Neuroweaver GUI Documentation

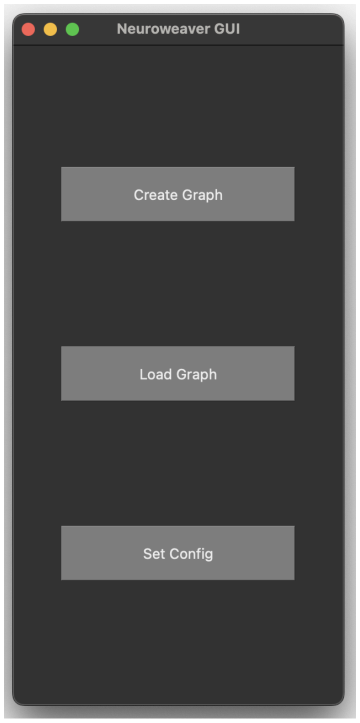
Overview

Neuroweaver is a framework designed to operate intelligent closed-loop neuromodulation systems. It enables users to construct models and workflows using a component-flow model and to build workflow graphs. Currently, Neuroweaver users must manually code each step of the component-flow graph construction. However, its GUI simplifies this process by allowing code-independent graph building. The GUI is designed to be lightweight, modular, independent of workflow processes, and extendable.

The GUI is structured to allow users to dynamically add and configure components, connect them with flows, and manage the overall configuration of the application. The main components of the GUI include the main window, graph creation tools, configuration editing, and components interaction.

This documentation introduces and explains the features of the GUI. Also certain design elements and decisions are discussed.

Main Window

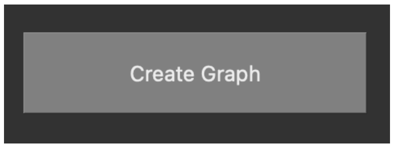


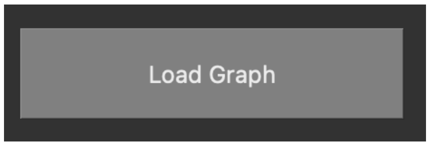
Description

The MainWindow class acts as the entry point and central hub of the application. It initializes the GUI components, sets up the layout, and handles the primary user interactions such as button clicks and menu selections.

Features

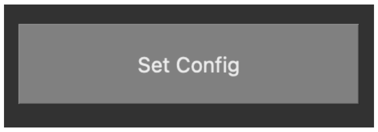
- Graph Management:





  Users can create new graphs or load existing ones from saved configurations. Create Graph button initiates a new graph creation session (a blank graph editor). Load Graph button, allows users to load a graph from a directory containing the necessary JSON files (nodes.json and flows.json).

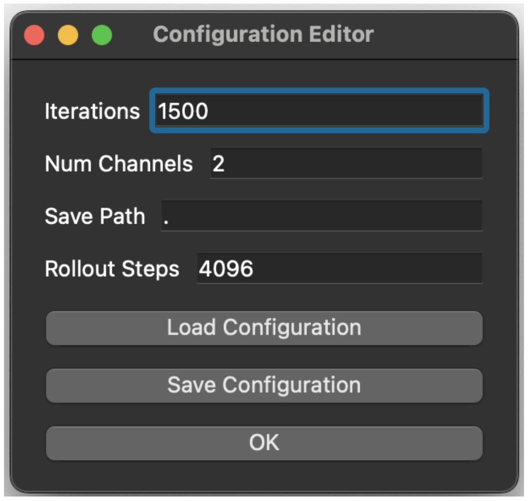
- Configuration Settings:



  Access to configuration settings through a dedicated editor. Set Config button opens the configuration editor to adjust settings such as iterations or number of channels, or save path for the results.

Configuration Editor

Description

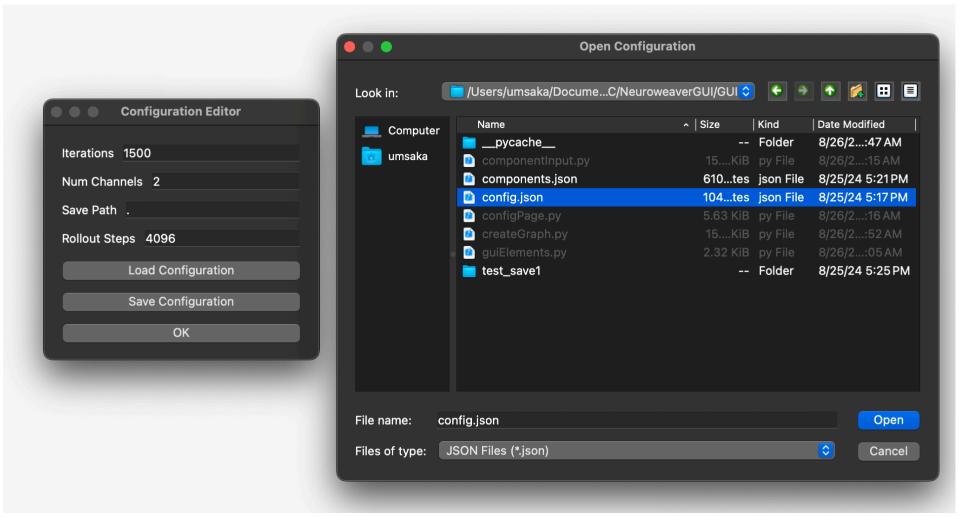


Iterations 1500
Num Channels 2
Save Path .
Rollout Steps 4096

The ConfigEditor class provides a GUI for editing JSON based configuration files. It allows users to load, modify, and save configuration settings that affect the behavior of the Neuroweaver application.

Features

- Modify Settings:
  Each setting can be modified through a simple text field interface.
- Load Configuration:



Iterations 1500
Num Channels 2
Save Path .
Rollout Steps 4096

Look in: /Users/umsaka/Docume...C/NeuroweaverGUI/GUI

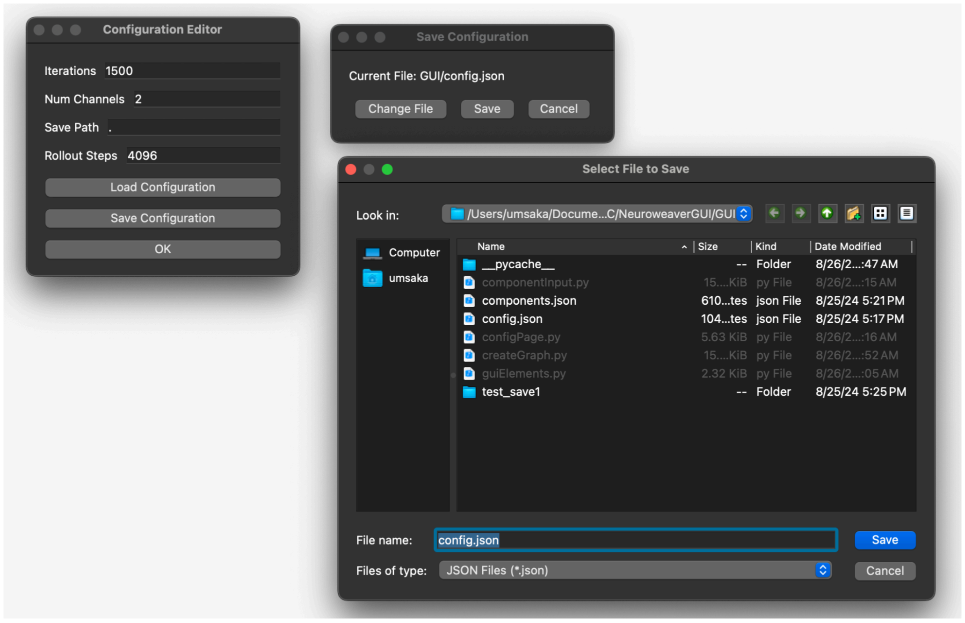
Name	Size	Kind	Date Modified
__pycache__	--	Folder	8/26/2...:47 AM
componentInput.py	15....KiB	py File	8/26/2...:15 AM
components.json	610...tes	json File	8/25/24 5:21 PM
config.json	104...tes	json File	8/25/24 5:17 PM
configPage.py	5.63 KiB	py File	8/26/2...:16 AM
createGraph.py	15....KiB	py File	8/26/2...:52 AM
guiElements.py	2.32 KiB	py File	8/26/2...:05 AM
test_save1	--	Folder	8/25/24 5:25 PM

File name: config.json
Files of type: JSON Files (*.json)

Configuration can be loaded from any JSON file in the following format. To select, just locate the configuration json file and open. The json file should have a format similar to here (String Keys and Values):

```
{
    "Iterations": "1500",
    "Num Channels": "2",
    "Save Path": ".",
    "Rollout Steps": "4096"
}
```

- Save Configuration:



Iterations 1500
Num Channels 2
Save Path .
Rollout Steps 4096

Current File: GUI/config.json

Look in: /Users/umsaka/Docume...C/NeuroweaverGUI/GUI

Name	Size	Kind	Date Modified
__pycache__	--	Folder	8/26/2...:47 AM
componentInput.py	15....KiB	py File	8/26/2...:15 AM
components.json	610...tes	json File	8/25/24 5:21 PM
config.json	104...tes	json File	8/25/24 5:17 PM
configPage.py	5.63 KiB	py File	8/26/2...:16 AM
createGraph.py	15....KiB	py File	8/26/2...:52 AM
guiElements.py	2.32 KiB	py File	8/26/2...:05 AM
test_save1	--	Folder	8/25/24 5:25 PM

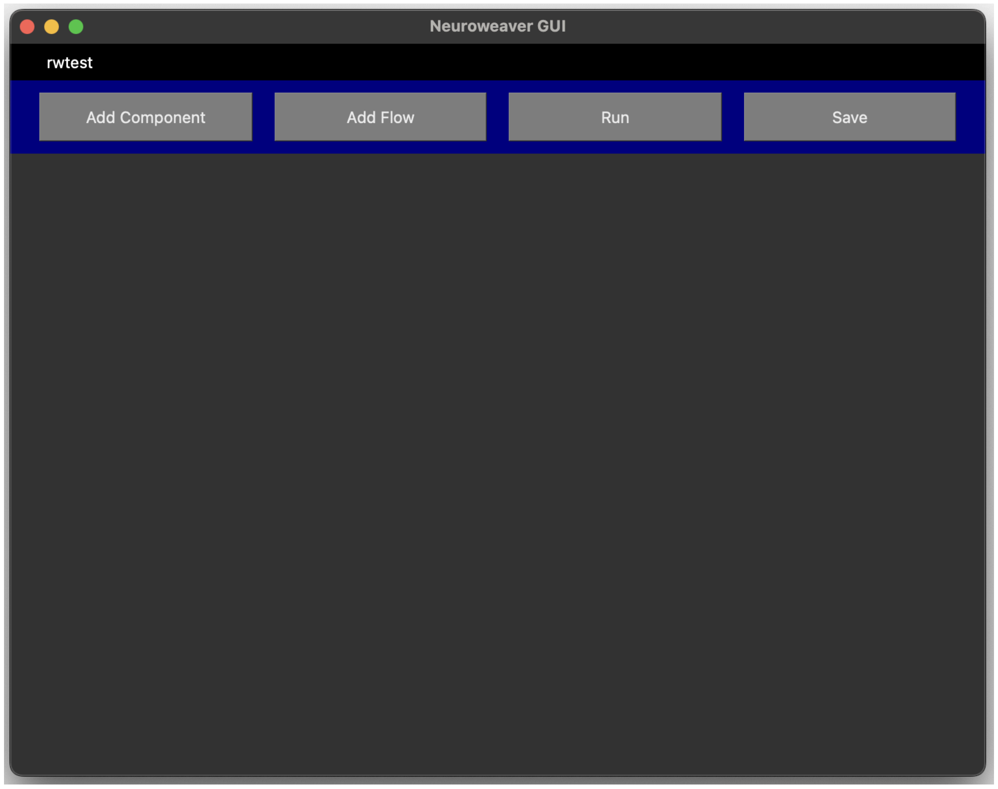
File name: config.json
Files of type: JSON Files (*.json)

Changes can be saved back to the original file or a new file and this locations is selected using a file dialog.

- Dynamic Fields:
  Configuration fields are generated dynamically based on the loaded JSON file, providing flexibility to handle various configuration structures without changing the source code.

Graph Editor

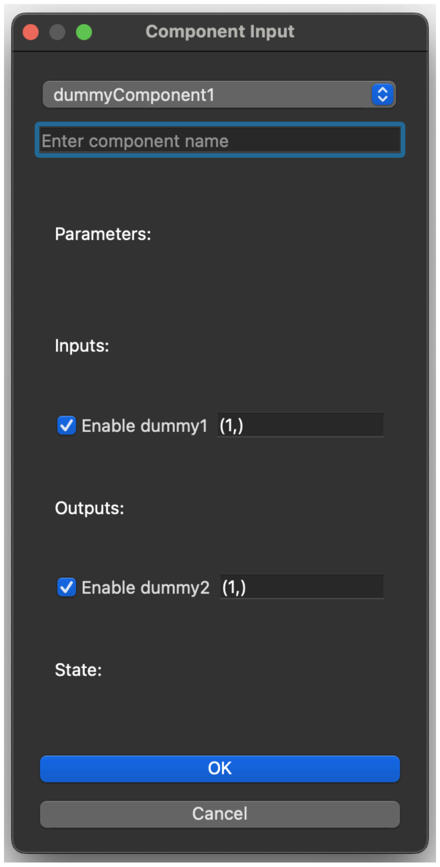
Description



The createGraph class is responsible for the visual creation and manipulation of graphs. It provides a canvas where nodes (components) can be added, configured, and connected with edges (flows).

Features

- Component Addition:



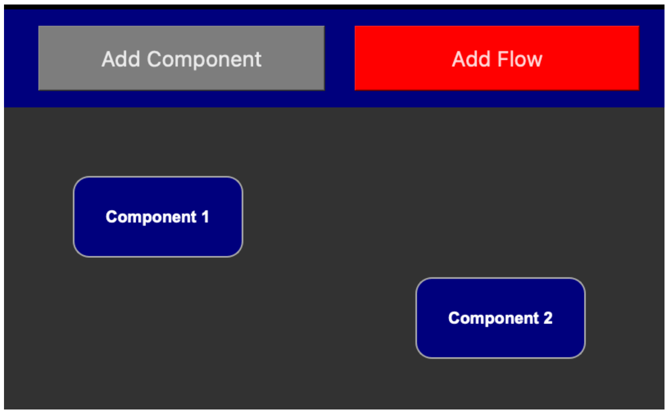
☑ Enable dummy1 (1,)

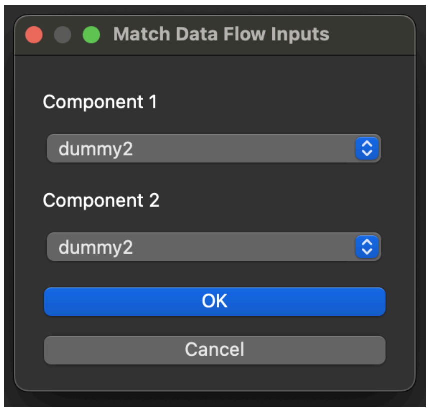
☑ Enable dummy2 (1,)

Users can add components by specifying their type, parameters, and connections. These fields changed based on the specifications of component types. To add a component, click on Add Component Button.

- Flow Creation:

  Flows can be created by connecting nodes. This represents the data flow or dependencies between components. To create a flow, the user should click on Add Flow, and while it's red, click on two components consecutively. If successful, a dialog will be opened to ask user, which output of the first connection will be connected to which output:



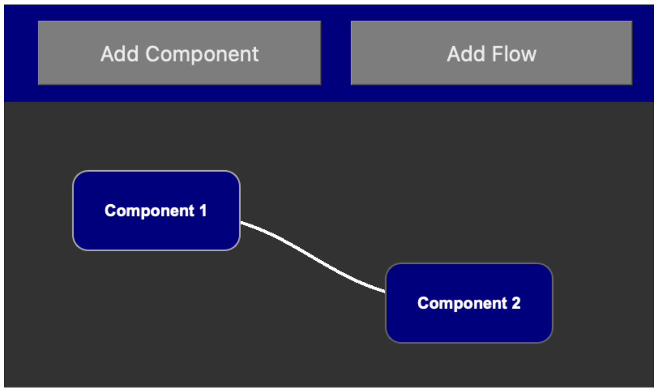

Match Data Flow Inputs

Component 1

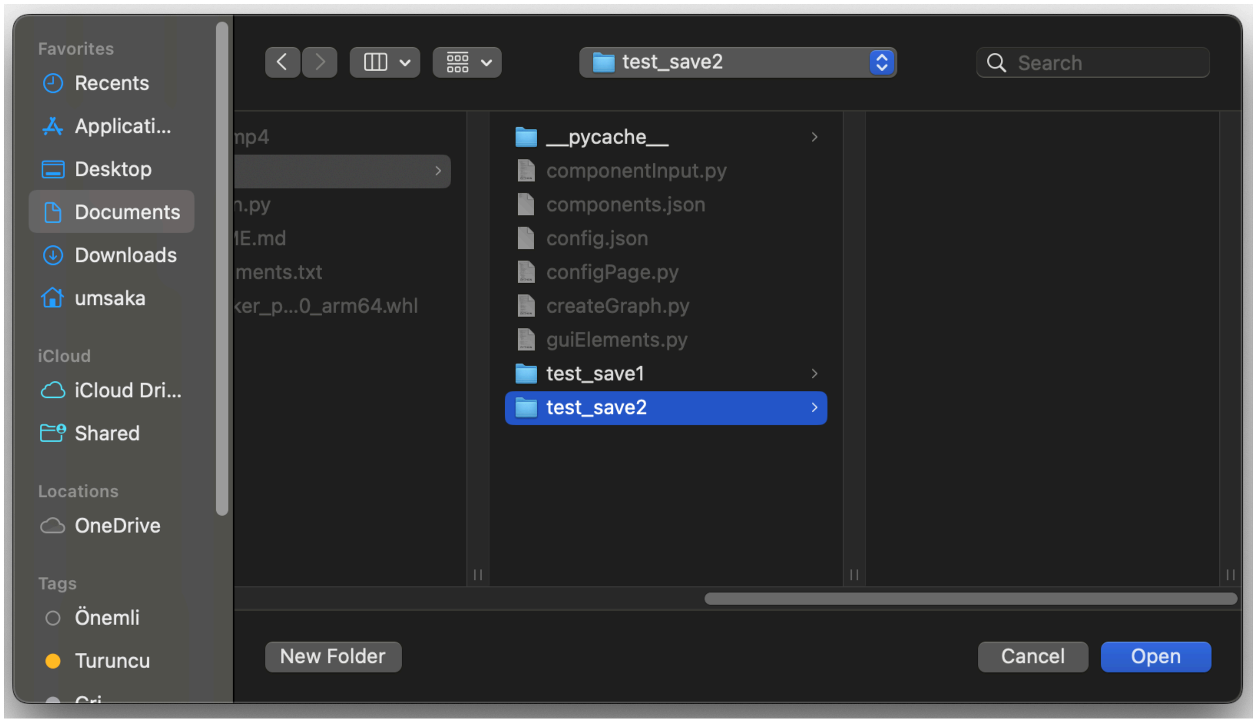
dummy2

Component 2

dummy2



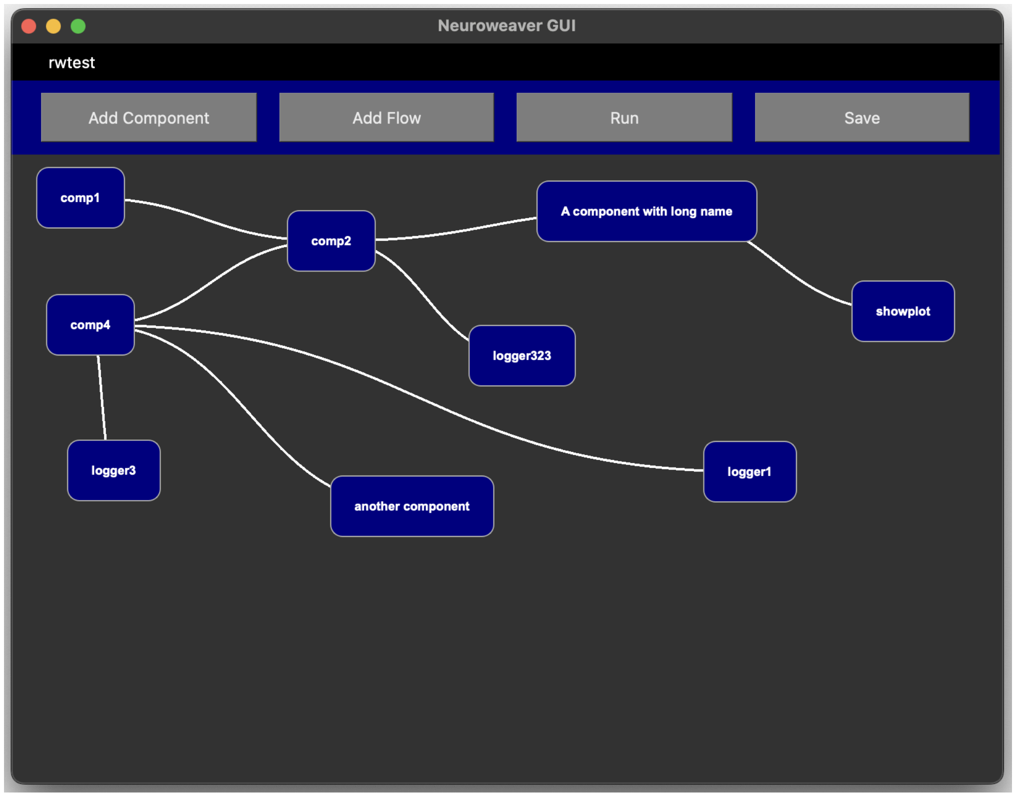
- Graph Saving:



  The entire graph configuration can be saved to a directory, allowing for persistence and reusability. To save, the user should click Save on the Editor. Then, specify the directory where the files are going to be stored. This process will generate two json files: nodes.json and flows.json.
- Run the model:
  If a user hits the run button, the drawn graph will be saved into a temporary file. Then the GUI will start a new python process to call the runner. The runner uses this temporary files and locates the component source codes from component.json, then imports these code and create the Neuroweaver graph structure and then run the graph.

  This feature will be further explained in the following months when the Neuroweaver code will become public.

Design Elements



- Draggable Nodes:
  Each component on the graph is represented as a draggable node, allowing for intuitive layout adjustments by the user.
- Curved Edges:
  Edges are drawn curved to make the GUI more modern and clean.
- Responsive node size:
  Node size is automatically arranged depending on the length of the node

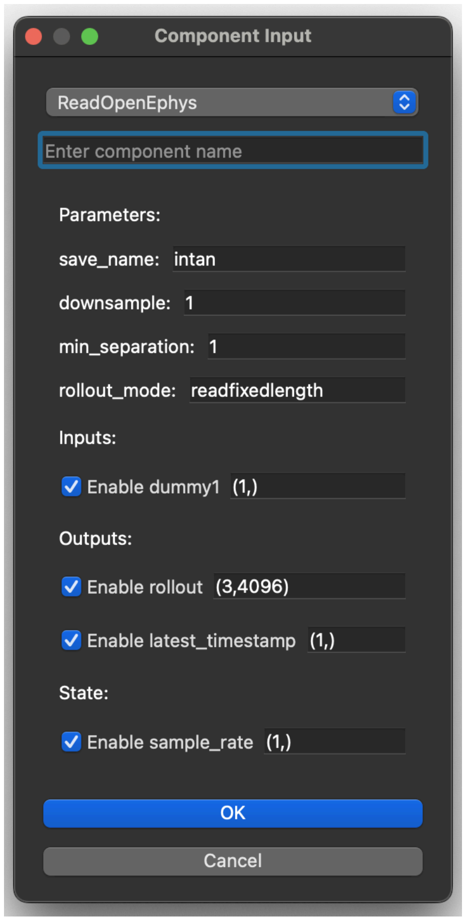
Component Interaction

Description

This module contains dialogs for inputting and configuring new nodes (components) and flows within the graph. It supports dynamic input fields based on the component type selected.

Features

- New Node Configuration:



ReadOpenEphys

Enter component name

Parameters:

save_name: intan

downsample: 1

min_separation: 1

rollout_mode: readfixedlength

Inputs:

✓ Enable dummy1 (1,)

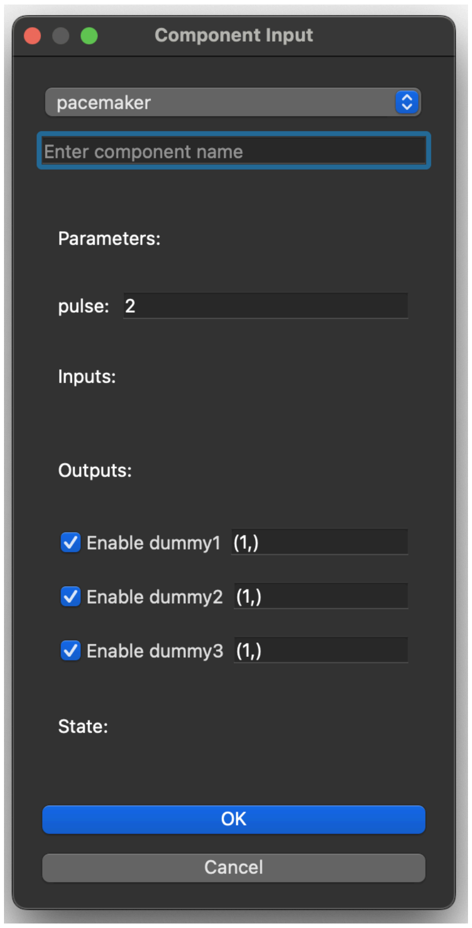
Outputs:

✓ Enable rollout (3,4096)

✓ Enable latest_timestamp (1,)

State:

✓ Enable sample_rate (1,)



pacemaker

Enter component name

Parameters:

pulse: 2

Inputs:

Outputs:

✓ Enable dummy1 (1,)

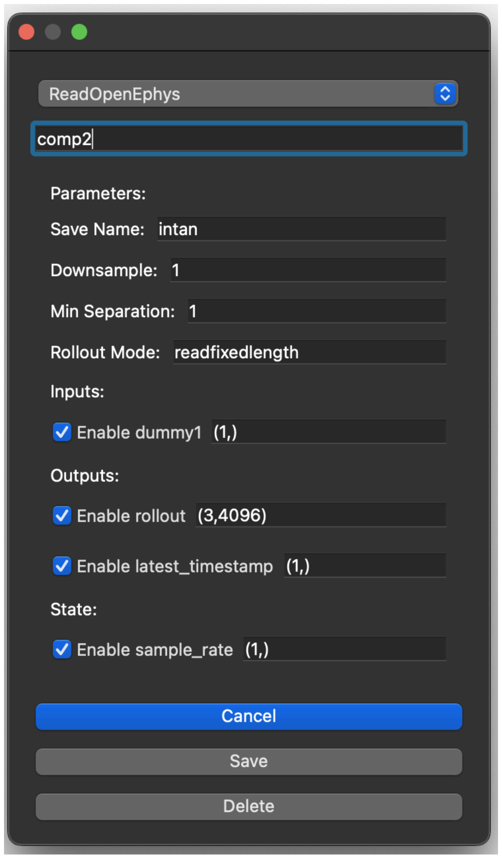
✓ Enable dummy2 (1,)

✓ Enable dummy3 (1,)

State:

Allows users to define new components with customizable parameters, inputs, outputs, and state. The fields changed based on the component type.

- Editing an Existing Node:



ReadOpenEphys

comp2

Parameters:

Save Name: intan

Downsample: 1

Min Separation: 1

Rollout Mode: readfixedlength

Inputs:

✓ Enable dummy1 (1,)

Outputs:

✓ Enable rollout (3,4096)

✓ Enable latest_timestamp (1,)

State:

✓ Enable sample_rate (1,)

If the user double-clicks on an existing node on graph editor, a node editor will pop up. This editor will show the current configuration of each parameter. The user can:

1. Change Name,
2. Change Parameters,
3. Change Input-Outputs
4. Change Component Type
5. Delete the Node

In the latter 3 option, the associated flows will also be automatically removed from the graph.

Component Definition

The user can define the component types in a json file. In this file, each component is defined under a dictionary. Whenever a new type of component is required, the user can locate the module that contains the source code of this new component and edit the components.json file to use that component in the Neuroweaver. With this feature, the user can add and work with new components without changing the source code of the GUI at all. Here’s an example components.json file contents:

```
{
        "dummyComponent1": {
            "inputs" : ["dummy1"],
            "outputs" : ["dummy2"],
            "state": [],
            "parameters":{},
            "module":"hardware.dummy_module",
            "function":"dummy1",
            "iterationAddition": 5,
            "kwargs":[]
        },
        "dummyComponent2": {
            "inputs" : ["dummy2"],
            "outputs" : ["dummy3"],
            "state": [],
            "Parameters":{"param1":"5"},
            "module":"hardware.dummy_module",
            "function":"dummy2",
            "iterationAddition": 5,
            "kwargs":[]
        }
}
```