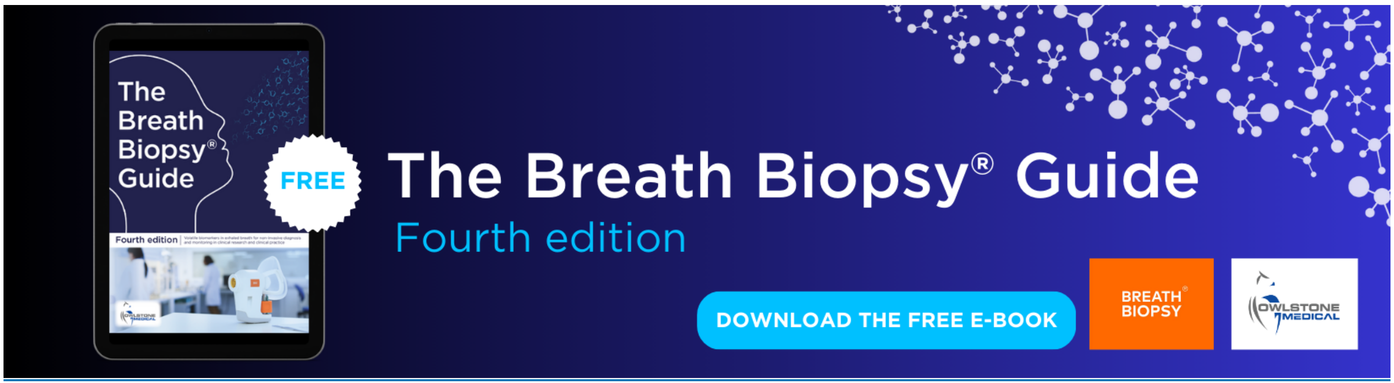
Open Ephys: an open-source, plugin-based platform for multichannel electrophysiology



OPEN ACCESS
IOP Publishing

Journal of Neural Engineering

J. Neural Eng. **14** (2017) 045003 (13pp)

https://doi.org/10.1088/1741-2552/aa5eea

Open Ephys: an open-source, plugin-based platform for multichannel electrophysiology

Joshua H Siegle[1], Aarón Cuevas López[2,6], Yogi A Patel[3], Kirill Abramov[4], Shay Ohayon[5] and Jakob Voigts[5]

[1] Allen Institute for Brain Science, 615 Westlake Ave N, Seattle, WA 98109, United States of America, jsiegle on GitHub
[2] Universitat Politècnica de València, Camí de Vera, s/n, 46022 València, Spain, aacuevas on GitHub
[3] Georgia Institute of Technology, North Ave NW, Atlanta, GA 30332, United States of America, yapatel on GitHub
[4] Zaporizhzhya State Engineering Academy, 69006, Sobornyi Ave, 226, Zaporizhia, Zaporiz'ka oblast, Ukraine, septen on GitHub
[5] Massachusetts Institute of Technology, 77 Massachusetts Ave, Cambridge, MA 02139, United States of America, shayo and jvoigts on GitHub
[6] Instituto de Neurociencias CSIC-UMH, Avenida Santiago Ramon y Cajal, s/n, 03550 Sant Joan d'Alacant, Alicante, Spain, aacuevas on GitHub

E-mail: joshs@alleninstitute.org (Joshua H Siegle)

Received 16 November 2016, revised 12 January 2017
Accepted for publication 7 February 2017
Published 5 June 2017

Abstract

Objective. Closed-loop experiments, in which causal interventions are conditioned on the state of the system under investigation, have become increasingly common in neuroscience. Such experiments can have a high degree of explanatory power, but they require a precise implementation that can be difficult to replicate across laboratories. We sought to overcome this limitation by building open-source software that makes it easier to develop and share algorithms for closed-loop control. *Approach.* We created the Open Ephys GUI, an open-source platform for multichannel electrophysiology experiments. In addition to the standard 'open-loop' visualization and recording functionality, the GUI also includes modules for delivering feedback in response to events detected in the incoming data stream. Importantly, these modules can be built and shared as plugins, which makes it possible for users to extend the functionality of the GUI through a simple API, without having to understand the inner workings of the entire application. *Main results.* In combination with low-cost, open-source hardware for amplifying and digitizing neural signals, the GUI has been used for closed-loop experiments that perturb the hippocampal theta rhythm in a phase-specific manner. *Significance*. The Open Ephys GUI is the first widely used application for multichannel electrophysiology that leverages a plugin-based workflow. We hope that it will lower the barrier to entry for electrophysiologists who wish to incorporate real-time feedback into their research.

Keywords: electrophysiology, software, open source, closed loop

(Some figures may appear in colour only in the online journal)

Introduction

One of the primary dilemmas faced by scientists is whether to spend time building new tools to address their questions of interest or to tailor their questions to the available tools. In the field of systems neuroscience, the disparity between the complexity of the system under investigation and the relative simplicity of the methods of observation means that the most impactful scientific discoveries often go hand-in-hand with engineering breakthroughs. In addition to thinking deeply about the brain, researchers must be able to modify their experimental tools in order to gain new insights from their limited views into neural circuitry.

Extracellular electrophysiology, one of the most commonly used techniques in systems neuroscience, uses voltage measurements to eavesdrop on the spiking activity of neural populations distributed across multiple brain regions. The data acquisition systems used for such experiments must detect, amplify, filter, and digitize dozens or hundreds of voltages in parallel, while offering ways to efficiently visualize and store these data. Currently, a majority of these systems come from commercial vendors, which sell high-quality hardware that requires minimal setup and configuration steps. These commercial systems are typically closed-source, which means that they can only be modified in ways that the vendors allow. Furthermore, any modifications can only be shared with others that own the same system. This limitation on sharing methods becomes more problematic as the experiments carried out by neuroscientists become more complex. For experiments that require specific types of data processing to be carried out during the experiment, they severely restrict the replicability of the findings of that experiment. So far, there has been little effort to make components of various commercial systems interoperable to overcome this hurdle.

In order to make it easier for electrophysiologists to customize and share their data acquisition pipelines, we developed an open-source suite of tools under the moniker 'Open Ephys.' The central focus of our platform is the Open Ephys graphical user interface (GUI), a plugin-based application for data acquisition, processing, and visualization. The GUI is written entirely in C++, using the JUCE library (www.juce.com) and has been developed with many current and common software development practices in mind [1]. All of the source code, design files, and documentation are available for free, via GitHub (https://gihub.com/open-ephys) and the Open Ephys Wiki (https://open-ephys.atlassian.net). Our software is covered by the GNU GPL 3.0 license (www.gnu.org/licenses/gpl-3.0.en.html) and our hardware is covered by the TAPR Open Hardware License (www.tapr.org/ohl.html). We chose these restrictive 'share-alike' licenses, as opposed to more permissive licenses such as BSD or MIT, in order to ensure that any modified versions of our tools (including new plugins) must also be open source. We believe that the advantages of keeping scientific software open-source strongly outweigh the costs, and we would like to encourage others to share their source code if they decide to build upon our work.

From the beginning, however, we knew that making our software and hardware open source would not be sufficient; we also had to include logical interfaces through which researchers could add their own modifications. This is especially important when designing closed-loop experiments, in which measured activity is used to determine the time and/or type (e.g. electrical [2, 3], optogenetic [4, 5], or environmental [6]) of perturbation fed back into the system under investigation. The space of potential closed-loop perturbations is enormous, so each experiment will likely require a unique protocol. In our view, an open-source model will help bring closed-loop paradigms into mainstream neuroscience. Tools that are well designed, open-source, and modular can enable scientists to make adjustments to fit the needs of their individual research programs.

Just as importantly, closed-loop experiments change the nature of how an experiment needs to be published and shared (figure 1). With 'open-loop' experiments, only the details of the experimental setup, a description of the analysis methods, and a graphical representation of the data are the 'scientific product' that is made public. Although it is becoming increasingly common for scientists to share their analysis code as well, the code itself is not generally expected to be shared in order to make sense of the conclusions of a given study. In contrast, closed-loop experiments are *defined* by the algorithm used to perturb neural activity and all resulting data are uninterpretable without access to this algorithm. Therefore, closed-loop algorithms need to be published and shared in full. This presents a technical challenge because these algorithms interact intimately with many other parts of the data acquisition system. This issue will become much easier to solve if neuroscientists adopt open standards that allow them to share and reuse algorithms for closed-loop control.

Our solution to this problem was to build the Open Ephys GUI around a plugin architecture. The GUI's signal processing chain is configured in a modular fashion, using modules which can be written, compiled, and distributed separately from the main 'host' application, thanks to a common data passing interface. This paradigm is commonly used in applications for music production, in which software instruments (signal generators) and effects (signal processors) can be mixed and matched. Anyone can write new plugins for the most common digital audio workstations (the host applications), and the same plugins can be loaded into multiple host applications at runtime. We felt that a similar approach applied to the domain of extracellular electrophysiology would allow scientists to maximize the flexibility of their experiments, while simplifying the process of publishing and sharing their modifications.

Below, we describe the GUI and its associated open-source hardware in the following sections:

- Overview of the GUI
- Plugin architecture
- Compatible hardware
- Application: closed-loop stimulation of hippocampus
- Advantages and disadvantages of Open Ephys
- Comparison to other open-source data acquisition systems
- Future directions

Overview of the GUI

The Open Ephys GUI was created, first and foremost, as a software interface for running and monitoring extracellular

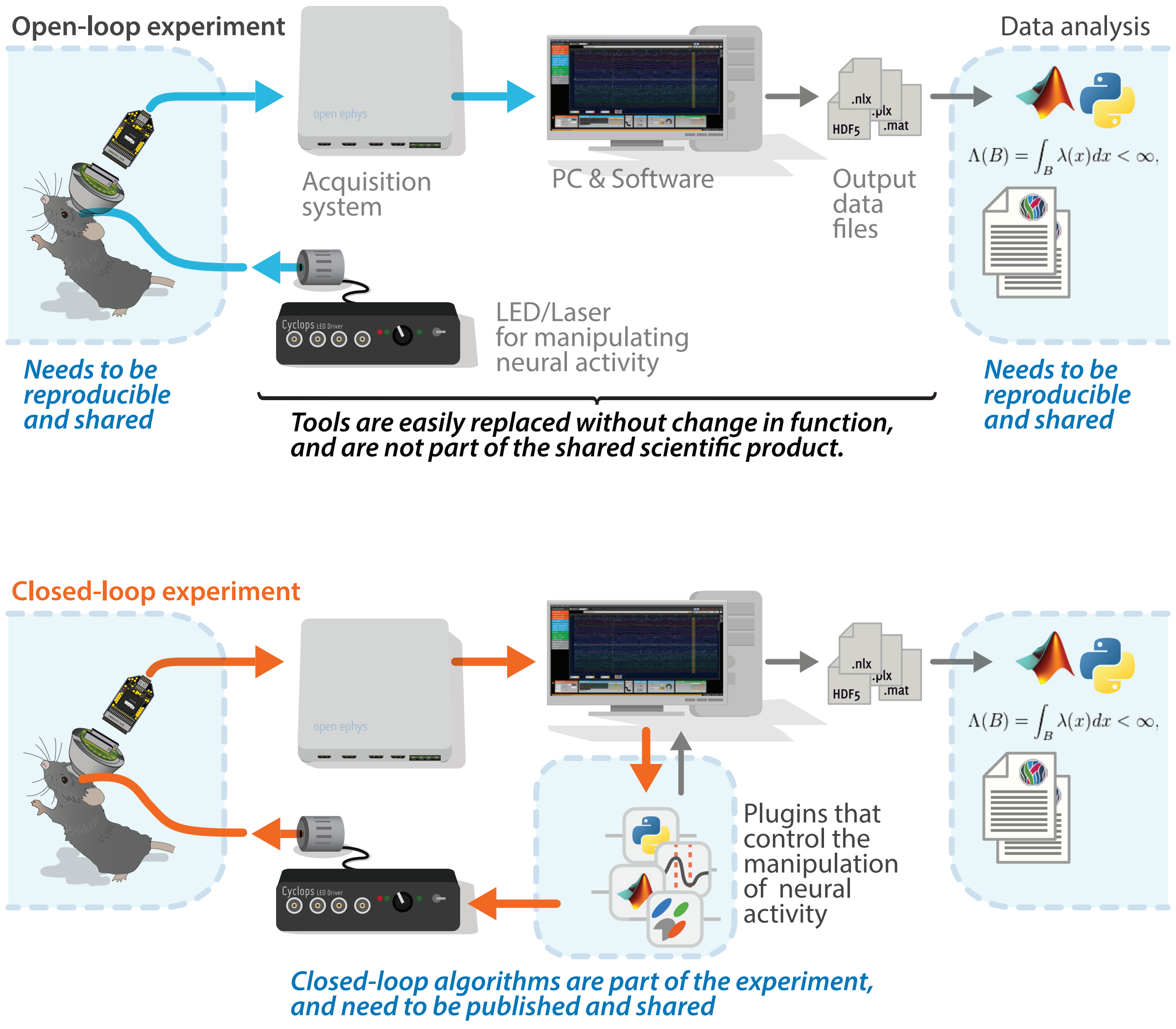

Open-loop experiment
open ephys
Acquisition system
PC & Software
.nlx
.plx
HDF5
.mat
Output data files
Data analysis
$\Lambda(B) = \int_B \lambda(x)dx < \infty,$
Cyclops LED Driver
LED/Laser for manipulating neural activity
Needs to be reproducible and shared
Needs to be reproducible and shared
Tools are easily replaced without change in function, and are not part of the shared scientific product.
Closed-loop experiment
Plugins that control the manipulation of neural activity
Closed-loop algorithms are part of the experiment, and need to be published and shared

Figure 1. Comparing open-loop and closed-loop experiments. In open-loop experiments (top), perturbations are carried out 'blindly,' without taking brain state into account. In closed-loop experiments (bottom), algorithms are used to update the perturbation in real time, based on the state of the brain at a given moment. Building closed-loop experiments on top of an open platform is essential for interpreting and reproducing their results.

electrophysiology experiments. The user interface consists of four main elements (figure 2):

- **The control panel**, at the top of the main application window, holds buttons to toggle acquisition and recording, a clock to track the amount of time spent acquiring data, controls for audio output and recording parameters, and displays that indicate CPU usage and available disk space.
- **The processor list**, at the left of the main application window, contains a list of data processing modules available for constructing the signal chain. These are organized as 'Sources,' 'Filters,' 'Sinks,' and 'Utilities.'
- **The editor viewport**, at the bottom of the main application window, provides access to the parameter editors for every processing module in the signal chain. Here, the user can control which channels are recorded and monitored, as well as alter custom settings for individual modules.
- **The data viewport**, which fills the remaining space in the main application window, holds tables for visualizers, as well as a graph that displays the layout of the signal chain at a glance.

When the GUI is launched for the first time, the user must configure the signal chain from scratch. The user interface for creating the signal chain is modeled after that of Ableton Live, a widely used audio production application [7]. Users drag and drop modules from the processor list onto the editor viewport to build a processing pipeline from left to right. Each module is meant to serve a precise, well-defined purpose, such as communicating with an external piece of hardware, extracting spike waveforms from an incoming data stream, or visualizing data in a specific way. This allows a data stream to be customized for each experiment, and encourages re-use of modules with commonly needed functions. The signal chain can be split or merged at any point, allowing multiple processing streams to run in parallel. Data can be saved from any module in the signal chain, a critical feature for monitoring

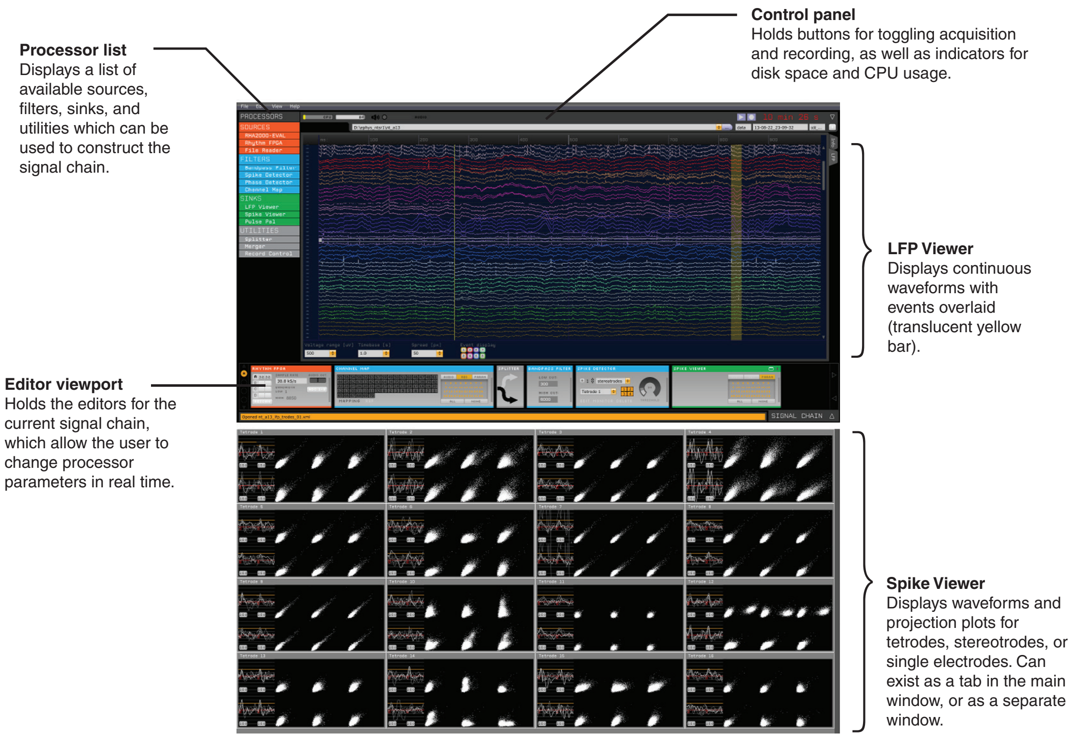

Processor list
Displays a list of available sources, filters, sinks, and utilities which can be used to construct the signal chain.
Control panel
Holds buttons for toggling acquisition and recording, as well as indicators for disk space and CPU usage.
LFP Viewer
Displays continuous waveforms with events overlaid (translucent yellow bar).
Editor viewport
Holds the editors for the current signal chain, which allow the user to change processor parameters in real time.
Spike Viewer
Displays waveforms and projection plots for tetrodes, stereotrodes, or single electrodes. Can exist as a tab in the main window, or as a separate window.

Figure 2. Layout of the Open Ephys graphical user interface (GUI). The major components of the software are labeled. The layout is very flexible; the Processor List and Editor Viewport can be hidden when not in use, and any visualizers can either inhabit a table within the main window or their own floating window. The software runs on Windows, Linux, and Mac OS X. Pre-compiled executable files are available at open-ephys.org, and the source code can be viewed on GitHub (github.com/open-ephys/plugin-GUI).

how individual processing elements behave during closed-loop experiments. On a given processing cycle, each module is handed a buffer of continuous data (typically around 20 ms long) and a buffer of events (including spikes) that occurred within the same timespan. Any modifications that a module makes to those buffers are automatically passed to the next module in the signal chain.

The GUI is built upon the JUCE Toolkit (www.juce.com), JUCE is a well-supported open-source development framework that provides the core C++ classes for data handling and creating versatile user interfaces. Thanks to JUCE, the GUI runs equally well on Windows, Linux, and Mac, with minimal need for code specialization across the three operating systems. JUCE was originally written to facilitate the development of high-performance audio processing applications. We have adapted many of its audio-specific functions to process neural data. Audio data is typically handled at 44.1 kHz and 16 bits, which is very similar to the ~30 kHz, 16 bit data streams that are the standard for most extracellular electrophysiology experiments. The GUI uses JUCE to interface with the computer's audio card, which generates the precise timing signals required to ensure that the signal processing chain can keep up with the incoming data stream.

For most electrophysiology experiments, the user will need to visualize the continuous local field potential, or LFP, from all channels in order to assess electrode placement and signal quality. The GUI includes an 'LFP Viewer' module that displays the signal from each recording site in real time (figure 2). The LFP Viewer streams data from left to right, like an oscilloscope, for windows of 0.25–10 s. Depending on the filter settings of the hardware and software, the LFP Viewer can be used to display spike waveforms as well as low-frequency signals.

Many experiments also require the detection and display of spike waveforms in real time. The GUI separates this functionality across three modules: a Filter Node, a Spike Detector, and a Spike Viewer. The Filter Node streams the incoming data through a bandpass filter; the default settings are 300 Hz for the low cut and 6000 Hz for the high cut. This allows the data to be thresholded by the Spike Detector, in order to extract the waveforms of candidate spikes. The Spike Detector can detect spikes on single electrodes, stereotrodes (two linked channels), or tetrodes (four linked channels). These spikes are sent as events—in parallel to the continuous data stream—to the Spike Viewer, which displays waveforms and peak-height projection plots for stereotrodes and tetrodes. The flexible plugin architecture also makes it possible to implement more sophisticated real-time spike analysis algorithms. For example, the Spike Sorting module, which encapsulates spike detection, visualization, and sorting into one module, can be used in real time to isolate single-unit activity based on spike shape. This feature is critically important for many acute experiments in which the isolation and characterization of single units in real-time is required.

The GUI has been optimized for multi-channel extracellular electrophysiology experiments that use twisted-wire tetrodes [8, 9] or silicon probes [10, 11] to detect voltage fluctuations inside the brain. Our primary target audience is researchers that use these implantable devices in nonhuman model organisms, although the flexible nature of the GUI makes it compatible with other sources of continuous voltage traces, such

Table 1. Available plugins. List of plugins currently included with the Open Ephys GUI.

Type	Name	Function
Source	Rhythm FPGA	Reads data from a device running Intan's Rhythm FPGA firmware; includes the Open Ephys acquisition board and the Intan RHD2000 Evaluation Board.
	File Reader	Reads data from a file.
	Network Events	Reads events from a TCP port, either from another program running locally, or another machine. Can be used to signal the start of an experimental epoch or stimulus condition.
	Serial Port	Reads data from a serial port. Can be used to track eye position or location in a virtual maze.
Filter	Spike Detector	Extracts spike events from continuous data.
	Common Avg Ref	Subtracts the average of all channels or a subset of channels. Can be used to remove common-mode noise from a recording.
	Channel Map	Re-orders channels in continuous data.
	Bandpass Filter	Filters data between two frequencies.
	Phase Detector	Emits events when it detects peaks, troughs, or zero-crossings in a continuous signal.
	Rectifier	Outputs the absolute value of a continuous signal.
	Spike Sorter	Extracts spike events from continuous data, and allows the user to define unit boundaries in PCA or waveform space.
Sink	Arduino Output	Events within the GUI are used to trigger digital pulses from an Arduino microcontroller (www.arduino.cc)
	Spike Viewer	Displays spike waveforms and peak heights for stereotrodes and tetrodes.
	Event Broadcaster	Broadcasts events over a network connection.
	LFP Viewer	Displays continuous signals.
	Pulse Pal	Events within the GUI are used to trigger channels on a Pulse Pal, an open-source stimulation device (www.sanworks.io)
Record engine	Open Ephys Format	Fault-tolerant data format designed for the Open Ephys GUI
	Binary Format	Flat binary file of int16s
	NWB Format	Neurodata Without Borders, an HDF5-based format designed to facilitate data-sharing for neurophysiology expriments
	Kwik Format	An HDF5-based format designed for KlustaKwik.

as human scalp EEG. By creating new software plugins, the GUI could be used as an interface for existing EEG hardware from companies such as OpenBCI (http://openbci.com), g.Tec (www.gtec.at), or EMOTIV (www.emotiv.com).

To obtain the software, executable files for the GUI can be downloaded from the Open Ephys website (http://open-ephys.org/gui). Alternatively, the GUI can be compiled from the source code downloaded from GitHub (https://github.com/open-ephys/plugin-GUI). On Windows, compiling the GUI requires Visual Studio; on Mac, Xcode is required; on Linux, the standard GNU command-line tools (e.g. *make*, *gcc*) are sufficient. Development of the GUI occurs in a distributed fashion at institutions around the world. Efforts are coordinated via GitHub, as well as the Open Ephys mailing list (open-ephys@googlegroups.com).

Plugin architecture

To ensure that the GUI can be adapted to the requirements of different experiments, we constructed it around a plugin architecture. Under this paradigm, everyone using the GUI shares the same host application, while data processing modules are compiled separately and loaded from within the host application by the user when desired. This means that the core application can remain free of the library dependencies introduced by the plugins.

A variety of useful plugins are provided when the application is downloaded (table 1). New modules can be built individually and distributed as binary files, provided they were compiled on the same operating system as the host application. The plugins are shared as dynamically loaded libraries (DLLs) on Windows, dynamic libraries (DYLIBs) on Mac, and shared objects (SOs) on Linux.

All plugins are classified at either Sources, Filters, and Sinks (figure 3):

- **Sources** bring data into the signal chain by filling an empty buffer of continuous samples and an empty buffer of events on each processing cycle. There is an option of creating a separate thread for each source, which simplifies the process of communicating with external devices that run on a different clock than the host application. Data acquired via the thread will be automatically fed into the GUI's signal chain. Example sources include the 'Rhythm FPGA' thread, which communicates with hardware devices running Intan's Rhythm firmware (see next section); the File Reader, which loads data from a file; and the Network Source, which receives events from another computer. If a developer wishes to create a Source module that interfaces with proprietary hardware, they can do so by calling a separate, closed-source DLL to communicate with the device.
- **Filters** modify the incoming data stream in some way or detect events in the continuous data, such as spikes or oscillations. Example filters include the Filter Node, which applies a bandpass filter to all channels; the common average reference (CAR), which subtracts the

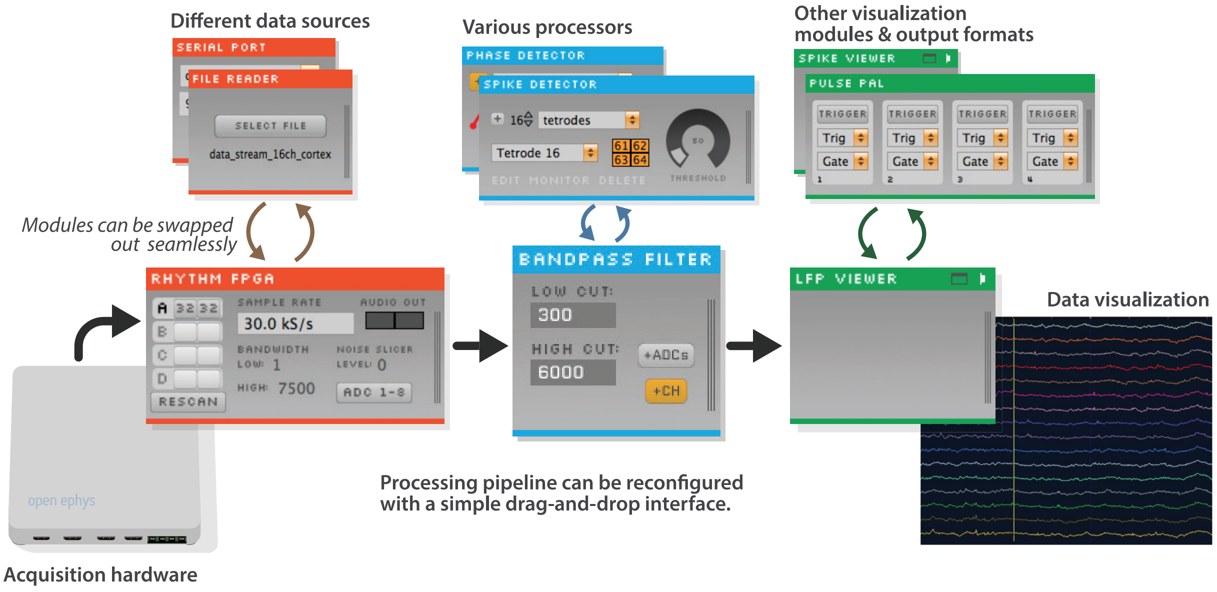


Figure 3. The Open Ephys plugin architecture. The Open Ephys GUI allows users to process data using a flexible, modular signal chain. 'Source' modules (orange) can be swapped out to allow the signal chain to receive data from different sources. 'Filter' modules (blue) can be mixed and matched to determine the real-time data processing steps that occur. And 'Sink' modules (green) can be used to control external hardware, such as a visual display or a stimulating laser.

average signal to remove artifacts; and the Spike Detector, which emits spike events whenever a threshold crossing is detected on a particular channel or subset of channels.

- **Sinks** interface with elements outside the signal chain, such as a display, a stimulation device, or a network port. Example sinks include the LFP Viewer, which allows the user to visualize continuous data streams; the Spike Viewer, which displays spike waveforms in real time; and the Pulse Pal Output, which uses events to trigger stimulation from a Pulse Pal, an open-source pulse generator created by Josh Sanders of Sanworks (www.sanworks.io).

In addition to Sources, Filters, and Sinks, it is also possible to create plugin 'Record Engines' and 'File Sources.' Record Engines and File Sources specify how continuous, spike, and event data is written to or loaded from disk, and make it possible for users to customize the data format of the GUI. The GUI already includes four Record Engines, which specify the output data formats:

(1) The Open Ephys Format, the default format, is a fault-tolerant format built specifically for the GUI. This format saves the data in blocks of 1024 samples, each of which includes a timestamp and a readily identifiable 'record marker,' so that data can still be recovered if part of the file becomes corrupted.
(2) The Binary Format saves continuous data as flat files of int16s, and is used by the spike sorting package Kilosort [12].
(3) Neurodata without borders (NWB) is an HDF5-based format designed to facilitate data sharing between neurophysiology laboratories [13].
(4) Kwik is a deprecated HDF5-based format originally developed for the KlustaKwik suite of spike sorting tools [14].

More information on data formats can be found on the Open Ephys wiki (https://open-ephys.atlassian.net/wiki/display/OEW/Data+format).

One important challenge when designing the plugin system was the application programming interface (API), which plugins use to communicate with the GUI. Since few neuroscientists are proficient with software development, the basic methods and workflow to create a plugin need to be straightforward and easy to understand. At the same time, the interface must also be flexible enough that it does not impose too many limitations on plugin capabilities.

To address this issue, we created a complete C++ interface for plugins, instead of using more traditional C-based plugin libraries. While this does impose some restrictions, such as the need to build the plugins with the same compiler as the core GUI, it makes it possible for plugins to take advantage of the class inheritance capabilities of C++. At its core, every plugin is simply a C++ class derived from a base class which contains all the needed functions for the plugin to integrate with the GUI. A plugin developer must only provide overrides to a small set of virtual methods to have a working processor. At the same time, there exist a number of optional methods, already defined in the base classes, that can be redefined in the plugin to achieve more complex functionality. C++ inheritance structure also allows the base classes to provide easy-to-use helper methods without the need for the programmer to know the internal implementation details. The use of a C++ interface also prevents bloating of the compiled binaries for the GUI and the modules. Initial implementations of the plugin architecture required compiling the JUCE library with each individual plugin, leading to significant increases in compiled binary sizes. The C++ interface removes the need to re-compile JUCE with each plugin and allows each plugin to directly access the JUCE classes through the GUI. This, coupled with the designed inheritance scheme, allows plugins to integrate seamlessly into the host application, both from a developer's and end-user's perspective.

To further simplify up the process of developing new plugins, we provide a Plugin Generator GUI. The Plugin Generator allows developers to walk through the process of plugin creation and configuration with a user-friendly visual interface. The application gives users the opportunity to specify the type of plugin they wish to create (Source, Filter, Sink, Record Engine, File Source), the name of the plugin, and the parameters that can be accessed by the user (boolean, continuous, or discrete). Once the configuration is finished, the Plugin Generator automatically generates the C++ header and implementation files needed to build the plugin. The Plugin Generator allows user to easily change the plugin's UI: they can add control components (sliders, buttons, etc) that can be bound to any of the plugin's parameters, change the look and feel of all components at once, or create the plugin's layout using one of the provided templates. From that point, the developer needs only to add C++ code into the 'process' method to define the plugin's functionality.

Those wishing to develop new plugins are encouraged to explore the following resources on our wiki:

- An overview of the plugin architecture (https://open-ephys.atlassian.net/wiki/display/OEW/Plugin+architecture)
- A tutorial for adding new plugins (https://open-ephys.atlassian.net/wiki/display/OEW/Tutorial%3A+Add+a+custom+processor)
- Instructions for using the Plugin Generator (https://open-ephys.atlassian.net/wiki/display/OEW/Plugin+Generator)

The process of checking which plugins are present and loading them is handled by a Plugin Manager class inside the host application. Since the Open Ephys GUI is frequently being improved, the plugin API may sometimes be updated, thus rendering plugins incompatible. The Plugin Manager's loading process includes a version check that ensures that all loaded plugins are compatible with the current version of the GUI, and informs the user when a plugin's API does not match that of the host applications.

The GUI makes it easy for signal chains to be saved and re-loaded at runtime, but this may not be possible when signal chain configurations are shared with other users who do not have the same plugins available. In this case, the Plugin Manager inserts a dummy processor into the chain with information about the required plugin. The user can then locate the plugin or decide to do without it. Data acquisition cannot proceed unless the dummy processor is replaced or removed.

Together, the modularity of the GUI, API, and Plugin Generator simplify the development process for users. If a module with the desired functionality is not available, a user can create a new one by copying the source code for any existing module and changing the functionality to suit their requirements. There is no need to understand the inner workings of the entire application (figure 4). Only knowledge of the standardized interfaces for passing data between modules is required, allowing the GUI to be modified by users with varying levels of programming skill. As of January 2017, the GUI's source code has been forked 130 times on GitHub, and 10 different research groups have contributed plugins to the main repository.

Compatible hardware

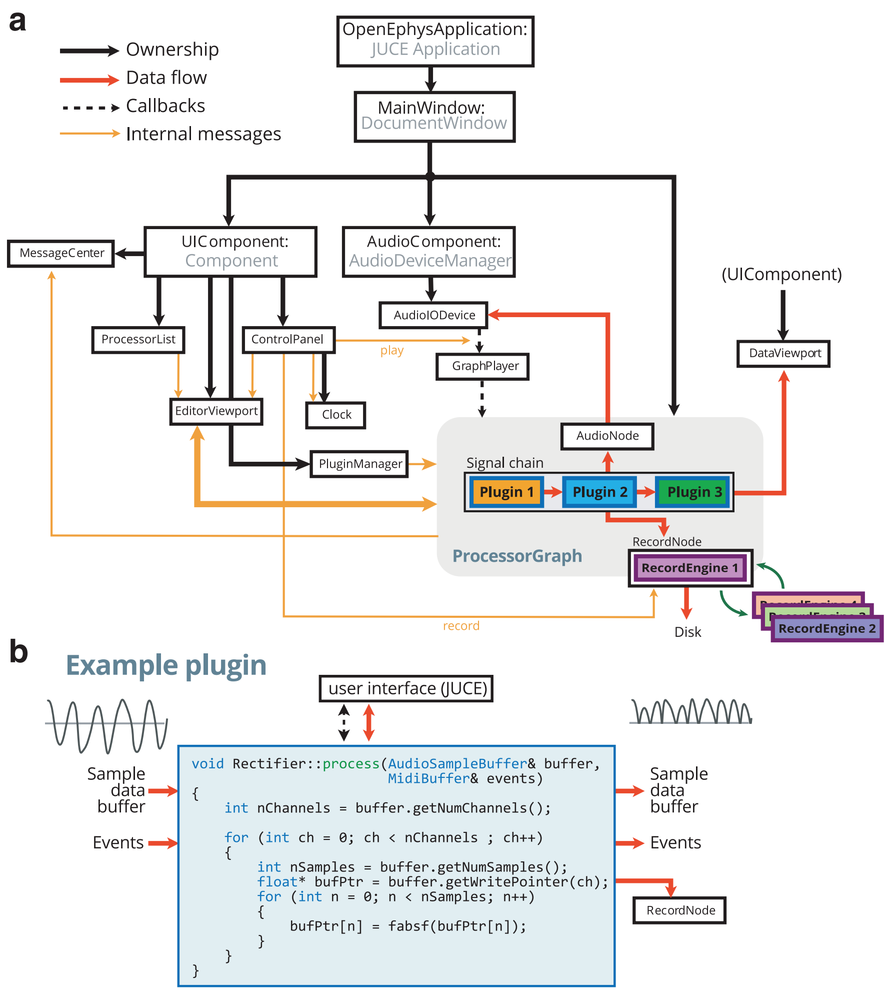
The Open Ephys GUI was designed to be agnostic to the origin of the incoming data. By creating a new Source plugin, it is possible to interface with virtually any hardware device that generates regularly sampled multichannel data. By swapping out the source modules, one can use an identical signal chain with a variety of different inputs.

Although they have the option to develop new Source plugins, most users prefer to collect data with the Open Ephys acquisition board, an open-source interface between up to 8 Intan amplifier chips and a computer's USB port (figure 5(a)). Intan chips encapsulate much of the functionality of traditional data acquisition systems inside a 8×8 mm package [15, 16]. Open Ephys uses Intan's RHD-series chips, first released in 2012, which include a bank of analog filters and amplifiers for each of 32 or 64 channels. The filtered and amplified signals are sent to a multiplexer, which connects them one by one to an analog-to-digital converter. Samples of each channel are represented as 16-bit integers, which are transmitted serially over a wire tether. The use of low-voltage differential signaling (LVDS) facilitates reliable data transmission over long, thin conductors.

Headstages compatible with the Open Ephys system consist of one or more Intan chips, an 'electrode-facing' connector, and a 'tether-facing' connector. The electrode-facing connector must include one conductor for each electrode, which has electrical continuity with one of the inputs on the Intan chip. The tether-facing connector must be a 12-pin Omnetics connector (product #A79623-001) that interfaces with cables conforming to the Intan SPI standard (www.intantech.com/RHD2000_SPI_cables.html).

Most of the existing headstage designs (either from Open Ephys or Intan) use 16 or 32-channel Omnetics connectors with 0.025″ pitch on the electrode-facing end. These connectors are already widely adopted in neuroscience, due to the reliability of their connection over a high number of mating cycles. Because they use the same reference and ground configuration, these headstages can be immediately swapped in for headstages made by Plexon, Neuralynx, Blackrock, Ripple, and Triangle Biosystems. Because the headstage designs are open source, it is possible to create versions that interface with other types of connectors. Recently, an isolation and waterproofing system was developed to make Intan-based headstages safe to use in clinical applications with high-density ECoG arrays [17].

The Open Ephys acquisition board provides four headstage ports, each of which can interface with up to two Intan chips. If eight 64-channel chips are used, a total of 512 channels can be recorded simultaneously. Data acquisition is driven by an Opal Kelly XEM-6310 FPGA module, running a modified version of Intan's Rhythm FPGA firmware



```
void Rectifier::process(AudioSampleBuffer& buffer,
                        MidiBuffer& events)
{
    int nChannels = buffer.getNumChannels();

    for (int ch = 0; ch < nChannels ; ch++)
    {
        int nSamples = buffer.getNumSamples();
        float* bufPtr = buffer.getWritePointer(ch);
        for (int n = 0; n < nSamples; n++)
        {
            bufPtr[n] = fabsf(bufPtr[n]);
        }
    }
}
```

Figure 4. Internal structure of the Open Ephys GUI. (a) Diagram of the major C++ classes that comprise the GUI. Words in black are names defined by the GUI; words in grey are JUCE classes on top which these components are built. The 'ProcessorGraph' is the class in which the signal chain is built. Developers can add new functionality by creating new plugins (Sources, Filters, or Sinks) for the ProcessorGraph, without needing to understand the structure of the rest of the application. (b) Example plugin code for implementing a rectifier. Only the code that is shown in the example needs to be edited to implement a minimal working plugin.

(https://github.com/open-ephys/rhythm). The FPGA ensures that data acquisition is synchronized between all of the Intan chips, and serializes the data for transmission via a USB 3.0 port.

The acquisition board can be synchronized with external devices via I/O boards, which are connected to the main board via standard HDMI cables. The I/O boards provide BNC terminal connectors for up to eight $\pm$ 5 V analog signals, or eight 5 V digital signals. These auxiliary inputs are sampled at the same rate as the neural data, typically 30 kHz.

The Open Ephys acquisition board has been used by over 100 labs to collect data from a variety of model organisms (figure 5(b)). Based on feedback from these users, as well as direct comparisons with commercial data acquisition hardware (www.open-ephys.org/blog/2014/7/9/open-ephys-and-neuralynx-a-head-to-head-comparison), we are confident that Open Ephys data quality is at least as good as that of proprietary systems. In addition, many commercial vendors now use Intan chips to drive data acquisition, meaning their analog signal processing front-end is identical to that of Open Ephys.

Application: closed-loop stimulation of hippocampus

The GUI's real-time feedback engine is built on top of JUCE's audio processing library, which can handle complex floating point data processing steps in real time. The requirements for audio processing—in terms of sample rate, bit depth, and channel count—are in a similar range as those for neural data. We were therefore able to develop closed-loop stimulation algorithms on top of the existing JUCE classes. The basic approach involves creating a Filter plugin that can detect events in the neural data, then using a Sink plugin to control external hardware capable of delivering feedback to the brain.

One of the first experiments we carried out demonstrated the efficacy of Open Ephys for closed-loop stimulation (figure 6(a)). In the lab of Matthew Wilson at MIT, we created a software module that could detect different phases of the hippocampal theta rhythm in real time, and deliver optogenetic stimulation with a ~20 ms delay (1/6th of an ~8 Hz theta cycle). Closed-loop feedback allowed us to probe the function of theta rhythms with enhanced precision relative

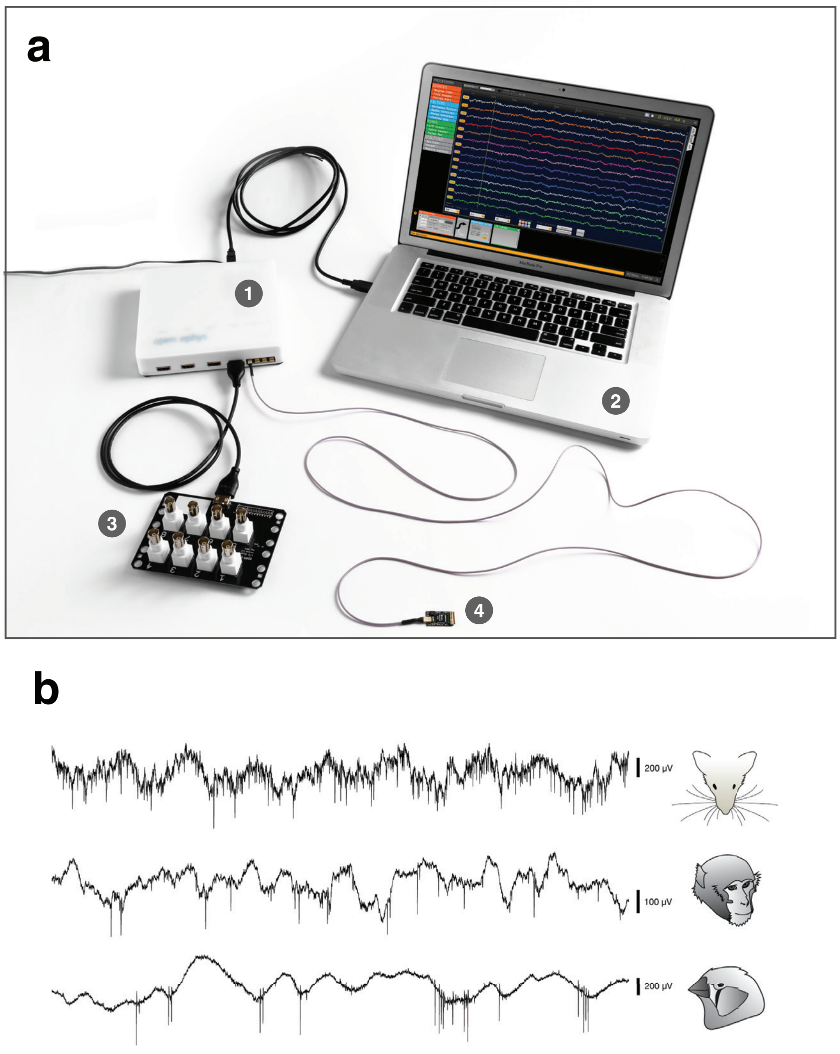
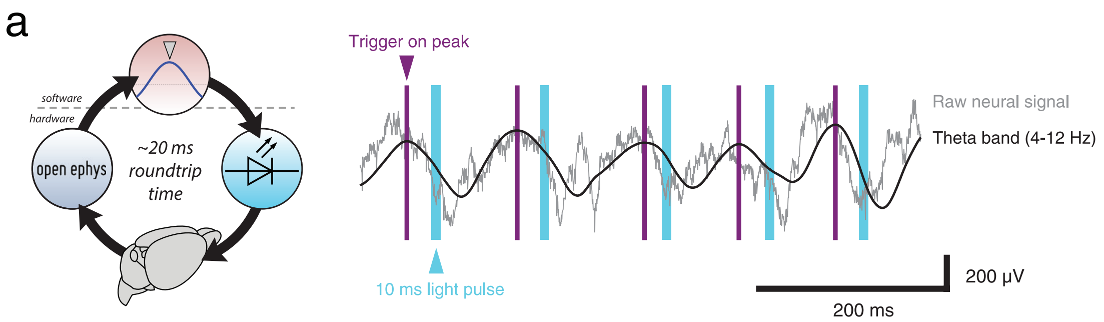


Figure 5. Open Ephys hardware and example recordings. (a) A typical hardware configuration include an acquisition board (1), computer running the Open Ephys GUI (2), an I/O board for synchronization with external devices (3), and a headstage containing an Intan chip that interfaces with electrodes implanted in the brain (4). Photo credit: Jeff Henkel. (b) 1 s of example data from three model organisms: mouse barrel cortex (top), macaque neocortex (middle), and zebra finch LMAN (bottom).

to previous open-loop interventions [4]. Carrying out this experiment involved the use of four plugins: (1) a Rhythm FPGA module, to acquire neural data from the Open Ephys acquisition board; (2) a Bandpass Filter module, to filter the incoming data in the theta range (4–12 Hz); (3) a Phase Detector module, to emit events at the peaks and troughs of the theta wave; and (4) a Pulse Pal module, to convert GUI events into 10 ms pulses capable of driving an external blue LED (Plexon). The LED was coupled to a fiber optic cable that terminated near the recording site, which resulted in phase-specific optogenetic stimulation of hippocampal inhibitory interneurons.

Minimum closed-loop latencies are constrained by both the size of the software buffer and the USB communication protocol. The software buffer determines the number of continuous data samples that are delivered to the plugins on each processing cycle. The default buffer size is 21 ms, but it can be manually configured at runtime to be anywhere between 3 ms and 42 ms. Decreasing the buffer size will lower the upper bound on closed-loop response times, but will increase the chance that processing will not be completed on a given callback. If the signal chain includes a high number of channels or complex closed-loop algorithms, a larger buffer may be necessary. To ensure that all buffers can be processed safely, the GUI includes a visual CPU meter that displays the amount of time spent processing each buffer as a percentage of the overall buffer size. So, if it takes 3 ms to process a 21 ms buffer, the CPU meter will be at 14%.

In addition to the software buffer, closed-loop latencies also limited by delays inherent in the USB communication protocol. When using the Open Ephys acquisition board, data is sent to the GUI in 10 ms chunks. Thus, even when a software buffer of 5 ms is used, mean closed-loop latency is still around 10 ms (figure 6(b)). There is substantial handshaking overhead involved in each transfer, so lowering the chunk size does not lead to a decrease in latency. This limitation can be overcome by using a faster hardware data transfer interface, such as Ethernet or PCI express.



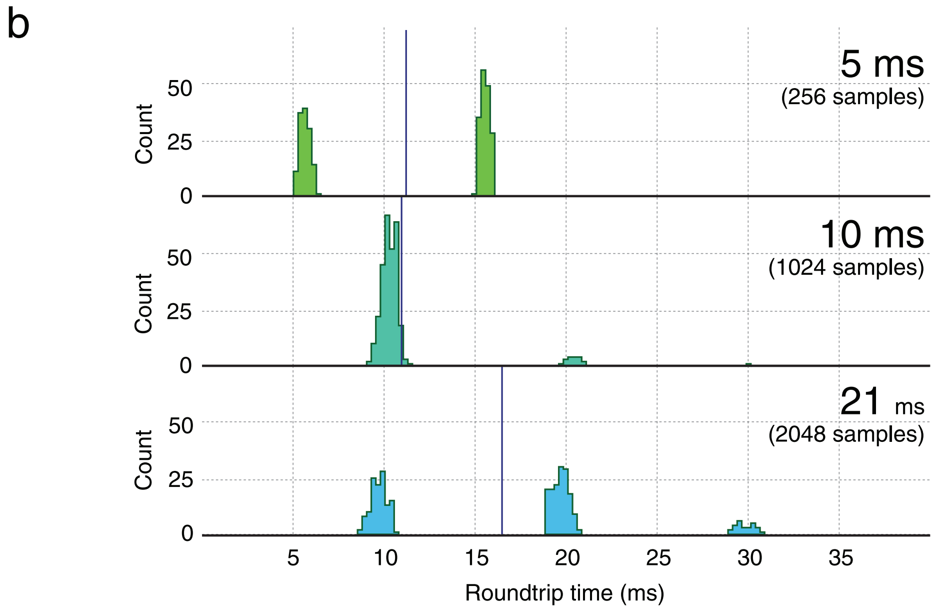


Figure 6. Closed-loop feedback with the GUI. (a) The first published closed-loop experiment carried out with Open Ephys. An electrode implanted in the CA1 region of mouse hippocampus was used to trigger an LED light pulse that activated local inhibitory neurons on the peak of the theta oscillation. The flexibility of the Open Ephys software made it straightforward implement the closed-loop algorithm for detecting peaks of the theta wave. Figure adapted from [4]. CC BY 4.0. (b) Histograms of round-trip (event-to-stimulation) latencies for different software buffer sizes. Mean latencies are shown as vertical blue lines. Input signal: 100 Hz sine wave passed through a 128-channel silicon probe in saline, sampled for 30 s at 30 kHz with the Open Ephys acquisition board. Stimulation occurred on each peak of the sine wave (Phase Detector plugin) using an Arduino Uno communicating via USB (Arduino Output plugin). Clustering around two to three values is likely due to the use of a regularly spaced input signal (100 ms between peaks).

Although implementing closed-loop algorithms on data transmitted via USB involves inherent and uncertain delays of approximately 20 ms, these latencies are acceptable for a wide range of applications. The ability to trigger stimulation based on brain states opens up a large class of experiments that are not accessible to the typical neuroscientist. These include protocols for implementing brain machine interfaces [18, 19], adaptive sampling of a stimulus space [20, 21], and entrainment or disruption of intrinsic oscillations [6, 22–24]. Making closed-loop experiments feasible for a wider range of researchers was one of the primary goals of developing the Open Ephys GUI. Having real-time access to data in software also makes it easier to prototype feedback algorithms that can later be transferred to hardware. And for certain types of experiments, such as real-time decoding of spike trains, hardware implementations are impractical due of the complexity of the generalized linear models or Bayesian inference algorithms required [25, 26].

Advantages and disadvantages of Open Ephys

For labs looking to purchase a new multichannel electrophysiology system, Open Ephys offers three advantages over its closed-source commercial counterparts:

- **Low cost.** A complete Open Ephys system can be obtained for less than $100 per channel, compared to commercial systems costing $1000 per channel or more. For labs on a tight budget, Open Ephys may be the only option for setting up high-channel-count experiments. For labs that want to add recording capabilities to multiple rigs in parallel, our system is an attractive choice. As the throughput of systems neuroscience research continues to expand—both in terms of the number of simultaneously recorded channels, and the number of subjects per experiment—Open Ephys may be the best choice for growing a lab's electrophysiology resources.

- **Transparency.** Because the designs are freely available, Open Ephys encourages scientists to look 'under the hood' and understand the details of its implementation. This not only makes for a better-educated user base, but also alleviates the dependency on a particular company to upgrade functionality or fix bugs. The one caveat to this is the Intan amplifier chips in the headstages, which do contain highly customized proprietary technology. But this is also a problem for the numerous commercial companies that are using Intan chips in their hardware.
- **Flexibility.** Not only is our hardware and software completely open source, but it was designed from the start with modularity in mind. The acquisition board is compatible with many types of headstages, and the software is built around processor modules that can be swapped in and out independently. Hardware and software modules designed by various labs can be made immediately accessible to the broader community, reducing the amount of time spent on redundant development. It is highly possible that, in the future, Open Ephys hardware will be widely used with a different piece of software. Or, the hardware could be made obsolete by new technologies, but the software will live on as a separate entity.

Of course, there are several drawbacks of our platform that must be taken into consideration:

- **No guarantee of support.** Open Ephys has grown its user base substantially in recent years, but we do not have any full-time employees capable of providing support. However, there are a number of proficient users around the world who are often happy to volunteer their time to answer questions posted on GitHub or the Open Ephys mailing list.
- **Developed by amateur engineers.** Open Ephys was developed by neuroscientists for neuroscientists, which means some of the design decisions may not be optimal from an engineering perspective. Anyone who chooses to use our system must accept responsibility for validating the features they plan to utilize. This should not be a substantial burden, but labs that aren't willing to put in the extra consideration may be better off buying a commercial system.
- **Steeper learning curve.** Most commercial recording systems have a strictly enforced workflow, which enables users to acquire data by pressing a single button. In its default state, the Open Ephys GUI cannot record data without adding at least one processor to the signal chain; many other common functions require selecting the appropriate plugins in the correct order. This may not be intuitive to users accustomed to pre-configured software.
- **Performance hit from modular architecture.** The plugin-based design of the Open Ephys GUI carries some degree of overhead, likely around a few milliseconds per buffer, although this will be highly context-dependent. Building the GUI with a more stripped-down architecture would speed up each processing cycle, but at the cost of reduced cross- modularity. Programming the GUI for use with a real-time operating system, as RTXI has done in their closed-loop electrophysiology software (see next section), would further reduce processing latencies, but would force us to forgo cross-platform compatibility.

Comparison to other open-source data acquisition platforms

How does Open Ephys compare to other open-source recording platforms? The most similar is NeuroRighter implemented by the Potter Lab at Georgia Tech, but no longer under development [27, 28]. Like Open Ephys, NeuroRighter offers open-source hardware and software optimized for multielectrode recordings and closed-loop stimulation. NeuroRighter is more mature, having been in use for over 7 years. However, there are two aspects of NeuroRighter that make it less flexible than Open Ephys: its reliance on National Instruments digitization hardware, and its use of the C# programming language, which is Windows-specific. The former also makes the system more costly: NeuroRighter costs around \$10 000 for 64 channels, whereas a 64-channel Open Ephys system can be purchased for less than \$3000. The use of Intan chips in our headstages makes Open Ephys more compact and more affordable. Nevertheless, the success of NeuroRighter was an inspiration during the early days of developing Open Ephys, especially their commitment to making tools for delivering closed-loop feedback more accessible.

The real-time experiment interface (RTXI) [29] is another mature (10+ years) open-source data acquisition platform available for intracellular and extracellular electrophysiology (http://rtxi.org). RTXI provides hard real-time closed-loop control, and has a complete plugin architecture and development API, with which users have contributed over 50 modules (http://github.com/rtxi). RTXI's core is written in C and uses the Qt/QwT GUI frameworks. RTXI's plugin architecture is written in C++ and has many similarities to the Open Ephys plugin architecture. A C++ class called DefaultGUIModel is abstracted by each custom plugin and provides virtual methods that can be overloaded to customize each state the plugin can be in (initialization, execution, pause, unload/halt) and the plugin's input/output connections. DefaultGUIModel automatically generates a GUI for the custom plugin when compiled, making it easier for novice users to create their own custom plugins quickly. Users interested in adding additional online data visualization elements can easily incorporate stock elements from the Qt and QwT GUI frameworks.

In RTXI, unlike Open Ephys, modules can be compiled and dynamically loaded and inserted into the signal chain without halting execution of the system. Template Makefiles are provided to users to simplify the process of compiling custom plugins. Loading and unloading of modules into the RTXI workspace is managed by the core PluginManager class, which provides a thread-safe mechanism for modifying the signal chain on-the-fly. Connections between loaded modules and the data acquisition card channels are created by connecting an output data stream from the data acquisition card or a loaded module to the input data stream of another module of data acquisition card channel. RTXI is feature-rich and flexible, with a mature plugin architecture and development API. However, its key limitation is the dependence upon commercial hardware (e.g. National Instruments) for data acquisition.

More recently, the Boyden Lab at MIT, in collaboration with LeafLabs, developed an open-source, *open-loop* 1024-channel recording system called Willow [30]. Willow also uses Intan chips to handle the front-end amplification and filtering, but it sends the data directly to a solid-state drive, bypassing the need for a separate computer to acquire data. A copy of the data can also be sent over Ethernet for real-time visualization. Willow is optimized for reliability when recording high channel counts, but it lacks the modularity and extensibility of Open Ephys at the software level.

At least two commercial data acquisition platforms provide open-source software: Ripple, LLC (http://rippleneuro.com) and SpikeGadgets (www.spikegadgets.com/main/home.html). This makes it easier for users to modify the software to suit their needs, and to better understand how their data is being processed. However, without a modular, plugin-based architecture, modifications are not as straightforward to make or share as they are with Open Ephys.

A key advantage of open-source tools is that they can generally be made to interoperate. If someone had the need for it, they could make Open Ephys interface with any of the platforms listed above. Our hardware is meant to be hacked, so we have made the necessary details of the communication protocols available to all. Likewise, the software was intended to be used with a variety of data sources. Creating a module to interface with a different type of hardware is a straightforward process, given adequate knowledge of C++.

Future directions

Our biggest challenge is further lowering the barrier to entry for creating new plugins for the Open Ephys GUI. Currently, developing a new module requires knowledge of C++, a low-level programming language that is not typically taught to neuroscientists. Researchers in systems neuroscience are more often familiar with high-level languages, such as Python, Julia, and Matlab. By making plugin development compatible with these languages, Open Ephys has the opportunity to take advantage of the massive amount of existing analysis code. Our software is already plugin-based and highly modular. Once we make it interface with programming languages everyone already uses for data analysis, this analysis code can be turned into modules for running closed-loop experiments.

Another limitation of the Open Ephys platform is the relatively high closed-loop roundtrip times necessitated by our use of the USB port for data transmission (~20 ms). By switching to a PCI express (PCIe) interface, which is included in most desktop computers and has a much higher bandwidth than USB, it will be possible to reduce roundtrip times to less than a millisecond. A prototype PCIe-based acquisition system is currently under development, which will be able to stream thousands of channels to the GUI with latencies in the hundreds of microseconds. Having access to the data on these timescales opens up even more possibilities for closed-loop feedback [31].

In general, our goal for the future is to make it as simple as possible for scientists to create new plugins for the GUI and to modify its existing functionality. Plugins should be able to be written in high-level languages and provide access to the data within a few samples, rather than the ~20 ms blocks of samples that are currently required. Given the convenience of a plugin architecture for building and sharing real-time analysis algorithms, we hope that Open Ephys will help to lower the barriers for electrophysiologists who wish to carry out closed-loop experiments in their labs.

Acknowledgments

We wish to thank the founders of the Allen Institute for Brain Science, Paul G. Allen and Jody Allen, for their vision, encouragement and support. We wish to thank Caleb Kemere, Christopher Moore, and Matthew Wilson for their work on the Open Ephys board of directors. We would like to thank Konstantinos Meletis and Marie Carlen for their donations to Open Ephys early in the development process. We thank Filipe Carvalho of the Champalimaud Institute and Andrew Seddon and Melissa Hough of CircuitHub, Inc. for helping with hardware distribution. We would like to thank Adam Kampff and Gonçalo Lopes for organizing open-source discussions. We thank Galen Lynch for zebrafinch recordings and Jon Newman for advice and inspiration. Finally, we would like to thank all users and developers for their invaluable contributions to testing and improving the Open Ephys toolkit.

Conflict of interest

JHS and JV are co-founders and board members of Open Ephys, Incorporated, a Massachusetts-based nonprofit. They do not receive any financial compensation as a result of their involvement. Revenue from the sale of Open Ephys acquisition boards is used to fund an annual stipend for ACL. KA was mentored by JHS as part of Google Summer of Code 2016, which provided a stipend for KA.

References

[1] Gewaltig M O and Cannon R 2014 Current practice in software development for computational neuroscience and how to improve it *PLoS Comput. Biol.* **10** e1003376

[2] Ego-Stengel V and Wilson M A 2010 Disruption of ripple-associated hippocampal activity during rest impairs spatial learning in the rat *Hippocampus* **20** 1–10

[3] Jadhav S P, Kemere C, German P W and Frank L M 2012 Awake hippocampal sharp-wave ripples support spatial memory *Science* **336** 1454–8

[4] Siegle J H and Wilson M A 2014 Enhancement of encoding and retrieval functions through theta phase-specific manipulation of hippocampus *eLife* **3** e03061

[5] Fong M F, Newman J P, Potter S M and Wenner P 2015 Upward synaptic scaling is dependent on neurotransmission rather than spiking *Nat Commun.* **6** 6339

[6] Ngo H V, Martinetz T, Born J and Mölle M 2013 Auditory closed-loop stimulation of the sleep slow oscillation enhances memory *Neuron* **78** 545–53

[7] DeSantis D, Gallagher I, Haywood K, Knudsen R, Behles G, Rang J, Henke R and Slama T 2013 Ableton Reference Manual Version 9 (https://www.ableton.com/en/manual/credits/)

[8] Gray C M, Maldonado P E, Wilson M and McNaughton B 1995 Tetrodes markedly improve the reliability and yield of multiple single-unit isolation from multi-unit recordings in cat striate cortex *J. Neurosci. Methods* **63** 43–54

[9] Harris K D, Henze D A, Csicsvari J, Hirase H and Buzsáki G 2000 Accuracy of tetrode spike separation as determined by simultaneous intracellular and extracellular measurements *J. Neurophysiol.* **84** 401–14

[10] Csicsvari J, Henze D A, Jamieson B, Harris K D, Sirota A, Barthó P, Wise K D and Buzsáki G 2003 Massively parallel recording of unit and local field potentials with silicon-based electrodes *J. Neurophysiol.* **90** 1314–23

[11] Blanche T J, Spacek M A, Hetke J F and Swindale N V 2005 Polytrodes: high-density silicon electrode arrays for large-scale multiunit recording *J. Neurophysiol.* **93** 2987–3000

[12] Pachitariu M, Steinmetz N, Kadir S, Carandini M and Harris K D 2016 Kilosort: realtime spike-sorting for extracellular electrophysiology with hundreds of channels bioRxiv to be published, https://doi.org/10.1101/061481

[13] Teeters J L 2015 Neurodata without borders: creating a common data format for neurophysiology *Neuron* **88** 629–34

[14] Rossant C *et al* 2016 Spike sorting for large, dense electrode arrays *Nat Neurosci.* **19** 634–41

[15] Harrison R R and Charles C 2003 A low-power low-noise CMOS amplifier for neural recording applications *IEEE J. Solid-State Circuits* **38** 958–65

[16] Harrison R R 2008 A versatile integrated circuit for the acquisition of biopotentials *Custom Integrated Circuits Conf. 2007* (IEEE) pp 115–22

[17] Hermiz J, Rogers N, Kaestner E, Ganji M, Cleary D, Snider J, Barba D, Dayeh S, Halgren E and Gilja V 2016 A clinic compatible, open source electrophysiology system *IEEE 38th Annual Int. Conf. of the Engineering in Medicine and Biology Society* https://doi.org/10.1109/EMBC.2016.7591730

[18] Schwartz A B 2004 Cortical neural prosthetics *Annu. Rev. Neurosci.* **27** 487–507

[19] Hatsopoulos N G and Donoghue J P 2009 The science of neural interface systems *Annu. Rev. Neurosci.* **32** 249–66

[20] Paninski L 2005 Asymptotic theory of information-theoretic experimental design *Neural Comput.* **17** 1480–507

[21] Benda J, Gollisch T, Machens C K and Herz A V 2007 From response to stimulus: adaptive sampling in sensory physiology *Curr. Opin. Neurobiol.* **17** 430–6

[22] Berényi A, Belluscio M, Mao D and Buzsáki G 2012 Closed-loop control of epilepsy by transcranial electrical stimulation *Science* **337** 735–7

[23] Paz J T, Davidson T J, Frechette E S, Delord B, Parada I, Peng K, Deisseroth K and Huguenard J R 2013 Closed-loop optogenetic control of thalamus as a tool for interrupting seizures after cortical injury *Nat. Neurosci.* **16** 64–70

[24] Buetfering C, Allen K and Monyer H 2014 Parvalbumin interneurons provide grid cell-driven recurrent inhibition in the medial entorhinal cortex *Nat. Neurosci.* **17** 710–8

[25] Lawhern V, Wu W, Hatsopoulos N and Paninski L 2010 Population decoding of motor cortical activity using a generalized linear model with hidden states *J. Neurosci. Methods* **189** 267–80

[26] Kloosterman F, Layton S P, Chen Z and Wilson M A 2014 Bayesian decoding using unsorted spikes in the rat hippocampus *J. Neurophysiol.* **111** 217–27

[27] Rolston J D, Gross R E and Potter S M 2009 A low-cost multielectrode system for data acquisition enabling real-time closed-loop processing with rapid recovery from stimulation artifacts *Frontiers Neuroeng.* **2** 12

[28] Newman J P, Zeller-Townson R, Fong M F, Arcot Desai S, Gross R E and Potter S M 2012 Closed-loop, multichannel experimentation using the open-source NeuroRighter electrophysiology platform *Frontiers Neural Circuits* **6** 98

[29] Lin R J, Bettencourt J, White J A, Christini D J and Butera R J 2010 Real-time experiment interface for biological control applications *Engineering in Medicine and Biology Society (EMBC), Annual Int. Conf. of the IEEE* pp 4160–3

[30] Kinney J P *et al* 2015 A direct-to-drive neural data acquisition system *Frontiers Neural Circuits* **9** 46

[31] Voigts J *et al* 2016 A open-source system and interface standards for very high data rate neurophysiology and low latency closed loop experiments *Neuroscience 2016* (San Diego, CA, 12-16 November 2016)