CICADA

Release 1.0.0

Cossart lab

Jul 04, 2023

GETTING STARTED

CHAPTER

ONE

CALCIUM IMAGING PIPELINE

CHAPTER

TWO

CONTENTS

2.1 Dependencies

PyCICADA has the following minimum requirements, which must be installed before you can get started using PyNWB.

1. Python 3.6, or 3.7
2. pip

PyCICADA has been tested on Ubuntu 18.04.1 LTS, Windows 10 and macOS Mojave, using Python 3.6 & 3.7

2.2 Installation

2.2.1 Install release from PyPI or from gitlab

1- Download cicada updated requirements from here, open an anaconda prompt / command prompt and change directories to where the cicada_updated_requirements is

2- Create an environment and install cicada dependencies:

```
conda create -n cicada_env python=3.6.7
conda activate cicada_env
conda install shapely
conda install -c conda-forge fa2
pip install -r cicada_updated_requirements.txt
pip install pandas==0.24.2
pip install seaborn==0.9.0
```

3a- To install or update PyCICADA distribution from PyPI (not recommended, this version needs updates) run:

```
pip install -U pycicada
```

3b- To install the latest version CICADA from gitlab (recommended) run:

```
pip install git+https://gitlab.com/cossartlab/cicada.git
```

2.2.2 Follow the latest updates

CICADA is under active development in the lab, if you have already installed CICADA and want to be sure you are using the latest version of the code, run the following:

```
conda activate cicada_env
pip uninstall pycicada
pip install git+https://gitlab.com/cossartlab/cicada.git
```

2.3 How to run

To run CICADA from PyPI or gitalb installation execute :

```
conda activate cicada_env
python -m cicada
```

2.4 Download cicada codes

- To download the codes, clone CICADA:

```
cd 'path_to_where_to_clone'
git clone https://gitlab.com/cossartlab/cicada.git
```

To run CICADA from the gitlab clone:

run the '__main__.py' file from 'path_to_where_to_clone/cicada/src/cicada/__main__.py'

- Option 1 : run (in a conda prompt)

```
conda activate cicada_env
cd 'path_to_where_to_clone/cicada/src/cicada
conda-develop 'path_to_where_to_clone/cicada/src
python __main__.py
```

- Option 2 : example given for PyCharm users

1/ Create a Python interpreter from an existing conda environment selecting the python.exe in the cicada_env folder from the 'envs' folder:

> Go to settings, show all interpreters, click on '+', select 'Conda Environment', 'Existing environment', select the python interpreter (python.exe from cicada_env)

2/ Add 'path_to_where_to_clone/cicada/src' to the interpreter paths:

> Go to settings, show all interpreters, select the one just created, click on the folders logo ('Show paths for the selected interpreter'), click on '+', select the 'src' folder from cicada

3/ Create a new configuration to execute the '__main__.py' file:

> In script path select : 'path_to_where_to_clone/cicada/src/cicada/__main__.py' for the python interpreter select the one added before

4/ Run

2.5 Introduction

CICADA stands for Calcium Imaging Complete Automated Data Analysis.

It's a Python pipeline aimed for analyzing calcium imaging data.

2.5.1 Motivation

We notice a lack of toolboxes or analysis pipelines for calcium imaging data, using open source language. Our motivation was to build an easy-to-use pipeline, which doesn't need programming skills. In that purpose, we offer a Graphical User Interface as well as a command line usage.

In order to tackle the broadness of scientists' needs, we built the pipeline to be easily extendable with a plugin-like interface, for data format and analyses.

2.5.2 Supported data format

The default data format chosen was Neurodata Without Borders: Neurophysiology (NWB:N). that is a data standard for neurophysiology, providing neuroscientists with a common standard to share, archive, use, and build analysis tools for neurophysiology data. NWB:N is designed to store a variety of neurophysiology data, including data from intracellular and extracellular electrophysiology experiments, data from optical physiology experiments, and tracking and stimulus data.

We provide some tools to convert their data into NWB, with a plugin-like interface. (Curently in Preprocessing)

2.5.3 Functionalities

- We offer a Graphical User Interface (GUI) as well as a command line usage.
- We have already implemented or aim to implement various kind of analyses such cell assemblies detection, network analyses, display functions (rasters, cells map) etc…
- The GUI offers specifics functionalities:
  - Subjects and recorded sessions can be filtered or grouped in order to easily select the data to be analysed, and saved for future use.
  - Subjects and and recorded sessions' informations can be displayed and modified depending on the data format.
  - Analyses are grouped by family. Each analysis provide a way to check the data see if they are compatible.
  - Analyses arguments can be saved in a yaml file allowing to easily load them later.
  - A multi-thread implementation allows multiple analysis at the same time.
  - A progression bar allows to follow the analysis current status.

2.6 Software architecture

2.6.1 Analysis

Each analysis is represented by a class that inherits CicadaAnalysis (in the module cicada.analysis).

Each CicadaAnalysis instance instanciate an ArgumentAnalysisHandler that communicates with the GUI and handle the arguments that will be passed to the analysis.

Each CicadaAnalaysis instance allows to check the compatibility of the data to analyze though the method `check_data()`.

The method `run_analysis()` will be called to start the analysis.

The abstract class CicadaAnalysisFormatWrapper allows,through inheritance, to write a wrapper for a given data format, such as nwb. It allows to get the content from specific data format, and adding a new format consists in creating a new instance of CicadaAnalysisFormatWrapper, without any change in the CicadaAnalysis instances.

2.6.2 Graphical User Interface

The GUI uses Qt through QtPy which wrapps PyQt5 and PySide.

The GUI is composed of 3 main modules.

1. A list that displays the loaded recorded sessions to analyse (SessionsWidget).
2. A tree that displays the available analyses, updated depending on the data to analyse. (AnalysisTreeApp)
3. A pop-up panel that allows to set arguments for the given analysis and to follow the status of the analysis. (AnalysisParametersApp). A corresponding overwiew of the opened panels is available on the main window (AnalysisOverview)

2.7 Data format

In order to use CICADA, you will need to have data in a format compatible with the wrapper provided or that you have written (instance of CicadaAnalysisFormatWrapper).

We provide a wrapper for NWB format so far. We will explain below how our NWB files are organized.

We also provide some scripts that allows to convert your data to NWB. You might have to write some code so it can fit to your data format. This pre-processing module provides a plugin-like interface.

It's a Python pipeline aimed for analyzing calcium imaging data.

2.7.1 NWB Format

NWB stands for Neurodata Without Borders: Neurophysiology (NWB:N).

NWB is organized in different modules.

So far, here are the kind of data we have included in the NWB files:

- **Calcium imaging movies**: the movie is added in acquisitions as a TwoPhotonSeries instance. It is possible to either save the movie as an external file (keeping only the reference to the file path and name) or keeping the whole data. You can either add only the motion corrected movie or both the original and corrected one. If the xy translations from the correction are available, they can be added as a CorrectedImageStack instance.

- **ABF file**: abf allows to extract timestamps and to synchronize our different acquisitions (calcium imaging data, behavior movies, speed on treadmill, LFP signal, piezo). Timestamps are saved as TimeSeries instances in the acquisition module of NWB
- **ROIs**: ROIs are instances of ImageSegmentation added in our case as a processing module called "ophys". But the name could be changed as the wrapper will look for ImageSegmentation in all processing modules.
- **Cell type**: Cell type is added in RoiResponseSeries of the Fluorescence instance containing in the field control and control_descrption. Control is a 1d array (uint8) containing a code for each ROIs (each code represent a cell type) Control_description is a list of string (the length equals the number of cells), the string represents the cell type description.
- **Neuronal activity**: We have different type of neuronal activity.
  - **Raw fluorescence signal**: Added as RoiResponseSeries in an instance of Fluorescence itself in the data_interface list of NWB.
  - **Cell activity**: what we call raster dur, is a binary matrix (n_cells * n_frames), and indicate for any given cell at any given frame if the cell is active or not (corresponding to the rise time of fluorescence signal). The same format can be used to record spikes inference, a 1 indicating a spike.
- **Behavior movies**: are added as ImageSeries in the NWB acquisitions. To save memory, we only keep the external reference to the file. The name of our behaviors movies are such as f'behavior_cam_{cam_id}'. The timestamps of each frame are available as TimeSeries such as described in the abf file section.
- **Behavior Epochs**: (time intervals) are recorded as BehavioralEpochs added in NWB data interfaces.
- **Invalid times**: Invalid times are added as Invalid time intervals in the NWB.
- **Other Epochs**: Are added as TimeIntervals using the create_time_intervals method.

2.7.2 NWB pre-processing

The pre-processing procedure aims at converting your data in the NWB format.

To do so we explore directories, each session data is contained in a given directory that will also contain a set of yaml files containing metadata and some configuration options.

To run the pre-processing you have to call the function convert_data_to_nwb() in cicada.preprocessing.cicada_data_to_nwb, it takes 3 arguments:

- data_to_convert_dir (str): Absolute path to the directory containing all data
- default_convert_to_nwb_yml_file (str): Absolute path to the default YAML file to convert an NWB file
- nwb_files_dir (str): Absolute path to the directory where to save the nwb file created

In order to run through the directories that contains data to be converted, you can use the function find_dir_to_convert() in from cicada.preprocessing.cicada_preprocessing_main which return a list of directories contains at least a file for each given keywords with a specific extensions. In our case, we are using the keywords [["session_data"], ["subject_data"]] with yaml extensions.

In each session directory, you should have two yaml files that will contain the metadata one regarding the subject, the other the session. We will put some examples online soon.

Then the pre-processing is run according to the yaml configuration file that is passed as argument in convert_data_to_nwb(). A default one is given in cicada.preprocessing with name pre_processing_default.yaml

You can make your own to fit your data.

Each step of the process is creating an instance of an implementation classes that inherit from ConvertToNWB (see in convert_to_nwb.py). You can create your own implementations.

Those classes are called in the order indicated in the yaml file in the order list. The py file names that contains the classes should respect a strict rule with only lower case and an underscore (_) added before a previously upper case letter. As example, ConvertBehaviorMovieToNWB will be in a file named convert_behavior_movie_to_nwb.py

Then for each instance of ConvertToNWB inherited classes that will be used, you need to indicate which arguments will be pass to the convert() methods.

Each entry of the yaml is the name of a class except for the entries order. So when the yaml will be read, we will get a dictionary with all those entries as keys.

First of all, if you want the class to be run for several different types of inputs, you can put the configuration for the class in a dict with key being numerical values (1, 2, etc…).

Then for each class, the entry in the yaml represents an argument (we will get it via the kwargs.get(arg_name) in the convert() method). Depending on the data, here are the sub-fields:

- **Files**:
  - keyword: list of keywords, only the files with those keywords will be selected
  - keyword_to_exclude: list of keywords, only files with non of those keywords will be returned.
  - dir: if given, means the file will be searched in this/those directory(ies)
  - extension: only files with one of those extensions will be returned
  - value: if file with .mat or .npz files, if a value is given, then the field with value as key is passed as argument instead of the file name.
- **Values**:
  - value: if no keywords or extensions are given, the content of value will given to the argument.
- **Values from other classes**:
  - from_other_converter: indicates from which other instance of ConvertToNWB we want to extract a value.
  - value: name of the attribute from the other converter that we want to be put in the argument. The other converter need to be called before the current one.

With this yaml file you can then create your NWB file step by step from each kind of data you have.

2.8 How to use

To get started with CICADA, you can download some of the NWB files from here on DANDI Archive or use your own NWB files.

Launch CICADA GUI

A first window named 'CICADA initial config' will open click on 'Run Cicada' or just press Enter.

A second window 'Cicada - CI_DATA' will open: this is CICADA main window. From here you can i) load the NWB files to analyse, ii) select the analysis to run, iii) open the results.

This main window is made of 3 panels.

- Let panel:

Contains the list of the NWB files to analyse. To populate this list click on 'file', 'Open new dataset…' and select the folder containing the NWB.

From here it is possible to select the files to include in the analysis (to help in the selection we have a sort option)

After NWB selection click on the cicada to go to the middle panel.

- Middle panel:

Contains the list of analysis implemented (some still need to be implemented) that are accessible. Based on the NWB files selected the analysis that can be performed turned white, others remain in grey.

Select one analysis and click on the cicada. A pop-up window will open.

- Pop-up window:

In this window the user is asked to set the parameters that will be used to run the selected analysis. Of note the last parameter is the path to the saving folder and need to be adjusted otherwise the analysis won't run.

Once everything is set, click on 'Run analysis'

Once the analysis is done, ckick on the home icon, it will display the main window.

- Right panel:

From the rigth panel of CICADA main window click on 'Open results folder' to access the results.

2.9 Pre-processing

Convert CI data to NWB file format

2.9.1 Data to NWB

cicada.preprocessing.cicada_data_to_nwb.**convert_data_to_nwb**(*data_to_convert_dir*, *default_convert_to_nwb_yml_file*, *nwb_files_dir*)

Convert all default_config_data_for_conversion located in dir_path and put it in NWB format then create the file. Use the yaml file contains in dir_path to convert the default_config_data_for_conversion. A yaml file with in its name session_data and one with subject_data must be in directory. Otherwise nothing will happend. A yaml file with abf in its name will need to be present to convert the abf default_config_data_for_conversion.

Parameters

- **data_to_convert_dir** (*str*) – Absolute path to the directory containing all data
- **default_convert_to_nwb_yml_file** (*str*) – Absolute path to the default YAML file to convert an NWB file
- **nwb_files_dir** (*str*) – Absolute path to the directory where to save the nwb file created

cicada.preprocessing.cicada_data_to_nwb.**create_convert_class**(*class_name*, *config_dict*, *converter_dict*, *nwb_file*, *yaml_path*, *files*, *dir_path*)

Parameters

- **class_name** –

- **config_dict** –
- **converter_dict** –
- **nwb_file** –
- **yaml_path** –
- **files** –
- **dir_path** –

Returns:

cicada.preprocessing.cicada_data_to_nwb.**create_nwb_file**(*subject_data_yaml_file*, *session_data_yaml_file*)

Create an NWB file object using all metadata containing in YAML file

Parameters

- **subject_data_yaml_file** (*str*) – Absolute path to YAML file containing the subject metadata
- **session_data_yaml_file** (*str*) – Absolute path to YAML file containing the session metadata

cicada.preprocessing.cicada_data_to_nwb.**filter_list_according_to_keywords**(*list_to_filter*, *keywords*, *keywords_to_exclude*)

Conditional loop to remove all files or directories not containing the keywords # or containing excluded keywords. Inplace list modification

Parameters

- **list_to_filter** (*list*) – List containing all files/directories to be filtered (full_path)
- **keywords** (*str*) – If the list doesn't contain the keyword, remove it from list
- **keywords_to_exclude** (*str*) – If the list contains the keyword, remove it from list

Exemples:

```
>>> print(filter_list_of_files(["file1.py", "file2.c", "file2.h"],"2","h"))
["file2.c"]
```

cicada.preprocessing.cicada_data_to_nwb.**filter_list_of_files**(*dir_path*, *files*, *extensions*, *directory=None*)

Take a list of file names and either no extensions (empty list or None) and remove the directory that starts by "." or a list of extension and remove the files that are not with this extension. It returns a new list with full paths

Parameters

- **dir_path** (*str*) – path in which the files ares
- **files** (*list*) – List of files to be filtered
- **extensions** (*str*) – File extension to use as a filter
- **directory** (*in which looking for files with a given extensions or return files in this*) – directory keyword, allows to identify directories
- **directory** –

Exemples:

```
>>> print(filter_list_of_files(["file1.py", "file2.c", "file3.h"],"py"))
["file1.py"]
```

2.9.2 Run preprocessing

2.9.3 NWB file class

class cicada.preprocessing.convert_to_nwb.**ConvertToNWB**(*nwb_file*)
NWB file object class

convert(***kwargs*)
Convert the data and add to the nwb_file

Parameters ****kwargs** – arbitrary arguments

2.9.4 Suite 2P ROIs

2.9.5 2D series

class cicada.preprocessing.convert_processed_2d_series_to_nwb.**ConvertProcessed2dSeriesToNWB**(*nwb_file*)
Class to convert 2D series to NWB

convert(***kwargs*)
Convert the data and add to the nwb_file

Parameters ****kwargs** – arbitrary arguments

static load_rasterplot_in_memory(*rasterplot_file_name*, *matlab_string*, *frames_to_add=None*)
Get 2D series data from file

Parameters

- **rasterplot_file_name** (*str*) – Absolute path to file containing 2D series
- **matlab_string** (*str*) – Key to the data from matlab and npz files
- **frames_to_add** (*dict*) – Key is the frame where you add blank frames and value is the number of frames to add
- **None.** (*Default is*) –

Returns Raster data as a 2d array

Return type raster (np.array)

2.9.6 Calcium imaging movie

class cicada.preprocessing.convert_ci_movie_to_nwb.**ConvertCiMovieToNWB**(*nwb_file*)
Class to convert Calcium Imaging movies to NWB

convert(***kwargs*)
Convert the data and add to the nwb_file

Parameters ****kwargs** – arbitrary arguments

2.9.7 ABF

class cicada.preprocessing.convert_abf_to_nwb.**ConvertAbfToNWB**(*nwb_file*)

Class to convert ABF data to NWB

convert(***kwargs*)

The goal of this function is to extract from an Axon Binary Format (ABF) file its content and make it accessible through the NWB file. The content can be: LFP signal, piezzo signal, speed of the animal on the treadmill. All, None or a few of these informations could be available. One information always present is the timestamps, at the abf sampling_rate, of the frames acquired by the microscope to create the calcium imaging movie. Such movie could be the concatenation of a few movies, such is the case if the movie need to be saved every x frames for memory issue for ex. If the movie is the concatenation of many, then there is an option to choose to extract the information as if 2 frames concatenate are contiguous in times (such as then LFP signal or piezzo would be match movie), or to add interval_times indicating at which time the recording is on pause and at which time it's starting again. The time interval containing this information is named "ci_recording_on_pause" and you can get it doing: if 'ci_recording_on_pause' in nwb_file.intervals: pause_intervals = nwb_file.intervals['ci_recording_on_pause']

Parameters

- ****kwargs** (*dict*) – kwargs is a dictionary, potentials keys and values types are:
- **abf_yaml_file_name** – mandatory parameter. The value is a string representing the path
- **abf** (*and file_name of the yaml file associated to this abf file. In the*) –
- **frames_channel** – mandatory parameter. The value is an int representing the channel
- **data.** (*of the abf in which is the frames timestamps*) –
- **abf_file_name** – mandatory parameter. The value is a string representing the path
- **file.** (*and file_name of the abf*) –

determine_ci_frames_indices()

Using the frames data channel, estimate the timestamps of each frame of the calcium imaging movie. If there are breaks between each recording (the movie being a concatenation of different movies), then there is an option to either skip those non registered frames that will be skept in all other data (lfp, piezzo, ...) or to determine how many frames to add in the movie and where so it matches the other data recording in the abf file

double_sign_transi(*chan1*, *chan2*, *thr=1*)

Extract signed ticks from two signals.

Parameters

- **chan1** (*npt.NDArray[np.float64]*) – First analog signal
- **chan2** (*npt.NDArray[np.float64]*) – Second analog signal
- **thr** (*float*) – Value for the transition detections

Returns The signed transitions

Return type npt.NDArray[np.int64]

position_ticks(*chan1*, *chan2*, *thr=1*)

Computes the position transitions.

Parameters

- **chan1** (*npt.NDArray[np.float64]*) – First position channel

- **chan2** (*npt.NDArray[np.float64] | None, optional*) – Second channel if present
- **thr** (*float*) – Crossing threshold value

Returns The transition array

Return type npt.NDArray[np.int64]

2.9.8 Utils

class cicada.preprocessing.utils.**ComparableItem**(*value*)
Make it possible to sort a list of items of different types, such as int and string

cicada.preprocessing.utils.**class_name_to_module_name**(*class_name*)
Transform the string representing a class_name, by removing the upper case letters, and inserting before them an underscore if 2 upper case letters don't follow. Underscore are also inserted before numbers ex: ConvertAbfToNWB -> convert_abf_to_nwb :param class_name: string :return:

cicada.preprocessing.utils.**flatten**(*list*)
Flatten a nested list no matter the nesting level

Parameters **list** (*list*) – List to flatten

Returns List without nest

Examples

```
>>> flatten([1,2,[[3,4],5],[7]])
[1,2,3,4,5,7]
```

cicada.preprocessing.utils.**get_continous_time_periods**(*binary_array*)
take a binary array and return a list of tuples representing the first and last position(included) of continuous positive period :param binary_array: :return:

cicada.preprocessing.utils.**load_tiff_movie_in_memory**(*tif_movie_file_name*, *frames_to_add=None*)
Load tiff movie from filename using Scan Image Tiff

Parameters **tif_movie_file_name** (*str*) – Absolute path to tiff movie

Returns Tiff movie as 3D-array

Return type tiff_movie (array)

cicada.preprocessing.utils.**load_tiff_movie_in_memory_using_pil**(*tif_movie_file_name*, *frames_to_add=None*)
Load tiff movie from filename using PIL library

Parameters

- **tif_movie_file_name** (*str*) – Absolute path to tiff movie
- **frames_to_add** – dict with key an int representing the frame index after which add frames. the value is the number of frames to add (integer)

Returns Tiff movie as 3D-array

Return type tiff_movie (array)

cicada.preprocessing.utils.**merging_time_periods**(*time_periods*, *min_time_between_periods*)
Take a list of pair of values representing intervals (periods) and a merging thresholdd represented by min_time_between_periods. If the time between 2 periods are under this threshold, then we merge those periods. It returns a new list of periods. :param time_periods: list of list of 2 integers or floats. The second value represent the end of the period, the value being included in the period. :param min_time_between_periods: a float or integer value :return: a list of pair of list.

cicada.preprocessing.utils.**module_name_to_class_name**(*module_name*)
Transform the string representing a module_name, by removing underscores, , and transforming as upper cases the following letter. ex: convert_abf_to_nwb -> ConvertAbfToNwb :param module_name: string :return:

cicada.preprocessing.utils.**sort_by_param**(*nwb_path_list*, *param_list*)
Sort NWB files depending on a list of parameters

Parameters

- **nwb_path_list** (*list*) – List of absolute path to NWB files
- **param_list** (*list*) – List of parameters to sort by

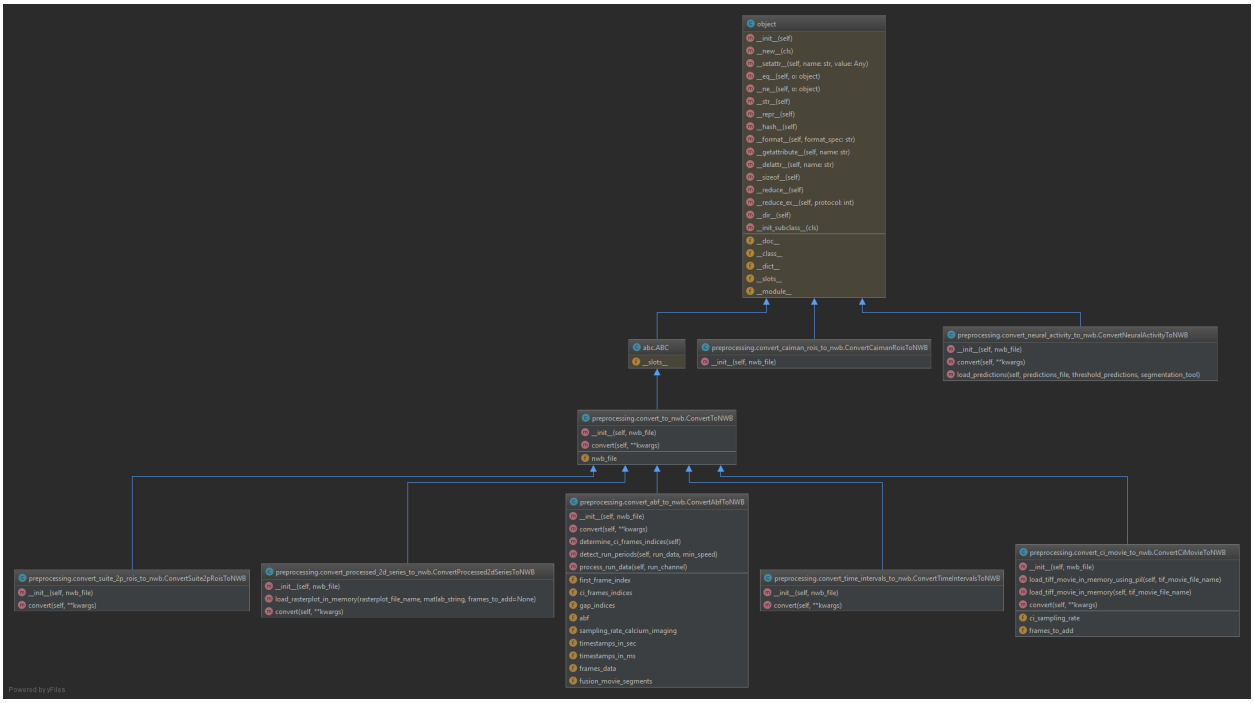
Returns List of NWB files sorted

Return type nwb_sorted_list (list)

cicada.preprocessing.utils.**update_frames_to_add**(*frames_to_add*, *nwb_file*, *ci_sampling_rate*)
Update frames_to_add (dict), based on pause_intervals and ci_frames_time_series :param frames_to_add: dict, with key an int representing the frame index after which add frames. :param the value is the number of frames to add: :type the value is the number of frames to add: integer :param nwb_file: nwb file, will get nwb_file.intervals['ci_recording_on_pause'] and :param nwb_file.get_acquisition: :type nwb_file.get_acquisition: "ci_frames"

Returns:

2.9.9 Appendix



2.10 Analysis

Analyses

2.10.1 Main analysis class

class cicada.analysis.cicada_analysis.**CicadaAnalysis**(*name*, *short_description*, *accepted_data_formats*, *family_id=None*, *long_description=None*, *data_to_analyse=None*, *data_format=None*, *config_handler=None*, *gui=True*)

An abstract class that should be inherit in order to create a specific analyse

add_analysis_description_for_gui(*description*)

Add a description that will be displayed as a widget on top of the arguments list :param description:

Returns:

add_argument_for_gui(*with_incremental_order=True*, ***kwargs*)

Parameters

- ****kwargs** –
- **with_incremental_order** – boolean, if True means the order of the argument will be the same as when added

Returns:

add_bool_option_for_gui(*arg_name*, *true_by_default*, *short_description*, *long_description=None*, *family_widget=None*)

Add an option which is a boolean to the menu :param arg_name: (string) used to get back the value :param true_by_default: (bool) :param short_description: (str) :param long_description: (string or None)

Returns:

add_choices_for_groups_for_gui(*arg_name*, *choices*, *with_color*, *short_description*, *long_description=None*, *family_widget=None*, *mandatory=False*, *add_custom_group_field=False*, *custom_group_validation_fct=None*, *custom_group_processor_fct=None*)

Allows to create group based composed of elements of choices.. :param arg_name: :param choices: :param with_color: If True, means that we can select a color for each group :param short_description: :param long_description: :param family_widget: :param mandatory: (bool) :param add_custom_group_field: if True, add a text field allowing to name the group to be added :param custom_group_validation_fct: if not None, called when a custom group is created to check the text :param field and validate the name. Return True or False. If False: :param the field is erased. Instead of False: :param can: :param return a string: :param in that case a message box will display a message indicating the error: :param custom_group_processor_fct: None or function that process the text in the custom group field and return :param a list of string representing the elements in the group. If this fct is None: :param then the new group will: :param just contain one element the content of the text field that can be then processed later on.:

Returns:

add_ci_movie_arg_for_gui(*long_description=None*)

Will add an argument for gui, named ci_movie that will list all calcium imaging available for each session Returns:

add_dpi_arg_for_gui(*family_widget*)
Add dpi value :param family_widget:

Returns:

add_field_float_option_for_gui(*arg_name*, *default_value*, *mandatory*, *short_description*, *long_description=None*, *family_widget=None*)
Add a field to add some float Returns:

add_field_text_option_for_gui(*arg_name*, *default_value*, *short_description*, *long_description=None*, *family_widget=None*)
Add a field to add some text Returns:

add_image_format_package_for_gui()
Add a few arguments at once: save_formats, dpi, width_fig & height_fig Returns:

add_open_dir_dialog_arg_for_gui(*arg_name*, *mandatory*, *short_description*, *long_description=None*, *key_names=None*, *family_widget=None*)
Allows to add widget to select a directory. If key_names is given then multiple option are added, one for each key_name :param arg_name: :param mandatory: (boolean) :param short_description: :param long_description: :param key_names: None or list of string :param family_widget:

Returns:

add_open_file_dialog_arg_for_gui(*arg_name*, *extensions*, *mandatory*, *short_description*, *long_description=None*, *key_names=None*, *family_widget=None*)
Allows to add widget to select a file. If key_names is given then multiple option are added, one for each key_name :param arg_name: :param extensions: str or list of str, ex: "txt" :param mandatory: (boolean) :param short_description: :param long_description: :param key_names: None or list of string :param family_widget:

Returns:

add_save_formats_arg_for_gui(*family_widget=None*)
Add save_formats option, True or False Returns:

add_segmentation_arg_for_gui()
Will add an argument for gui, named segmentation that will list all segmentations available for each session Returns:

add_verbose_arg_for_gui()
Add verbose option, True or False Returns:

add_with_timestamp_in_filename_arg_for_gui()
Add with_timestamp_in_file_name option, True or False Returns:

abstract check_data()
Check the data given one initiating the class and return True if the data given allows the analysis implemented, False otherwise. :return: a boolean

abstract copy()
Make a copy of the analysis Returns:

create_results_directory(*dir_path*)
Will create a directory in dir_path with the name of analysis and time at which the directory is created so it can be unique. The attribute _results_path will be updated with the path of this new directory :param dir_path: path of the dir in which create the results dir

Returns: this new directory

get_data_identifiers()
Return a list of string representing each data to analyse Returns:

get_data_to_analyse()

Returns a list of the data to analyse

get_results_path()
Return the path when the results from the analysis will be saved or None if it doesn't exist yet Returns:

is_data_format_accepted(*data_format*)

Parameters **data_format** –

Returns: boolean, True is the data format is accepted for this analysis

abstract run_analysis(***kwargs*)
Run the analysis :param kwargs: :return:

set_arguments_for_gui()
Need to be implemented in order to be used through the graphical interface. super().set_arguments_for_gui() should be call first to instantiate an AnalysisArgumentsHandler and create the attribution for results_path :return: None

set_data(*data_to_analyse*)
A list of :param data_to_analyse: list of data_structure

transfer_attributes_to_tabula_rasa_copy(*analysis_copy*)
Transfers attributes values from self to a new copy that doesn't have all information yet :param analysis_copy:

Returns:

update_original_data()
To be called if the data to analyse should be updated after the analysis has been run. :return: boolean: return True if the data has been modified

update_progressbar(*time_started*, *increment_value=0*, *new_set_value=0*)

Parameters

- **time_started** (*float*) – Start time of the analysis
- **increment_value** (*float*) – Value that should be added to the current value of the progress bar
- **new_set_value** (*float*) – Value that should be set as the current value of the progress bar

2.10.2 Argument handler

class cicada.analysis.cicada_analysis_arguments_handler.**AnalysisArgumentsHandler**(*cicada_analysis*)
Handle the AnalysisArgument instances for a given CicadaAnalysis instance. Allows to create the widgets and get the values to pass to run_analysis() of the CicadaAnalysis instance.

add_argument(***kwargs*)

Parameters ****kwargs** –

Returns:

check_arguments_validity()
Check if all mandatory arguments have been filled Returns: True if we can run the analysis

get_analysis_argument(*arg_name*)

Parameters **arg_name** –

Returns:

get_analysis_arguments(*sorted=False*)

Parameters **sorted** –

Returns:

get_gui_widgets(*group_by_family=False*)
Get the list of widgets necessary to fill the arguments for the cicada analysis associated :param group_by_family: if True, group the widgets in one widget to be grouped together if they belong to the same :param family. AnalysisArgument will have an attribute named family_widget whose value is a string.:

Returns:

load_analysis_argument_from_yaml_file(*file_name*)
Set the analysis argument value based on the value in the yaml file. The :param file_name:

Returns:

save_analysis_arguments_to_yaml_file(*path_dir*, *yaml_file_name*)
Save the arguments value to a yaml file. The first key will represent the argument name then the value will be a dict with the argument details such as the type etc… :param path_dir: directory in which save the yaml file :param yaml_file_name: yaml file name, with the extension or without (will be added in that case)

Returns:

set_argument_value(*arg_name*, ***kwargs*)
Set an argument values, will be use to run analysis :param arg_name: :param **kwargs:

Returns:

set_widgets_to_default_value()
Set the widgets to the default value of their AnalysisArgument Returns:

2.10.3 Format wrapper

class cicada.analysis.cicada_analysis_format_wrapper.**CicadaAnalysisFormatWrapper**(*data_ref*, *data_format*, *load_data=True*)
An abstract class that should be inherit in order to create a specific format wrapper Two constants should be added on the classe WRAPPER_ID and DATA_FORMAT

property data_ref
Return the path of the file or directory corresponding to this data session :return:

static grouped_by()
Indicate by which factor the data can be grouped ex: age, species, contains behavior, categories of age.. :return: a dictionary with key the name of the group (str) that will be displayed in the GUI, and as value the a sring representing either an argument or a method of the wrapper class (hasattr() and callable() will be use to get them)

abstract property identifier
Identifier of the session :return:

abstract static is_data_valid(*data_ref*)
Check if the data can be an input for this wrapper as data_ref :param data_ref: file or directory

Returns: a boolean

load_data()
Load data in memory Returns:

2.10.4 NWB wrapper

class cicada.analysis.cicada_analysis_nwb_wrapper.**CicadaAnalysisNwbWrapper**(*data_ref*, *load_data=True*)

WRAPPER_ID = 'NWB_CI'
Allows to communicate with the nwb format

abf_status()
Using neuronal data and timestamps to infer if the abf has been used to create the NWB :return:

property age
Age of the subject (int), in days or months :return: None if age unknown, int otherwise, months or days is found with age_unit

property age_unit
Unit for the animal age :return:'D' for days, 'M' for months

cell_types_status()
Return the cells types in a string, or a message saying no cell types :return:

contains_ci_movie(*consider_only_2_photons*)
Indicate if the data object contains at least one calcium imaging movie represented by an instance of pynwb.image.ImageSeries :param consider_only_2_photons: boolean, it True means we consider only 2 photons calcium imaging movies, :param if other exists but not 2 photons: :param then False will be return:

Returns: True if it's the case, False otherwise

property genotype
Genotype of the subject :return: None if age unknown

get_acquisition_names()
Returns:

get_all_cell_types()
Return a list of all cell types identified in this session. If the list is empty, it means the type of the cells is not identified Returns:

get_behavior_movies(*key_to_identify='behavior'*)
Return a dict with as key a string identifying the movie, and as value a dict of behavior movies a string as file_name if external, or a 3d array :param key_to_identify: string, key to identify that a movie is a behavior movie

Returns:

get_behavioral_epochs_names()
The name of the different behavioral Returns:

get_behavioral_epochs_times(*epoch_name*)
Return an interval times (start and stop in seconds) as a numpy array of 2*n_times. :param epoch_name: Name of the interval to retrieve

Returns: None if the interval doesn't exists or a 2d array

get_behavioral_events_names()
The name of the different behavioral Returns:

get_behavioral_events_times(*event_name*)
The name of the different behavioral Returns:

get_behavioral_time_series_names()
The name of the different behavioral Returns:

get_behaviors_movie_time_stamps()
return a dict with key the cam id and value np.array with the timestamps of each frame of the behavior movie return None if non available Returns:

get_cell_indices_by_cell_type(*roi_serie_keys*)
Return a dict with key the cell_type name and value an array of int representing the cell indices of this type :param roi_serie_keys:

Returns:

get_ci_movie_sampling_rate(*only_2_photons=False*, *ci_movie_name=None*)

Parameters

- **only_2_photons** – if True only 2 photons one are considere
- **ci_movie_name** – (string) if not None, return the sampling rate for a given ci_movie, otherwise the first
- **found** (*one*) –

Returns: (float) sampling rate of the movie, return None if no movie is found

get_ci_movie_time_stamps()
return a np.array with the timestamps of each frame of the CI movie return None if non available Returns:

get_ci_movies(*only_2_photons*)
Return a dict with as key a string identifying the movie, and as value a dict of CI movies a string as file_name if external, or a 3d array :param only_2_photons: return only the 2 photon movies

Returns:

get_ci_movies_sample(*only_2_photons*, *n_frames=2000*)
Return a dict with as key a string identifying the movie, and as value a dict of CI movies a string as file_name if external, or a 3d array :param only_2_photons: return only the 2 photon movies :param n_frames: number of frames to take from movie to serve as data sample

Returns:

get_identifier(*session_data*)
Get the identifier of one of the data to analyse :param session_data: Data we want to know the identifier

Returns: A hashable object identfying the data

get_imaging_plan_location(*only_2_photons=False*, *ci_movie_name=None*)

Parameters

- **only_2_photons** – if True only 2 photons one are considere
- **ci_movie_name** – (string) if not None, return the sampling rate for a given ci_movie, otherwise the first
- **found** (*one*) –

Returns: (str) imaging plan location

get_interval_as_data_frame(*interval_name*)
Return an interval time as a pandas data frame. :param interval_name: Name of the interval to retrieve

Returns: None if the interval doesn't exists or a pandas data frame otherwise

get_interval_original_frames(*interval_name*)
Return an interval times (start and stop in frames) as a numpy array of 2*n_times. :param interval_name: Name of the interval to retrieve

Returns: None if the interval doesn't exists or a 2d array

get_interval_times(*interval_name*)
Return an interval times (start and stop in seconds) as a numpy array of 2*n_times. :param interval_name: Name of the interval to retrieve

Returns: None if the interval doesn't exists or a 2d array

get_intervals_names()
Return a list representing the intervals contains in this data Returns:

get_mouse_position_info()
Returns:

get_mouse_speed_info()
Returns:

get_opto_stimulation_time_serie()
Returns:

get_pixel_mask(*segmentation_info*)
Return pixel_mask which is a list of list of pair of integers representing the pixels coordinate (x, y) for each cell. the list length is the same as the number of cells. :param segmentation_info: a list of 3 elements: first one being the name of the module, then the name :param of image_segmentation and then the name of the segmentation plane.:

Returns:

get_roi_response_serie_data(*keys*)

Parameters **keys** – lsit of string allowing to get the roi repsonse series wanted

Returns:

get_roi_response_serie_data_by_keyword(*keys*, *keyword*)
Return a dict with other last key data :param keys: list of string allowing to get the roi response series in data_interfaces :param keyword:

Returns:

get_roi_response_serie_timestamps(*keys*, *verbose=True*)

Parameters

- **keys** – lsit of string allowing to get the roi repsonse series wanted

- **verbose** – print info

Returns:

get_roi_response_series(*keywords_to_exclude=None*)
param: keywords_to_exclude: if not None, list of str, if one of neuronal data has this keyword, then we don't add it to the choices

Returns: a list or dict of objects representing all roi response series (rrs) names rrs could represents raw traces, or binary raster, and its link to a given segmentation. The results returned should allow to identify the segmentation associated. Object could be strings, or a list of strings, that identify a rrs and give information how to get there.

get_roi_response_series_list(*keywords_to_exclude=None*)
param: keywords_to_exclude: if not None, list of str, if one of neuronal data has this keyword, then we don't add it to the list

Returns: a list with all roi response series (rrs) names

get_rrs_sampling_rate(*keys*)
Args:

Returns: (float) sampling rate of the movie, return None if no sampling rate is found

get_segmentations()
Returns: a dict that for each step till plane_segmentation represents the different option. First dict will have as keys the name of the modules, then for each modules the value will be a new dict with keys the ImageSegmentation names and then the value will be a list representing the segmentation plane

get_signal_by_keyword(*keyword*, *exact_keyword=False*)
Look for a signal with this keyword. Returns the first instance found that matches it. :param keyword: (str) :param exact_keyword: (bool) if True, the name of the TimeSeries representing the signal should be the same :param as keyword: :param otherwise keyword should be in the name of the TimeSeries:

Returns: a two 1d array representing a signal and its timestamps. If no sginal with this keyword found, return None, None

get_signal_keys()
Return a list of str representing the key of the signals available. Signals are TimeSeries instance registered as data interfaces in modules. Returns:

get_signals_info()
Returns: a dict that for each step till the TimeSeries name represents the different option. First dict will have as keys the name of the modules, then for each modules the value will be the name of the TimeSeries representing the signal

get_timestamps_range()
Return a tuple of float representing the first and last time stamp with movie recording (behavior or ci movie) Returns:

static grouped_by()
Indicate by which factor the data can be grouped ex: age, species, contains behavior, categories of age.. :return: a dictionary with key the name of the group (str) that will be displayed in the GUI, and as value the a sring representing either an argument or a method of the wrapper class (hasattr() and callable() will be use to get them)

property identifier
Identifier of the session :return:

static is_data_valid(*data_ref*)
Check if the data can be an input for this wrapper as data_ref :param data_ref: file or directory

Returns: a boolean

load_data()
Load data in memory Returns:

property session_id
Id of the subject :return: None if subject_id unknown

property sex
Sex (gender) of the subject :return: None if sex unknown

property species
Species of the subject :return: None if age unknown

property subject_id
Id of the subject :return: None if subject_id unknown

property weight
Id of the subject :return: None if weight unknown

2.10.5 Cells count

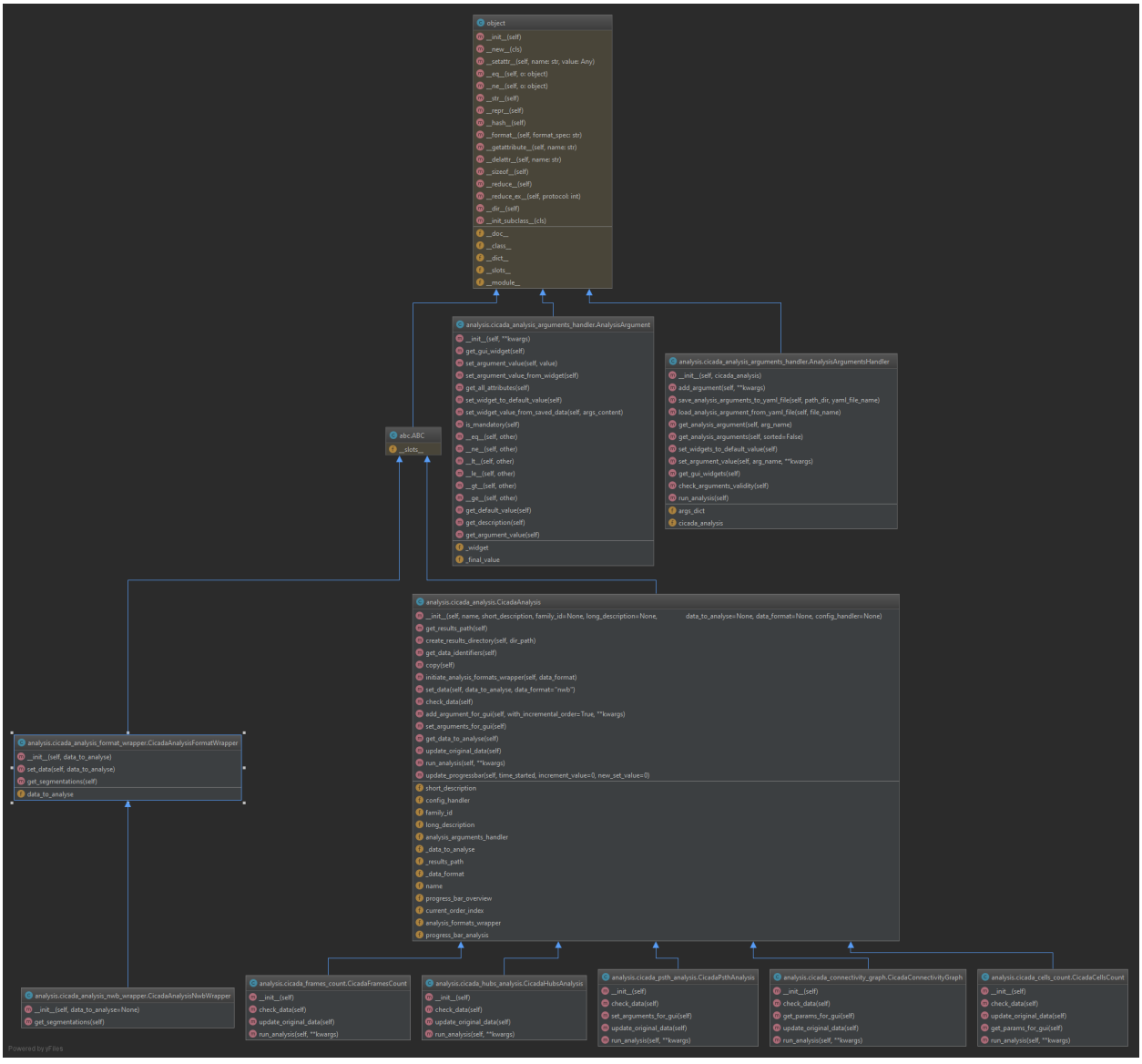
2.10.6 Connectivity graph

2.10.7 Frames count

2.10.8 Hubs analysis

2.10.9 PSTH analysis

2.10.10 Annexe



2.11 GUI

Graphic interface to launch analyses

2.11.1 Main Window

2.11.2 Filter/group sessions

2.11.3 Display metadata

2.11.4 Analysises list

class cicada.gui.cicada_analysis_tree_gui.**AnalysisTreeApp**(*parent*, *config_handler*, *analyses_instances*, *to_parameters_button=None*)

create_tree_model()
Create the tree model Returns: the tree model, an instance of QAnalysisTreeModel

doubleClickedItem(*idx*)
Method called when the user double click in the tree :param idx: Index of the branch clicked

Returns:

init_ui()
Set some of the elements used to build the tree Returns:

invalidate_all_items()
invalidate all items if the tree Returns:

load_arguments_parameters_section()
Used to load the parameters section with widgets, based on the current selection in the tree. If the selection is not on any valid tree item, nothing will happen Returns: None

set_data(*data_to_analyse*, *data_format*)
Give to the tree the data that the analysis classes will be given to analyze. Allows the tree to inactivate the analyses for which the data don't fulfill the requierements :param data_to_analyse: a list of data in a given format (could be nwb or other) :param data_format: format of the data, must be a string. So far only "nwb" is supported

Returns: None

class cicada.gui.cicada_analysis_tree_gui.**QAnalysisTreeModel**(*tree_item*, *parent=None*)

columnCount(*self*, *parent: QModelIndex = QModelIndex()*) → int

data(*index*, *role*)

Parameters

- **index** –
- **role** –

Returns

flags(*self*, *index: QModelIndex*) → Qt.ItemFlags

headerData(*self*, *section: int*, *orientation: Qt.Orientation*, *role: int = Qt.ItemDataRole.DisplayRole*) → Any

index(*self*, *row: int*, *column: int*, *parent: QModelIndex = QModelIndex()*) → QModelIndex

parent(*self*, *child: QModelIndex*) → QModelIndex
parent(*self*) → QObject

rowCount(*self*, *parent: QModelIndex = QModelIndex()*) → int

class cicada.gui.cicada_analysis_tree_gui.**QAnalysisTreeView**(*tree_item*, *config_handler*, *parent=None*)

drawBranches(*self*, *painter: QPainter*, *rect: QRect*, *index: QModelIndex*)

isIndexHidden(*q_model_index*)
Avoid to select the line that display the family name :param q_model_index: :return:

keyPressEvent(*event*)

Parameters **event** – P -> set background picture

Returns:

cicada.gui.cicada_analysis_tree_gui.**fill_tree_item_with_dict**(*root_tree*, *instances_dict*)
Recursive function that fills the root_tree according to data in instances_dict. :param root_tree: instance of TreeItem :param instances_dict: Contains instance of cicada_analysis, in a hierarchy similar to the one we want the tree to be :return:

2.11.5 Overview of analysises

class cicada.gui.cicada_analysis_overview.**AnalysisOverview**(*config_handler*, *parent=None*)
Class containing the overview linked to an analysis

add_analysis_overview(*cicada_analysis*, *analysis_id*, *obj*)
Add widgets to track the corresponding analysis :param cicada_analysis: CicadaAnalysis instance :type cicada_analysis: CicadaAnalysis :param analysis_id: Randomly generated ID linked to the analysis :type analysis_id: str :param obj: The analysis window's object itself :type obj: object

keyPressEvent(*self*, *a0: QKeyEvent*)

class cicada.gui.cicada_analysis_overview.**AnalysisState**(*analysis_id*, *cicada_analysis*, *parent=None*, *without_bringing_to_front=False*)
Class containing the name of the analysis and the subjects analysed

bring_to_front(*window_id*, *event*)
Bring corresponding analysis window to the front (re-routed from the double click method)

Parameters

- **window_id** (*QWidget*) – Analysis Widget object
- **event** (*QEvent*) – Double click event

deleteLater()
Re-implementation of the deleteLater method to properly delete the whole widget

class cicada.gui.cicada_analysis_overview.**ResultsButton**(*cicada_analysis*)
Class containing the button to open the result folder

deleteLater()
Re-implementation of the deleteLater method to properly delete the widget

open_explorer()
Open the file explorer depending on the OS

2.11.6 Analysis parameters

class cicada.gui.cicada_analysis_parameters_gui.**AnalysisData**(*cicada_analysis*, *arguments_section_widget*, *config_handler*, *parent=None*)

keyPressEvent(*event*)

Parameters **event** – P -> set background picture

Returns:

populate_session_list(*session_list*)
Add all session to the QListWidget :param session_list: List of all sessions' identifier :type session_list: list

class cicada.gui.cicada_analysis_parameters_gui.**AnalysisPackage**(*cicada_analysis*, *analysis_name*, *name*, *main_window*, *config_handler*, *parent=None*)
Widget containing the whole analysis window

bring_to_front(*window_id*, *event*)
Bring corresponding window to the front (re-routed from the double click method)

Parameters

- **window_id** (*QWidget*) – Analysis Widget object
- **event** (*QEvent*) – Double click event

errOutputWritten(*text*, *path*)
Append std.err text to the QLabel and create an err file.

Parameters

- **text** (*str*) – Output of the standard output in python interpreter
- **path** (*str*) – path where we will output the err file

normalOutputWritten(*text*, *path*)
Append std.out text to the QLabel and create a log file.

Parameters

- **text** (*str*) – Output of the standard output in python interpreter
- **path** (*str*) – path where we will output the log file

on_close(*event*)
Check if an analysis is still on going and prompt the user to let him know then ask whether he still wants to close. If yes, delete the associated overview and stop the thread

Parameters **event** (*QEvent*) – Qt Event triggered when attempting to close the window

class cicada.gui.cicada_analysis_parameters_gui.**AnalysisParametersApp**(*thread_name*, *progress_bar*, *analysis_name*, *config_handler*, *parent=None*)

Class containing the parameters widgets

create_widgets(*cicada_analysis*)

Parameters **cicada_analysis** (CicadaAnalysis) – Chosen analysis

keyPressEvent(*event*)

Parameters **event** – P -> set background picture

Returns:

load_arguments()
Will open a FileDialog to select a yaml file used to load arguments used for a previous analysis

reset_arguments()
Reset all arguments to default value

run_analysis()
Check if the parameters are valid and then create a thread which will run the analysis

save_yaml_with_name()
Save parameters as a YAML file under the name given by the user. The path is retrieved from the config file and if it doesn't exist a QFileDialog will be displayed to select the path

tabula_rasa()
Erase the widgets and make an empty section

class cicada.gui.cicada_analysis_parameters_gui.**CheckBoxWidget**(*analysis_arg*, *parent=None*)
Used to set a boolean value

get_value()
Return the value of the widget Returns:

set_value(*value*)
Set the widget value to the value passed Returns:

to_stretch()
Indicate if the widget should take all the space of a horizontal layout how might share the space with another widget Returns: Boolean

class cicada.gui.cicada_analysis_parameters_gui.**ColorDialogWidget**(*analysis_arg*, *show_alpha_channel*, *parent=None*)

Widget used to select a color

get_value()
Returns: a tuple of 4 floats representing RGBA with values from 0.0 to 1.0

open_dialog()
Open the color dialog Returns:

set_value(*value*)

Parameters `value` – a list or tuple of 3 or 4 float between 0.0 to 1.0, RGB or RGBA values, the A representing the alpha

Returns:

`to_stretch()`
Indicate if the widget should take all the space of a horizontal layout how might share the space with another widget Returns: Boolean

`update_button_color()`
Returns:

`class cicada.gui.cicada_analysis_parameters_gui.ComboBoxWidget(analysis_arg, parent=None)`

`add_multiple_combo_boxes(session_id, choices_dict, legends, index)`
Allows to add multiple combo boxes, each changing the content of the next one for on given session_id :param session_id: :param choices_dict: each key represent a content to put in the list and the value could be either None, either :param another dict which keys will be the content of the next ComboBox etc… or instead of a dict as value it: :param could be a list that will define the content.: :param legends: :param index:

Returns:

`get_value()`
Returns:

`set_value(value)`
Set a new value. Either value is None and nothing will happen If value is a list instance, :param value:

Returns:

`to_stretch()`
Indicate if the widget should take all the space of a horizontal layout how might share the space with another widget Returns: Boolean

`class cicada.gui.cicada_analysis_parameters_gui.EmittingErrStream(parent=None)`
Class managing the std.err redirection

`write(text)`
Override of the write function used to display output :param text: Python output from stdout :type text: str

`class cicada.gui.cicada_analysis_parameters_gui.EmittingStream(parent=None)`
Class managing the std.out redirection

`write(text)`
Override of the write function used to display output :param text: Python output from stdout :type text: str

`class cicada.gui.cicada_analysis_parameters_gui.FileDialogWidget(analysis_arg, directory_only, extensions=None, parent=None)`
Create a widget that will contain a button to open a FileDialog and a label to display the file or directory choosen A label will also explain what this parameter do

`get_value()`
Return the argument

Returns Dictionary with the set value

Return type result_dict (dict)

`one_for_all()`
Allows to select the same dir of files for all session Returns:

set_value(*value*)
Set the value :param value: either None, either a string or etiher a dictionary

Returns:

to_stretch()
Indicate if the widget should take all the space of a horizontal layout how might share the space with another widget Returns: Boolean

class cicada.gui.cicada_analysis_parameters_gui.**FinalMeta**(*name*, *bases*, *namespace*, ***kwargs*)

class cicada.gui.cicada_analysis_parameters_gui.**FloatLineEditWidget**(*analysis_arg*, *parent=None*)

get_value()
Return the value of the widget Returns:

set_value(*value*)
Set the widget value to the value passed Returns:

to_stretch()
Indicate if the widget should take all the space of a horizontal layout how might share the space with another widget Returns: Boolean

class cicada.gui.cicada_analysis_parameters_gui.**GroupElements**(*group_widget*, *group_name*, *group_elements*, *grid_layout*, *with_color=False*, *default_color=None*)
Used by GroupsFromCheckboxesWidget to display groups content with the possibility to remove one

get_color()
Returns: a tuple of 4 floats representing RGBA with values from 0.0 to 1.0, or None if no color is define

open_color_dialog()
Open the color dialog Returns:

remove_widgets()
Remove widgets. implementation of the deleteLater method to properly delete the whole widget Will be called by self.ti_manager.delete_interval_name method Returns:

update_button_color()
Returns:

class cicada.gui.cicada_analysis_parameters_gui.**GroupsFromCheckboxesWidget**(*analysis_arg*, *choices_attr_name*, *with_color=False*, *add_custom_group_field=False*, *custom_group_validation_fct=None*, *custom_group_processor_fct=None*, *parent=None*)
Display multiple choice that can be selected individually or as group to create new instances. There is also a text field to select all items based on their content (string). Option for a text field allowing to name elements that composed the group, without being in the list

get_value()
Returns: a dict with key is a string representing the group and value is a list with first a list with the elements of the group (strings), and second a color (a tuple of 4 floats representing RGBA with values from 0.0 to 1.0, or None if no color is define)

search_action()
Call when the search button is clicked Returns:

set_value(*value*)
Set the value. :param value: value is a dict with key is a string representing the group and value is a list with first a list

with the elements of the group (strings), and second a color (a tuple of 4 floats representing RGBA with values from 0.0 to 1.0, or None if no color is define) Returns: None

to_stretch()
Indicate if the widget should take all the space of a horizontal layout how might share the space with another widget Returns: Boolean

class cicada.gui.cicada_analysis_parameters_gui.**LineEditWidget**(*analysis_arg*, *parent=None*)

get_value()
Return the value of the widget Returns:

set_value(*value*)
Set the widget value to the value passed Returns:

to_stretch()
Indicate if the widget should take all the space of a horizontal layout how might share the space with another widget Returns: Boolean

class cicada.gui.cicada_analysis_parameters_gui.**ListCheckboxWidget**(*analysis_arg*, *choices_attr_name*, *parent=None*)

Allows multiple choices

get_value()
Returns:

select_all(*session_id*)
select all items :param session_id: None or session of which select items :return:

set_value(*value*)
Set the value. :param value: value is either a string or integer or float, or a list. If a list, then item whose value matches :param one of the elements in the list will be checkeds:

Returns: None

to_stretch()
Indicate if the widget should take all the space of a horizontal layout how might share the space with another widget Returns: Boolean

unselect_all(*session_id*)
Unselect all items :return:

class cicada.gui.cicada_analysis_parameters_gui.**MyFileDialogQButton**(*key_name*, *file_dialog*, *file_dialogs_dict*, *parent=None*)

Special button for opening file dialog

open_dialog()
Open the QFileDialog :param key_name:

Returns:

class cicada.gui.cicada_analysis_parameters_gui.**MyQComboBox**
Special instance of ComboBox allowing to handle change so that it is connected to other combo_boxes

selection_change(*index*)
Called if the selection is changed either by the user or by the code :param index:

Returns:

class cicada.gui.cicada_analysis_parameters_gui.**MyQFrame**(*analysis_arg=None*, *parent=None*, *with_description=True*)

change_mandatory_property(*value*)
Changing the property allowing to change the style sheet depending on the mandatory aspect of the argument :param value:

Returns:

set_property_to_missing()
Allows the change the stylesheet and indicate the user that a Returns:

class cicada.gui.cicada_analysis_parameters_gui.**MySelectButton**(*parent*, *title*, *tooltip_text*, *select_all=False*, *session_id=None*)

class cicada.gui.cicada_analysis_parameters_gui.**ParameterWidgetModel**

abstract get_value()
Return the value of the widget Returns:

abstract set_value(*value*)
Set the widget value to the value passed Returns:

abstract to_stretch()
Indicate if the widget should take all the space of a horizontal layout how might share the space with another widget Returns: Boolean

class cicada.gui.cicada_analysis_parameters_gui.**ProgressBar**(*remaining_time_label*, *parent=None*)
Class containing the progress bar of the current analysis

update_progress_bar(*time_elapsed*, *increment_value=0*, *new_set_value=0*)
Update the progress bar in the analysis widget and the corresponding remaining time :param time_elapsed: Time elepased since beginning of analysis, in seconds :type time_elapsed: float :param increment_value: Value that should be added to the current value of the progress bar :type increment_value: float :param new_set_value: Value that should be set as the current value of the progress bar :type new_set_value: float

Returns:

update_progress_bar_overview(*name*, *increment_value=0*, *new_set_value=0*)
Update the overview progress bar

Parameters

- **name** (*str*) – Analysis ID
- **time_started** (*float*) – Start time of the analysis
- **increment_value** (*float*) – Value that should be added to the current value of the progress bar
- **new_set_value** (*float*) – Value that should be set as the current value of the progress bar

class cicada.gui.cicada_analysis_parameters_gui.**RemainingTime**(*parent=None*)
Class containing the remaining time of the analysis

static correct_time_converter(*time_to_convert*)
Convert a float in a correct duration value :param time_to_convert: Float value representing seconds to be converted in a correct duration with MM.SS :type time_to_convert: float

Returns String of the correct duration

Return type time_text (str)

update_remaining_time(*progress_value*, *time_elapsed*, *done=False*)
Update the remaining time :param progress_value: Current progress bar value :type progress_value: float :param time_elapsed: Time elepased since the beginning of the analysis (in sec) :type time_elapsed: float :param done: True if the analysis is done and false if still running :type done: bool

class cicada.gui.cicada_analysis_parameters_gui.**SameFamilyWidgetsContainer**(*widgets*, *parent=None*)
A QFrame used to group widgets that belongs to a same group. Just useful for visual purposes

to_stretch()
Indicate if the widget should take all the space of a horizontal layout how might share the space with another widget Returns: Boolean

class cicada.gui.cicada_analysis_parameters_gui.**SliderWidget**(*analysis_arg*, *parent=None*)
Used to set a numerical value

get_value()
Return the value of the widget Returns:

set_value(*value*)
Set the widget value to the value passed Returns:

to_stretch()
Indicate if the widget should take all the space of a horizontal layout how might share the space with another widget Returns: Boolean

class cicada.gui.cicada_analysis_parameters_gui.**TextEditWidget**(*analysis_arg*, *parent=None*)

get_value()
Return the value of the widget Returns:

set_value(*value*)
Set the widget value to the value passed Returns:

to_stretch()
Indicate if the widget should take all the space of a horizontal layout how might share the space with another widget Returns: Boolean

class cicada.gui.cicada_analysis_parameters_gui.**Worker**(*name*, *cicada_analysis*, *analysis_arguments_handler*, *parent*)
Thread to manage multiple analysises at the same time

run()
Run the analysis

setProgress(*name*, *time_elapsed=0*, *increment_value=0*, *new_set_value=0*)
Emit the new value of the progress bar and time remaining

Parameters

- **name** (*str*) – Analysis ID
- **time_elapsed** (*float*) – Start elpased (in sec)

- **increment_value** (*float*) – Value that should be added to the current value of the progress bar
- **new_set_value** (*float*) – Value that should be set as the current value of the progress bar

set_results_path(*results_path*)

Set the selected path to the results in the "Open result folder" button in the corresponding overview

Parameters **results_path** (*str*) – Path to the results

cicada.gui.cicada_analysis_parameters_gui.**except_hook**(*cls*, *exception*, *traceback*)

Redirect exception to std.err so we can display stack trace on exceptions

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

H

I

K

L

M

N

O

P

Q

R

S

T

U

W