timestamp)]:** End time for the analysis.
- **[normalise (bool, optional)]:** If True, noramlises the correlogram. Defaults to the class setting.
- **[check_for_triggers (bool, optional)]:** If enabled, checks for triggers within the provided time windows. Defaults to False.
- **[keep_dfs (bool, optional)]:** If True, returns the dataframes used in the calculation. Defaults to True.
- **[hide_center_bin (bool, optional)]:** If True, removes the center bin. Defaults to False.

#2: Multiple Cross Correlogram

To compute a grid of cross correlograms between sources and targtets from the database, you need to use the method [compute_cc_grid_from_db].

Parameters:

- **[Sources (list)]:** List of source channels.
- **[Targets (list)]:** List of target channels.
- **[Fs_id (str)]:** Experiment name associated with your token.
- **[start_time (pd.Timestamp)]:** Start time for the analysis.
- **[end_time (pd.Timestamp)]:** End time for the analysis.
- **[check_for_triggers (bool, optional)]:** If enabled, checks for triggers within the provided time windows. Defaults to False.

2.2 - Cross Correlograms Analysis

To install the dependencies for this utility, use the command:

```
pip install git+https://github.com/OrganaSys-np.utils.git#egg=np_utils[SSG]
```

Select an electrode and a time window, and the algorithm will return the detected spikes within the specific window, along with their clustering. The spikes are categorised into two groups:

1. Spikes:
There are spikes confirmed in the time window. Their labels range from 0 and X amount of numbers of clusters.

These are spikes hat could not be assigned to any cluster. They will always be labelled as cluster –1.

2. Artifacts:
These are spikes identified as artifacts.

They're always assigned to cluster –2, which is reserved for them
They're also detected based on their amplitude. If a spike exceeds the **[artifact_threshold]** it will be classified as an artifact.

Usage:

Import the utility by using this command:

```
from np_utils.spike_sorting import SpikeSorting
```

Creating a SpikeSorting
Use the following parameters:

Pre-processing:

- **[realign] –** Specifies whether to realign spikes based on amplitude. The default is True. This is intended to help align spikes that are not perfectly aligned, ensuing that all peaks are centered around time 0.
- **[time_before] –** The time before the event to include in the window. Default is 0.5ms, with maximum value of 1ms.
- **[time_after] –** The time after the event to include in the window. Default is 1ms, with maximum value of 2ms.
- **[artifact_threshold] –** The threshold for detecting artifacts in uV. The default is 1000 uV, which is intended to filter out spikes that are too large to be real spikes.
- **[filter_artifacts] –** A flag indicating whether to filter out artifacts. The default is True. If otherwise, the artifacts will be treated as regular spikes.

Hyper parameters:

- **[dimred_method] –** The dimensionality reduction method to use. Options are either PCA or ICA. Default always being PCA.
- **[n_components] –** The number of components to keep after dimensionality reduction with the default number being 3.
- **[cluster_method] –** The clustering method to use with options including HDBSCAN or OPTICS. HDBSCAN is the default option and generally recommended.
- **[clustering_kwargs] –** Additional keyword arguments for the clustering method. The default is None.

Miscellaneous:

- **[n_jobs] –** The number of jobs to run in parallel. Default number being 8.

  - Setting it too high can make the database reject many requests at once.

```
from dateutil import parser
from datetime import timedelta

from np_utils.spike_sorting import SpikeSorting

start = parser.parse(
    "2025-05-13 13:00:00"
)   # Change this to the start time of the recording
```

```
EXP_NAME = "fs001"
N_COMP = 3
ELECTRODE = 9
sorter = SpikeSorting(dimred_method="ICA", n_components=N_COMP)
```

To run the SpikeSorting, you need to specify a start time, stop time and the experiment ID.

```
processed_spike_events_df = sorter.run_spike_sorting(
    start, start + timedelta(minutes=20), fs_id=EXP_NAME, electrode_nb=ELECTRODE
)
```

Results Plotting

Spike Sorting offers multiple plotting options:

- **[plot_clustered_spikes] –** Displays the average waveform for each cluster, including confidence intervals.
- **[plot_clustered_artifacts] –** Displays the average waveform for each artifact cluster, along with confidence intervals.
- **[plot_raw_spikes_for_clusters] –** Displays the raw trace of a specific cluster.
- **[plot_raw_outlier_spikes] –** Displays the raw trace of all outliers or artifacts.
- **[plot_spike_clustering_in_latent_space] –** Displays the clustering results in the latent space.
- **[plot_explained_variance] –** Displays the explained variance of the PCA.

2.3 - Real Time Data Recording

This code allows you to record live data using OrganaSys' Live MEA webview and save it to an HDF5 file. It gives around every second, just under 5000 samples of data just from 32 electrodes.

Installation:

We highly recommend installing miniconda to create a new environment.

1. Clone the repository:

```
git clone https://github.com/OrganaSys-np/LiveMEA.git
cd live-mea
```

2. Create and activate a conda environment:

```
conda create --name livemea-env python=3.11
conda active livemea-env
```

3. Install the required packages:

```
pip install -r requirements.txt
```

LiveMEA Class:

This is how the initialisation works using this command:

```
from MEA_live import livemea-envfrom pathlib import pathlib

live_mea = LiveaMEA(save_path: str | Path, recording_duration: int = 5, mea_id: int = 1)
```

- **[save_path]** - Path to save the HDF5 file.
- **[recording_duration]** - Duration of recording in seconds.
- **[mea_id]** - MEA ID to use. Defaults to 1 but must be between 1 and 4.
- **[record()]** - Starts recording live data and saves it to the file.
- **[plot_data(h5f_path: str | Path)]** - Plots data from the file.

Here's an example below:

```
from MEA_live import LiveMEA

# Initialise the LiveMEA instance
live_mea = LiveMEA(save_pat"live_data.h5", recording_duration=10, mea_id=1)

# Record data
live_mea.record()

# Plot recorded data
LiveMEA.plot_data("live_data.h5")
```

CLI Usage:

This is how the initialisation works using this starting command:
Run this from the repository root directory

```
python live-mea -d <duration> -p <path> -m <MEA>
```

- **[-d]**, **[--duration]** - Duration of the recording in seconds. Defaults to 5.
- **[-p]**, **[--path]** - Path to save the recorded data.
- **[-m]**, **[--MEA]** - MEA ID to record data from. Defaults to 1.

Here's the final example of this:

```
python live-mea -d 10 -p "live_data.h5" -m 1
```

2.4 - Parameter Loading & Scanning for Stimulation

Parameter Loader

The loader is used for managing stimulation parameters and for previewing them with an interactive plot. It has key features such as:

- Validate parameters
- Enable/disable parameters
- Preview electrode & parameter settings in an interactive plot
- Send parameters to the Intan Software.

The utility can be imported using the command:

```
from np_utils.parameters_loader import StimParamLoader
```

Creating a Parameter Loader:

- **[stimparams]** - List of StimParam instances, which can be none or empty.
- **[intan]** - Intan software instance that can be none if you don't want to send parameters immediately.
- **[must_connect]** - If True, raise an error if the Intan is not connected. Defaults to False.
- **[verbose]** - If True, print out log messages and defaults to True.

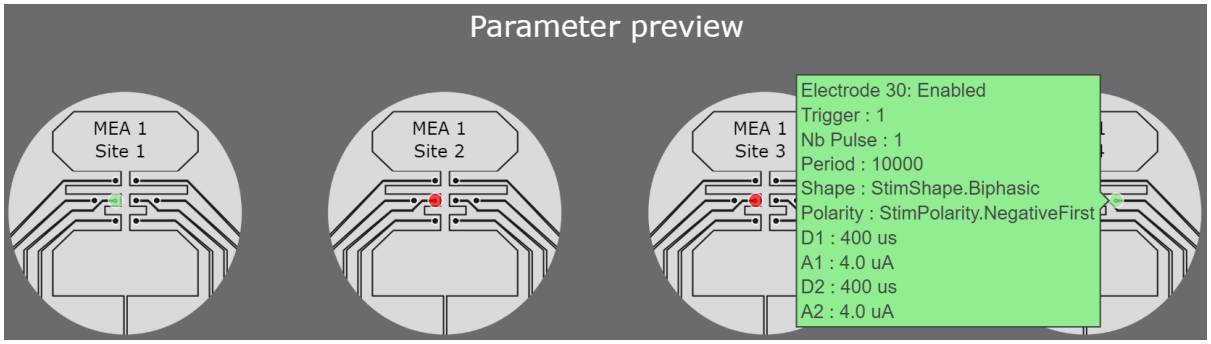
Changing Parameters:

- Create the instance with a list of **[stimparams]** instances.
- Modify the **[stimparams]** attribute directly.

- Use [reset()] to clear all parameters.

Previewing Parameters:

To preview the parameters, use the [preview_parameters] method. Here's an example of this below:



Sending & Logging Parameters:

To send the parameters to the Intan, use the [send_parameters] method. When finished, disable all parameters and send the update to the Intan using the [disable_all_and_send] method.

If using the [verbose] option, all the log messages will be shown.

Examples can be shown below this message:

```
from np_utils.parameters_loader import StimParamLoader
from neuroplatform import Stimparam, StimPolarity
```

```
stimparam1 = StimParam()
stimparam2 = StimParam()
stimparam3 = StimParam()
stimparam4 = StimParam()
###
stimparam1.index = 2
stimparam1.enable = True
stimparam1.trigger_key = 0
stimparam1.nb_pulse = 1
stimparam1.polarity = StimPolarity.PostiveFirst
stimparam1.phase_duration1 = 100
stimparam1.phase_duration2 = 100
stimparam1.phase_amplitude1 = 1.0
stimparam1.phase_amplitude2 = 1.0
###
stimparam2.index = 100
stimparam2.enable = False
stimparam2.trigger_key = 0
stimparam2.nb_pulse = 1
stimparam2.polarity = StimPolarity.NegativeFirst
stimparam2.phase_duration1 = 200
stimparam2.phase_duration2 = 200
stimparam2.phase_amplitude1 = 2.0
stimparam2.phase_amplitude2 = 2.0
###
stimparam3.index = 18
stimparam3.enable = False
stimparam3.trigger_key = 0
stimparam3.nb_pulse = 1
stimparam3.polarity = StimPolarity.PositiveFirst
stimparam3.phase_duration1 = 300
stimparam3.phase_duration2 = 300
stimparam3.phase_amplitude1 = 3.0
stimparam3.phase_amplitude2 = 3.0
###
stimparam4.index = 30
stimparam4.enable = True
stimparam4.trigger_key = 1
stimparam4.nb_pulse = 1
stimparam4.polarity = StimPolarity.NegativeFirst
stimparam4.phase_duration1 = 400
stimparam4.phase_duration2 = 400
stimparam4.phase_amplitude1 = 4.0
stimparam4.phase_amplitude2 = 4.0
```

```
# Create a list of stimparams
stimparams = [stimparam1, stimparam2, stimparam3, stimparam4]
```

```
loader = StimParamLoader(
    stimparams, intan=None, verbose=False
) # Not connecting to Itan. Otherwise, book and set Intan=IntanSoftware()
```

```
# Preview the stimparams with an interactive plot
loader.preview_parameters(colorblind=False)
```

```
# Enabling all stimparams
loader.enable_all()
```

```
# Book the system before enabling the stimaprams
loader.send_parameters()
```

```
# Disable all stimparams
loader.disable_all_and_send()
```

```
# Show the log
loader.get_log()
```

Parameter Scanner:

This utility allows you to explore different stimulation settings on a MEA to find optimal parameters. To install, run the command below:

```
pip install git+https://github.com/OrganaSys-np/np-utils.git#egg=np.utils[SSN]
```

Usage:

Follow these steps:

1. Imports

```
from neuroplatform import Experiment, StimPolarity
from np_utils.stim_scan import StimParamGrid, StimScan, MEAType
```

2. Defining a Parameter Grid

```
grid = StimParamGrid(
    amplitudes=[1, 2, 3],
    durations=[100, 200],
    polarities=[StimPolarity.NegativeFirst, StimPolarity.PostiveFirst],
    mea_type=MEAType.MEA32,
)
print(grid.total_combinations())
grid.display_grid()
```

3. Running the scan

```
EXPERIMENT = Experiment("...") # Use your token here
scan = StimScan(
    fs_experiment=EXPERIMENT,
    parameter_grid=grid,
    delay_btw_stim=1,   # Delay between stimulations in seconds
    delay_btw_channels=5,   # Delay between channels in seconds
    repeats_per_channel=5,   # number of repeats per channel for a single parameter set
    scan_channels=range(
        32, 64
    ),   # channels to scan, here MEA2. Must match electrodes allowed by your token
)
```

4. *Optional – Displaying duration of the scan before running it*

```
scan.get_scan_duration()
# Example of outcome:
# Predicted duration of the scan: 0h 6m 41s
```

5. Confirm

```
scan.run()
```

Results:

To display the list of all stimulations and associated parameters, use:

```
stim_df = scan.get_stimulation_parameter_history()
stim_df
```

And to show all events after the stimulation, the command is:

```
scan.plot_all_stims(
   s_before=1, s_after=2.5
)   # arguments define how long to show before the first stim and after the last respectively
```