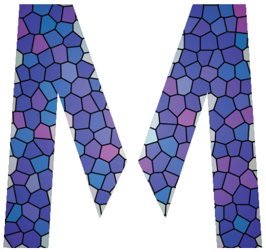


MOSAIC Manual
Release v2.2

NIST
National Institute of
Standards and Technology
U.S. Department of Commerce

Questions/Suggestions

Subscribe to our mailing list: visit https://groups.google.com/a/list.nist.gov/group/mosaic/.
Once subscribed, send messages by emailing mosaic@list.nist.gov.
To unsubscribe: email mosaic+unsubscribe@list.nist.gov.

Report problems with MOSAIC using the issue tracker on GitHub
https://github.com/usnistgov/mosaic/issues

Terms of Use

Disclaimer

MOSAIC Developers

Arvind Balijepalli, *NIST*

Kyle Briggs, *Univeristy of Ottawa*

Jacob Forstater, *NIST*

Joseph Robertson, *NIST*

Canute Vaz

Contents

1 Introduction **1**

2 Data Processing Algorithms in *MOSAIC* **3**
- 2.1 ADEPT 2-State . . . 3
- 2.2 ADEPT . . . 4
- 2.3 CUSUM+ . . . 5

3 Getting Started **7**
- 3.1 Binary Installation . . . 7
- 3.2 Source Installation . . . 7
- 3.3 Docker Installation . . . 14

4 *MOSAIC* GUI **19**
- 4.1 Interface Overview . . . 19
- 4.2 Panels A & B: Analysis Setup and Trajectory Viewer . . . 20
- 4.3 Panels C,D, & E: Blockade Depth Histogram, Statistics, and Event Viewer . . . 23

5 *MOSAIC* WEB **27**
- 5.1 Interface Overview . . . 27
- 5.2 Data Source Path . . . 29
- 5.3 Analysis Settings . . . 30
- 5.4 Analysis Results . . . 32

6 Settings File **35**
- 6.1 Settings Layout . . . 35
- 6.2 Trajectory Settings . . . 36
- 6.3 Optimizing Settings . . . 41
- 6.4 Default Settings . . . 42

7 Database Structure and Query Syntax **45**
- 7.1 Metadata Table . . . 46
- 7.2 Analysis Settings Table . . . 47
- 7.3 Work with SQLite . . . 47
- 7.4 Export to CSV . . . 48

8 Scripting and Advanced Features **51**
- 8.1 Import Data and Run an Analysis . . . 51

8.2 Advanced Scripting . . . 54

9 **Extend *MOSAIC*** **59**
9.1 Read Arbitrary Binary Data Files . . . 59
9.2 Define Top-Level Functionality . . . 61

10 **Publication Quality Figures** **63**
10.1 Timeseries Plots . . . 63
10.2 Histogram Plots . . . 66
10.3 Contour Plots . . . 73

11 **Advanced Analysis** **77**
11.1 Capture Rate . . . 77

12 **Addons** **81**
12.1 Mathematica . . . 81
12.2 Matlab . . . 90
12.3 IGOR . . . 92

13 **Developer Tools** **95**
13.1 Debug Logs . . . 95
13.2 Function Timing and Profiling . . . 97

14 **API Documentation** **99**
14.1 *MOSAIC* Modules . . . 100

Python Module Index **143**

Index **145**

CHAPTER 1

Introduction

MOSAIC is a modular toolbox for analyzing data from single molecule experiments. Primarily developed to analyze data from nanopore experiments [Reiner:2012bg], *MOSAIC* can analyze any data that fit the form [Balijepalli:2014ft]:

$$i(t) = i_0 + \sum_{j=1}^{N} a_j \left(1 - e^{-(t-\mu_j)/\tau_j}\right) H\left(t - \mu_j\right)$$

The above functional form, which represents the response to a step change from one state to another is ubiqutous in many disciplines. By fitting individual state changes to the equation above, *MOSAIC* is able to automatically identify the states corresponding to each change. Moreover this approach allows us to accurately characterize transient events before they asymptotically approach a steady state. In nanopore applications, this has resulted in a 20-fold improvement in the number of states identified per unit time [Balijepalli:2014ft].

MOSAIC offers tremendous flexibility in how it can be used. Nanopore data can be analyzed and visualized using the *MOSAIC GUI* (GUI), which is available as a stand-alone application (download binaries). This is a convenient way for most users to analyze nanopore data. Advanced users can write their own Python scripts to include *MOSAIC* in their analysis workflow (see *Scripting and Advanced Features*). Finally, because *MOSAIC* was designed from the start using object oriented design, developers can easily extend it by combining existing classes to define new functionality or writing their own classes (see *Extend MOSAIC*).

CHAPTER 2

Data Processing Algorithms in *MOSAIC*

There are three primary algorithms available in *MOSAIC* to process time-series data from single-molecule nanopore experiments. Fitting-based approaches are outlined in the *Introduction*, are implemented in *MOSAIC* using two separate algorithms, i) StepResponseAnalysis is used for events that exhibit a single state, and ii) MultistateAnalysis for *N*-state events. In addition, the CUSUM algorithm is available for *N*-state events.

2.1 ADEPT 2-State

This algorithm limits the generalized algorithm for state-detection [Balijepalli:2014ft] to cases with a single state as seen in the figure below. This simplified approach speeds up the analysis considerably and is appropriate to use for many applications, for example the detection of PEG, small molecules, DNA homopolymers, etc. The `adept2State` class uses a simplified form of the expression for the ionic current across a nanopore as shown below. Settings that control the fit are defined through the settings file and are described in more detail in the *Optimizing Settings* section. This functional form is fit to a time-series from a single event to recover optimal parameters for the mdoel.

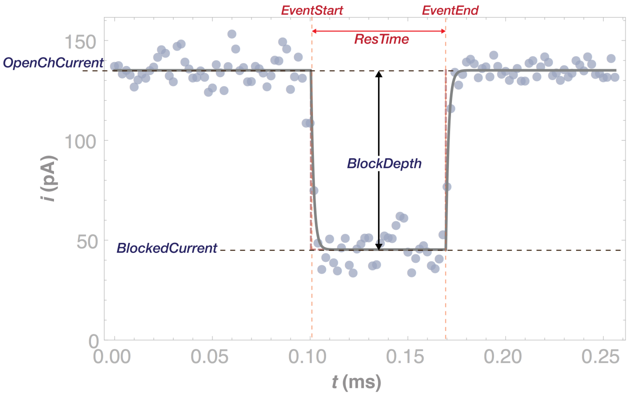
$$i(t) = i_0 + a\left[\left(e^{-(t+\mu_1)/\tau} - 1\right) H\left(t-\mu_1\right) + \left(1 - e^{-(t+\mu_2)/\tau}\right) H\left(t-\mu_2\right)\right]$$

This simplification speeds up the analysis for two state events like the PEG event in the figure below. The figure shows the results of the fit (or meta-data) superimposed on the time-series of a single event.

2.1.1 Algorithm Settings

2.1.2 Metadata Output

Meta-data for individual events generated by `adept2State` can be queried using SQLite as described in the *Database Structure and Query Syntax* section. A list of meta-data stored by the step response algorithm is given below.



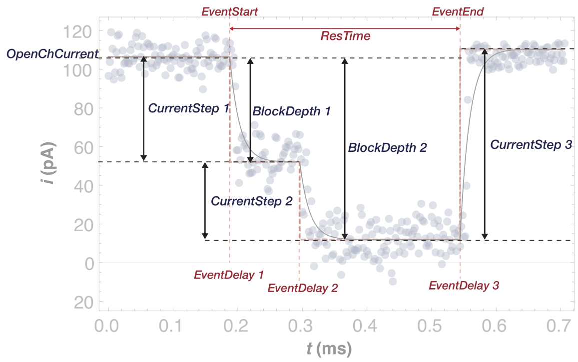
Column Name	Column Type	Description
recIDX	INTEGER	Record index.
ProcessingStatus	TEXT	Status of the analysis.
OpenChCurrent	REAL	Open channel current in pA.
BlockedCurrent	REAL	Blocked state current in pA.
EventStart	REAL	Event start in ms.
EventEnd	REAL	Event end in ms.
BlockDepth	REAL	BlockedCurrent/OpenChCurrent.
ResTime	REAL	EventEnd-EventStart in ms.
RCConstant1	REAL	Downstroke RC constant in ms.
RCConstant2	REAL	Upstroke RC constant in ms.
AbsEventStart	REAL	Global event start time in ms.
ReducedChiSquared	REAL	Reduced Chi-squared of fit.
ProcessTime	REAL	Event processing time in ms.
TimeSeries	REAL_LIST	(OPTIONAL) Event time-series.

2.2 ADEPT

The multistate algorithm implements the general case for identifying states in nanopore data [Balijepalli:2014ft]. The general form of the equation used in this algorithm is shown below, where N is the number of states. This functional form is fit to a time-series from a single event to recover optimal parameters for the mdoel.

$$i(t) = i_0 + \sum_{j=1}^{N} a_j \left(1 - e^{-(t-\mu_j)/\tau_j}\right) H\left(t - \mu_j\right)$$

Settings that control the fit are defined through the settings file and are described in more detail in the *Optimizing Settings* section. Upon successfully fitting the model to an event, `adept` generates meta-data the describes the individual states in the event. A representative example of one such event is shown in the figure below.



2.2.1 Algorithm Settings

2.2.2 Metadata Output

The `adept` algorithm outputs meta-data that characterizes every processed event. Similar to the stepresponse-page algorithm, this information is stored in a SQLite database and is available for further processing (see *Database Structure and Query Syntax*). Notably, the data output by `adept` differs from `adept2State` in one important way. Because the number of states (*NStates*) detected in each event is not pre-determined, key meta-data (e.g. *BlockDepth*, *EventDelay*, etc.) are stored as arrays of real numbers with length equal to *NStates*.

Column Name	Column Type	Description
recIDX	INTEGER	Record index.
ProcessingStatus	TEXT	Status of the analysis.
OpenChCurrent	REAL	Open channel current in pA.
NStates	INTEGER	Number of detected states.
CurrentStep	REAL_LIST	Blocked current steps in pA.
BlockDepth	REAL_LIST	BlockedCurrent/OpenChCurrent for each state.
EventStart	REAL	Event start in ms.
EventEnd	REAL	Event end in ms.
EventDelay	REAL_LIST	Start time of each state in ms.
StateResTime	REAL_LIST	Residence time of each state in ms.
ResTime	REAL	EventEnd-EventStart in ms.
RCConstant	REAL_LIST	System RC constant in ms.
AbsEventStart	REAL	Global event start time in ms.
ReducedChiSquared	REAL	Reduced Chi-squared of fit.
ProcessTime	REAL	Event processing time in ms.
TimeSeries	REAL_LIST	(OPTIONAL) Event time-series.

2.3 CUSUM+

The CUSUM algorithm (used by OpenNanopore for example) [Raillon:2012is] is available in *MOSAIC*. In contrast with other algorithms available in *MOSAIC*, this approach does not leverage system information in the analysis. This however results in a faster estimation of single- and multi-level events, compared with stepresponse-page and *ADEPT*. You can read about the CUSUM algorithm here.

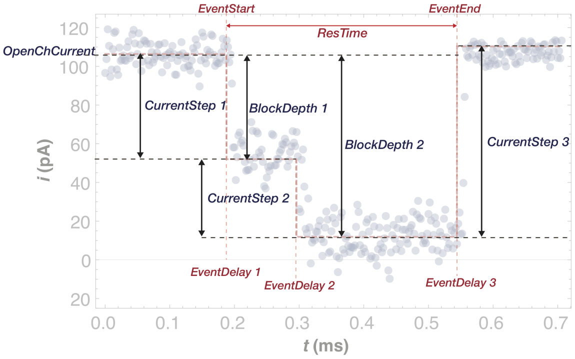
Some known issues with CUSUM:

1. If the duration of a sub-event is shorter than a five RC constants, the averaging will underestimate the extent of the current change. For longer events, CUSUM should achieve very similar output to the fitting employed

elsewhere in *MOSAIC*.

2. CUSUM assumes an instantaneous transition between current states. As a result, if the RC rise time of the system is large, CUSUM can trigger and detect intermediate states. This can usually be mitigated by optimizing the algorithm sensitivity settings.
3. If an event is very long, CUSUM will detect a state transistion even if there is no real change, leading to an artificially high number of states. This is a consequence of false positives from using a statistical t-test. In some cases this can be mitigated by reducing the sensitivity.

Settings that control the algorithm are defined through the settings file, as described the *Optimizing Settings* section. Upon successfully analyzing an event, `cusumPlus` generates meta-data the describes the individual states in the event. A representative example of one such event is shown in the figure below.



2.3.1 Algorithm Settings

2.3.2 Metadata Output

The `cusumPlus` algorithm outputs meta-data that characterizes every processed event. Similar to the *ADEPT* algorithm, this information is stored in a SQLite database and is available for further processing (see *Database Structure and Query Syntax*).

Column Name	Column Type	Description
recIDX	INTEGER	Record index.
ProcessingStatus	TEXT	Status of the analysis.
OpenChCurrent	REAL	Open channel current in pA.
NStates	INTEGER	Number of detected states.
CurrentStep	REAL_LIST	Blocked current steps in pA.
BlockDepth	REAL_LIST	BlockedCurrent/OpenChCurrent for each state.
EventStart	REAL	Event start in ms.
EventEnd	REAL	Event end in ms.
EventDelay	REAL_LIST	Start time of each state in ms.
StateResTime	REAL_LIST	Residence time of each state in ms.
ResTime	REAL	EventEnd-EventStart in ms.
AbsEventStart	REAL	Global event start time in ms.
ProcessTime	REAL	Event processing time in ms.
TimeSeries	REAL_LIST	(OPTIONAL) Event time-series.

CHAPTER 3

Getting Started

3.1 Binary Installation

MOSAIC is available as a pre-compiled binary for Windows and Mac OS X (download binaries). *MOSAIC* binaries do not need special installation. Under **Mac OS X** open the the downloaded disk image and drag the *MOSAIC* executable to the Applications folder. Under **Windows**, unzip downloaded zip file and move the *MOSAIC* executable to your hard disk.

Note: *MOSAIC* binaries are 64-bit. If you need 32-bit support, please build *MOSAIC* from source as described in the *Source Installation* section.

3.2 Source Installation

3.2.1 Install *MOSAIC* on Mac OS X

In the following guide, we provide step-by-step instructions on setting up and running *MOSAIC* on OS X. To simplify the isntallation, we use Homebrew to install some required dependencies. Homebrew requires Apple command line tools, but will directly prompt you to install it on set up.

1. Installing Homebrew

First we will install Homebrew, a useful package manager, to help install some of the dependencies required by *MOSAIC*. You will need administrator access for this step. In the OS X Terminal, run the following command:

```
$  ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/
↪install)"
```

Note, if the Apple command line tools are not installed, Homebrew will prompt you do so during installation.

Hint: To test if Homebrew is properly installed, run the following in the terminal: `brew doctor`

To ensure that Homebrew is set up correctly, add the Homebrew directory to ~/.bash_profile. This can be done using the following command:

```
$  echo 'export PATH="/usr/local/bin:$PATH"' >> ~/.bash_profile
```

Hint: If you don't have a .bash_profile file in your home directory, you can create one manually using a text editor.

Restart the terminal to update your shell.

2. Installing brewed Python and other neccessary packages

MOSAIC is written in Python and utilizes a number of different packages and utilities. In the following we'll install a number of these (specifically, python, gcc, gfortran, qt, and pyQt4). With homebrew this is easy to do in one line! Run the following in the terminal:

```
$  brew install python gcc gfortran qt pyqt
```

At this point, it is a good idea to update the PYTHONPATH environment variable in ~/.bash_profile:

```
$  export PYTHONPATH=$PYTHONPATH:/usr/local/lib/python3.7/site-packages
```

3. (Optional) Install and Setup Virtual Environment

It is generally a good practice to run *MOSAIC* from within a dedicated virtual environment. This minimizes conflicts with other installed programs. While we highly recommend this approach, it is not required to run *MOSAIC*. If you prefer to skip this, move on to the next step now.

To setup a virtual environment, we need two different packages: *virtualenv*, which creates the virtual environments, and *virtualenvwrapper*, a wrapper for *virtualenv* that simplifies set up and use.

To install these and set up the virtual enviroment wrapper, run the following in a shell:

```
$  pip install virtualenv virtualenvwrapper
```

Hint: Under Ubuntu, you may need install virtualenv and virtualenvwrapper as root. Simply prefix the command above with sudo.

If you would like virtualenvwrapper to be available each time you open a new terminal window, add the line below to ~/.bash_profile on OS X or ~/.bashrc on Linux.

```
source /usr/local/bin/virtualenvwrapper.sh
```

Hint: Depending on the process used to install *virtualenv*, the path to virtualenvwrapper.sh may vary. Find the approporiate path by running `$ find /usr -name virtualenvwrapper.sh`. Adjust the line in your .bash_profile or .bashrc script accordingly.

Open a new shell to make the new virtual environment available. Now we are ready to create a virtual environment. You can choose any name for your virtual environment, here we name it *MOSAIC*:

```
$  mkvirtualenv -p <path to python>/python MOSAIC
```

Hint: We explicitly specify the Python installation to use. This is not mandatory, but is useful if you have multiple Python installations on your computer. The *<path to python>* may vary according to the specific version of python you wish to use. In most cases, this will be either */usr/local/bin/* or */usr/bin*

---

4. Installing MOSAIC

Install using Setuptools

The command-line version of *MOSAIC* can be installed using *pip* as shown below. Any additional dependencies required by *MOSAIC* will be installed automatically.

```
pip install mosaic-nist
```

Note: Installing the graphical interface requires one to install *MOSAIC* from the source distribution as outlined below.

Install from a Downloaded Source Distribution

First we need to obtain the *MOSAIC* source code. For analyzing publication data, we recommend downloading the latest stable version of the source code (download source). Alternatively, the latest development version can be downloaded from the MOSAIC page on Github. Here we will show you how to set up *MOSAIC* from the latest stable release:

1. Download the latest release (download source)
2. Create a directory for the project source. In this case we will create a directory called MOSAIC, located in `~/projects/`, where '~' is your home directory.

```
$ mkdir ~/projects/MOSAIC
```

3. Navigate to the directory:

```
$ cd ~/projects/MOSAIC
```

4. Extract the source into this folder.
5. Make sure you are working in the virtual environment we set up in the previous step by typing:

```
$ workon MOSAIC
```

Note: You will notice that (*MOSAIC*) now appears in front of the $ prompt in your shell. This inidicates that the virtual environment is active. We have employed this notation to indicate commands that should be run from inside the virtual environment.

---

6. *MOSAIC* and its dependencies are built using setuptools. Navigate to ~/projects/MOSAIC/ and run the following:

```
(MOSAIC)$ python setup.py mosaic_deps
```

7. Finally, add the installation directory (~/projects/MOSAIC as set up previously) to your *PYTHONPATH* as shown below. This addition can be made permanent by adding the line below to your *.bash_profile* (OS X) or *.bashrc* (Ubuntu) script.

```
(MOSAIC)$ export PYTHONPATH=$PYTHONPATH:~/projects/MOSAIC
```

5. Testing MOSAIC

To test the *MOSAIC* installation, in the *MOSAIC* directory type

```
$  python setup.py test -q
```

Verify that all tests pass as seen below

3.2.2 Install *MOSAIC* on Ubuntu(14.04)

MOSAIC can be run under Ubuntu using a procedure very similar to *Install MOSAIC on Mac OS X*.

1. Prerequisites

Several prerequisites must be installed prior to building *MOSAIC* dependencies. This is easily accomplished in Ubuntu using the *aptitude* package manager.

Hint: *superuser* privileges are needed when installing *MOSAIC* prerequisites.

```
$  sudo apt-get install python python-dev python-pip python-qt4
   pkg-config freetype* gfortran liblapack-dev libblas-dev
```

Next add the following to ~/.bashrc

```
export PYTHONPATH=/usr/lib/python3.7/dist-packages
```

2. (Optional) Install and Setup Virtual Environment

It is generally a good practice to run *MOSAIC* from within a dedicated virtual environment. This minimizes conflicts with other installed programs. While we highly recommend this approach, it is not required to run *MOSAIC*. If you prefer to skip this, move on to the next step now.

To setup a virtual environment, we need two different packages: *virtualenv*, which creates the virtual environments, and *virtualenvwrapper*, a wrapper for *virtualenv* that simplifies set up and use.

To install these and set up the virtual enviroment wrapper, run the following in a shell:

```
$  pip install virtualenv virtualenvwrapper
```

Hint: Under Ubuntu, you may need install virtualenv and virtualenvwrapper as root. Simply prefix the command above with sudo.

If you would like virtualenvwrapper to be available each time you open a new terminal window, add the line below to ~/.bash_profile on OS X or ~/.bashrc on Linux.

```
source /usr/local/bin/virtualenvwrapper.sh
```

Hint: Depending on the process used to install *virtualenv*, the path to virtualenvwrapper.sh may vary. Find the approporiate path by running `$ find /usr -name virtualenvwrapper.sh`. Adjust the line in your .bash_profile or .bashrc script accordingly.

Open a new shell to make the new virtual environment available. Now we are ready to create a virtual environment. You can choose any name for your virtual environment, here we name it *MOSAIC*:

```
$  mkvirtualenv -p <path to python>/python MOSAIC
```

Hint: We explicitly specify the Python installation to use. This is not mandatory, but is useful if you have multiple Python installations on your computer. The *<path to python>* may vary according to the specific version of python you wish to use. In most cases, this will be either */usr/local/bin/* or */usr/bin*

3. Installing MOSAIC

Install using Setuptools

The command-line version of *MOSAIC* can be installed using *pip* as shown below. Any additional dependencies required by *MOSAIC* will be installed automatically.

```
pip install mosaic-nist
```

Note: Installing the graphical interface requires one to install *MOSAIC* from the source distribution as outlined below.

Install from a Downloaded Source Distribution

First we need to obtain the *MOSAIC* source code. For analyzing publication data, we recommend downloading the latest stable version of the source code (download source). Alternatively, the latest development version can be downloaded from the MOSAIC page on Github. Here we will show you how to set up *MOSAIC* from the latest stable release:

1. Download the latest release (download source)
2. Create a directory for the project source. In this case we will create a directory called MOSAIC, located in `~/projects/`, where '~' is your home directory.

```
$ mkdir ~/projects/MOSAIC
```

3. Navigate to the directory:

```
$ cd ~/projects/MOSAIC
```

4. Extract the source into this folder.
5. Make sure you are working in the virtual environment we set up in the previous step by typing:

```
$ workon MOSAIC
```

Note: You will notice that (*MOSAIC*) now appears in front of the $ prompt in your shell. This inidicates that the virtual environment is active. We have employed this notation to indicate commands that should be run from inside the virtual environment.

6. *MOSAIC* and its dependencies are built using setuptools. Navigate to ~/projects/MOSAIC/ and run the following:

```
(MOSAIC)$ python setup.py mosaic_deps
```

7. Finally, add the installation directory (~/projects/MOSAIC as set up previously) to your *PYTHONPATH* as shown below. This addition can be made permanent by adding the line below to your *.bash_profile* (OS X) or *.bashrc* (Ubuntu) script.

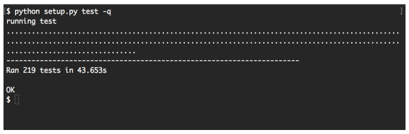
```
(MOSAIC)$ export PYTHONPATH=$PYTHONPATH:~/projects/MOSAIC
```

4. Testing MOSAIC

To test the *MOSAIC* installation, in the *MOSAIC* directory type

```
$  python setup.py test -q
```

Verify that all tests pass as seen below



```
$ python setup.py test -q
running test
...............................................................................................
...............................................................................................
.............................
----------------------------------------------------------------------
Ran 219 tests in 43.653s

OK
$
```

3.2.3 Install *MOSAIC* on Windows

In the following guide, we provide step-by-step instructions on setting up and running *MOSAIC* on Windows. To simplify the isntallation, we use Anaconda to install some required dependencies.

1. Installing Anaconda

First we will install Anaconda to easily install the dependencies required by *MOSAIC*. Download the 64-bit Anaconda installer for Python 3.7 and use the graphical installer.

2. Installing MOSAIC dependencies within Anaconda

MOSAIC is written in Python and utilizes a number of different packages and utilities. In the following we'll install a number of these (specifically, python, gcc, gfortran, qt, and pyQt4). With Anaconda this is easy to do.

First, we create a self-contained environment to host the *MOSAIC* installation. Open the Anaconda prompt (Start Menu–>Anaconda 2–>Anaconda Prompt) and type:

```
$  conda create -n mosaicENV python=3.7
```

Activate the new environment:

```
$  conda activate mosaicENV
```

Add a new installation source (conda-forge) for packages that are not included with Anaconda out of the box:

```
$  conda config --add channels conda-forge
```

Install all the dependencies by typing:

```
$  conda install
                    cython=0.29
                    pandas=0.20.3
                    nose=1.3.7
                    numpy=1.11.1 o
                    scipy=0.18.1
                    docutils=0.14
                    flask=0.12.2
                    matplotlib=1.5.3
                    pyqt=4
                    lmfit=0.9.3
                    uncertainties=2.4.8.1
                    PyWavelets=0.5.2
                    coverage=4.5.1
                    codecov=2.0.15
```

Hint: The latest dependency version numbers can be obtained from the requirements.txt file.

Install the *MOSAIC* source by cloning the Github repository or using one of the methods below.

3. Installing MOSAIC

Install using Setuptools

The command-line version of *MOSAIC* can be installed using *pip* as shown below. Any additional dependencies required by *MOSAIC* will be installed automatically.

```
pip install mosaic-nist
```

Note: Installing the graphical interface requires one to install *MOSAIC* from the source distribution as outlined below.

Install from a Downloaded Source Distribution

First we need to obtain the *MOSAIC* source code. For analyzing publication data, we recommend downloading the latest stable version of the source code (download source). Alternatively, the latest development version can be downloaded from the MOSAIC page on Github. Here we will show you how to set up *MOSAIC* from the latest stable release:

1. Download the latest release (download source)
2. Create a directory for the project source. In this case we will create a directory called MOSAIC, located in `~/projects/`, where '~' is your home directory.

```
$ mkdir ~/projects/MOSAIC
```

3. Navigate to the directory:

```
$ cd ~/projects/MOSAIC
```

4. Extract the source into this folder.
5. Make sure you are working in the virtual environment we set up in the previous step by typing:

```
$ workon MOSAIC
```

Note: You will notice that (*MOSAIC*) now appears in front of the $ prompt in your shell. This inidicates that the virtual environment is active. We have employed this notation to indicate commands that should be run from inside the virtual environment.

6. *MOSAIC* and its dependencies are built using setuptools. Navigate to ~/projects/MOSAIC/ and run the following:

```
(MOSAIC)$ python setup.py mosaic_deps
```

7. Finally, add the installation directory (~/projects/MOSAIC as set up previously) to your *PYTHONPATH* as shown below. This addition can be made permanent by adding the line below to your *.bash_profile* (OS X) or *.bashrc* (Ubuntu) script.

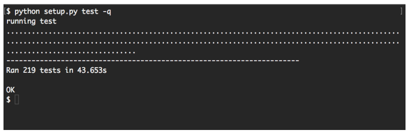
```
(MOSAIC)$ export PYTHONPATH=$PYTHONPATH:~/projects/MOSAIC
```

4. Testing MOSAIC

To test the *MOSAIC* installation, in the *MOSAIC* directory type

```
$  python setup.py test -q
```

Verify that all tests pass as seen below



```
$ python setup.py test -q
running test
..............................................................................................
..............................................................................................
...............................
----------------------------------------------------------------------
Ran 219 tests in 43.653s

OK
$ 
```

3.3 Docker Installation

MOSAIC can be installed using Docker Desktop. This may be desirable in many cases because it provides a consistent experience across all operating systems. Installing *MOSAIC* using Docker is relatively straightforward and requires only a few steps as described below.

1. Install Docker Desktop

Download and install Docker Desktop. Follow instructions for either Windows or Mac OS X installation.

2. Create a configuration file

Copy the text below and paste into a file called `docker-compose.yml`.

Note: The filename and extension should be `docker-compose.yml`.

Modify the data path and log file paths under the `volumes` line appropriately for your PC.

Also note the version number in the `image` line.

Place the `docker-compose.yml` anywhere on your PC.

```
version: '3'
services:
mosaic:
```

(continues on next page)

(continued from previous page)

```yaml
    image: ghcr.io/usnistgov/mosaic:v2.2
    ports:
        - "5000:5000"
    volumes:
        - C:\\Users\\arvind\\Desktop:/src/data
        - C:\\Users\\arvind\\Desktop:/var/logs
```

3. Run MOSAIC using Docker

Open the `Command Prompt` in Windows or `Terminal` on Mac OS.

Hint: You can launch the `Command Prompt` by typing `cmd` in the search bar on Windows.

In the `Command Prompt`, navigate to the directory that you placed the `docker-compose.yml` file. For example if you place the file at `C:\\Users\\arvind\Desktop` then type:

```
c:
cd Users\arvind\Desktop
```

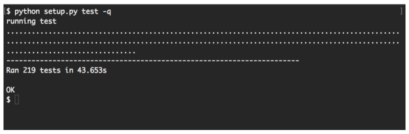
Run the Docker *MOSAIC* image by typing:

```
docker compose up
```

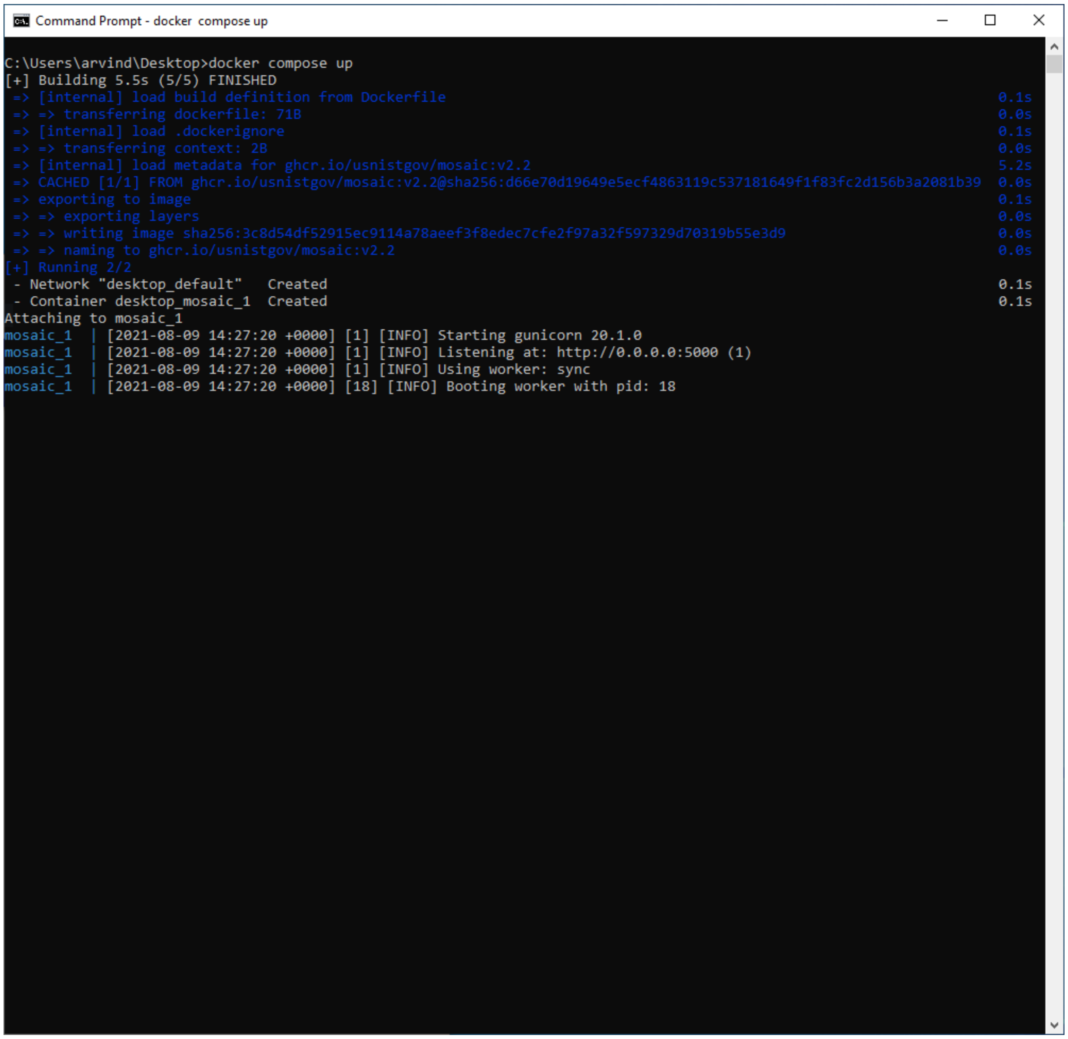
You should see Docker download the image file and start *MOSAIC* similar to the figure below

4. Run MOSAIC Web UI

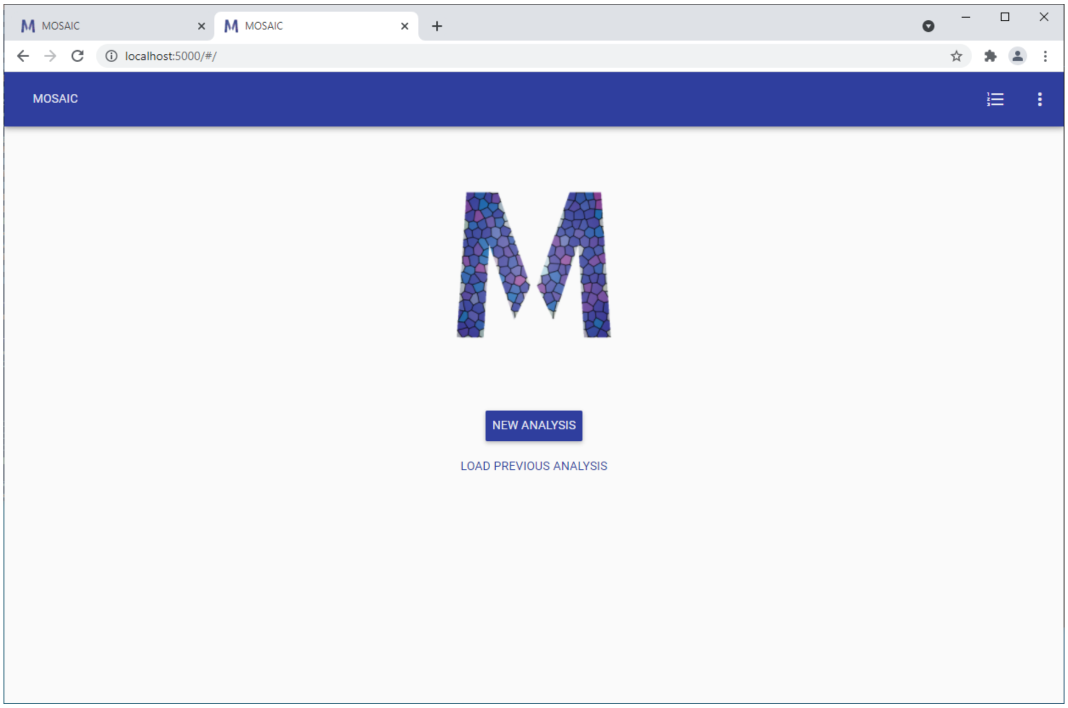
Open a browser and type `localhost:5000` in the address bar. This should load the MOSAIC web UI as shown below.



```
$ python setup.py test -q
running test
..............................................................................................
..............................................................................................
...............................
----------------------------------------------------------------------
Ran 219 tests in 43.653s

OK
$
```



```
C:\Users\arvind\Desktop>docker compose up
[+] Building 5.5s (5/5) FINISHED
 => [internal] load build definition from Dockerfile                                                                        0.1s
 => => transferring dockerfile: 71B                                                                                         0.0s
 => [internal] load .dockerignore                                                                                           0.1s
 => => transferring context: 2B                                                                                             0.0s
 => [internal] load metadata for ghcr.io/usnistgov/mosaic:v2.2                                                              5.2s
 => CACHED [1/1] FROM ghcr.io/usnistgov/mosaic:v2.2@sha256:d66e70d19649e5ecf4863119c537181649f1f83fc2d156b3a2081b39 0.0s
 => exporting to image                                                                                                      0.1s
 => => exporting layers                                                                                                     0.0s
 => => writing image sha256:3c8d54df52915ec9114a78aeef3f8edec7cfe2f97a32f597329d70319b55e3d9                                0.0s
 => => naming to ghcr.io/usnistgov/mosaic:v2.2                                                                              0.0s
[+] Running 2/2
 - Network "desktop_default"   Created                                                                                      0.1s
 - Container desktop_mosaic_1  Created                                                                                      0.1s
Attaching to mosaic_1
mosaic_1  | [2021-08-09 14:27:20 +0000] [1] [INFO] Starting gunicorn 20.1.0
mosaic_1  | [2021-08-09 14:27:20 +0000] [1] [INFO] Listening at: http://0.0.0.0:5000 (1)
mosaic_1  | [2021-08-09 14:27:20 +0000] [1] [INFO] Using worker: sync
mosaic_1  | [2021-08-09 14:27:20 +0000] [18] [INFO] Booting worker with pid: 18
```


MOSAIC
MOSAIC
localhost:5000/#/
MOSAIC
NEW ANALYSIS
LOAD PREVIOUS ANALYSIS

CHAPTER 4

MOSAIC GUI

MOSAIC's GUI interface is designed to allow you to easily setup and run an analysis and to analyze the results of prior trials via a graphical interface; it contains the most commonly used features of *MOSAIC*. The GUI contains modular panels for setting up an analysis, running it, and analyzing the results. Here we give you a brief overview of the graphical interface and its basic use. You can learn more in the examples-page section.

Opening the GUI

If you installed *MOSAIC* from a precomiled binary, you can open the GUI by double clicking the *MOSAIC* icon. Alternatively, if you compiled *MOSAIC* from source code, you can run the GUI from the terminal window – navigate to the installation directory and type:

```
python runMOSAIC
```

Hint: Having trobule getting the GUI to start? Frequently, this arises because your PYTHONPATH environment variable is set up incorrectly. To fix this error, first type `echo $PYTHONPATH` in the terminal. If you don't see the path to the *MOSAIC* installation in *PYTHONPATH*, consult the operating-system specific instructions (*OSX* or *Ubuntu*) to help resolve this issue.

4.1 Interface Overview

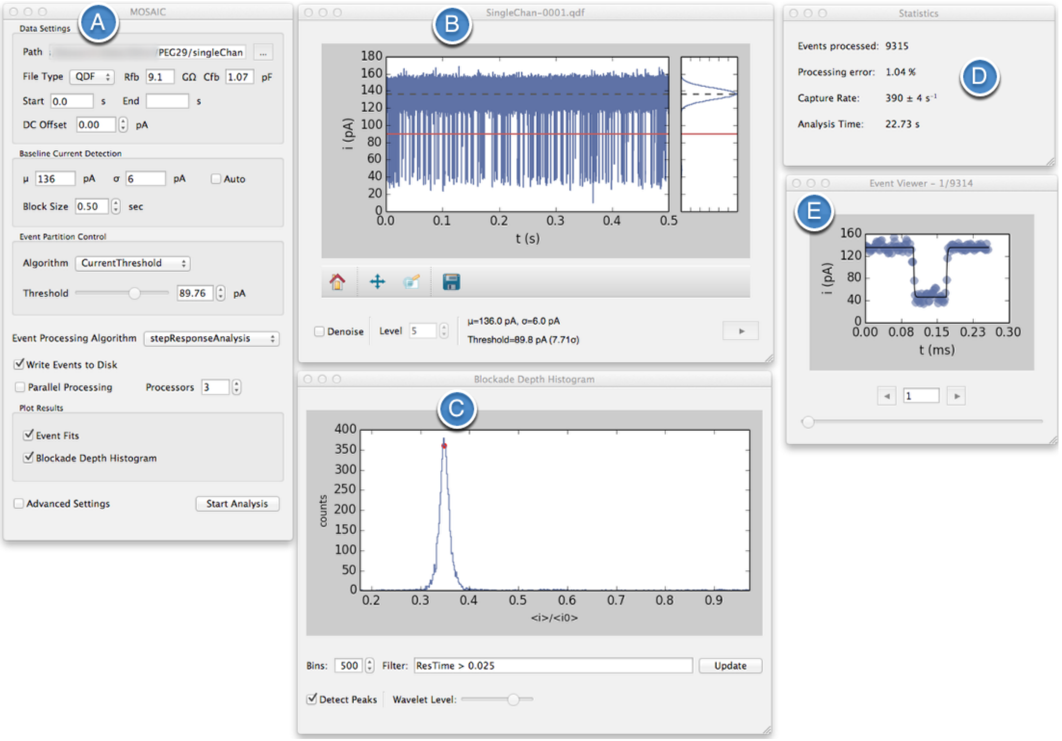
The main interface consists of five panels which we go over in detail later in this document. Briefly, these are:

A) *Analysis Setup*: This panel is used to set up the analysis parameters.

B) *Trajectory Viewer*: This panel shows a snippet of the ionic current time-series and an all points histogram, used to set the baseline and threshold parameters found in *Panel A: Analysis Setup*.

C) *Blockade Depth Histogram*: Once the data processing has started, this panel shows a live blockade depth histogram; a query can be defined to restrict the histogram to data which fulfills a user-defined criteria.

D) *Analysis Statistics*: Displays live statistics about the data processed.



MOSAIC

Data Settings

Path [redacted]PEG29/singleChan

File Type QDF Rfb 9.1 GΩ Cfb 1.07 pF

Start 0.0 s End s

DC Offset 0.00 pA

Baseline Current Detection

μ 136 pA σ 6 pA ☐ Auto

Block Size 0.50 sec

Event Partition Control

Algorithm CurrentThreshold

Threshold 89.76 pA

Event Processing Algorithm stepResponseAnalysis

☑ Write Events to Disk

☐ Parallel Processing Processors 3

Plot Results

☑ Event Fits

☑ Blockade Depth Histogram

☐ Advanced Settings

SingleChan-0001.qdf

☐ Denoise Level 5

μ=136.0 pA, σ=6.0 pA

Threshold=89.8 pA (7.71σ)

Statistics

Events processed: 9315

Processing error: 1.04 %

Capture Rate: 390 ± 4 s^{-1}

Analysis Time: 22.73 s

Event Viewer – 1/9314

Blockade Depth Histogram

Bins: 500 Filter: ResTime > 0.025

☑ Detect Peaks Wavelet Level:

Fig. 1: Primary panels in *MOSAIC*: (A)Analysis Setup (B) Trajectory Viewer (C) Live Blockade Depth Histogram (D) Live Analysis Statistics (E) Event Viewer.

E) *Event Viewer*: Displays the partitioned events and their fit. This panel is active only if "Write Events to Disk" is enabled in the *Analysis Setup*.

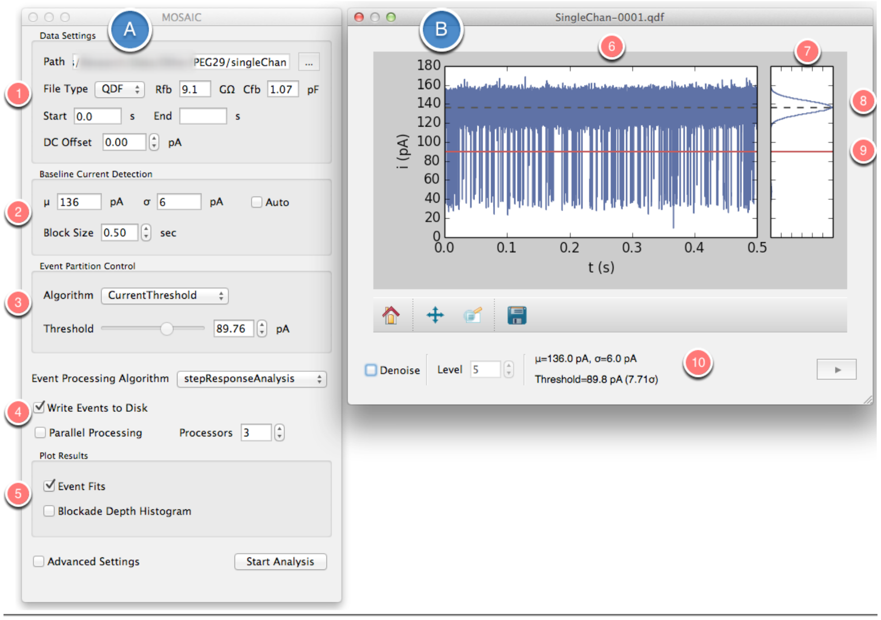
4.2 Panels A & B: Analysis Setup and Trajectory Viewer

4.2.1 Panel A: Analysis Setup

1. Data Settings

- **Path**: Allows user to set the directory containing files to analyze. Click the "…" icon to navigate to the directory.
- **File Type**: The GUI is natively compatible with either ABF or QDF Files, this field is automatically populated based on the files in the directory you've chosen.The **Rfb** and **Cfb** parameters are needed to correctly analyze QDF files (see `qdfTrajIO` for more information)
- **Rfb & Cfb**: *MOSAIC* supports the QUB QDF file format used by the Electronic Biosciences Nanopatch system. Two additional parameters, the feedback resistance (Rfb) in Ohms and capacitance (Cfb) in Farads are required to appropriately convert the measurements to ionic current.
- **Start** and **End**: These parameters allow you to analyze a range of your data. Choose the starting and ending times if you'd like to analyze a small time segement of your data. If this is left blank, all data will be analyzed.
- **DC Offset**: If your measurement contains a systematic bias, it can be manually corrected by entering the DC offset here.

2. Baseline Current Detection



Path [redacted]PEG29/singleChan
File Type QDF Rfb 9.1 GΩ Cfb 1.07 pF
Start 0.0 s End s
DC Offset 0.00 pA
μ 136 pA σ 6 pA ☐ Auto
Block Size 0.50 sec
Algorithm CurrentThreshold
Threshold 89.76 pA
Event Processing Algorithm stepResponseAnalysis
☑ Write Events to Disk
☐ Parallel Processing Processors 3
☑ Event Fits
☐ Blockade Depth Histogram
☐ Advanced Settings
☐ Denoise Level 5
μ=136.0 pA, σ=6.0 pA
Threshold=89.8 pA (7.71σ)

Fig. 2: *Overview of Panels A & B*: (A) Analysis setup panel (B) Trajectory viewer panel

- μ: Mean baseline current, in picoamperes (pA). This is shown schematically in the trajectory viewer (see Label #8). When *Auto* is selected, this will be greyed out and labeled `<auto>`
- σ: Noise level (in pA). This is expected noise level of your baseline. Typically one would set this to the measured RMS noise of the open channel state at the cutoff frequency. When *Auto* is selected, this will be greyed out and labeled `<auto>`.
- **Auto**: Checking this box enables automatic dectection of the mean baseline current (μ) and noise level (σ). When auto is enabled, the values chosen by the software will be displayed in the trajectory viewer panel (see Label #10)
- **Block Size**: Controls the amount of data examined to determine the baseline. This also controls the amount of data shown in the trajectory viewer.

3. Event Partition Control

This panel is used to set the current threhold used for event detection

- **Algorithm**: Currently,the only event partitioning algorithm enabled is *CurrentThreshold*.
- **Threshold**: This is used to set the minimum current threshold used to partition events with the *CurrentThreshold* algorithm.

4. Event Processing Setup

Event Processing Algorithm: The GUI supports two event processing algorithms, i) *StepResponseAnalysis* and ii) *MultiStateAnalysis*. *StepResponseAnalysis* is the default analysis, and should be used with data sets with unimodal events. For events with mutliple states or steps the *MultiStateAnalysis* algorithm, which is capable of automatically analyzing events with *N* states, should be used. Note that *StepResponseAnalysis* is a restricted case of *MultiStateAnalysis* and is more computationally efficient to run if you have unimodal (or single states) data.

- **Write Events to Disk**: When this box is checked, the data points for each partition events are written to the SQLite database. When this is checked it is possible to view the individual fits of each in the *Event Fits* panel.

Hint: When *Write Evens to Disk* is checked, your database can become extremely large! This is because *MOSAIC* is effectively writing most of your time-series to the database. Note that the fit parameters are *always* written to the database.

- **Parallel Processing** and **Processors**: Parallel processing can be enabled by checking this box. This box will be greyed out if the python module ZeroMQ is not installed. The *Processors* box allows you to select the number of processors used in the analysis. It is important to note that the GUI will occupy one processor, so choosing 3 processors will actually use a total of 4 processors.

5. Plot Results and Advanced Settings

- **Event Fits**: Checking this box will show the events viewer (Panel E). This can also be accessed from the file menu `View>Plots>Event Fits`. If *Write Events to Disk* is not enabled this checkbox will be greyed out.
- **Blockade Depth Histogram**: Checking this box will show the blockade depth histogram (Panel C). This can also be accessed through the file menu `View>Plots>Blockade Depth Histogram`.
- **Advanced Settings**: This opens a dialog window to manually edit settings not otherwise accessible in the GUI. See the *Settings File* section for further details.

4.2.2 Panel B: Trajectory Viewer

This panel shows a segment of the data time series. The file currently being displayed is shown at the top of the window. If data from multiple files are loaded, the last filename is displayed. The length of time displayed in the window is controlled by *BlockSize* in Panel A (see #2).

6. Time Series (Trajectory)

- This plot shows the ionic current time series, of length *BlockSize*. Other features in the panel (such as histogram, denoising, etc.) only utilize the data in the window for their calculations.

7. All Points Histogram

- This shows a histogram of the time series data shown in #6.

8. Dashed line indicates mean baseline current

9. Detection threshold level indicated by solid red line

10. Navigation, Denoising, and Statistics

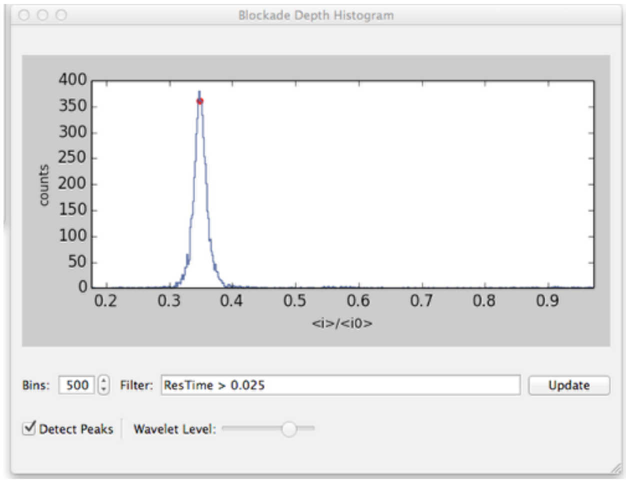
- **Navigation Tools:** Tools to navigate the plot window are shown below the time-series plot. These can be applied to either the trajectory or all points histogram plots. The arrow bar on the bottom right of the trajectory viewer can be used to advance to the next data block.
- **Denoising** Wavelet denoising can be activated by clicking , the denoising level is enabled here, the level of denoising can be varied between 1 and 5.

Warning: Wavelet-based denoising is currently an experimental feature and should be used with caution.

- **Baseline Statistics**: The mean baseline current, standard deviation, and the threshold used for event detection (specified as a multiple of the standard deviation in parenthesis) correspond to the settings in the main window. If the baseline current detection is set to *auto* these values will update as each data segment is examined. The size of this segment is determined by the *Block Size* setting. In the figure above, the *Block Size* is set to 0.5 s.

4.3 Panels C,D, & E: Blockade Depth Histogram, Statistics, and Event Viewer

4.3.1 Panel C: Blockade Depth Histogram



Bins: 500 Filter: ResTime > 0.025

☑ Detect Peaks Wavelet Level:

Fig. 3: Blockade depth histogram

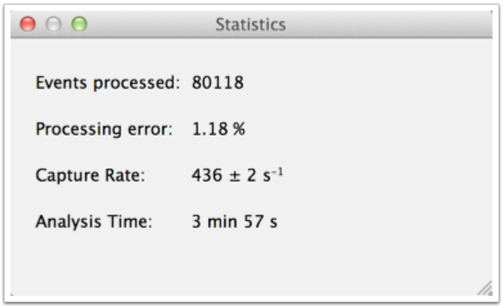
This window shows the blockade depth histogram calculated from the meta-data output by *MOSAIC*.

- **Filter**: The data displayed in the histogram can be restricted to events that fulfill specific user-defined criteria. For instance, the default filter `ResTime > 0.025` only includes events longer than 0.025 ms (or 25 μs). The GUI uses a SQL *select* statement to restrict the events included in the histogram. The text in the *Filter* field represents the part of the query after the *where* clause, and allows the user to use standard SQL syntax to narrow the results in the plot. See the *Work with SQLite* section for details on SQL syntax.
- **Bins**: The number of bins in the histogram are defined here. By default, 500 bins are used, but the user can change this necessary.
- **Detect Peaks**: Checking *Detect Peaks* enables a wavelet-based peak detection algorithm. The wavelet level slider controls the sensitivity of the peak detection. Sliding it to the right will decrease the number of peaks picked up. The peaks detected are represented with red dots. Mousing over the detected peaks cause the coordinates of the peak to be displayed in the lower right hand corner of the window. The detected peaks can also be exported to a CSV file from the file menu `File>Save Histogram`.

4.3.2 Panel D: Statistics

The *Statistics Window* is displayed when a new analysis is started and displays:

- **Events Processed**: The number of events processed.
- **Processing Error**: The processing error rate (i.e. the percentage of events for which fit has failed).
- **Capture Rate**: An estimate of the mean capture rate.
- **Analysis Time**: The amount of data processed (in seconds).



Events processed: 80118

Processing error: 1.18 %

Capture Rate: 436 ± 2 s^{-1}

Analysis Time: 3 min 57 s

Fig. 4: Live statistics window

4.3.3 Panel E: Event Viewer

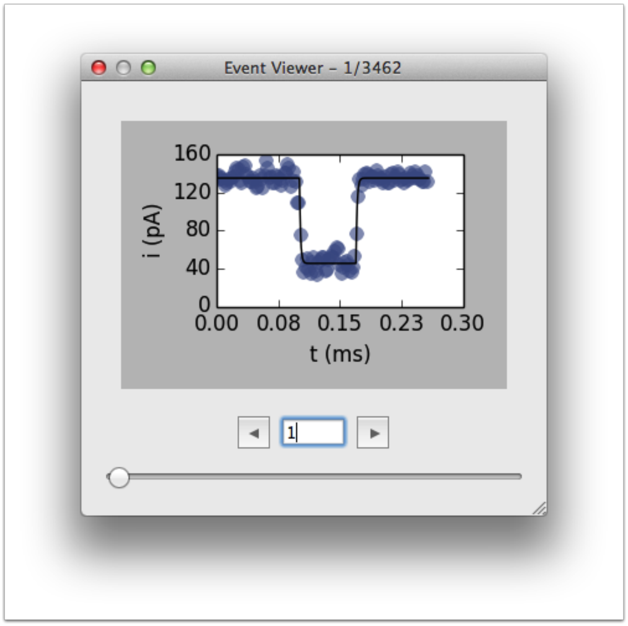
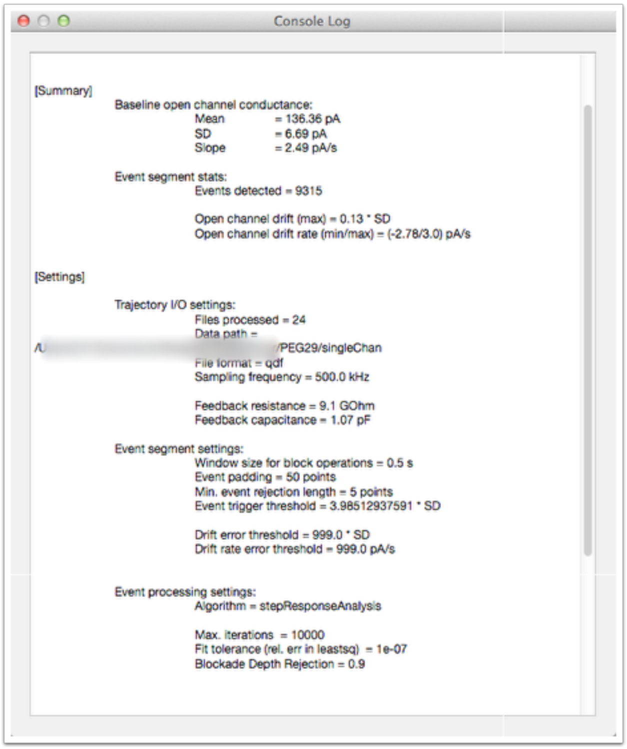


Fig. 5: Event viewer window

If *Write to Disk* is enabled, this panel allows you to view the first 10,000 events processed. This is useful to ensure the quality of the analysis and to debug potential problems with the settings.

4.3.4 Console Log

When processing is complete, this panel displays a log of the analysis. This log contains useful information such as the analysis settings, the number of events fit, baseline drift, open channel conducatance, etc. This file is written to the database and can be accessed later.

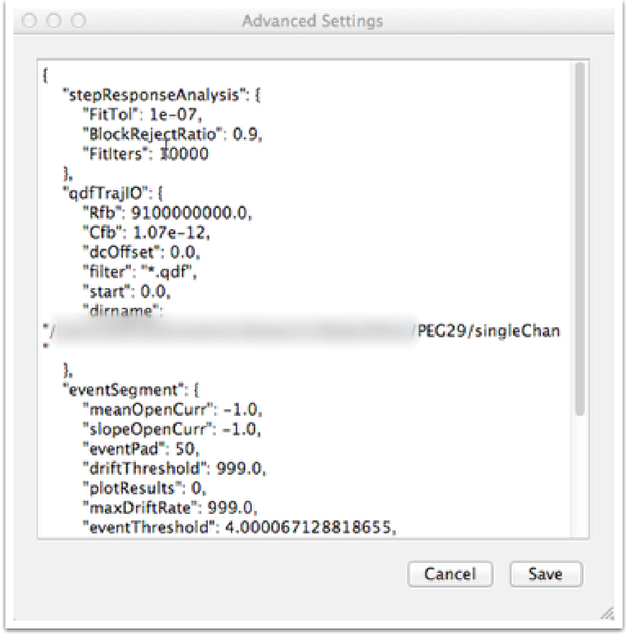


```
[Summary]
          Baseline open channel conductance:
                    Mean          = 136.36 pA
                    SD            = 6.69 pA
                    Slope         = 2.49 pA/s

          Event segment stats:
                    Events detected = 9315

                    Open channel drift (max) = 0.13 * SD
                    Open channel drift rate (min/max) = (-2.78/3.0) pA/s


[Settings]

          Trajectory I/O settings:
                    Files processed = 24
                    Data path =
[redacted]/PEG29/singleChan
                    File format = qdf
                    Sampling frequency = 500.0 kHz

                    Feedback resistance = 9.1 GOhm
                    Feedback capacitance = 1.07 pF

          Event segment settings:
                    Window size for block operations = 0.5 s
                    Event padding = 50 points
                    Min. event rejection length = 5 points
                    Event trigger threshold = 3.98512937591 * SD

                    Drift error threshold = 999.0 * SD
                    Drift rate error threshold = 999.0 pA/s


          Event processing settings:
                    Algorithm = stepResponseAnalysis

                    Max. iterations  = 10000
                    Fit tolerance (rel. err in leastsq)  = 1e-07
                    Blockade Depth Rejection = 0.9
```

Fig. 6: Console log window

4.3.5 Advanced Settings

This dialog allows you to manually edit advanced settings for uncommon use cases not natively accessible from within the GUI. Further information can be found in the *Settings File*.



```
{
   "stepResponseAnalysis": {
      "FitTol": 1e-07,
      "BlockRejectRatio": 0.9,
      "FitIters": 10000
   },
   "qdfTrajIO": {
      "Rfb": 9100000000.0,
      "Cfb": 1.07e-12,
      "dcOffset": 0.0,
      "filter": "*.qdf",
      "start": 0.0,
      "dirname":
"/[redacted]/PEG29/singleChan
"
   },
   "eventSegment": {
      "meanOpenCurr": -1.0,
      "slopeOpenCurr": -1.0,
      "eventPad": 50,
      "driftThreshold": 999.0,
      "plotResults": 0,
      "maxDriftRate": 999.0,
      "eventThreshold": 4.000067128818655,
```

Fig. 7: Advanced settings window

CHAPTER 5

MOSAIC WEB

MOSAIC's web interface is designed to allow you to easily setup and run an analysis, or to visualize and analyze the results of previous experiments. It contains the most commonly used features of *MOSAIC*. The web interface consists of a series of screens that allow you to set up and run an analysis. The interface keeps track of all analysis within the current session so it is easy to go back and compare data sets. The web interface is designed to allow you to easily run the most common use cases. For more complicated analysis, please refer to the *Scripting and Advanced Features* section.

Opening the Web Interface

The web interface can be run locally, allowing you to run a *MOSAIC* analysis from within any modern web browser. If you installed *MOSAIC* from a precomiled binary, you can start the web interface by double clicking the *MOSAIC* icon. Alternatively, if you compiled *MOSAIC* from source code, you can run the GUI from the terminal window – navigate to the installation directory and type:

```
python runMOSAIC
```

This should start a local copy of the *MOSAIC* web server and open a new browser window that launches the web interface. Running servers can be accessed at anytime by opening a browser and entering http://localhost:5000 in the adress bar.

5.1 Interface Overview

The main screen of the *MOSAIC* web interface is the starting point for running new analyses or reviewing previous runs.

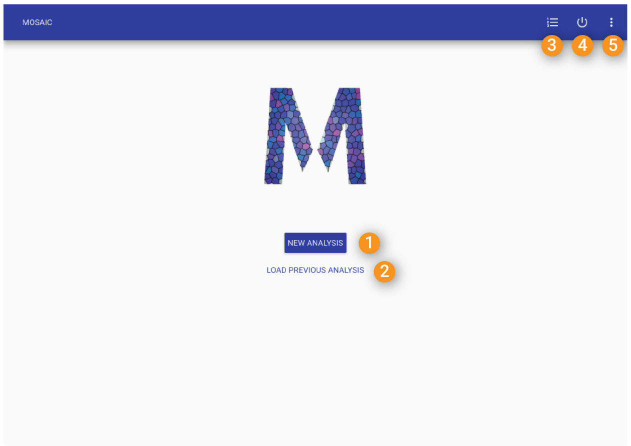
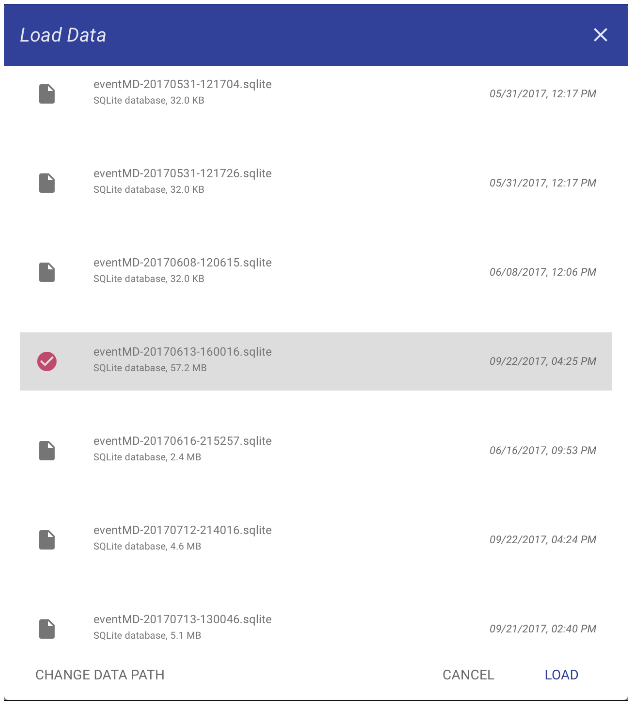


Fig. 1: The main screen of the *MOSAIC* web interface allows new analyses to be set up or previous runs to be retrieved.


Load Data

eventMD-20170531-121704.sqlite
SQLite database, 32.0 KB
05/31/2017, 12:17 PM

eventMD-20170531-121726.sqlite
SQLite database, 32.0 KB
05/31/2017, 12:17 PM

eventMD-20170608-120615.sqlite
SQLite database, 32.0 KB
06/08/2017, 12:06 PM

eventMD-20170613-160016.sqlite
SQLite database, 57.2 MB
09/22/2017, 04:25 PM

eventMD-20170616-215257.sqlite
SQLite database, 2.4 MB
06/16/2017, 09:53 PM

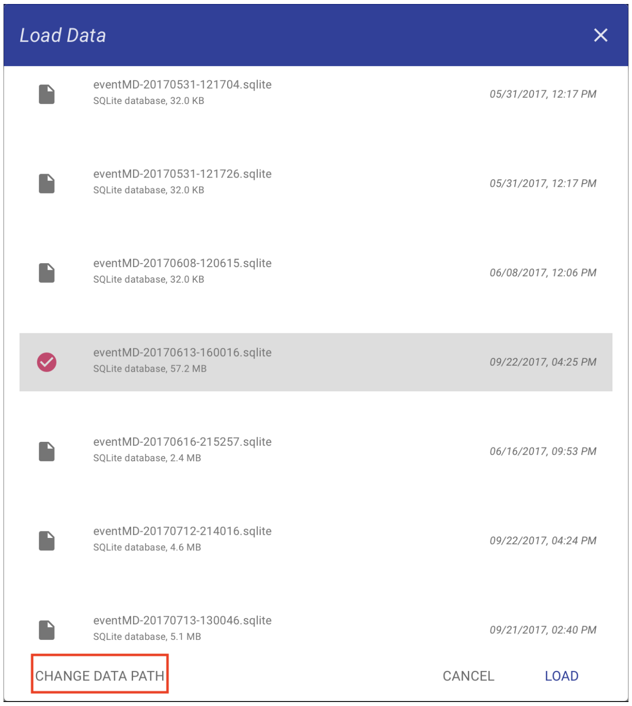
eventMD-20170712-214016.sqlite
SQLite database, 4.6 MB
09/22/2017, 04:24 PM

eventMD-20170713-130046.sqlite
SQLite database, 5.1 MB
09/21/2017, 02:40 PM

5.2 Data Source Path

The path of the data source can be changed when *MOSAIC* is run in local mode, i.e., when it is run entirely on a local machine. This is the default configuration of the pre-compiled binaries. When running *MOSAIC* from source it can be enabled by editing the global.json file.

The data source path can be edited when starting a new analysis or loading a previous analysis by clicking the CHANGE DATA PATH button shown in the figure below.



Load Data

eventMD-20170531-121704.sqlite
SQLite database, 32.0 KB
05/31/2017, 12:17 PM

eventMD-20170531-121726.sqlite
SQLite database, 32.0 KB
05/31/2017, 12:17 PM

eventMD-20170608-120615.sqlite
SQLite database, 32.0 KB
06/08/2017, 12:06 PM

eventMD-20170613-160016.sqlite
SQLite database, 57.2 MB
09/22/2017, 04:25 PM

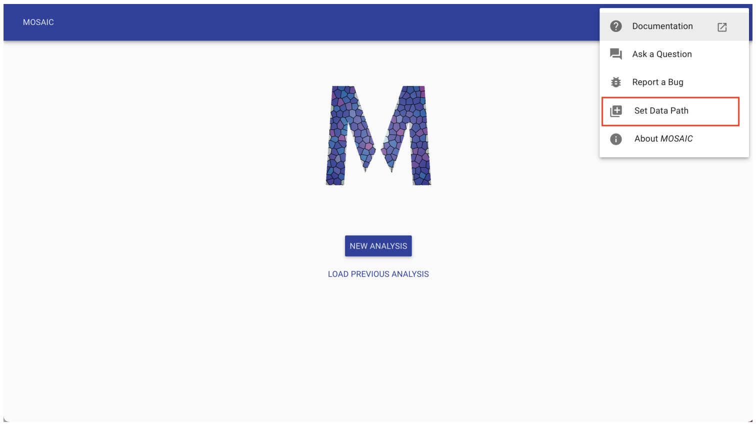
eventMD-20170616-215257.sqlite
SQLite database, 2.4 MB
06/16/2017, 09:53 PM

eventMD-20170712-214016.sqlite
SQLite database, 4.6 MB
09/22/2017, 04:24 PM

eventMD-20170713-130046.sqlite
SQLite database, 5.1 MB
09/21/2017, 02:40 PM

CHANGE DATA PATH   CANCEL   LOAD

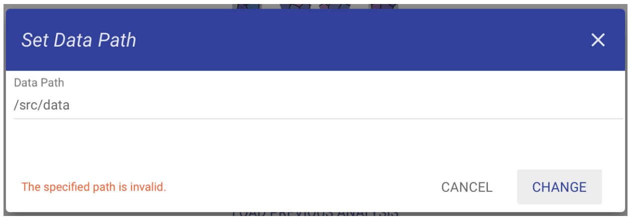
The setting can also be accessed from the overflow menu at the top right and then clicking Set Data Path as seen from the figure below.



The new source data path can be entered in the resulting dialog. Click CHANGE to save the new path.

If the new path is invalid, an error message will be displayed and the path will not be updated as seen below.

Data Path
/src/data



Data Path
/src/data

The specified path is invalid.

5.3 Analysis Settings

The analysis settings interface displays the data trajectory and allows one to set up and run an analysis. Below we present an overview of the different settings available on this screen. The numbered sections correspond to the numbers in the figure below.

1. Time-Series

This section of the analysis settings shows a segment of the data time series. The length of time-series displayed is determined by *BlockSize* parameter in #2.

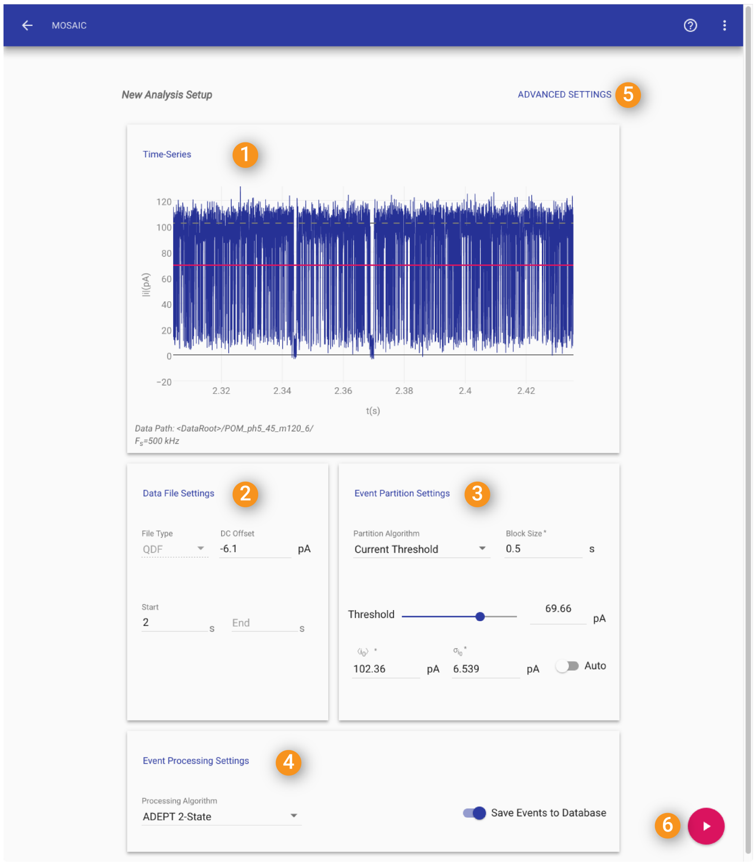
- **Gray Dashed line:** The gray dashed line in the time-series display indicates the mean baseline current ($\langle i_0 \rangle$) as estimated in #3.
- **Solid Red line:** The red line in the time-series indicates the threshold current set in #3.
- **Data Path:** The location of the data filesand currently being displayed is shown at the top of the window. If data from multiple files are loaded, the last filename is displayed.
- **Fs:** The sampling frequency of the time-series data in kHz.

2. Data File Settings

- **File Type**: The web interface is natively compatible with ABF, BIN or QDF Files. If the data path contains any of those file types this field is automatically populated.
- **DC Offset**: If your measurement contains a systematic bias, it can be manually corrected by entering the DC offset here.
- **Start** and **End**: These parameters allow you to analyze a range of your data. Choose the starting and ending times if you'd like to analyze a small time segement of your data. If both fields are left blank, all data will be analyzed. If the *End* field is left blank, the data segment from *Start* to the end of the data will be analyzed.

3. Event Partition Control

- **Partition Algorithm**: Currently, the only event partitioning algorithm available is *CurrentThreshold*.
- **Block Size**: Controls the amount of data used to determine the baseline. This setting also controls the amount of data shown in the trajectory viewer.
- $\langle i_0 \rangle$: Mean open channel current, in picoamperes (pA). This is shown using the gray dashed line in the time-series viewer (see #1). When *Auto* is selected, this input will be disabled.



Data Path: <DataRoot>/POM_ph5_45_m120_6/

F_s=500 kHz

File Type: QDF

DC Offset: -6.1 pA

Start: 2 s

End: s

Partition Algorithm: Current Threshold

Block Size*: 0.5 s

Threshold: 69.66 pA

$\langle i_0 \rangle$*: 102.36 pA

σ_{i_0}*: 6.539 pA

Auto

Processing Algorithm: ADEPT 2-State

Save Events to Database

- σ_{i_0}: Standardard deviation of the open channel current noise in pA. This is expected noise level of your baseline. Typically one would set this to the measured standard deviation of the open channel current at the cutoff frequency. When *Auto* is selected, this input will be disabled.
- **Auto**: Checking this box enables automatic dectection of $\langle i_0 \rangle$ and σ_{i_0}.
- **Threshold**: The slider and correspnding text input can be used to set the current threshold used to determine the start of an event. This setting is used by the *CurrentThreshold* algorithm to perform an initial partition of the time-series into individual events.

4. Event Processing Setup

Event Processing Algorithm: The GUI supports two event processing algorithms, i) *StepResponseAnalysis* and ii) *MultiStateAnalysis*. *StepResponseAnalysis* is the default analysis, and should be used with data sets with unimodal events. For events with mutliple states or steps the *MultiStateAnalysis* algorithm, which is capable of automatically analyzing events with *N* states, should be used. Note that *StepResponseAnalysis* is a restricted case of *MultiStateAnalysis* and is more computationally efficient to run if you have unimodal (or single states) data.

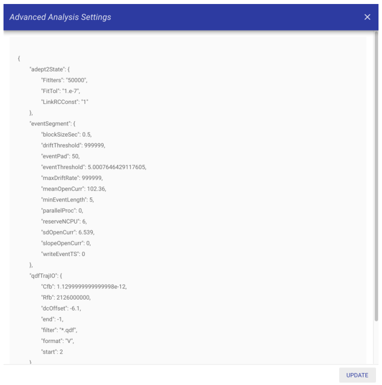
- **Write Events to Disk**: When this box is checked, the data points for each partition events are written to the SQLite database. When this is checked it is possible to view the individual fits of each in the *Event Fits* panel.

Hint: When Write Events to Disk is checked, your database can become extremely large! This is because *MOSAIC* is effectively writing most of your time-series to the database. Note that the fit parameters are always written to the database.

---

5. Advanced Settings

This opens a dialog window to manually edit settings not otherwise accessible in the GUI. See the *Settings File* section for further details.



```
{
    "adept2State": {
        "FitIters": "50000",
        "FitTol": "1.e-7",
        "LinkRCConst": "1"
    },
    "eventSegment": {
        "blockSizeSec": 0.5,
        "driftThreshold": 999999,
        "eventPad": 50,
        "eventThreshold": 5.0007646429117605,
        "maxDriftRate": 999999,
        "meanOpenCurr": 102.36,
        "minEventLength": 5,
        "parallelProc": 0,
        "reserveNCPU": 6,
        "sdOpenCurr": 6.539,
        "slopeOpenCurr": 0,
        "writeEventTS": 0
    },
    "qdfTrajIO": {
        "Cfb": 1.1299999999999998e-12,
        "Rfb": 2126000000,
        "dcOffset": -6.1,
        "end": -1,
        "filter": "*.qdf",
        "format": "V",
        "start": 2
    }
```

6. Start Analysis/Update Settings

Use this button to either start the analysis when it has a *Play* symbol or to update and validate any settings when it displays a *Check Mark*.

5.4 Analysis Results

1. Analysis Control

Statistics

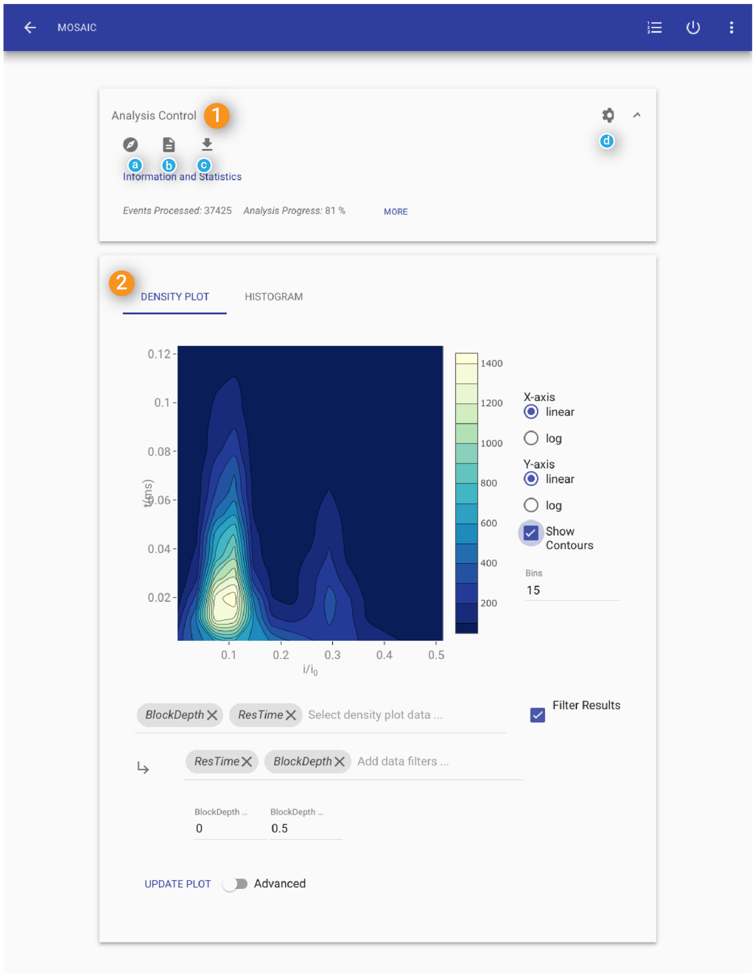
The *Statistics Window* is displayed when a new analysis is started and displays:

- **Events Processed**: The number of events processed.
- **Processing Error**: The processing error rate (i.e. the percentage of events for which fit has failed).
- **Capture Rate**: An estimate of the mean capture rate.
- **Analysis Time**: The amount of data processed (in seconds).

2. Results View

This window shows the blockade depth histogram calculated from the meta-data output by *MOSAIC*.

- **Filter**: The data displayed in the histogram can be restricted to events that fulfill specific user-defined criteria. For instance, the default filter `ResTime > 0.025` only includes events longer than 0.025 ms (or 25 μs). The GUI uses a SQL *select* statement to restrict the events included in the histogram. The text in the *Filter* field represents the part of the query after the *where* clause, and allows the user to use standard SQL syntax to narrow the results in the plot. See the *Work with SQLite* section for details on SQL syntax.
- **Bins**: The number of bins in the histogram are defined here. By default, 500 bins are used, but the user can change this necessary.
- **Detect Peaks**: Checking *Detect Peaks* enables a wavelet-based peak detection algorithm. The wavelet level slider controls the sensitivity of the peak detection. Sliding it to the right will decrease the number of peaks picked up. The peaks detected are represented with red dots. Mousing over the detected peaks cause the coordinates of the peak to be displayed in the lower right hand corner of the window. The detected peaks can also be exported to a CSV file from the file menu `File>Save Histogram`.



Events Processed: 37425 Analysis Progress: 81 %

X-axis
● linear
○ log

Y-axis
● linear
○ log

☑ Show Contours

Bins
15

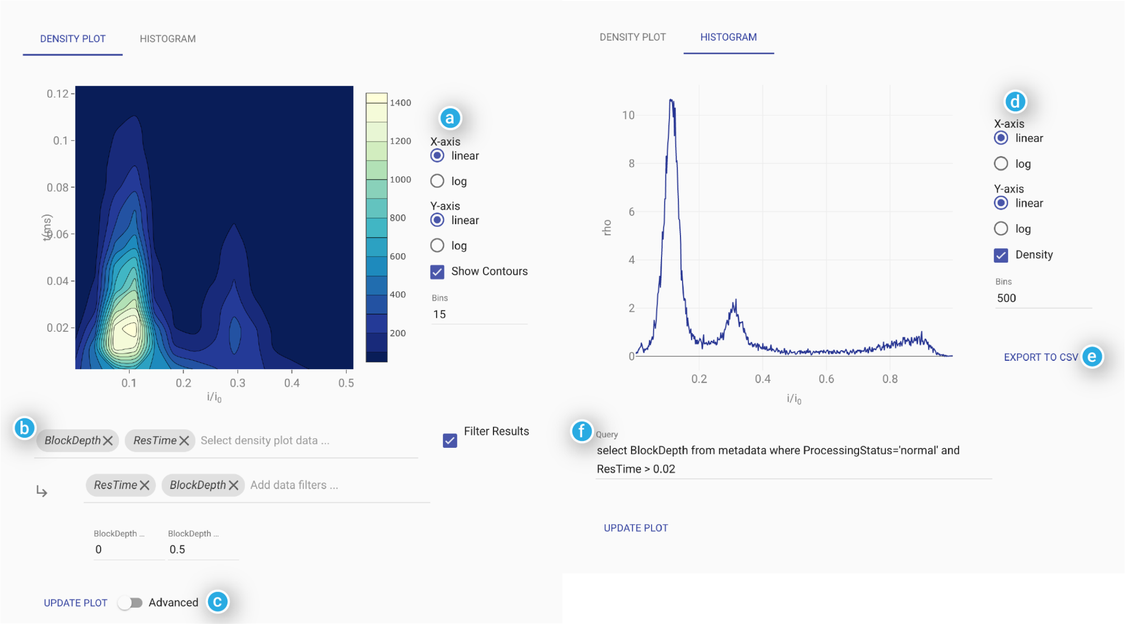
BlockDepth ✕ ResTime ✕ Select density plot data ...

☑ Filter Results

↳ ResTime ✕ BlockDepth ✕ Add data filters ...

BlockDepth ... 0

BlockDepth ... 0.5



X-axis
● linear
○ log

Y-axis
● linear
○ log

☑ Show Contours

Bins
15

BlockDepth ✕ ResTime ✕ Select density plot data ...

☑ Filter Results

↳ ResTime ✕ BlockDepth ✕ Add data filters ...

BlockDepth ... 0

BlockDepth ... 0.5

X-axis
● linear
○ log

Y-axis
● linear
○ log

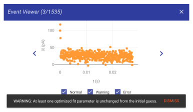
☑ Density

Bins
500

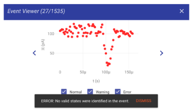
Query
select BlockDepth from metadata where ProcessingStatus='normal' and ResTime > 0.02

Fit Parameters

j	$\langle i_j \rangle$ / $\langle i_0 \rangle$	t_j (μs)
1	0.6031	38.37
2	0.2152	84.68



WARNING: At least one optimized fit parameter is unchanged from the initial guess.



ERROR: No valid states were identified in the event.

CHAPTER 6

Settings File

MOSAIC stores its settings in the JSON format. When using the graphical interface, a settings file is generated automatically upon starting an analysis, or by clicking *Save Settings* in the *File menu* (see *MOSAIC GUI*).

6.1 Settings Layout

JSON is a human readable file format that consists of key-value pairs separated by sections. Each section in a JSON object consists of a section name and a list of string key-value pairs.

```
{
        "<section name>" : {
                "key1" : "value1",
                "key2" : "value2",
                ...
        }

}
```

MOSAIC settings define a new section for each class, with key-value pairs corresponding to class attributes that are set upon initialization. This is illustrated below for the adept2State class. The *adept2State* section in the settings file holds parameters corresponding to the `adept2State` class. Note that that the section name in the settings file is identical to the corresponding class name. Three parameters are then defined within the section that control the behavior of the class.

```
{
        "adept2State" : {
                "FitTol"                        : "1.e-7",
                "FitIters"                      : "50000",
                "BlockRejectRatio"              : "0.9"
        }
}
```

Finally, `adept2State` is initialized by defining class attributes corresponding to the key-value pairs in the settings file.

```
try:
                self.FitTol=float(self.settingsDict.pop("FitTol", 1.e-7))
                self.FitIters=int(self.settingsDict.pop("FitIters", 5000))

                self.BlockRejectRatio=float(self.settingsDict.pop("BlockRejectRatio",␣
↪0.8))

except ValueError as err:
        raise commonExceptions.SettingsTypeError( err )
```

6.2 Trajectory Settings

6.2.1 Common Settings (`metaTrajIO`)

class `mosaic.trajio.metaTrajIO.`**`metaTrajIO`**(***kwargs*)

> **Warning:** This metaclass must be sub-classed. All abstract methods within this metaclass must be implemented.

Initialize a TrajIO object. The object can load all the data in a directory, N files from a directory or from an explicit list of filenames. In addition to the arguments defined below, implementations of this meta class may require the definition of additional arguments. See the documentation of those classes for what those may be. For example, the qdfTrajIO implementation of metaTrajIO also requires the feedback resistance (Rfb) and feedback capacitance (Cfb) to be passed at initialization.

Parameters

- *dirname* : all files from a directory ('<full path to data directory>')
- *nfiles* : if requesting N files (in addition to dirname) from a specified directory
- *fnames* : explicit list of filenames ([file1, file2,. . .]). This argument cannot be used in conjuction with dirname/nfiles. The filter argument is ignored when used in combination with fnames.
- *filter* : '<wildcard filter>' (optional, filter is '*' if not specified)
- *start* : Data start point in seconds.
- *end* : Data end point in seconds.
- *datafilter* : Handle to the algorithm to use to filter the data. If no algorithm is specified, datafilter is None and no filtering is performed.
- *dcOffset* : Subtract a DC offset from the ionic current data.
- *filtersettings*: Dict containing low pass filter settings (optional: if not provided filter settings will be loaded from the settings file. If no settings are found, *datafilter* will be turned off.)

Properties

- *FsHz* : sampling frequency in Hz. If the data was decimated, this property will hold the sampling frequency after decimation.

- *LastFileProcessed* : return the data file that was last processed.
- *ElapsedTimeSeconds* : return the analysis time in sec.

Errors

- *IncompatibleArgumentsError* : when conflicting arguments are used.
- *EmptyDataPipeError* : when out of data.
- *FileNotFoundError* : when data files do not exist in the specified path.
- *InsufficientArgumentsError* : when incompatible arguments are passed

6.2.2 QDF Files (`qdfTrajIO`)

class `mosaic.trajio.qdfTrajIO.`**`qdfTrajIO`**(***kwargs*)

Use the readqdf module from EBS to read individual QDF files.

In addition to `metaTrajIO` args, check if the feedback resistance (`Rfb`) and feedback capacitance (`Cfb`) are defined to convert qdf binary data into pA.

A typical settings section to read QDF files is shown below. Note, that the values for `Rfb` and `Cfb` are specific to the amplifier used.

```
"qdfTrajIO": {
"Rfb"                           : 9.1e+12,
"Cfb"                           : 1.07e-12,
"dcOffset"                      : 0.0,
"filter"                        : "*.qdf",
"start"                         : 0.0
}
```

Parameters

In addition to metaTrajIO.__init__ args,

- *Rfb* : feedback resistance of amplifier
- *Cfb* : feedback capacitance of amplifier
- *format* : 'V' for voltage or 'pA' for current. Default is 'V'

Returns None

Errors

- *InsufficientArgumentsError* : if the mandatory arguments `Rfb` and `Cfb` are not set.

6.2.3 ABF Files (`abfTrajIO`)

class `mosaic.trajio.abfTrajIO.`**`abfTrajIO`**(***kwargs*)

Read ABF1 and ABF2 file formats. Currently, only gap-free mode and single channel recordings are supported.

A typical settings section to read ABF files is shown below.

```
"abfTrajIO" : {
"filter"                        : "*.abf",
"start"                         : 0.0,
"dcOffset"                      : 0.0
}
```

Parameters

In addition to `metaTrajIO` args, None

6.2.4 Binary Files (`binTrajIO`)

`class mosaic.trajio.binTrajIO.binTrajIO(**kwargs)`

Read a file that contains interleaved binary data, ordered by column. Only a single column that holds ionic current data is read. The current in pA is returned after scaling by the amplifier scale factor (`AmplifierScale`) and removing any offsets (`AmplifierOffset`) if provided.

Usage and Assumptions Binary data is interleaved by column. For three columns (*a*, *b*, and *c*) and *N* rows, binary data is assumed to be of the form:

[a_1, b_1, c_1, a_2, b_2, c_2, … … …, a_N, b_N, c_N]

The column layout is specified with the `ColumnTypes` parameter, which accepts a list of tuples. For the example above, if column **a** is the ionic current in a 64-bit floating point format, column **b** is the ionic current representation in 16-bit integer format and column **c** is an index in 16-bit integer format, the `ColumnTypes` paramter is a list with three tuples, one for each column, as shown below:

[(‘curr_pA’, ‘float64’), (‘AD_V’, ‘int16’), (‘index’, ‘int16’)]

The first element of each tuple is an arbitrary text label and the second element is a valid Numpy type.

Finally, the `IonicCurrentColumn` parameter holds the name (text label defined above) of the column that holds the ionic current time-series. Note that if an integer column is selected, the `AmplifierScale` and `AmplifierOffset` parameters can be used to convert the voltage from the A/D to a current.

Assuming that we use a floating point representation of the ionic current, and a sampling rate of 50 kHz, a settings section that will read the binary file format defined above is:

```
"binTrajIO": {
        "AmplifierScale" : "1",
        "AmplifierOffset" : "0",
        "SamplingFrequency" : "50000",
        "ColumnTypes" : "[('curr_pA', 'float64'), ('AD_V', 'int16'), (
↪'index', 'int16')]",
        "IonicCurrentColumn" : "curr_pA",
        "dcOffset": "0.0",
        "filter": "*.bin",
        "start": "0.0",
        "HeaderOffset": 0
}
```

Settings Examples Read 16-bit signed integers (big endian) with a 512 byte header offset. Set the amplifier scale to 400 pA, sampling rate to 200 kHz.

```
"binTrajIO": {
        "AmplifierOffset": "0.0",
        "SamplingFrequency": 200000,
        "AmplifierScale": "400./2**16",
        "ColumnTypes": "[('curr_pA', '>i2')]",
        "dcOffset": 0.0,
        "filter": "*.dat",
```

(continues on next page)

(continued from previous page)

```
        "start": 0.0,
        "HeaderOffset": 512,
        "IonicCurrentColumn": "curr_pA"
}
```

Read a two-column file: 64-bit floating point and 64-bit integers, and no header offset. Set the amplifier scale to 1 and sampling rate to 200 kHz.

```
"binTrajIO": {
        "AmplifierOffset": "0.0",
        "SamplingFrequency": 200000,
        "AmplifierScale": "1.0",
        "ColumnTypes" : "[('curr_pA', 'float64'), ('AD_V',
↪'int64')]",
        "dcOffset": 0.0,
        "filter": "*.bin",
        "start": 0.0,
        "HeaderOffset": 0,
        "IonicCurrentColumn": "curr_pA"
}
```

Parameters

In addition to `metaTrajIO` args,

- *AmplifierScale* : Full scale of amplifier (pA/2^nbits) that varies with the gain (default: 1.0).
- *AmplifierOffset* : Current offset in the recorded data in pA (default: 0.0).
- *SamplingFrequency* : Sampling rate of data in the file in Hz.
- *HeaderOffset* : Ignore first *n* bytes of the file for header (default: 0 bytes).
- *ColumnTypes* : A list of tuples with column names and types (see Numpy types). Note only integer and floating point numbers are supported.
- *IonicCurrentColumn* : Column name that holds ionic current data.

Returns None

Errors None

6.2.5 Chimera Files (`binTrajIO`)

class `mosaic.trajio.chimeraTrajIO.`**`chimeraTrajIO`**`(`***kwargs*`)`

Read a file generated by the Chimera VC100. The current in pA is returned after scaling by the amplifier scale factors.

Usage and Assumptions Binary data is in a single column of unsigned 16 bit integers:

The column layout is specified with the `ColumnTypes` parameter, which accepts a list of tuples.

[('curr_pA', '<u2')]

The option is left in in case of future changes to the platform, but can be left alone in the settings file for now. The first element of each tuple is an arbitrary text label and the second element is a valid Numpy type.

Chimera gain settings are used to convert the integers stored by the ADC to current values. These values are automatically read in from matched MAT files generated by the Chimera software.

```
"chimeraTrajIO": {
        "filter": "*.log",
        "start": "0.0",
        "HeaderOffset": "0"
}
```

Parameters In addition to `metaTrajIO` args,

- *HeaderOffset* : Ignore first *n* bytes of the file for header (currently fixed at: 0 bytes).

Returns None

Errors None

6.2.6 TSV Files (`binTrajIO`)

class `mosaic.trajio.tsvTrajIO.`**`tsvTrajIO`**`(**kwargs)`

Read tab separated valued (TSV) files.

Parameters

In addition to `metaTrajIO` args,

- *headers* : If True, the first row is ignored (default: True)
- *separator* : set the data separator (defualt: ‘”"t’)
- *scale* : set the data scale (default: 1). For example to convert from to pA set `scale=1e12`.

Either:

- *Fs* : Sampling frequency in Hz. If set, all other options are ignored and the first column in the file is assumed to be the current in pA.

Or:

- *nCols* : number of columns in TSV file (default:2, first column is time in ms and second is current in pA)
- *timeCol* : explicitly set the time column (default: 0, first col)
- *currCol* : explicitly set the position of the current column (default: 1)

If neither `Fs` nor {`nCols`, `timeCol`, `currCol`} are set then the latter is assumed with the listed default values.

6.3 Optimizing Settings

MOSAIC classes are controlled through the JSON settings files as defined above. In most cases, running *MOSAIC* through the GUI (see *MOSAIC GUI*) should generate satisfactory results. However, settings can be further optimized either by editing a file named `.settings` stored within the data directory, or by clicking on the `Advanced Settings` check-box in the *Panel A: Analysis Setup* section of the GUI.

6.3.1 Initial Event Detection (`eventSegment`)

The first step when analyzing an ionic-current time series is to perform a quick partition to identify events. This is accomplished by overriding the `eventPartition` class. Currently, the only implementation of event partitioning is the `eventSegment` algorithm. This algorithm uses a thresholding technique to detect the start and end of an event. When an event is detected the ionic current time-series associated with that event is passed to a processing algorithm for fitting. Settings that can be passed to `eventSegment` are given below followed by their descriptions.

```
"eventSegment" : {
        "blockSizeSec"                  : "0.5",
        "eventPad"                      : "50",
        "minEventLength"                : "5",
        "eventThreshold"                : "6.0",
        "driftThreshold"                : "999.0",
        "maxDriftRate"                  : "999.0",
        "meanOpenCurr"                  : "-1",
        "sdOpenCurr"                    : "-1",
        "slopeOpenCurr"                 : "-1",
        "writeEventTS"                  : "1",
        "parallelProc"                  : "0",
        "reserveNCPU"                   : "2"
}
```

Setting	Description
blockSizeSec	Time-series length (in sec) for block operations.
eventPad	Pad an event with the specified number of points.
minEventLength	Discard events with fewer than the specfied points.
eventThreshold	Event detection threshold.
meanOpenCurr	Set the mean open channel current (i0) in pA. -1 computes i0 automatically.
sdOpenCurr	Set the open channel std. dev. in pA. -1 computes SD automatically.
slopeOpenCurr	Set the open channel drift in pA/ms. -1 automatically computes the slope.
driftThreshold	Aborts the analysis when the open channel drift exceeds the specified value.
maxDriftRate	Aborts the analysis when the open channel slope exceeds the specified value (pA/ms).
writeEventTS	Write the event time-series to the output database.
parallelProc	Enable parallel processing.
reserveNCPU	Use N-reserveNCPU for parallel processing.

6.3.2 Two-State Identification (`adept2State`)

Once the time-series is partitioned, individual events are processed by a processing algorithm. For simple event patterns (e.g. homopolymers of DNA, PEG, etc.), one can use the stepresponse-page algorithm. Settings that can be passed to this algorithm are below, followed by their descriptions. For a vast majority of cases, the settings below can be used without modification.

```
"adept2State" : {
        "FitTol"                        : "1.e-7",
        "FitIters"                      : "50000"
}
```

6.3.3 Multi-State Identification (`adept`)

For more complex signals with multiple states, the *ADEPT* algorithm yields better results. The settings passed to this algorithm (described below) are largely similar to *Two-State Identification (adept2State)*.

```
"adept" : {
        "FitTol"                        : "1.e-7",
        "FitIters"                      : "50000",
        "InitThreshold"                 : "3.0"
}
```

Hint: The parameter `InitThreshold` is used for preliminary state identification within multi-state events. As a rule of thumb, this value should be set to roughly half that of `eventThreshold` in the *Initial Event Detection (eventSegment)* section. However, the final value may be adjusted further for optimal results.

6.4 Default Settings

```
{
        "eventSegment" : {
                "blockSizeSec"                  : "0.5",
                "eventPad"                              : "50",
                "minEventLength"                : "5",
                "eventThreshold"                : "6.0",
                "driftThreshold"                : "999.0",
                "maxDriftRate"                  : "999.0",
                "meanOpenCurr"                  : "-1",
                "sdOpenCurr"                    : "-1",
                "slopeOpenCurr"                 : "-1",
                "writeEventTS"                  : "1",
                "parallelProc"                  : "0",
                "reserveNCPU"                   : "2"
        },
        "singleStepEvent" : {
                "binSize"                               : "1.0",
                "histPad"                               : "10",
                "maxFitIters"                   : "5000",
                "a12Ratio"                              : "1.e4",
                "minEvntTime"                   : "10.e-6",
                "minDataPad"                    : "75"
```

(continues on next page)

(continued from previous page)

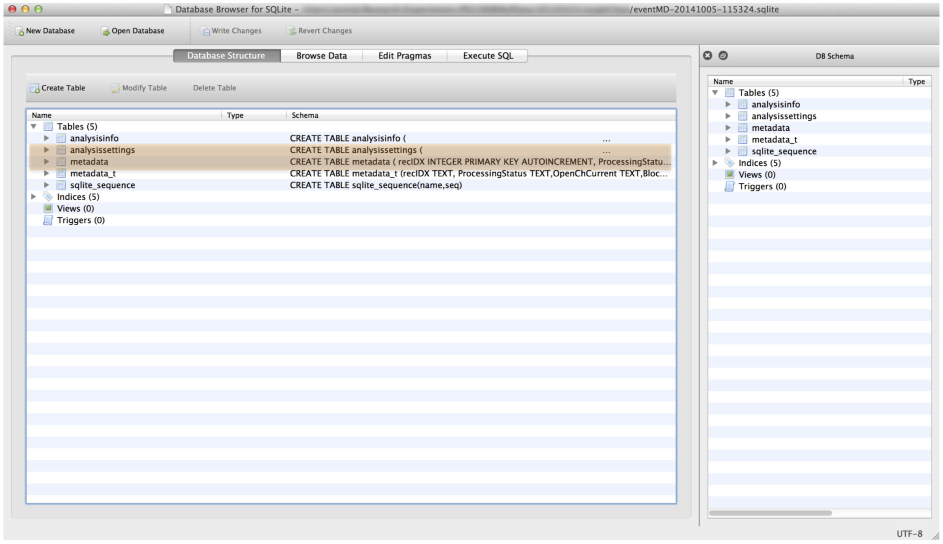
```
        },
        "adept2State" : {
                "FitTol"                                : "1.e-7",
                "FitIters"                              : "50000"
        },
        "adept" : {
                "FitTol"                : "1.e-7",
                "FitIters"              : "50000",
                "InitThreshold" : "3.0"
     },
     "cusumPlus": {
                        "StepSize"              : 3.0,
                        "Threshold"             : 3.0
},
        "besselLowpassFilter" : {
                "filterOrder"                   : "6",
                "filterCutoff"                  : "10000",
                "decimate"                              : "1"
        },
        "waveletDenoiseFilter" : {
                "wavelet"                               : "sym5",
                "level"                                 : "5",
                "thresholdType"                 : "soft",
                "thresholdSubType"              : "sqtwolog"
        },
        "abfTrajIO" : {
                "filter"                                : "*.abf",
                "start"                                 : 0.0,
                "dcOffset"                              : 0.0
        },
        "qdfTrajIO": {
                "Rfb": 9.1e+12,
                "Cfb": 1.07e-12,
                "dcOffset": 0.0,
                "filter": "*.qdf",
                "start": 0.0
        },
        "binTrajIO": {
                "AmplifierScale": "1.0",
                "AmplifierOffset": "0.0",
                "SamplingFrequency": "50000",
                "HeaderOffset": "0",
                "ColumnTypes": "[('curr_pA', 'float64')]",
                "IonicCurrentColumn" : "curr_pA",
                "dcOffset": "0.0",
                "filter": "*.bin",
                "start": "0.0"
        }
}
```

CHAPTER 7

Database Structure and Query Syntax

MOSAIC stores the output of an analysis in a SQLite database. Database files are stored in the same directory as the data being processed. Each analysis creates a new database file named *eventMD-<date>-<time>.sqlite*, where *<date>* is the date the analysis was performed (e.g. 20140929 for Sep 29, 2014) and *<time>* is the analysis start time (e.g. 112937 for 11:29:37 AM).

SQLite databases store data in tables similar to spreadsheets, where each table is analogous to a sheet in an Excel spreadsheet. Databased generated by *MOSAIC* can be inspected using a database viewer, for example the open source DB browser for SQLite. *MOSAIC* outputs databases with multiple tables as seen from the figure below. Databases output by *MOSAIC* contain four tables: i) *analysisinfo* contains general information about the anlysis such as the data path, analysis algorithm etc., ii) *analysissettings* contains a JSON formatted string with the analysis settings, iii) *metadata* holds the output of the analysis, and iv) *metadata_t* lists the data types for each column in *metadata*. Two tables most relevant to the analysis (*metadata* and *analysissettings*) are discussed in detail below.



Name	Type	Schema
Tables (5)		
analysisinfo		CREATE TABLE analysisinfo (...
analysissettings		CREATE TABLE analysissettings (...
metadata		CREATE TABLE metadata (recIDX INTEGER PRIMARY KEY AUTOINCREMENT, ProcessingStatu…
metadata_t		CREATE TABLE metadata_t (recIDX TEXT, ProcessingStatus TEXT,OpenChCurrent TEXT,Bloc…
sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)
Indices (5)		
Views (0)		
Triggers (0)		

Name	Type
Tables (5)	
analysisinfo	
analysissettings	
metadata	
metadata_t	
sqlite_sequence	
Indices (5)	
Views (0)	
Triggers (0)	

7.1 Metadata Table

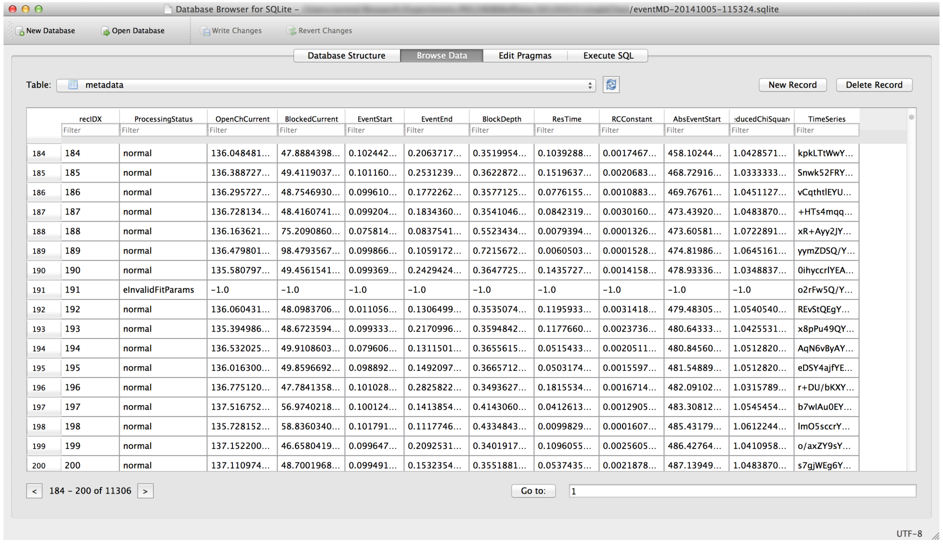
The *metadata* table contains the primary output of the analysis. *MOSAIC* processes individual blockade events from a time-series of ionic current. The parameters describing each event (or metadata) are stored in individual rows of the *metadata* table in the database file. The column names describe the metadata and are unique to the processing algorithm used. For example, the column names for the stepresponse-page algorithm are shown below. The column names for *ADEPT* differ from this list.

```
{
    ProcessingStatus,
    OpenChCurrent,
    BlockedCurrent,
    EventStart,
    EventEnd,
    BlockDepth,
    ResTime,
    RiseTime,
    AbsEventStart,
    RedChiSq,
    TimeSeries
}
```

Note that the column names can be used in constructing queries passed to SQLite, and is described in more detail in the *Work with SQLite* section and the *Scripting and Advanced Features* section. The first example SQL query below returns the *BlockDepth* column (ratio of *BlockedCurrent* to *OpenChCurrent*). One can imagine assembling more complex queries for example restricting the results to events whose residence time is greater than 0.2 ms as seen from the second example query below.

```sql
select BlockDepth from metadata where ProcessingStatus='normal'

select BlockDepth from metadata where ProcessingStatus='normal' and ResTime > 0.2
```

A typical *metadata* table for the stepresponse-page algorithm is shown below. The *ProcessingStatus* column is a text field that should read *normal* if the fit for a particular event was successful. If a failure occurred the corresponding error code (e.g. *eInvalidFitParameters*) is stored and all other columns (except *TimeSeries*) are set to -1. If event time-series storage was requested, then the *TimeSeries* column will store the ionic current data for that entry in binary format.

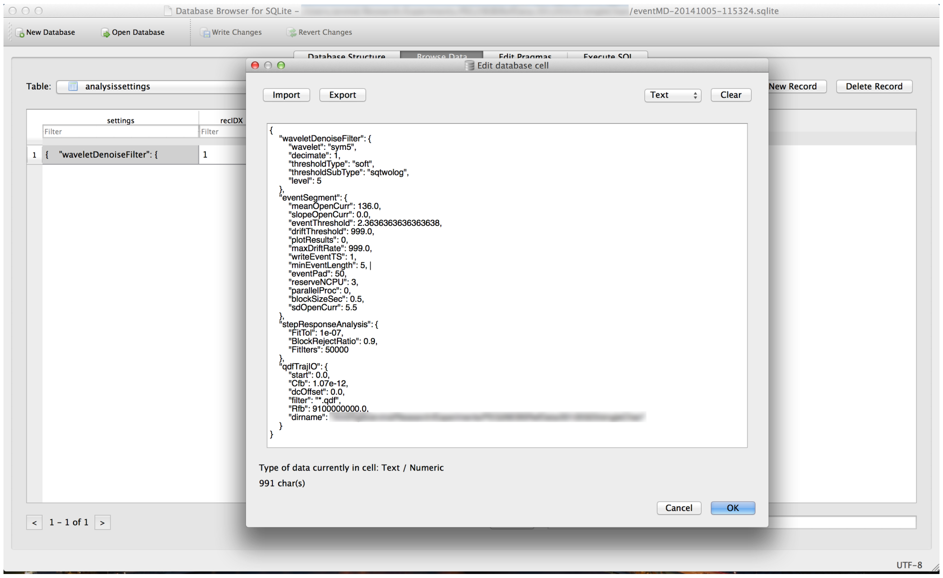


	recIDX	ProcessingStatus	OpenChCurrent	BlockedCurrent	EventStart	EventEnd	BlockDepth	ResTime	RCConstant	AbsEventStart	:ducedChiSquar	TimeSeries
184	184	normal	136.048481...	47.8884398...	0.102442...	0.2063717...	0.3519954...	0.1039288...	0.0017467...	458.10244...	1.0428571...	kpkLTtWwY...
185	185	normal	136.388727...	49.4119037...	0.101160...	0.2531239...	0.3622872...	0.1519637...	0.0020683...	468.72916...	1.0333333...	Snwk52FRY...
186	186	normal	136.295727...	48.7546930...	0.099610...	0.1772262...	0.3577125...	0.0776155...	0.0010883...	469.76761...	1.0451127...	vCqthtlEYU...
187	187	normal	136.728134...	48.4160741...	0.099204...	0.1834360...	0.3541046...	0.0842319...	0.0030160...	473.43920...	1.0483870...	+HTs4mqq...
188	188	normal	136.163621...	75.2090860...	0.075814...	0.0837541...	0.5523434...	0.0079394...	0.0001326...	473.60581...	1.0722891...	xR+Ayy2JY...
189	189	normal	136.479801...	98.4793567...	0.099866...	0.1059172...	0.7215672...	0.0060503...	0.0001528...	474.81986...	1.0645161...	yymZDSQ/Y...
190	190	normal	135.580797...	49.4561541...	0.099369...	0.2429424...	0.3647725...	0.1435727...	0.0014158...	478.93336...	1.0348837...	0ihyccrlYEA...
191	191	eInvalidFitParams	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	o2rFw5Q/Y...
192	192	normal	136.060431...	48.0983706...	0.011056...	0.1306499...	0.3535074...	0.1195933...	0.0031418...	479.48305...	1.0540540...	REvStQEgY...
193	193	normal	135.394986...	48.6723594...	0.099333...	0.2170996...	0.3594842...	0.1177660...	0.0023736...	480.64333...	1.0425531...	x8pPu49QY...
194	194	normal	136.532025...	49.9108603...	0.079606...	0.1311501...	0.3655615...	0.0515433...	0.0020511...	480.84560...	1.0512820...	AqN6vByAY...
195	195	normal	136.016300...	49.8596692...	0.098892...	0.1492097...	0.3665712...	0.0503174...	0.0015597...	481.54889...	1.0512820...	eDSY4ajfYE...
196	196	normal	136.775120...	47.7841358...	0.101028...	0.2825822...	0.3493627...	0.1815534...	0.0016714...	482.09102...	1.0315789...	r+DU/bKXY...
197	197	normal	137.516752...	56.9740218...	0.100124...	0.1413854...	0.4143060...	0.0412613...	0.0012905...	483.30812...	1.0545454...	b7wIAu0EY...
198	198	normal	135.728152...	58.8360340...	0.101791...	0.1117746...	0.4334843...	0.0099829...	0.0001607...	485.43179...	1.0612244...	lmO5sccrY...
199	199	normal	137.152200...	46.6580419...	0.099647...	0.2092531...	0.3401917...	0.1096055...	0.0025605...	486.42764...	1.0410958...	o/axZY9sY...
200	200	normal	137.110974...	48.7001968...	0.099491...	0.1532354...	0.3551881...	0.0537435...	0.0021878...	487.13949...	1.0483870...	s7gjWEg6Y...

< 184 – 200 of 11306 >

7.2 Analysis Settings Table

The *analysissettings* table contains a single text entry that stores the settings file for the analysis. This allows any database opened with the *MOSAIC* GUI to retrieve settings that correspond to the analysis results in the file. As seen from the figure below, the settings file is in the JSON format as described in the *Settings File* documentation.



```
{
    "waveletDenoiseFilter": {
        "wavelet": "sym5",
        "decimate": 1,
        "thresholdType": "soft",
        "thresholdSubType": "sqtwolog",
        "level": 5
    },
    "eventSegment": {
        "meanOpenCurr": 136.0,
        "slopeOpenCurr": 0.0,
        "eventThreshold": 2.3636363636363638,
        "driftThreshold": 999.0,
        "plotResults": 0,
        "maxDriftRate": 999.0,
        "writeEventTS": 1,
        "minEventLength": 5, |
        "eventPad": 50,
        "reserveNCPU": 3,
        "parallelProc": 0,
        "blockSizeSec": 0.5,
        "sdOpenCurr": 5.5
    },
    "stepResponseAnalysis": {
        "FitTol": 1e-07,
        "BlockRejectRatio": 0.9,
        "FitIters": 50000
    },
    "qdfTrajIO": {
        "start": 0.0,
        "Cfb": 1.07e-12,
        "dcOffset": 0.0,
        "filter": "*.qdf",
        "Rfb": 9100000000.0,
        "dirname": [redacted]
    }
}
```

Type of data currently in cell: Text / Numeric

991 char(s)

7.3 Work with SQLite

MOSAIC stores the output of an analysis in a SQLite database as described in the *Database Structure and Query Syntax* section. Interacting with the data through the Structured Query Language (SQL) is a flexible approach to further analyze or plot the output. Here we provide a few detailed examples of the common ways in which the output of *MOSAIC* can be queried for further processing. While this section is not a comprehensive SQL tutorial, it provides common use cases to allow you to get started.

One way to retrieve data from a SQLite database is to use the *select* command. In its simplest form, a *select* query can return the entire contents of a table using the syntax below. The statement below selects all columns *(select *)* from the table specified by <*tablename*>.

```
select * from <tablename>
```

The power of SQL lies in its ability to restrict results to match specific criteria. This is accomplished with the *where* clause described next. SQL queries can be very fast event for large databases. It is often desirable to only include events that were successfully fit in a plot or other analysis. All eventprocess-page algorithms implemented in *MOSAIC* store a *ProcessingStatus* column in the output database. This enables one to easily query events that were successfully processed. This is easily accomplished with the query below, which returns all columns for events that were successfully processed (*ProcessingStatus=normal*).

```
select * from metadata where ProcessingStatus='normal'
```

It is not always necessary to retrieve every column for events that fit a certain criteria. For example, gui-blockdepth-sec in the GUI displays a histogram of the blockade depths that match a user specified criteria. This is accomplished within the GUI by a query similar to the one shown below. There are two important differences between the query below and previous examples: i) by replacing * with *BlockDepth*, we only retrieve the *BlockDepth* column for events that meet the criteria specified after the *where* clause, and ii) selection criteria specified after where can be compound statements or even nested as seen in the examples below.

```
select BlockDepth from metadata where ProcessingStatus='normal' and ResTime > 0.2
```

```
select BlockDepth from metadata where ProcessingStatus='normal' and ResTime > 0.2
and BlockDepth between 0.1 and 0.5
```

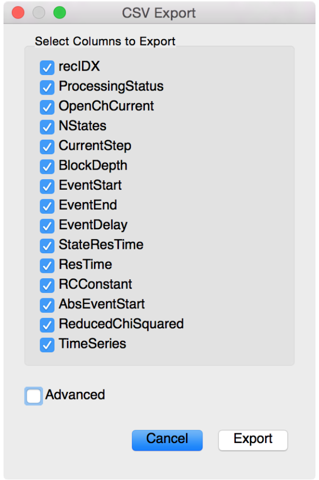
Multiple columns can be retrieved from a table by providing a comma separated list of column names after the *select* clause. As in previous cases, only events that meet a specified criteria are returned. The results can be ordered using *order*. In this example we sort the results in ascending order by the *AbsEventStart* column.

```
select BlockDepth, ResTime, AbsEventStart from metadata where ProcessingStatus='normal
↪'
order by AbsEventStart ASC
```

Finally, SQL allows the number of results returned to be limited using the *limit* clause. In this example, we limit the query results to the first 500 rows that meet our criteria.

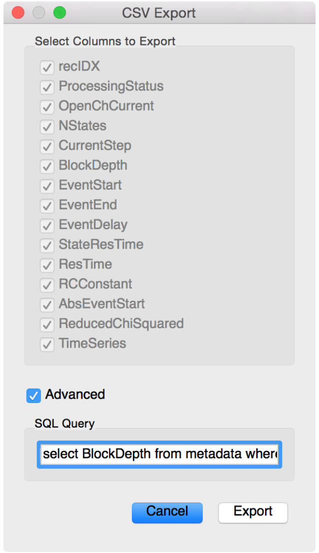
```
select AbsEventStart from metadata where ProcessingStatus='normal'
order by AbsEventStart ASC limit 500
```

7.4 Export to CSV



☑ recIDX
☑ ProcessingStatus
☑ OpenChCurrent
☑ NStates
☑ CurrentStep
☑ BlockDepth
☑ EventStart
☑ EventEnd
☑ EventDelay
☑ StateResTime
☑ ResTime
☑ RCConstant
☑ AbsEventStart
☑ ReducedChiSquared
☑ TimeSeries

☐ Advanced



Select Columns to Export

- ☑ recIDX
- ☑ ProcessingStatus
- ☑ OpenChCurrent
- ☑ NStates
- ☑ CurrentStep
- ☑ BlockDepth
- ☑ EventStart
- ☑ EventEnd
- ☑ EventDelay
- ☑ StateResTime
- ☑ ResTime
- ☑ RCConstant
- ☑ AbsEventStart
- ☑ ReducedChiSquared
- ☑ TimeSeries

☑ Advanced

SQL Query

```
select BlockDepth from metadata wher
```

CHAPTER 8

Scripting and Advanced Features

The analysis can be run from the command line by setting up a Python script. Scripting allows one to build additional analysis tools on top of *MOSAIC*. The first step is to import *MOSAIC* as shown below.

```
import mosaic
```

Alternatively, one can import sub-modules of *MOSAIC* directly into a script to access other parts of the system as shown below.

```
import mosaic.qdfTrajIO as qdf
import mosaic.abfTrajIO as abf

import mosaic.SingleChannelAnalysis
import mosaic.eventSegment as es
import mosaic.adept2State
import mosaic.besselLowpassFilter as bessel
```

8.1 Import Data and Run an Analysis

Once the required modules are imported, a basic analysis can be run with the code snippet below. The top-level object that is used to configure and run a new analysis is `SingleChannelAnalysis`, which takes five arguments: i) the path to the data directory, ii) a handle to a *TrajIO* object that reads in data (e.g. `abfTrajIO`), iii) a handle to a data filtering algorithm (e.g. `besselLowpassFilter` or *None* for no filtering), iv) a handle to a partitioning algorithm (e.g. `eventSegment`) that partitions the data and v) a handle to a processing algorithm (e.g. `adept2State`) that processes individual blockade events.

```
# Process all ABF files in a directory
analysisObj=mosaic.SingleChannelAnalysis.SingleChannelAnalysis(
            '~/ReferenceData/abfSet1',
            abf.abfTrajIO,
            None,
            es.eventSegment,
```

(continues on next page)

(continued from previous page)

```
            moasaic.adept2State.adept2State
        )
```

The analysis is started by calling the *Run()* function.

```
analysisObj.Run()
```

The code listing above analyzes all ABF files in the specified directory. Handles to trajectory I/O, data filtering, event partitioning and event processing are controlled with their corresponding sections in the *Settings File*. Default settings used to read ABF files are shown below.

```
"abfTrajIO" : {
    "filter"             : "*.abf",
    "start"              : 0.0,
    "dcOffset"           : 0.0
}
```

MOSAIC also supports the QUB QDF file format used by the Electronic Biosciences Nanopatch system. This is accomplished by replacing `abfTrajIO` in the previous example with `qdfTrajIO`. Settings for QDF files require two additional parameters to be specified in the settings file, the feedback resistance (Rfb) in Ohms and capacitance (Cfb) in Farads as described in the *API Documentation*. A sample section of the settings file to read QDF files, followed by Python code required to run an anlysis, is shown below.

```
"qdfTrajIO": {
        "Rfb"                : 9.1e+12,
        "Cfb"                : 1.07e-12,
        "dcOffset"           : 0.0,
        "filter"             : "*.qdf",
        "start"              : 0.0
    }
```

```
# Process all QDF files in a directory
mosaic.SingleChannelAnalysis.SingleChannelAnalysis(
            '~/ReferenceData/qdfSet1',
            qdf.qdfTrajIO,
            None,
            es.eventSegment,
            mosaic.adept2State.adept2State
        ).Run()
```

Upon completion the analysis writes a log file to the directory containing the data. The log file summarizes the conditions under which the analysis were run, the settings used and timing information.

```
Start time: 2014-10-05 11:53 AM

[Status]
    Segment trajectory: ***USER STOP***
    Process events: ***NORMAL***


[Summary]
    Baseline open channel conductance:
        Mean    = 136.0 pA
        SD  = 5.5 pA
        Slope   = 0.0 pA/s
```

(continues on next page)

(continued from previous page)

```

    Event segment stats:
        Events detected = 11306

        Open channel drift (max) = 0.0 * SD
        Open channel drift rate (min/max) = (-2.77/3.0) pA/s


[Settings]
    Trajectory I/O settings:
        Files processed = 27
        Data path = ~/ReferenceData/qdfSet1
        File format = qdf
        Sampling frequency = 500.0 kHz

        Feedback resistance = 9.1 GOhm
        Feedback capacitance = 1.07 pF

    Event segment settings:
        Window size for block operations = 0.5 s
        Event padding = 50 points
        Min. event rejection length = 5 points
        Event trigger threshold = 2.36363636364 * SD

        Drift error threshold = 999.0 * SD
        Drift rate error threshold = 999.0 pA/s


    Event processing settings:
        Algorithm = adept2State

        Max. iterations  = 50000
        Fit tolerance (rel. err in leastsq)  = 1e-07
        Blockade Depth Rejection = 0.9



[Output]
    Output path = ~/ReferenceData/qdfSet1
    Event characterization data = ~/ReferenceData/qdfSet1/eventMD-20141005-115324.
↪sqlite
    Event time-series = ***enabled***
    Log file = eventProcessing.log

[Timing]
    Segment trajectory = 98.03 s
    Process events = 0.0 s

    Total = 98.03 s
    Time per event = 8.67 ms
```

8.1.1 Filter Data

```
# Filter data with a Bessel filter before processing
mosaic.SingleChannelAnalysis.SingleChannelAnalysis(
            '~/ReferenceData/abfSet1',
            abf.abfTrajIO,
            bessel.besselLowpassFilter,
            es.eventSegment,
            mosaic.adept2State.adept2State
        ).Run()
```

MOSAIC supports filtering data prior to analysis. This is achieved by passing the *dataFilterHnd* argument to the `SingleChannelAnalysis` object. In the code above, the ABF data is filtered using a `besselLowpassFilter`. Parameters for the filter are defined within the settings file as described in the *Settings File* section.

```
"besselLowpassFilter" : {
    "filterOrder"    : "6",
    "filterCutoff"   : "10000",
    "decimate"       : "1"
}
```

A similar approach can be used to filter data using a `waveletDenoiseFilter` or a tap delay line (`convolutionFilter`). Additional filters can be easily added to *MOSAIC* as described in *Extend MOSAIC*.

8.2 Advanced Scripting

Scripting with Python allows transforming the output of the *MOSAIC* further to generate plots, perform additional analysis or extend functionality. Moreover, individual components of the *MOSAIC* module, which forms the back end code executed in the data processing pipeline, can be used for specific tasks. In this section, we highlight a few typical use cases.

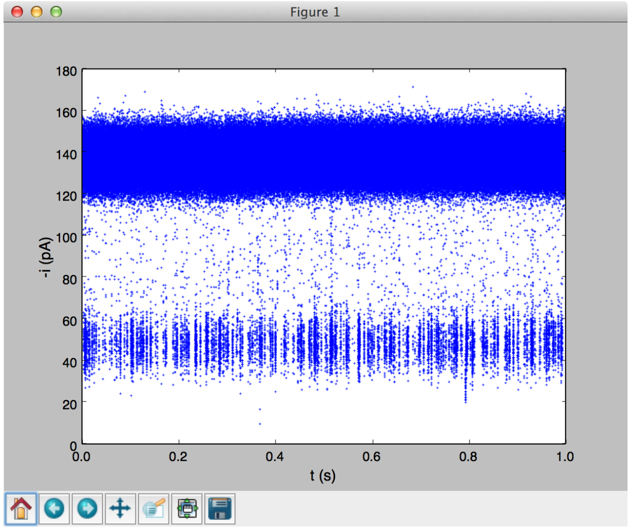
Plot the Ionic Current Time-Series

```
import mosaic.abfTrajIO as abf
import matplotlib.pyplot as plt
import numpy as np

abfDat=abf.abfTrajIO(dirname='~/abfSet1/', filter='*.abf')
plt.plot( np.arange(0,1,1/500000.), b.popdata(500000), 'b.', markersize=2 )
plt.xlabel("t (s)", fontsize=14)
plt.ylabel("-i (pA)", fontsize=14)
plt.show()
```

It is useful to visualize time-series data to highlight unique characteristics of a sample. For example the sample code above was used to load 1 second of monodisperse PEG28 data, sampled at 500 kHz. The data was read using a `abfTrajIO` object similar to the examples above. The `popdata()` command was used to take 500k data points (or 1 second) and then plot a time-series using matplotlib (see figure below). Calling `popdata()` again will return the next *n* points.

We have packaged time-series plotting into an easy to use module *timeseries*. Run interactive examples in an IPython notebook: **|timeseries|**

Estimate the Channel Gating Duration



Scripting can be used to obtain statistics from the raw time-series. In the code snippet below, we estimate the amount of time a channel spends in a gated state by combining modules defined within *MOSAIC*. The analysis is performed in blocks for efficiency. We first define a Python function that takes multiple arguments including *TrajIO* object, the threshold at which we want to define the gated state in pA (gatingcurrentpa), the block size in seconds (blocksz), the total time of the time-series being processed in seconds (totaltime) and the sampling rate of the data in Hz (fshz). The function then calculates the number of blocks in which the channel was in a gated state and returns the time spent in that state in seconds.

```
import mosaic.abfTrajIO as abf
import numpy as np

def estimateGatingDuration( trajioobj, gatingcurrentpa, blocksz, totaltime, fshz ):
    npts = int((fshz)*blocksz)
    nblk = int(totaltime/blocksz)-1

    # Iterate over the blocks of data and check if the channel was in a gated state.
    # The code below returns the mean ionic current of blocks that are below the␣
↪gating
    # threshold (gatingcurrentpa)
    gEvents = filter(  lambda x:x<float(gatingcurrentpa),
                       [ np.mean(trajioobj.popdata(npts)) for i in range(nblk) ])

    return len(gEvents)*blocksz

abfObj=abf.abfTrajIO(dirname='~/abfSet1',filter='*.abf')
print estimateGatingDuration( abfObj, 20., 0.25, 100, abfObj.FsHz )
```

Plot the Output of an Analysis

This final example shows how one can use *MOSAIC* to process an ionic current time-series and then build a custom script that further analyses and plots the results. This example uses single-molecule mass spectrometry (SMMS) data [Robertson:2007jo], described in more detail in the smms-sec section .

In the code below, we first process all the ABF files in a specified directory similar to the examples in previous sections. Upon completion of the analysis, the results are stored in a SQLite database, which can be then queried using the structured query language (SQL).

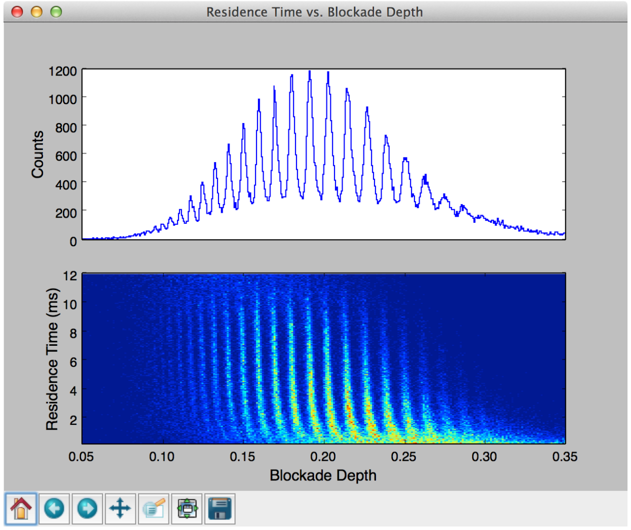
```
import mosaic.qdfTrajIO as qdf
```

(continues on next page)

(continued from previous page)

```
import mosaic.abfTrajIO as abf

import mosaic.SingleChannelAnalysis
import mosaic.eventSegment as es
import mosaic.adept2State

import glob
import pylab as pl
import numpy as np
import mosaic.sqlite3MDIO as sql

# Process all ABF files in a directory
mosaic.SingleChannelAnalysis.SingleChannelAnalysis(
            '~/ReferenceData/abfSet1',
            abf.abfTrajIO,
            None,
            es.eventSegment,
            mosaic.adept2State.adept2State
        ).Run()


# Load the results of the analysis
s=sql.sqlite3MDIO()
s.openDB(glob.glob("~/ReferenceData/abfSet1/*sqlite")[-1])

# We first set up a string that holds the query to retrieve the analysis results. Note that {col}
# will be replaced with the name of the database column when we run the query below.
q = "select {col} from metadata where ProcessingStatus='normal' and ResTime > 0.2 \
     and BlockDepth between 0.15 and 0.55"

# Now we run two separate queries - the first returns the blockade depth
# and the second returns the residence time. Note that we simply take the query
# string 'q' above and replace {col} with the column name.
x=np.hstack( s.queryDB( q.format(col='BlockDepth') ) )
y=np.hstack( s.queryDB( q.format(col='ResTime') ) )

# Use matplotlib to plot the results with 2 views:
# i)  a 1D histogram of blockade depths and
# ii) a 2D histogram of the residence times vs. blockade depth
fig = pl.gcf()
fig.canvas.set_window_title('Residence Time vs. Blockade Depth')

pl.subplot(2, 1, 1)
pl.hist(x, bins=500, histtype='step', rwidth=0.1)
pl.xticks(())
pl.ylabel("Counts", fontsize=14)

pl.subplot(2, 1, 2)
pl.hist2d(x,y, bins=500)

pl.xlabel("Blockade Depth", fontsize=14)
pl.ylabel("Residence Time (ms)", fontsize=14)
pl.ylim([0.2, 20])

pl.show()
```

Running the code above generates a two pane plot using matplotlib. The top pane contains a histogram of the blockade depth, while the bottom pane plots a 2D histogram of residence time vs. blockade depth.



CHAPTER 9

Extend *MOSAIC*

MOSAIC was designed from the start using object oriented tools, which makes it easy to extend. *Meta-Classes* define interfaces to five key parts of *MOSAIC*: time-series IO (`metaTrajIO`), time-series filtering (`metaIOFilter`), analysis output (`metaMDIO`), event partition and segmenting (`metaEventPartition`), and event processing (`metaEventProcessor`). Sub-classing any of these meta classes and implementing their interface functions allows one to extend *MOSAIC* while maintaining compatibility with other parts of the program. We highlight these capabilities via two examples. In the first example, we show how one can extend `metaTrajIO` to read arbitrary binary files. In the second example, we implement a new top-level class that converts files to the comma separated value (CSV) format.

9.1 Read Arbitrary Binary Data Files

In this first example, we implement a class that can read an arbitrary binary data file and make its data available via the interface functions in `metaTrajIO`. This allows the newly implemented binary data to be used across *MOSAIC*. A complete listing of the code used in this example (`binTrajIO`) is available in the API documentation.

The new binary IO class is implemented by sub-classing `metaTrajIO` as shown in the listing below.

```
class binTrajIO(metaTrajIO.metaTrajIO):
```

Next, we must fully implement the `metaTrajIO` interface functions (`_init()`, `readdata()` and `_formatsettings()`). Note that the arguments of each function must match their corresponding base-class versions. For example the `_init()` function only accepts keyword arguments and is defined as shown below.

```
def _init(self, **kwargs):
```

The `_init()` function checks the arguments passed to *kwargs* and raises an exception if they are not defined.

```
        if not hasattr(self, 'SamplingFrequency'):
                raise metaTrajIO.InsufficientArgumentsError("{0} requires the␣
↪sampling rate in Hz to be defined.".format(type(self).__name__))
        if not hasattr(self, 'PythonStructCode'):
                raise metaTrajIO.InsufficientArgumentsError("{0} requires the Python␣
↪struct code to be defined.".format(type(self).__name__))
```

(continues on next page)

(continued from previous page)

Next we define the `readdata()` function that reads in the data and stores the results in a numpy array. This array is then passed back to the calling function.

```
     def readdata(self, fname):

             tempdata=np.array([])
             # Read binary data and add it to the data pipe
             for f in fname:
                     tempdata=np.hstack(( tempdata, self.readBinaryFile(f) ))

             return tempdata
```

Finally, we implement the `_formatsettings()` that returns a formatted string of the settings used to read in binary data.

```
def _formatsettings(self):
        """
                Return a formatted string of settings for display
        """
        fmtstr=""

        fmtstr+='\n\t\tAmplifier scale = {0} pA\n'.format(self.AmplifierScale)
        fmtstr+='\t\tAmplifier offset = {0} pA\n'.format(self.AmplifierOffset)
        fmtstr+='\t\tHeader offset = {0} bytes\n'.format(self.HeaderOffset)
        fmtstr+='\t\tData type code = \'{0}\'\n'.format(self.PythonStructCode)

        return fmtstr
```

The newly defined `binTrajIO` class can then be used as shown below and in *Scripting and Advanced Features*.

```
# Process all binary files in a directory
mosaic.SingleChannelAnalysis.SingleChannelAnalysis(
            "~/RefData/binSet1/",
            bin.binTrajIO,
            None,
            es.eventSegment,
            mosaic.adept2State.adept2State
        ).Run()
```

Similar to other *TrajIO* objects, parameters for `binTrajIO` are obtained from the settings file when used with `SingleChannelAnalysis`. Example settings for `binTrajIO` that read 16-bit intgers from a binary data file, assuming 50 *kHz* sampling, are shown below.

```
"binTrajIO" : {
        "filter"                : "*bin",
        "AmplifierScale"        : "1.0",
        "AmplifierOffset"       : "0.0",
        "SamplingFrequency"     : "50000",
        "HeaderOffset"          : "0",
        "PythonStructCode"      : "'h'"
}
```

9.2 Define Top-Level Functionality

New functionality can be added to *MOSAIC* by combining other parts of the code. One way of accomplishing this is by defining new top-level functionality as shown in the following example. We define a new class that converts data from one of the supported data formats to comma separated text files (CSV). A complete listing of the `ConvertToCSV` class in this example is available in the API documentation.

The *__init__* function of `ConvertToCSV` class accepts two arguments: a trajIO object and the location to save the converted files. If the output directory is not specified, the data is saved in the same folder as the input data. The data conversion is performed by the `Convert()` function, which saves the data in blocks controlled by the *blockSize* parameter. `Convert()` saves each block to a new CSV file, named with the filename of the input data followed by an integer number (see the API documentation for `_filename()` for additional details).

```
class ConvertToCSV(object):
        def __init__(self, trajDataObj, outdir=None):
                self.trajDataObj=trajDataObj
                self.datPath=trajDataObj.datPath

                # If outdir is None, save the CSV files to the same directory as the␣
↪data.
                if outdir==None:
                        self.outDir=self.datPath
                else:
                        self.outDir=outdir

                self.filePrefix=None
                self._creategenerator()

        def Convert(self, blockSize):
                data=numpy.array([], dtype=numpy.float64)

                try:
                        while(True):
                                (self.trajDataObj.popdata(blockSize)).tofile(
                                                self._filename(),
                                                sep=','
                                        )
                except EmptyDataPipeError:
                        pass
```

The `ConvertToCSV` class can now be used with any trajIO object as seen below.

```
ConvertToCSV( abfTrajIO(dirname="~/RefData/abfSet1/", filter="*abf") ).Convert(
                        blockSize=50000)

ConvertToCSV( qdfTrajIO(dirname="~/RefData/qdfSet1/", filter="*qdf", Rfb="2.1E+9",
                        Cfb="1.16E-12") ).Convert(blockSize=50000)

ConvertToCSV( binTrajIO(dirname="~/RefData/binSet1/", filter="*bin", AmplifierScale=1.
↪0,
                        AmplifierOffset=0.0, SamplingFrequency=50000, HeaderOffset=0,
                        PythonStructCode='h') ).Convert(blockSize=50000)
```

Since `ConvertToCSV` accepts a trajIO object, we can apply a lowpass filter to the data before converting it to the CSV format. This is accomplished by passing the *datafilter* option to the trajIO object as described in the *Filter Data* section. In the example below, we convert ABF files to the CSV format after applying a lowpass Bessel filter to the data.

```
ConvertToCSV( abfTrajIO(dirname="~/RefData/abfSet1/", filter="*abf",
                datafilter=mosaic.besselFilter
        ) ).Convert(blockSize=50000)
```

Finally, the `ConvertToCSV` class can be further extended to output arbitrary binary files in place of CSV by the simple extension shown below.

```
"""
        Extend the MOSAIC ConvertToCSV class to export arbitrary binary files.

        :Created:       02/25/2015
        :Author:        Arvind Balijepalli <arvind.balijepalli@nist.gov>
        :ChangeLog:
        .. line-block::
                02/25/15        AB      Initial version
"""
import mosaic.ConvertToCSV as conv
import mosaic.binTrajIO as bin
import mosaic.settings as sett
import numpy as np

from mosaic.metaTrajIO import EmptyDataPipeError

class ConvertToBin(conv.ConvertToCSV):
        def Convert(self, blockSize, binType):
                """
                        Start converting data

                        :Parameters:
                                - `blockSize`   : number of data points to convert.
                                - `binType`     : Numpy binary type.
                """
                try:
                        while(True):
                                np.array( self.trajDataObj.popdata(blockSize),␣
↪dtype=binType ).tofile(self._filename())
                except EmptyDataPipeError:
                        pass


if __name__ == '__main__':
        s={
        "AmplifierOffset": 0.0,
        "SamplingFrequency": 250000,
        "AmplifierScale": "1.0",
        "ColumnTypes": "[('curr_pA', '>f8'), ('volts', '>f8')]",
        "dcOffset": 0.0,
        "filter": "*.bin",
        "start": 0.0,
        "HeaderOffset": 0,
        "IonicCurrentColumn": "curr_pA"
    }
        ConvertToBin(
                        bin.binTrajIO(dirname=".", **s ),
                        outdir="convert",
                        extension="bin"
                        ).Convert(blockSize=10000000, binType='f4')
```

CHAPTER 10

Publication Quality Figures

We provide packaged functions for publication quality plots using Python and matplotlib. Example plots using these modules are below.

10.1 Timeseries Plots

Generate publication time-series plots using the mosaicscripts.plots.timeseries module.

```
:Created:   11/19/2015
:Author:    Arvind Balijepalli <arvind.balijepalli@nist.gov>
:License:   See LICENSE.TXT
:ChangeLog:
    12/12/15    AB  Generalized plot function to allow different data types
    11/20/15    AB  Initial version
```

```
import mosaicscripts.plots.timeseries as ts
```

Plots are generated using the `mosaicscripts.plots.timeseries` module. See the timeseries module for additional details.

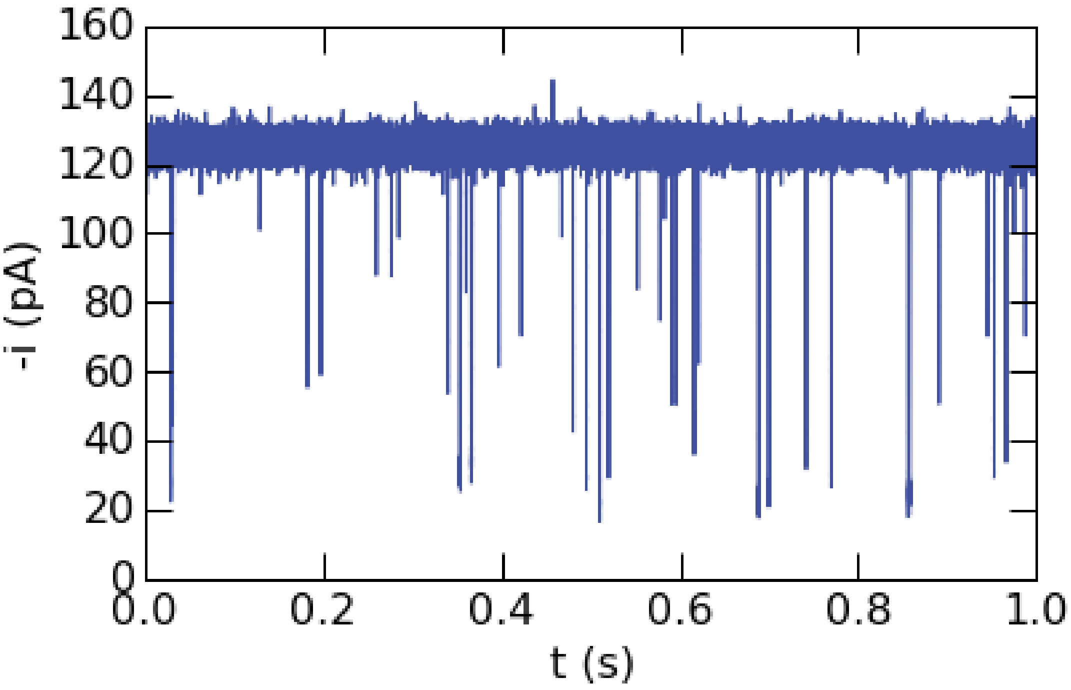
Basic usage to plot 1 second of ionic current vs. time is shown below. The `plotopts` argument is used to style the curve in the plot.

```
ts.PlotTimeseries(
    "../data/",
    "abf",
    5.0,
    6.0,
    50000,
    labels=["t (s)", "-i (pA)"],
    axes=True,
    polarity=-1,
    plotopts={
```

(continues on next page)

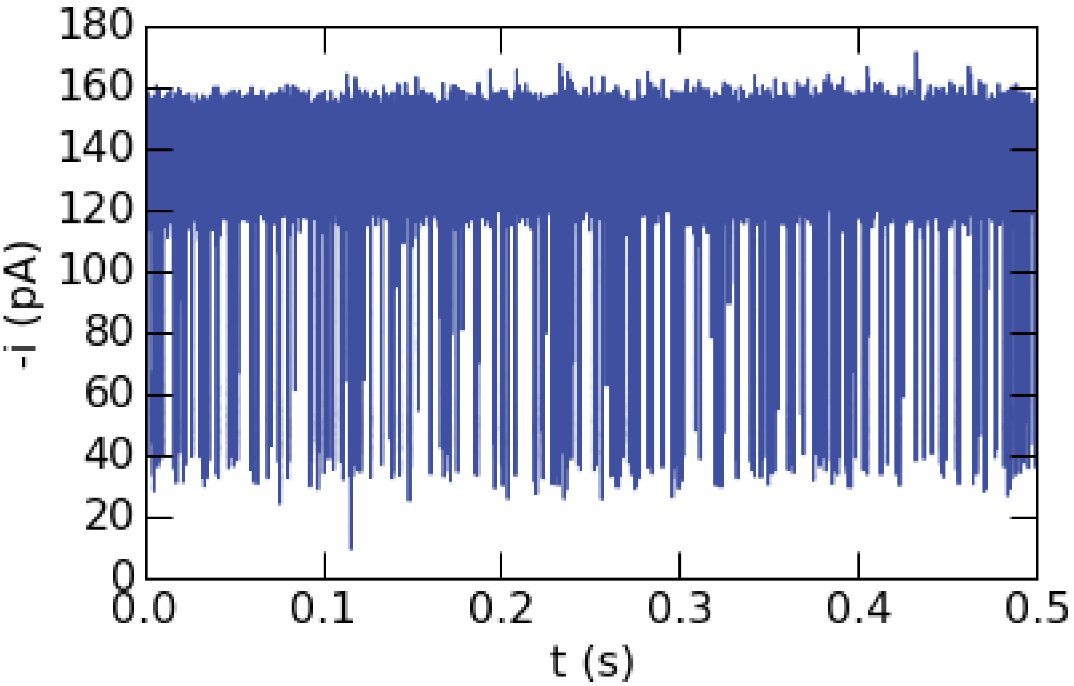
(continued from previous page)

```
        'color' : '#3F50A0',
        'marker' : '.',
        'markersize' : 0.2
    }
)
```



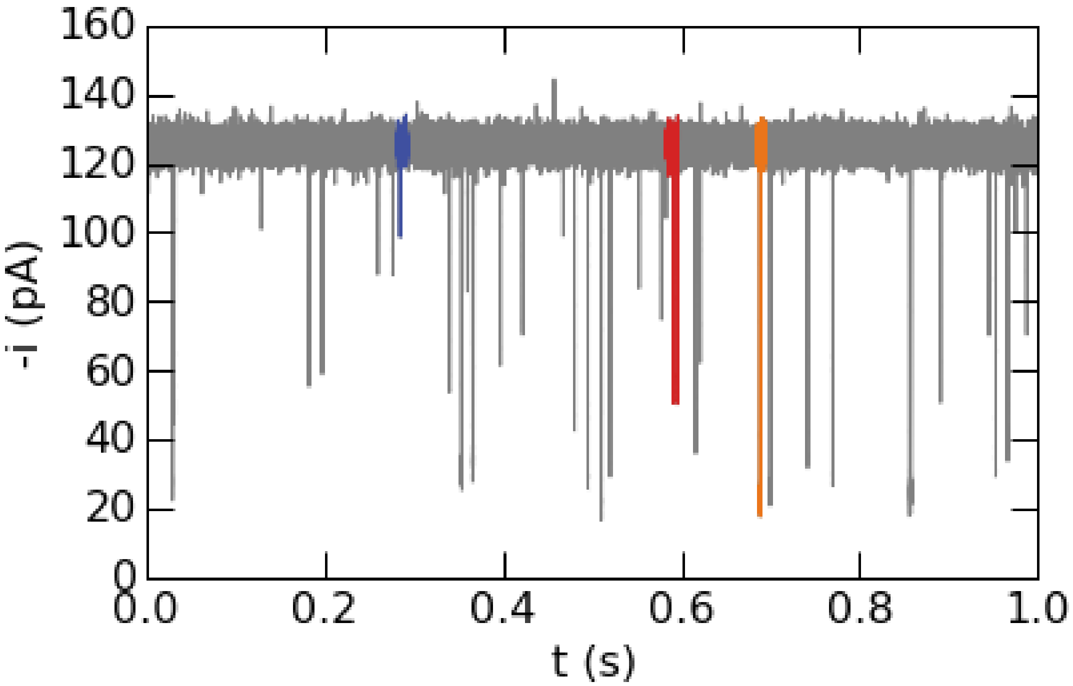
Plotting other data types is also straightforward. The next example demonstrates plotting a time-series in the QUB data format (QDF). Note that the conversion to voltage to current is performed with the `Rfb` and `Cfb` parameters, passed to the plotting function as a dictionary.

```
ts.PlotTimeseries(
    "../data/",
    "qdf",
    0.25,
    0.75,
    500000,
    labels=["t (s)", "-i (pA)"],
    axes=True,
    polarity=1,
    plotopts={
        'color' : '#3F50A0',
        'marker' : '.',
        'markersize' : 0.2
    },
    data_args={"Rfb" : 9.1e9, "Cfb": 1.07e-12}
)
```



Segments of timeseries can be highlighted for emphasis or to show specific features. Three blockade events are plotted using different colors in the example below.

```
ts.PlotTimeseries(
    "../data/",
    "abf",
    5.0,
    6.0,
    50000,
    labels=["t (s)", "-i (pA)"],
    axes=True,
    polarity=-1,
    plotopts={
        'color' : 'gray',
        'marker' : '.',
        'markersize' : 0.2
    },
    highlights=[
        [[0.282, 0.293], {'color' : '#3F50A0', 'marker' : '.', 'markersize' : 0.1}],
        [[0.584, 0.597], {'color' : '#D42324', 'marker' : '.', 'markersize' : 0.1}],
        [[0.685, 0.695], {'color' : '#EB751A', 'marker' : '.', 'markersize' : 0.1}]
    ],
    figname="timeseries.png"
)
```



10.2 Histogram Plots

Generate publication quality histogram plots using the mosaicscripts.plot.histogram module.

```
:Created:   12/14/2015
:Author:    Arvind Balijepalli <arvind.balijepalli@nist.gov>
:License:   See LICENSE.TXT
:ChangeLog:
    01/09/16         AB    Added a plot overlay example.
    12/14/15        AB   Initial version
```

```
import numpy as np
from scipy.optimize import curve_fit
```

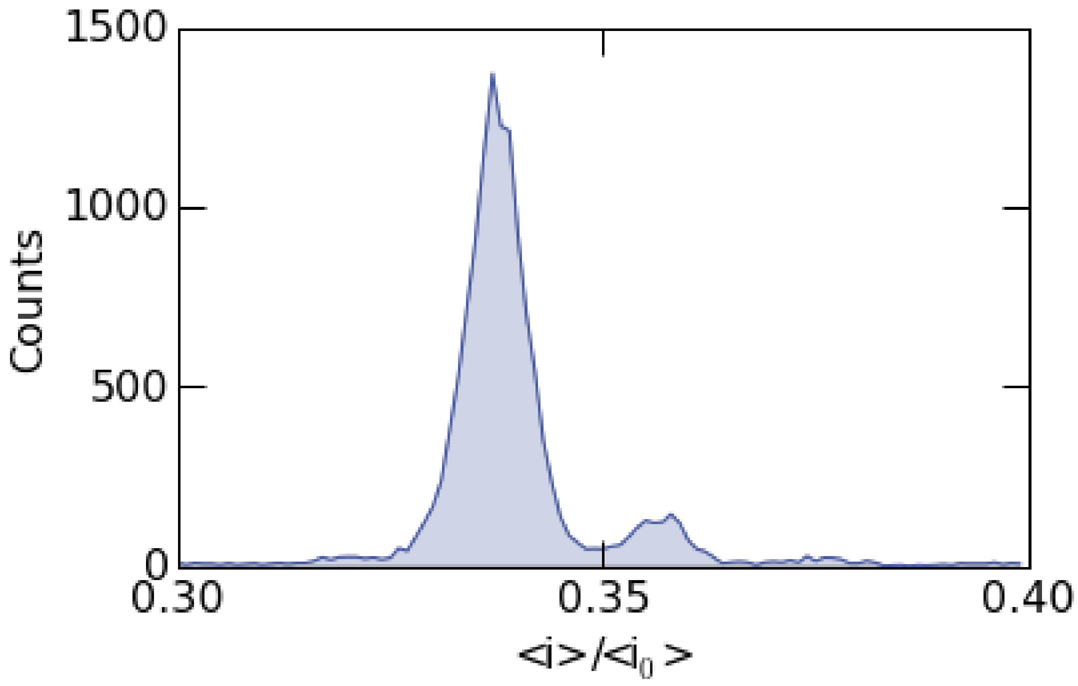
```
import mosaicscripts.plots.histogram as histogram
from mosaic.utilities.sqlQuery import query
```

```
q="select BlockDepth from metadata where ProcessingStatus='normal' and ResTime > 0.25␣
↪and BlockDepth between 0.3 and 0.4"
```

10.2.1 Basic Histogram Plots

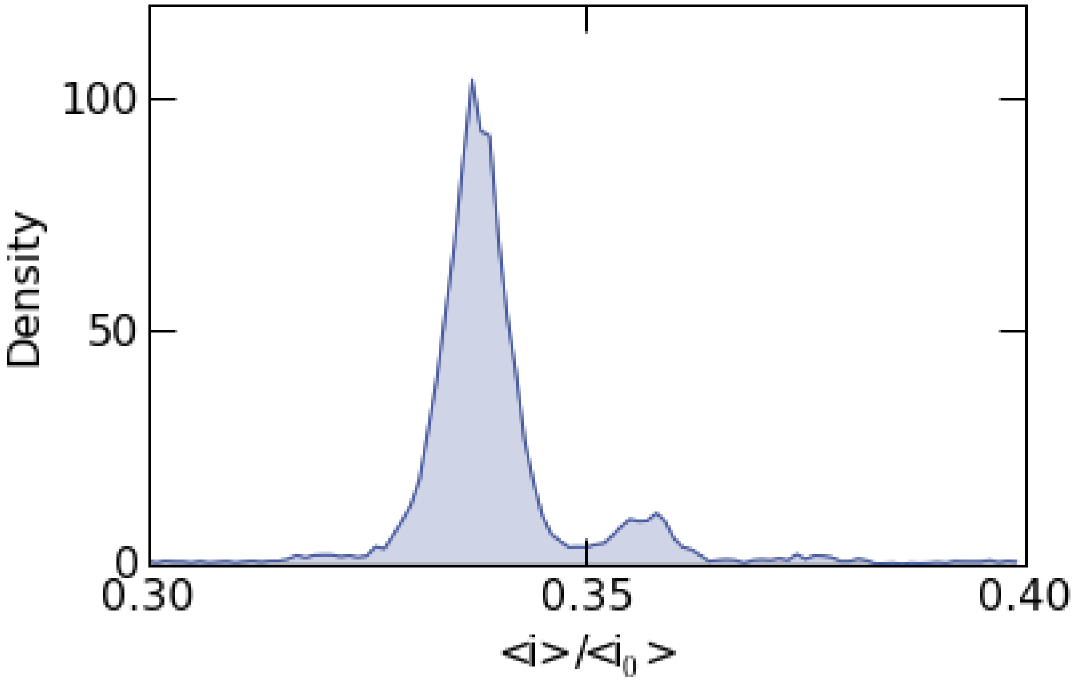
Plots are generated using the `mosaicscripts.plots.histogram.histogram_plot()` function. See the histogram module for additional details.

```
histogram.histogram_plot(
                            query("../data/eventMD-P28-bin.sqlite", q),
                            100,
                            (0.3, 0.4),
                            xticks= (0.3,0.35,0.4),
                            yticks=(0,500,1000,1500),
                            xlabel=r"<i>/<i$_0$>",
                            ylabel=r"Counts"
                    )
```



To plot the probability density, supply the argument `density`=True as shown below.

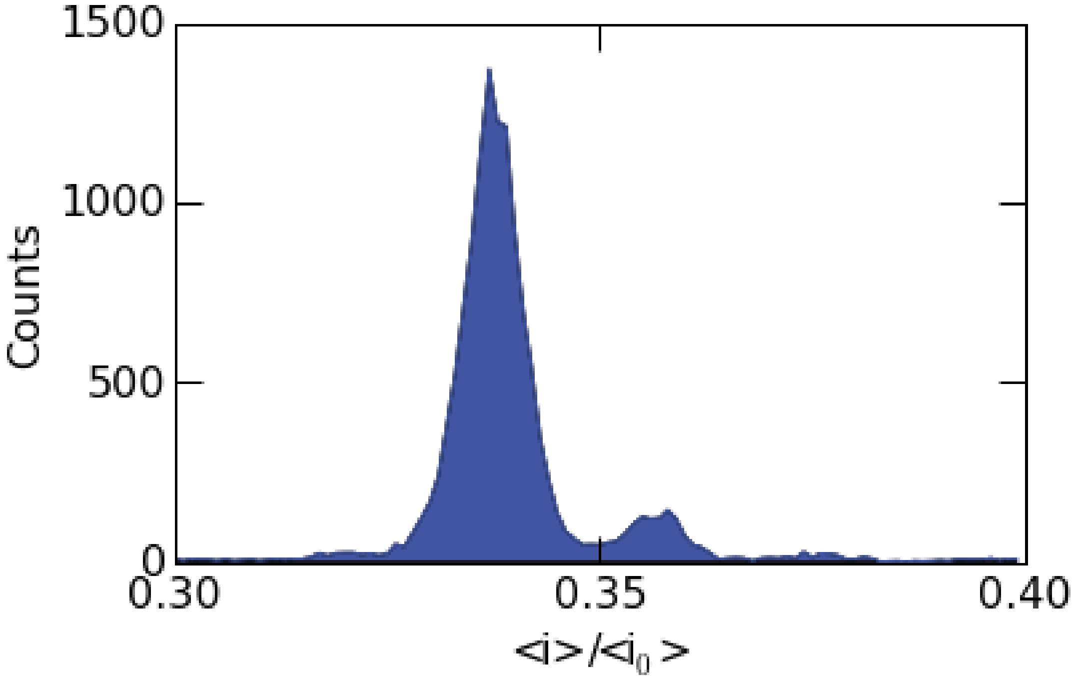
```
histogram.histogram_plot(
                            query("../data/eventMD-P28-bin.sqlite", q),
                            100,
                            (0.3, 0.4),
                            xticks= (0.3,0.35,0.4),
                            yticks=(0,50,100),
                            xlabel=r"<i>/<i$_0$>",
                            ylabel=r"Density",
                density=True
                    )
```



10.2.2 Custom Styles

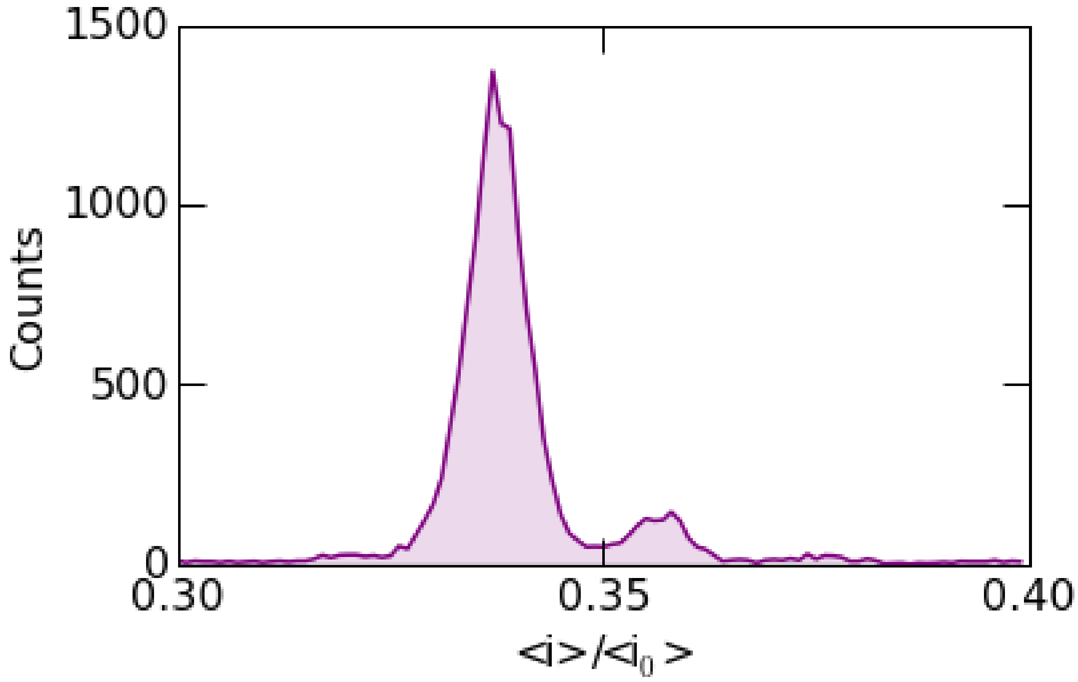
The fill transparency can be controlled with the `fill_alpha` argument. When set to `1`, it results in a filled plot as seen below. To turn off filling, simply set `fill_alpha`=0

```
histogram.histogram_plot(
                                query("../data/eventMD-P28-bin.sqlite", q),
                                100,
                                (0.3, 0.4),
                                xticks= (0.3,0.35,0.4),
                                yticks=(0,500,1000,1500),
                                xlabel=r"<i>/<i$_0$>",
                                ylabel=r"Counts",
                fill_alpha=1
                    )
```



Matplotlib plotting directies can be supplied to `histogram_plot()` using the `advanced_opts` argument. See the Matplotlib plot documentation for additional details. In the example below, the plot linewidth is set to 1.5 points.

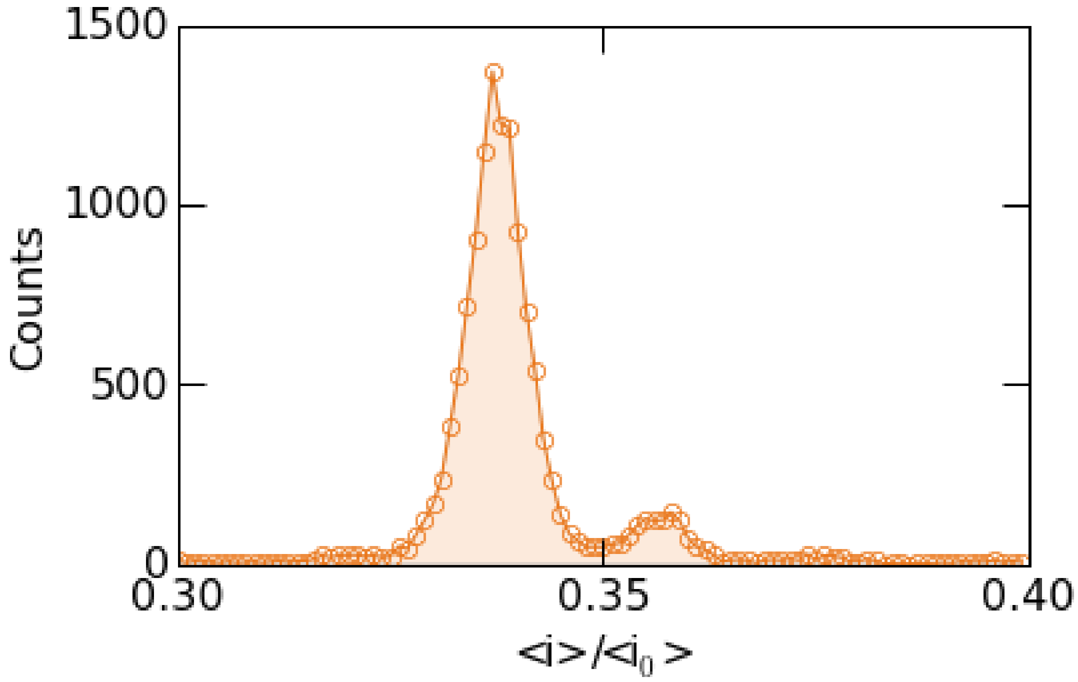
```
histogram.histogram_plot(
                            query("../data/eventMD-P28-bin.sqlite", q),
                            100,
                            (0.3, 0.4),
                            xticks= (0.3,0.35,0.4),
                            yticks=(0,500,1000,1500),
                            xlabel=r"<i>/<i$_0$>",
                            ylabel=r"Counts",
                color='purple',
                dpi=600,
                fill_alpha=0.15,
                advanced_opts={'linewidth': 1.5}
                    )
```



The example below shows more advanced styling. Circular markers can be placed at the center of each bin using the Matplotlib marker keywords.

Finally, images can be saved by supplying the `figname` argument as seen in the example below. Optionally, the figure resolution can be set with the `dpi` argument.

```
histogram.histogram_plot(
                                query("../data/eventMD-P28-bin.sqlite", q),
                                100,
                                (0.3, 0.4),
                                xticks= (0.3,0.35,0.4),
                                yticks=(0,500,1000,1500),
                                figname="histogram.png",
                                xlabel=r"<i>/<i$_0$>",
                                ylabel=r"Counts",
                color='#EB771A',
                dpi=600,
                fill_alpha=0.15,
                advanced_opts={
                                'marker': 'o',
                                'markersize': 6,
                                'markeredgecolor' : '#EB771A',
                                'markeredgewidth' : 0.75,
                                'markerfacecolor': 'none',
                                'linewidth': 1.
                        }
                    )
```



10.2.3 Advanced Analysis and Plot Overlays

The `mosaicscripts.plots.histogram.histogram_plot()` function allows one to overlay additional curves on top of the histogram data. This is useful, for example, to fit the histogram to a known functional form. Below we describe, how to fit the histogram data to a sum of two Gaussians.

First we must define the fit function as shown below. We sum two Gaussians of the form: $a_1 exp(-(x-\mu_1)^2/2\sigma_1^2) + a_2 exp(-(x-\mu_2)^2/2\sigma_2^2)$, where x is the independent variable, μ is the mean of the distribution, σ is the standard deviation, a is the amplitude and the subscripts denote the peak number.

```
def gauss_sum_fit(x, a1, mu1, sigma1, a2, mu2, sigma2):
    return a1*np.exp(-(x-mu1)**2/(2*sigma1**2)) + a2*np.exp(-(x-mu2)**2/(2*sigma2**2))
```

Next, we call the `histogram_plot` function as before. Note however there are two additional options we must provide to enable us to add the peak fits to the plot. The first is `show=False`, which suppresses plotting the histogram to allow additional plots to be added to the figure (see the Matplotlib documentation for details), and the second is `return_histogram=True`, which returns the raw histogram data that we fit to.

Next, we perform the least squares fit using the Scipy curve_fit function. The optimized parameters and covariance are stored in `popt` and `pcov` respectively.

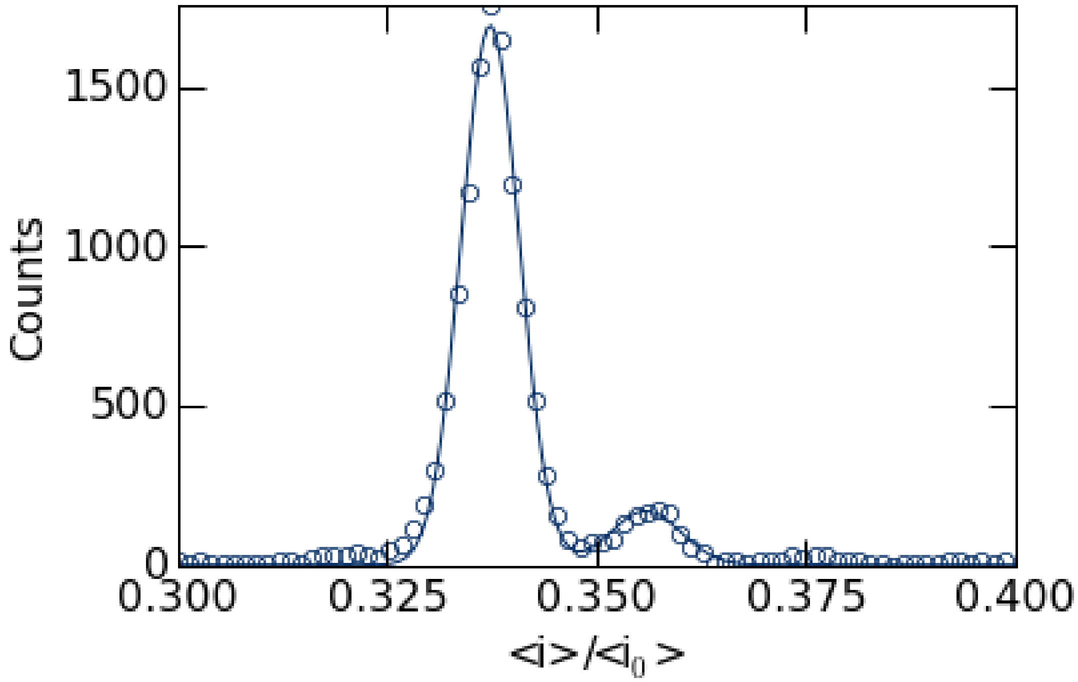
Finally, we plot the fit function and call show() to display the figure.

```
hist,bins=histogram.histogram_plot(
                            query("../data/eventMD-P28-bin.sqlite", q),
                            75,
                            (0.3, 0.4),
                            xticks= (0.3,0.325,0.35,0.375,0.4),
                            yticks=(0,500,1000,1500),
                            figname="histogram.png",
                            xlabel=r"<i>/<i$_0$>",
                            ylabel=r"Counts",
```

(continues on next page)

(continued from previous page)

```
                                fill_alpha=0.,
                                show=False,
                                return_histogram=True,
                                advanced_opts={
                                    'marker': 'o',
                                    'markersize': 6,
                                    'markeredgecolor' : '#002A63',
                                    'markeredgewidth' : 0.75,
                                    'markerfacecolor': 'none',
                                    'linewidth': 0.
                            }
         )

popt,pcov=curve_fit(gauss_sum_fit, bins, hist, [1200, 0.34,0.003, 100, 0.36,0.003])

xdat=np.arange(0.3, 0.4,0.0005)
ydat=gauss_sum_fit(xdat, *popt)

histogram.plt.plot(xdat, ydat, color="#002A63")
histogram.plt.show()
```



The `popt` variable holds the optimized fit parameters, stored in the order defined by the `gauss_sum_fit` above. We can extract these values from this list. For example, the peak positions can be retrieved as shown below.

```
popt[1], popt[4]
```

```
(0.33733498827022712, 0.35592403517767944)
```

10.3 Contour Plots

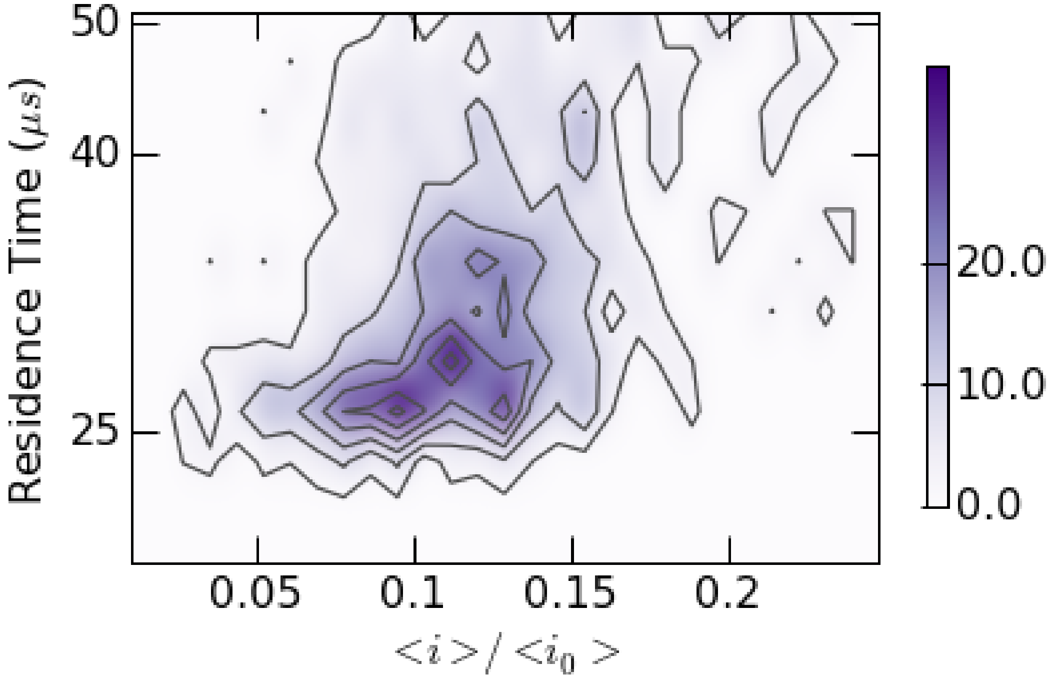
Generate publication quality contour plots using the mosaicscripts.plot.contour module.

```
:Created:   11/19/2015
:Author:    Arvind Balijepalli <arvind.balijepalli@nist.gov>
:License:   See LICENSE.TXT
:ChangeLog:
    11/19/15        AB  Initial version
```

```
import matplotlib.pyplot as plt
import mosaicscripts.plots.contour as contour
from mosaic.utilities.sqlQuery import query
```

Plots are generated using the `mosaicscripts.plots.contour_plot()` function. See the contour module for additional details.

```
contour.contour_plot(
                                    query(
                                            "../data/eventMD-20150404-221533_MSA.
↪sqlite",
                                            "select BlockDepth, StateResTime from␣
↪metadata where ProcessingStatus='normal' and BlockDepth > 0 and ResTime > 0.025"
                                            ),
                                    x_range=[0.01, 0.26],
                                    y_range=[0.02, 0.06],
                                    bin_size=0.0085,
                                    contours=6,
                                    colormap=plt.get_cmap('Purples'),
                                    img_interpolation='nearest',
                                    xticks=[
                                                    (0.05, '0.05'),
                                                    (0.1, '0.1'),
                                                    (0.15, '0.15'),
                                                    (0.2, '0.2')
                                                    ],
                                    yticks=[
                                                    (0.025, '25'),
                                                    (0.04, '40'),
                                                    (0.05, '50')
                                                    ],
                                    axes_type=['linear', 'log', 'linear'],
                    figname="contour.png",
                                    colorbar_num_ticks=4,
                                    cb_round_digits=-1,
                                    min_count_pct=0.08,     # Set bins with < 7% of␣
↪max to 0,
                                    xlabel=r"$<i>/<i_0>$",
                                    ylabel=r"Residence Time ($\mu s$)"
                    )
```



Plot styling can be controlled with custom colormaps. Examples are found within the `contour.gen_colormap()` function. Calling this function makes two additional colormaps (`mosaicBlue` and `mosaicOrange`) available as seen below.

```
contour.gen_colormaps()
```

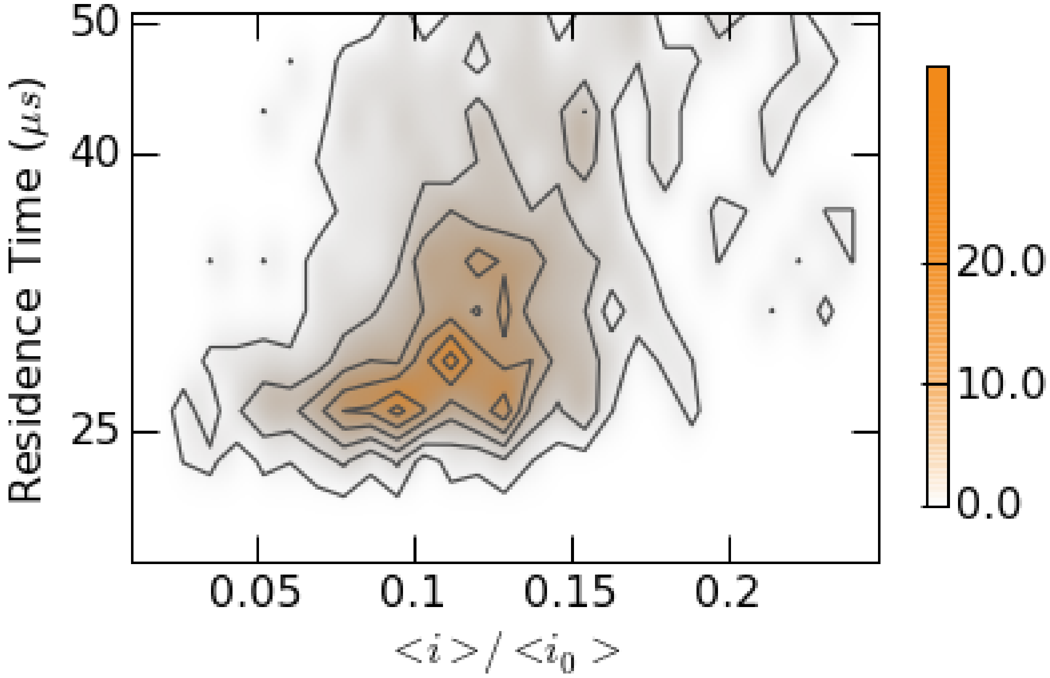
Note: The `colormap` argument is now uses `Orange1` as opposed to `Purples` above.

```
contour.contour_plot(
                                    query(
                                            "../data/eventMD-20150404-221533_MSA.
↪sqlite",
                                            "select BlockDepth, StateResTime from␣
↪metadata where ProcessingStatus='normal' and BlockDepth > 0 and ResTime > 0.025"
                                            ),
                                    x_range=[0.01, 0.26],
                                    y_range=[0.02, 0.06],
                                    bin_size=0.0085,
                                    contours=6,
                                    colormap=plt.get_cmap('mosaicOrange'),
                                    img_interpolation='nearest',
                                    xticks=[
                                                    (0.05, '0.05'),
                                                    (0.1, '0.1'),
                                                    (0.15, '0.15'),
                                                    (0.2, '0.2')
                                                    ],
                                    yticks=[
                                                    (0.025, '25'),
                                                    (0.04, '40'),
                                                    (0.05, '50')
                                                    ],
                                    axes_type=['linear', 'log','linear'],
```

(continues on next page)

(continued from previous page)

```
                        figname="contour.png",
                                    colorbar_num_ticks=4,
                                    cb_round_digits=-1,
                                    min_count_pct=0.08,      # Set bins with < 7% of␣
↪max to 0,

                                    xlabel=r"$<i>/<i_0>$",
                                    ylabel=r"Residence Time ($\mu s$)"
                        )
```



CHAPTER 11

Advanced Analysis

We provide packaged functions for advanced analysis such as calculating the capture rate of molecules, determining the residene time, etc as described below.

11.1 Capture Rate

Estimate the capture rate of molecules partitioning into a nanopore.

```
:Created:    12/27/2015
:Author:     Arvind Balijepalli <arvind.balijepalli@nist.gov>
:License:    See LICENSE.TXT
:ChangeLog:
    12/27/15        AB     Initial version
```

```
import numpy as np

from mosaicscripts.analysis.kinetics import CaptureRate
```

11.1.1 Wrapper Function to Estimate the Capture Rate

The capture rate can be estimated directly by calling the `CaptureRate` function in `mosaicscripts.analysis.kinetics`. The function returns a list with two elements: the capture rate (s:math:^*{-1}*), and the standard error of the capture rate (s:math:^*{-1}*).

```
np.round(
        CaptureRate(
            "../data/eventMD-P28-bin.sqlite",
            "select AbsEventStart from metadata where ProcessingStatus='normal' and␣
↪ResTime > 0.02 order by AbsEventStart ASC"
           ),
```

(continues on next page)

(continued from previous page)

```
        decimals=1
    )
```

```
array([ 27.9,    0.2])
```

11.1.2 Capture Rate Details

```
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

from mosaicscripts.analysis.kinetics import query1Col
import mosaicscripts.plots.mplformat as mplformat
from mosaic.utilities.fit_funcs import singleExponential
```

```
mplformat.update_rcParams()
```

Continue reading to dig deeper into how the capture rate is estimated within the `CaptureRate` function.

The first step is to read in the start times for each event. This is easily done with a query to the MOSAIC database as shown below. The start times are stored in the `AbsEventStart` column. We limit the events we use to estimate the capture rate to ones that were successfully fit (`ProcessingStatus`='normal') and those whose residence times (`ResTime`) in the pore are longer than 20 μs.

Finally, we sort the `AbsEventStart` to ensure the event start times are in ascending order.

```
start_times=query1Col(
        "../data/eventMD-P28-bin.sqlite",
        "select AbsEventStart from metadata where ProcessingStatus='normal' and␣
↪ResTime > 0.02 order by AbsEventStart ASC"
    )
```

Next, we calculate the arrival times, i.e. the time between the start of successive events. This is done with the `Numpy` diff function. Note that `AbsEventStart` is stored in milliseconds within the database. Here, we also convert the arrival times to seconds.

```
arrival_times=np.diff(start_times)/1000.
```

The partitioning of molecules into the pore is a stochastic process. There are however a couple properties related to stochastic process that we will leverage that makes the estimation of the capture rate more robust. With randomly occuring events that have some mean rate, the number of events scales linearly with time. Therefore, the distribution of these events follows a single exponential form. We can easily test this by calculating the probability density function (PDF) using the `Numpy` histogram function. Note that the `density=True` argument normalizes the histogram resulting in a PDF.

```
density,bins=np.histogram(arrival_times, bins=100, density=True)
```

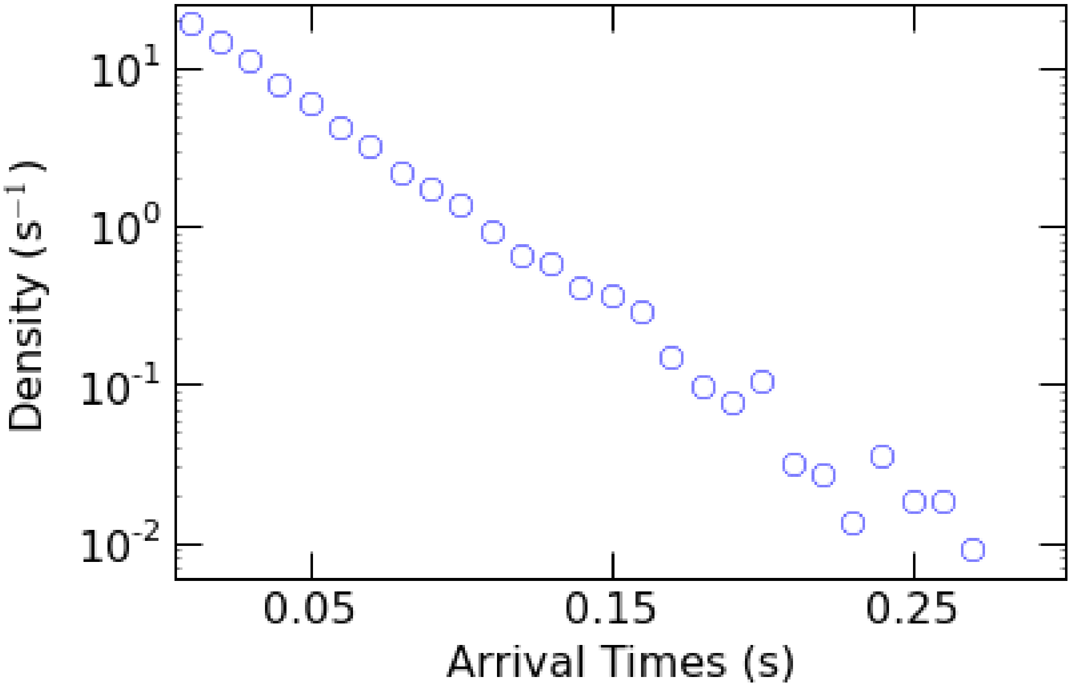
Plot the resulting PDF with `Matplotlib` to verify the distribution. Sure enough on a semilog scale, the resulting distribution appears linear suggesting an exponential form.

```
plt.semilogy(
        bins[:len(density)], density,
        linestyle='None',
        marker='o',
```

(continues on next page)

(continued from previous page)

```
        markersize=8,
        markeredgecolor='blue',
        markerfacecolor='None'
    )
plt.xlim(0.005,0.3)
plt.ylim(0.006,25)
plt.xticks([0.05,0.15,0.25])
plt.yticks([1e-2,0.1,1,1e1])
plt.axes().set_xlabel("Arrival Times (s)")
plt.axes().set_ylabel("Density (s$^{-1}$)")
plt.show()
```



Next we fit the PDF to a single exponential function of the form $a\ e^{-t/\tau}$, where a is a scaling factor and τ is the mean time of the distribution (with a rate of $1/\tau$). This is accomplished with the `curve_fit` function within `Scipy`.

```
popt, pcov = curve_fit(singleExponential, bins[:len(density)], density, p0=[1, np.
↪mean(arrival_times)])
```

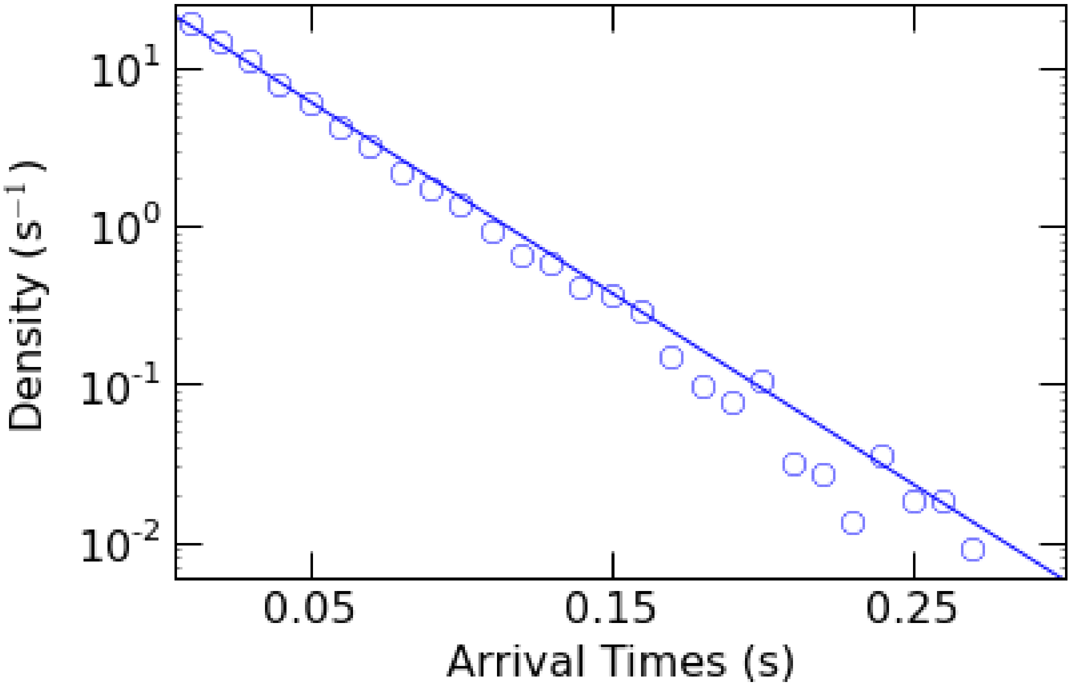
We then visually check the fit, by superimposing the resulting fit function over the PDF.

```
plt.semilogy(
        bins[:len(density)], density,
        linestyle='None',
        marker='o',
        markersize=8,
        markeredgecolor='blue',
        markerfacecolor='None'
    )
plt.semilogy(
            np.arange(0.001,0.4,0.02),
            singleExponential(np.arange(0.001,0.4,0.02), *popt),
```

(continues on next page)

(continued from previous page)

```
            color='blue'
        )
plt.xlim(0.005,0.3)
plt.ylim(0.006,25)
plt.xticks([0.05,0.15,0.25])
plt.yticks([1e-2,0.1,1,1e1])
plt.axes().set_xlabel("Arrival Times (s)")
plt.axes().set_ylabel("Density (s$^{-1}$)")
plt.show()
```



Finally, we can extract the capture rate ($1/\tau$) from the optimal fit parameters.

```
np.round([1/popt[1]], decimals=1 )
```

```
array([ 27.9])
```

CHAPTER 12

Addons

The output of *MOSAIC* is often processed further to generate plots or performe more sophisticated analysis. We facilitate this process by providing addon packages that make it easy to import the SQLite database generated by a *MOSAIC* analysis into mathematica-addons-sec, matlab-addons-sec or igor-addons-sec. The interfaces for these programs are described in more detail in this section.

12.1 Mathematica

12.1.1 Installation

The analysis output generated by *MOSAIC* can be imported into Mathematica for further processing. This accomplished with two packages: the low level mathematicaMosaicutilsSec and mathematica-mosaicanalysis-sec, which contains additional analysis routines. The addon package must first be installed to one of the locations in the Mathematica path. Alternatively, the required package files can be installed to the *Applications* folder using *setuptools* on Mac OS X and Linux by issuing the command below in the root folder of the *MOSAIC* code. Instructions for installing the package files for Windows are available here.

```
python setup.py mosaic_addons --mathematica
```

12.1.2 MosaicUtils

MosaicUtils provides low level functions to interact with a database output by *MOSAIC*. *MosaicUtils* can use a native Mathematica (**slower, default**) or Python (**faster, but experimental**) backend to query databases output by *MOSAIC*.

Note: To select the Python backend, please call the *SetQueryBackend* function as described below. If you use a virtual environment with Python, please call the *SetVirtualEnv* function after you install this addon.

SetQueryBackend[*backend*]

Args

- *backend* : select the *Mathematica* (default) or *Python* (faster, experimental) backend to run SQLite queries.

Returns None

ReadQueryBackend[]

Args None

Returns

- The *backend* used to run SQLite queries.

SetVirtualEnv[*virtualenv*]

Args

- *virtualenv* : name of the virtual environment configured for use with MOSAIC

Returns None

PrintMDKeys[*dbfile*]

Returns a list of column headings from the *metadata* table.

Args

- *dbfile* : full path to the database file

Returns A list of column names in the table *metadata*.

PrintMDTypes[*dbfile*]

Returns a list of column types from the *metadata* table.

Args

- *dbfile* : full path to the database file

Returns A list of column types in the table *metadata*.

QueryDB[*dbfile*, *query*]

Queries the *metadata* table using the supplied SQL query.

Args

- *dbfile* : full path to the database file
- *query* : a SQL query

Returns A nested list of query results.

PlotEvents[*dbfile*, *FsKHz*, *nEvents*]

Plot the event-time series if stored in the database (see the *Settings File* section for details on saving time-series to the analysis output).

Args

- *dbfile* : full path to the database file
- *FsKHz* : sampling frequency in kHz.
- *nEvents* : (optional) limit the plot to the first `n` entries in the database

Returns A dynamic object that allows the user to browse event time-series and fits.

GetAnalysisAlgorithm[*db*]

Returns the analysis algorithm used to process the current data set.

Args

- *db* : full path to a database file

Returns Algorithm used to analyze data.

MosaicUtils Examples

Once installed as described above, *MosaicUtils* must be imported as shown below.

```
In[1]= <<MosaicUtils`
```

SQL queries require the exact column names when querying data from a table (see *Database Structure and Query Syntax*). Column names in the *metadata* table, which stores the main results from the analaysis can be retrieved using the *PrintMDKeys* function as shown below. In this example, the column names returned correspond to an analysis performed using the *adept2State* algorithm.

```
In[2]= PrintMDKeys["<mosaicroot>/data/eventMD-PEG29-Reference.sqlite"]

Out[2]= {"recIDX", "ProcessingStatus", "OpenChCurrent",
        "BlockedCurrent", "EventStart", "EventEnd",
        "BlockDepth", "ResTime", "RCConstant", "AbsEventStart",
        "ReducedChiSquared", "TimeSeries"
        }
```

The *MosaicUtils* package allows the output of *MOSAIC* to be queried just like from Python. This accomplished using the *QueryDB* function. In the example below, we retrieve a column that returns the start time of the first 10 entries in the *metadata* table that have their *ProcessingStatus* set to *normal*. The results are then returned in a standard list. Note that *QueryDB* accepts a standard SQL query as described in more detail in the *Database Structure and Query Syntax* section.

```
In[3]= QueryDB[
        "<mosaicroot>/data/eventMD-PEG29-Reference.sqlite",
        "select AbsEventStart from metadata where ProcessingStatus='normal' limit 10"
        ]

Out[3]= {
        {1.84376}, {4.54439}, {5.26933}, {6.01253}, {6.80369},
        {8.48988}, {10.841}, {11.2246}, {13.2892}, {16.3983}
        }
```

Finally, the addon package allows us to plot individual events if time-series data was stored in the database. This is accomplished using the *PlotEvents* function, and provides a convenient tool to visually inspect the output of a *MOSAIC* analysis. In the example below, we inspect the events stored in the reference PEG28 data set included with *MOSAIC*. *PlotEvents* returns a dynamic object that allows the user to inspect all the events in a database. An event that was properly characterized by the code is plotted with *blue* markers (*left*). The plot is overlaid with the optimized fit function (*black*) and an idealized pulse (*red dashed*). Events that were not properly fit are plotted with *red* markers (*right*).

```
In[4]= PlotEvents["<mosaicroot>/data/eventMD-PEG29-Reference.sqlite", 500]
```

12.1.3 MosaicAnaly

MosaicAnalysis builds on the *MosaicUtils* package and provides basic analysis functions such as estimating the capture rate of molecules partitioning into a channel, or the mean residence time. Additionally, new functionality can be created by combining the functions defined below.

ScaledSingleExponentialFit[*hist*, *lambda*, *lambda0*]

Scale the histogram with the number of counts in the first bin. Fit a single exponential of the form *a exp(-t/tau)* to the scaled histogram.

Args

- *hist* : a histogram with format {{bin1, counts1}, {bin2, counts2}, . . . , {binN,countsN}}

- *lambda* : parameter of the distribution. This symbol must be passed from the calling function.
- *lambda0* : initial guess for *lambda*.

PlotScaledSingleExponentialFit[*hist*, *ftfunc*, *plotopts*]

Args

- *hist* : a histogram with format {{bin1, counts1}, {bin2,

counts2}, . . . , {binN,countsN}}

- *ftfunc* : an optimized fit, defined as a virtual function.
- *plotopts*: a list of options to control

the plot output.

CaptureRate[*arrtimes*, *stime*, *etime*, *nbins*, *plotopts*]

Estimate the capture rate of molecules by a channel by analyzing the arrival times of individual molecules. The arrival times of a stochastic process follow a single exponential distribution. This function first calculate a histogram of arrival times and then fits a single exponential function to the data.

Args

- *arrtimes* : a list of absolute start times (*AbsEventStart*) queried from a database.
- *stime* : lower limit of the arrival times distribution
- *etime* : upper limit of the arrival times distribution

- *nbins* : number of bins
- *plotopts* : a list of options to control the plot output.

Returns The mean capture rate, a plot of the underlying distribution of arrival times, the arrival times distribution and the optimized fit function.

ArrivalTimes[*abseventstart*]

Calculate the arrival times from a list of the absolute start time of each event in a data set.

Args

- *abseventstart* : a list of absolute start times (*AbsEventStart*) queried from a database.

Returns A list of arrival times.

Mosaic-Analysis Examples

```
In[1]=␣
↪
↪<
↪<MosaicUtils`
In[2]=␣
↪
↪<
↪<MosaicAnalysis`
```

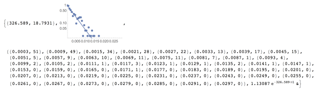
In the following example, we estimate the capture rate of PEG28 from the reference data set included with the *MOSAIC* source. The first argument fo *CaptureRate* is a list of the absolute start time of each event in the database. This data can be obtained using the query shown below. The remaining arguments to *CaptureRate* define the parameters of the arrival times distribution, the lower and upper limit of the arrival times and the number of bins. The function returns the mean capture rate and standard error, a plot that shows the underlying arrival times distribution, raw data used to generate the capture rate histogram, and a pure best-fit function.

```
In[3]=␣
↪CaptureRate[
␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪QueryDB[
␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪
↪"
↪<mosaicroot>
↪/
↪data/eventMD-PEG29-Reference.sqlite",
```

(continues on next page)

(continued from previous page)

```
␣
 ↪␣
 ↪␣
 ↪␣
 ↪␣
 ↪␣
 ↪␣
 ↪␣
 ↪
 ↪"select␣
 ↪AbsEventStart␣
 ↪from␣
↪metadata where
␣
 ↪␣
 ↪␣
 ↪␣
 ↪␣
 ↪␣
 ↪␣
 ↪␣
 ↪␣
 ↪␣
 ↪ ␣
 ↪   ␣
↪ProcessingStatus='normal' and ResTime > 0.01"
  ␣
 ↪   ␣
 ↪ ],␣
 ↪0.0,
 ↪ 0.
 ↪05,
 ↪ 50
]
```



The capture rate plot above can be formatted by supplying the optional *plotopts* argument, which uses standard Mathematica plot options, as seen in the example below. This is particulary helpful to customize the output of the plot,, for example for publication ready graphics.

```
        In[4]= CaptureRate[
                QueryDB[
                "<mosaicroot>/data/eventMD-PEG29-Reference.sqlite",
                "select AbsEventStart from metadata where
                        ProcessingStatus='normal' and ResTime > 0.01"
                ], 0.0, 0.05, 50,
                {Frame -> True, FrameLabel -> {Style["t (ms)", 16], ""},
FrameTicks -> {{{0.05, 0.1, 0.5, 1}, None}, {{0.005, 0.015, 0.025},
   None}}, FrameTicksStyle -> 14, PlotStyle -> {Black, Thick},
ImageSize -> 400}
        ]
```

12.2 Matlab

The SQLite database output by MOSAIC can be further processed using MATLAB. The data can then be stored in an array in the MATLAB Workspace, and then manipulated as desired.

The features, of opening, querying, and storing as an array, are made available in the MATLAB script openandquery.m. The script does not use the MATLAB Database Manager GUI, a part of the Database Toolbox, which requires a paid license. Instead, an open-source alternative, mksqlite, an interface between MATLAB and SQLite is used.

This section of the manual provides information on how to set up the mksqlite- package for use with MATLAB, and how to use the openandquery.m script.

All code has been successfully tested with MATLAB 2013a, MATLAB 2014a, G++ 4.7 in Ubuntu 14.04 LTS, and Windows Visual C++ 2010. Also, SQLite must be installed prior to performing the following steps.

12.2.1 mksqlite Documentation

Information about mksqlite, such as function calls and examples, is available in the MKSQLITE: A MATLAB Interface to SQLite documentation.

12.2.2 Installing mksqlite in Ubuntu 14.04 LTS

Download the latest mksqlite source files from SourceForge Unzip the files to a folder, and note the path to that folder (e.g., /home/mksqlitefolder) Open MATLAB, and change the current path to that of the mksqlite folder In the Command Window, type *buildit*, and press Enter to build mksqlite (this will run the buildit.m script). If the MEX files do not build, one of the following two problems may be why: i) a compiler may not be installed – see the MathWorks page on Supported and Compatible Compilers to select and install a compiler, or ii) errors are generated during compilation of mksqlite.cpp. In the latter case, see the "How to build mksqlite MEX file mksqlite.mexa64 in Linux?" thread in the MathWorks MATLAB Answers forum. If the build proceeds without errors, you will first see the notification "compiling release version of mksqlite…" in the Command Window, followed by "completed."

Note: GCC/G++ Version (in Linux)

You may have to install a version of GCC/G++ that is compatible with with your specific MATLAB release. If so, check out the linked discussion thread on MATLAB Central on how to set up a MEX Compiler.

12.2.3 Installing mksqlite in Windows 7

The installation steps are essentially the same as for Ubuntu, except a different compiler (e.g, contained in Windows SDK 7) may instead have to be installed. If the SDK installer say it cannot proceed, quit the installation, uninstall previous instances of Microsoft Visual C++ 2010, and then install Windows SDK 7 again.

12.2.4 Opening, Querying, and Closing the MOSAIC Output Database

The MATLAB script openandquery.m contains all of the commands to: Open a MOSAIC database (e.g., eventMD-PEG29-Reference.sqlite) Query the database Save queried data elements into a structure Close the database Convert the structure into a multi-dimensional array, that can be easily manipulated in MATLAB

Two changes must be made to the openandquery m-file by the end-user: The path to the database file must be changed for each database you wish to access. An example path in Linux would be /home/Data/eventMD-PEG29-Reference.sqlite, and in Windows C:\Data\eventMD-PEG29-Reference.sqlite. The query string can be changed as needed. More information about queries in available in the *Database Structure and Query Syntax* section.

12.2.5 Example

The reference database file provided with MOSAIC is *eventMD-PEG29-Reference.sqlite*, located in the data folder of the source code root directory. This database contains the results of an analysis performed using the *adept2State* and consists of the data fields:

```
{recIDX, ProcessingStatus, OpenChCurrent,
↪ BlockedCurrent, EventStart, EventEnd,
↪ BlockDepth, ResTime, RCConstant,
↪ AbsEventStart, ReducedChiSquared, and␣
↪TimeSeries}
```

In the openandquery script modify line 20 by typing in, within the quotes, the correct path to the database file.

```
dbname = '/home/
↪Data/eventMD-PEG29-Reference.sqlite';
```

The query in line 23 is to read the names of all fields in the database. The names, along with column ID, and data type, are stored in the structure fieldnames. You may double-click on the variable fieldnames in the Workspace, which will open the structure for you to read the field names in which you are interested.

```
fieldnames = mksqlite(
↪'PRAGMA table_info(metadata)');
```

Next, modify line 24 to include the query. In this example we want to select (and later manipulate) the data stored in the fields AbsEventStart and BlockDepth. This is where mksqlite comes in: the query are arguments to the mksqlite() function. For more information about using the mksqlite.m function check out the mksqlite documentation.

```
querytemp␣
↪= mksqlite('select AbsEventStart,
↪ BlockDepth from metadata');
```

No other changes are required. Run the script. The queried data are stored in the variable data, seen in the MATLAB Workspace (with value 418x2 double). This variable is a 2-column matrix. The first column contains all 418 data elements of the field AbsEventStart, and the second column contains all elements of the field BlockDepth. Note that the query above can be replaced with any standard SQL query as outlined in the *Work with SQLite* section.

12.3 IGOR

Data extraction in IGOR is a work in progress, but a number of users have found a successful route to querying the data and manipulating it in the IGOR environment. The installation and setup for these features requires an understanding of setup and use of ODBC drivers as well as rudimentary programming within the IGOR environment. To date, this has been tested on Mac OS X 10.9. Details may vary for other systems.

12.3.1 Activating SQL Database Access in IGOR

Database functionality in IGOR is preloaded, but not activated for use in the standard installation of SQL.xop. To activate this feature follow the instructions detailed in the “Igor Pro Folder/More Extensions/utilities/SQL Help.ihf”. The next few steps are reproduced from the IGOR instructions. First, activate the step in the activation process is open the folder, “Igor Pro Folder/More Extensions/utilities” and create an alias for SQL.xop. Then move the alias to “Igor Pro/Igor Extensions” or a similar folder that is in the search path for IGOR. It may be necessary to delete the “alias” text from the file name for functionality. Restart IGOR to activate.

IGOR relies on an external ODBC driver for database access. Depending on the operating system, it may be necessary to install a stand alone ODBC driver administrator package. First check your machine for the *ODBC administrator.app* in the *~/Applications/Utilities* folder. If not present ODBC administrator can be downloaded directly from the Apple support pages. To test the functionality, it is useful to follow the *Installing MySQL ODBC Driver…* instructions on the IGOR help page. The MySQL drivers are not necessary for functionally within *MOSAIC*.

With the ODBC administrator program installed, the next step is to install the SQLite driver for IGOR necessary to interface with the database. Once downloaded run the installation package in “sqlite3-odbc-0.93.dmg” and follow the setup instructions within the disk image. The driver should be ready to use within IGOR.

Hint: The IGOR addon installation (described above) can be activated automatically on *Mac OS X* by issuing the command `python setup.py mosaic_addons --igor` from the *MOSAIC* root directory. Note that administrator privileges are required.

12.3.2 Simple Database Query in IGOR

IGOR operates on databases with a single High Level operation command. This one command handles the database connection, query, export of data and closing of the database in one simple function or macro. To access this functionality, first open the procedure window and create the following function:

```
#include <SQLUtils>

Function QuerySQLData()

␣
↪String connectionStr= "DRIVER={SQLite3␣
↪Driver};DATABASE='database path';"
        String statement = "select␣
↪Blockdepth, ResTime from metadata␣
↪where ProcessingStatus ='normal'"

␣
↪    SQLHighLevelOp/CSTR={connectionStr,
↪ SQL_Driver_COMPLETE}/O/E=1 statement
End
```

Running this function will extract all normal events and create two waves containing the Blockade depth and Residence time of the events in sequence for further processing in IGOR. Two IGOR functions are included in the */addon/IGOR/* folder that import the data into IGOR waves for further processing. To open these functions to run, simply double click the file and the procedures will be opened in a new IGOR project. Once open, the procedure file can be compiled within IGOR to enable the code. A new menu “Mosaic” should then appear in the title bar within IGOR. A function “Fetch SQL data” will bring up a dialog box to manually enter a search string. After entering the string and clicking continue, you will be prompted to locate the database file you wish to access. The data will be imported into waves with the name automatically imported from the database. *Warning:* this will overwrite any existing data that is called by identical wave names.

CHAPTER 13

Developer Tools

We provide several tools to simplify developing and extending *MOSAIC* including debug information logging and function timing and profiling tools. Developer settings can be modified from `mosaic/_globals.py`. The master switch to turn on developer mode is the `DeveloperMode` attribute. **NOTE:** When `DeveloperMode=False`, all remaining attributes in `mosaic/_globals.py` are ignored.

```
# Control global settings
DeveloperMode=True              # Turn on developer options.

CodeProfiling='summary'         # Either 'summary' to print a summary at the end of a␣
↪run,
                                # 'none' for no timing, or
                                # 'all' to print timing of every function call␣
↪profiled.
LogProperties=False             # Log all class properties defined with mosaic_
↪property.
LogSizeBytes=int(2<<20)         # 2 MB
```

13.1 Debug Logs

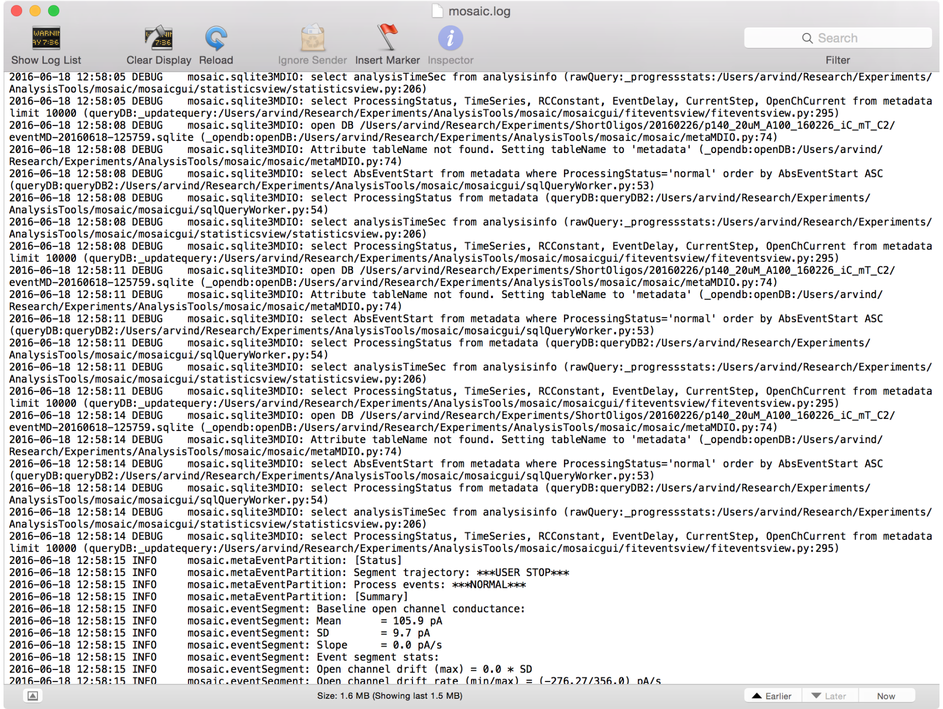
When `DeveloperMode` is active, logs are simultaneously saved the SQLite database, the console, and to a file using the Python logging module as described below. Details of the logging module can be found here. The logging facility provides different classes of messages with increasing severity, ranging from `debug` to help trace problems with code, `info` to provide feedback, `warn` to generate warnings, and `error` for error messages.

By default log messages in *MOSAIC* with a level of `info` and higher are saved to the console and to the SQLite database. On the other hand, `debug` messages are saved to a log file (see table below for log file locations). Since `debug` output can be verbose, the log file size is limited to the `LogSizeBytes` attribute in `mosaic/_globals.py`. The default value for this attribute is 2 MB. The log file is implemented with a rotating file structure, where only the previous five log files are saved to conserve disk space.

Platform	Log File Location
macOS	~/Library/Logs/MOSAIC/mosaic.log
linux	~/mosaic.log
linux (with sudo)	/var/logs/mosaic.log
Windows	<user home>/mosaic.log

Log messages can be added by first creating a logger instance and then logging a message as seen in the code sample below. For debug logs, the helper functions `_d()` or `_dprop()`, defined in the `mosaicLogFormat` module should be used. The helper functions append stack information to debug logs, allowing users to trace the calling function and its location in the source code.

```
import mosaic.utilities.mosaicLogging as mlog
from mosaic.utilities.mosaicLogFormat import _d


queryString="select ProcessingStatus, TimeSeries, RCConstant, EventDelay, CurrentStep,
↪ OpenChCurrent from metadata limit 10000"
logger=mlog.mosaicLogging().getLogger(name=__name__, dbHnd=self.mdioDBHnd)

logger.debug(_d("{0}", queryString))
```



```
2016-06-18 12:58:05 DEBUG    mosaic.sqlite3MDIO: select analysisTimeSec from analysisinfo (rawQuery:_progressstats:/Users/arvind/Research/Experiments/
AnalysisTools/mosaic/mosaicgui/statisticsview/statisticsview.py:206)
2016-06-18 12:58:05 DEBUG    mosaic.sqlite3MDIO: select ProcessingStatus, TimeSeries, RCConstant, EventDelay, CurrentStep, OpenChCurrent from metadata
limit 10000 (queryDB:_updatequery:/Users/arvind/Research/Experiments/AnalysisTools/mosaic/mosaicgui/fiteventsview/fiteventsview.py:295)
2016-06-18 12:58:08 DEBUG    mosaic.sqlite3MDIO: open DB /Users/arvind/Research/Experiments/ShortOligos/20160226/p140_20uM_A100_160226_iC_mT_C2/
eventMD-20160618-125759.sqlite (_opendb:openDB:/Users/arvind/Research/Experiments/AnalysisTools/mosaic/mosaic/metaMDIO.py:74)
2016-06-18 12:58:08 DEBUG    mosaic.sqlite3MDIO: Attribute tableName not found. Setting tableName to 'metadata' (_opendb:openDB:/Users/arvind/
Research/Experiments/AnalysisTools/mosaic/mosaic/metaMDIO.py:74)
2016-06-18 12:58:08 DEBUG    mosaic.sqlite3MDIO: select AbsEventStart from metadata where ProcessingStatus='normal' order by AbsEventStart ASC
(queryDB:queryDB2:/Users/arvind/Research/Experiments/AnalysisTools/mosaic/mosaicgui/sqlQueryWorker.py:53)
2016-06-18 12:58:08 DEBUG    mosaic.sqlite3MDIO: select ProcessingStatus from metadata (queryDB:queryDB2:/Users/arvind/Research/Experiments/
AnalysisTools/mosaic/mosaicgui/sqlQueryWorker.py:54)
2016-06-18 12:58:08 DEBUG    mosaic.sqlite3MDIO: select analysisTimeSec from analysisinfo (rawQuery:_progressstats:/Users/arvind/Research/Experiments/
AnalysisTools/mosaic/mosaicgui/statisticsview/statisticsview.py:206)
2016-06-18 12:58:08 DEBUG    mosaic.sqlite3MDIO: select ProcessingStatus, TimeSeries, RCConstant, EventDelay, CurrentStep, OpenChCurrent from metadata
limit 10000 (queryDB:_updatequery:/Users/arvind/Research/Experiments/AnalysisTools/mosaic/mosaicgui/fiteventsview/fiteventsview.py:295)
2016-06-18 12:58:11 DEBUG    mosaic.sqlite3MDIO: open DB /Users/arvind/Research/Experiments/ShortOligos/20160226/p140_20uM_A100_160226_iC_mT_C2/
eventMD-20160618-125759.sqlite (_opendb:openDB:/Users/arvind/Research/Experiments/AnalysisTools/mosaic/mosaic/metaMDIO.py:74)
2016-06-18 12:58:11 DEBUG    mosaic.sqlite3MDIO: Attribute tableName not found. Setting tableName to 'metadata' (_opendb:openDB:/Users/arvind/
Research/Experiments/AnalysisTools/mosaic/mosaic/metaMDIO.py:74)
2016-06-18 12:58:11 DEBUG    mosaic.sqlite3MDIO: select AbsEventStart from metadata where ProcessingStatus='normal' order by AbsEventStart ASC
(queryDB:queryDB2:/Users/arvind/Research/Experiments/AnalysisTools/mosaic/mosaicgui/sqlQueryWorker.py:53)
2016-06-18 12:58:11 DEBUG    mosaic.sqlite3MDIO: select ProcessingStatus from metadata (queryDB:queryDB2:/Users/arvind/Research/Experiments/
AnalysisTools/mosaic/mosaicgui/sqlQueryWorker.py:54)
2016-06-18 12:58:11 DEBUG    mosaic.sqlite3MDIO: select analysisTimeSec from analysisinfo (rawQuery:_progressstats:/Users/arvind/Research/Experiments/
AnalysisTools/mosaic/mosaicgui/statisticsview/statisticsview.py:206)
2016-06-18 12:58:11 DEBUG    mosaic.sqlite3MDIO: select ProcessingStatus, TimeSeries, RCConstant, EventDelay, CurrentStep, OpenChCurrent from metadata
limit 10000 (queryDB:_updatequery:/Users/arvind/Research/Experiments/AnalysisTools/mosaic/mosaicgui/fiteventsview/fiteventsview.py:295)
2016-06-18 12:58:14 DEBUG    mosaic.sqlite3MDIO: open DB /Users/arvind/Research/Experiments/ShortOligos/20160226/p140_20uM_A100_160226_iC_mT_C2/
eventMD-20160618-125759.sqlite (_opendb:openDB:/Users/arvind/Research/Experiments/AnalysisTools/mosaic/mosaic/metaMDIO.py:74)
2016-06-18 12:58:14 DEBUG    mosaic.sqlite3MDIO: Attribute tableName not found. Setting tableName to 'metadata' (_opendb:openDB:/Users/arvind/
Research/Experiments/AnalysisTools/mosaic/mosaic/metaMDIO.py:74)
2016-06-18 12:58:14 DEBUG    mosaic.sqlite3MDIO: select AbsEventStart from metadata where ProcessingStatus='normal' order by AbsEventStart ASC
(queryDB:queryDB2:/Users/arvind/Research/Experiments/AnalysisTools/mosaic/mosaicgui/sqlQueryWorker.py:53)
2016-06-18 12:58:14 DEBUG    mosaic.sqlite3MDIO: select ProcessingStatus from metadata (queryDB:queryDB2:/Users/arvind/Research/Experiments/
AnalysisTools/mosaic/mosaicgui/sqlQueryWorker.py:54)
2016-06-18 12:58:14 DEBUG    mosaic.sqlite3MDIO: select analysisTimeSec from analysisinfo (rawQuery:_progressstats:/Users/arvind/Research/Experiments/
AnalysisTools/mosaic/mosaicgui/statisticsview/statisticsview.py:206)
2016-06-18 12:58:14 DEBUG    mosaic.sqlite3MDIO: select ProcessingStatus, TimeSeries, RCConstant, EventDelay, CurrentStep, OpenChCurrent from metadata
limit 10000 (queryDB:_updatequery:/Users/arvind/Research/Experiments/AnalysisTools/mosaic/mosaicgui/fiteventsview/fiteventsview.py:295)
2016-06-18 12:58:15 INFO     mosaic.metaEventPartition: [Status]
2016-06-18 12:58:15 INFO     mosaic.metaEventPartition: Segment trajectory: ***USER STOP***
2016-06-18 12:58:15 INFO     mosaic.metaEventPartition: Process events: ***NORMAL***
2016-06-18 12:58:15 INFO     mosaic.metaEventPartition: [Summary]
2016-06-18 12:58:15 INFO     mosaic.eventSegment: Baseline open channel conductance:
2016-06-18 12:58:15 INFO     mosaic.eventSegment: Mean      = 105.9 pA
2016-06-18 12:58:15 INFO     mosaic.eventSegment: SD        = 9.7 pA
2016-06-18 12:58:15 INFO     mosaic.eventSegment: Slope     = 0.0 pA/s
2016-06-18 12:58:15 INFO     mosaic.eventSegment: Event segment stats:
2016-06-18 12:58:15 INFO     mosaic.eventSegment: Open channel drift (max) = 0.0 * SD
2016-06-18 12:58:15 INFO     mosaic.eventSegment: Open channel drift rate (min/max) = (-276.27/356.0) pA/s
```

The above code results in the following log message.

```
2016-06-18 12:58:11 DEBUG    mosaic.sqlite3MDIO:
        select ProcessingStatus, TimeSeries, RCConstant, EventDelay, CurrentStep,␣
↪OpenChCurrent from metadata limit 10000
        (queryDB:_updatequery:mosaic/mosaicgui/fiteventsview/fiteventsview.py:295)
```

```
2016-06-17 13:36:30 WARNING  mosaic.metaEventPartition:
        WARNING: Automatic open channel state estimation has been disabled.
```

```
2016-06-20 22:56:38 CRITICAL mosaic.utilities.mosaicLogging:
        Traceback (most recent call last):
                File "mosaic/utilities/mosaicTiming.py", line 180, in <module>
                        raise NotImplementedError("Feature not implemented.")
        NotImplementedError: Feature not implemented.
```

13.2 Function Timing and Profiling

```
import mosaic.utilities.mosaicTiming as mosaicTiming

partitionTimer=mosaicTiming.mosaicTiming()

class metaEventPartition(object):
        ... ... ...

        @metaEventPartition.partitionTimer.FunctionTiming
        def _processEvent(self, eventobj):
                ... ... ...

        def Stop(self):
                ... ... ...

                partitionTimer.PrintStatistics()

                ... ... ...
```

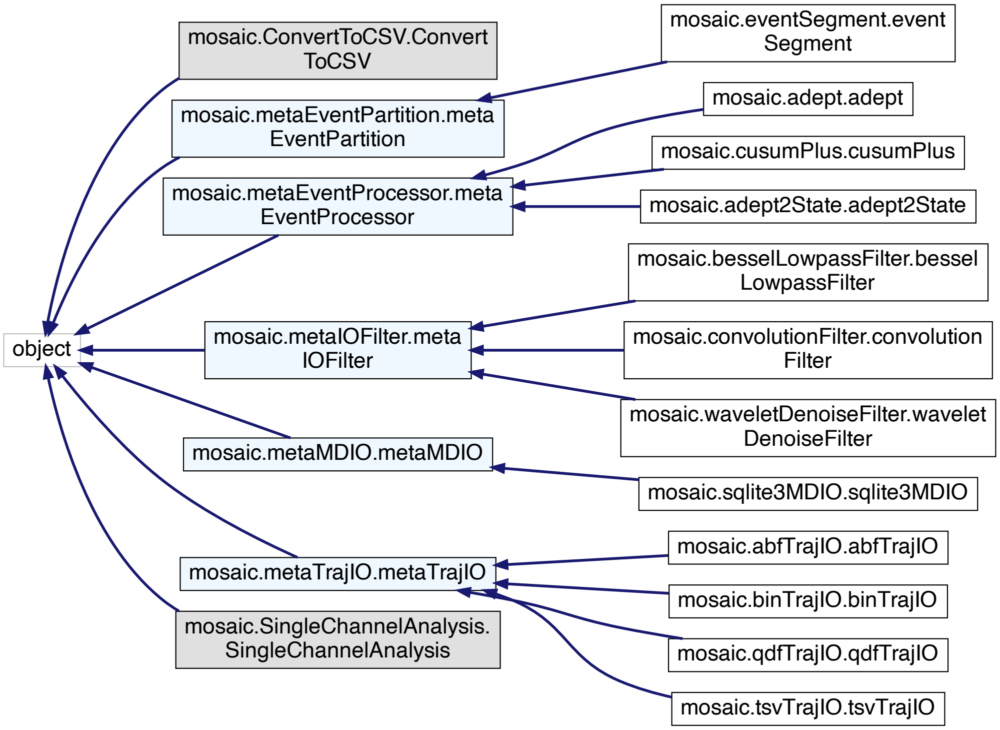
```
import mosaic.utilities.mosaicTiming as mosaicTiming

with mosaicTiming.mosaicTiming() as funcTimer:
        @funcTimer.FunctionTiming
        def someFunc():
                ... ... ...

                do something

                ... ... ...
```

```
2016-06-18 12:58:16 DEBUG    mosaic.utilities.mosaicTiming:
        Summary timing for "_processEvent": iterations=582, total=13475.599 ms,␣
↪maximum=296.641 ms, average=23.154 ms
        (PrintStatistics:Stop:mosaic/metaEventPartition.py:171)
```

CHAPTER 14

API Documentation

MOSAIC is designed using object oriented tools, which makes it easy to extend. The API documentation provides class level descriptions of the different modules that can be used in customized code. Meta-Classes (*blue*) define interfaces to five key parts of MOSAIC: time-series IO (`metaTrajIO`), time-series filtering (`metaIOFilter`), analysis output (`metaMDIO`), event partition and segmenting (`metaEventPartition`), and event processing (`metaEventProcessor`). Sub-classing any of these meta classes and implementing their interface functions allows one to extend *MOSAIC*, while maintaining compatibility with other parts of the program. The diagram below shows the class inheritence in *MOSAIC*, with top-level classes in *gray*.



14.1 *MOSAIC* Modules

14.1.1 Top-Level Interfaces

mosaic.apps.SingleChannelAnalysis module

Top level module to run a single channel analysis.

Created 05/15/2014

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

License See LICENSE.TXT

ChangeLog

3/25/17 AB Allow an optional argument to pass a database name.
5/15/14 AB Initial version

class `mosaic.apps.SingleChannelAnalysis.`**`SingleChannelAnalysis`**(*dataPath*, *trajDataHnd*, *dataFilterHnd*, *eventPartitionHnd*, *eventProcHnd*, ***kwargs*)

Bases: `object`

Run a single channel analysis. This is the entry point class for the analysis.

Parameters

- *dataPath* : full path to the data directory
- *trajDataHnd* : a handle to an implementation of `metaTrajIO`
- *dataFilterHnd* : a handle to an impementation of `metaIOFilter`
- *eventPartitionHnd* : a handle to a sub-class of `metaEventPartition`
- *eventProcHnd* : a handle to a sub-class of `metaEventProcessor`
- *dbFilename* : explicitly set the database name (optional kwarg).

`Run`(*forkProcess=False*)

Start an analysis.

Parameters

- *forkProcess* : start the analysis in a separate process if *True*. This option is useful when the main thread is used for other processing (e.g. GUI implementations).

`Stop`()

Stop a running analysis.

mosaic.apps.ConvertTrajIO module

Top level module to convert any data file readble by TrajIO objects into a comma separated value text file.

Created 10/13/2014

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

License See LICENSE.TXT

`ConvertTrajIO.`**`_filename`**()

Return a output filename that contains the data file prefix and and the block index.

`ConvertTrajIO.`**`_creategenerator`**()

Create a new filename generator if the file prefix has changed. The generator returns a filename incremented by a counter each time its next() function is called.

class `mosaic.apps.ConvertTrajIO.`**`ConvertTrajIO`**(*trajDataObj*, *outdir=None*, *extension='csv'*)

Bases: `object`

Convert data from a sub-class of metaTrajIO to either a delimited text file or binary file format.

Parameters

- *trajDataObj* : a trajIO data object
- *outdir* : the output directory. Default is *None*, which causes the output to be saved in the same directory as the input data.

- *extension* : 'csv' for comma separated values (default), 'tsv' for tab separated values, or 'bin' for 64-bit double precision binary.

Convert (*blockSize*)
Start converting data

Parameters

- *blockSize* : number of data points to convert.

14.1.2 Meta-Classes

mosaic.partition.metaEventPartition module

class mosaic.partition.metaEventPartition.**metaEventPartition** (*trajDataObj*, *eventProcHnd*, *eventPartitionSettings*, *eventProcSettings*, *settingsString*, ***kwargs*)

Bases: `object`

Warning: This metaclass must be sub-classed. All abstract methods within this metaclass must be implemented.

A class to abstract partitioning individual events. Once a single molecule event is identified, it is handed off to to an event processor. If parallel processing is requested, detailed event processing will commence immediately. If not, detailed event processing is performed after the event partition has completed.

Parameters

- ***trajDataObj*** [properly initialized object instantiated from a sub-class] of metaTrajIO.
- ***eventProcHnd*** [handle to a sub-class of metaEventProcessor. Objects of] this class are initialized as necessary
- *eventPartitionSettings* : settings dictionary for the partition algorithm.
- *eventProcSettings* : settings dictionary for the event processing algorithm.
- *settingsString* : settings dictionary in JSON format

Common algorithm parameters from settings file (.settings in the data path or current working directory)

- *writeEventTS* : Write event current data to file. (default: 1, write data to file)
- *parallelProc* : Process events in parallel using the pproc module. (default: 1, Yes)
- *reserveNCPU* : Reserve the specified number of CPUs and exclude them from the parallel pool.
- ***driftThreshold*** [Trigger a drift warning when the mean open channel current deviates by 'driftThreshold'*] SD from the baseline open channel current (default: 2)
- ***maxDriftRate*** [Trigger a warning when the open channel conductance changes at a rate faster] than that specified. (default: 2 pA/s)
- *minBaseline* : Minimum value for the ionic current baseline.
- *maxBaseline* : Maximum value for the ionic current baseline.

abstract _init(*trajDataObj*, *eventProcHnd*, *eventPartitionSettings*, *eventProcSettings*)

> **Important: Abstract method:** This method must be implemented by a sub-class.

This function is called at the end of the class constructor to perform additional initialization specific to the algorithm being implemented. The arguments to this function are identical to those passed to the class constructor.

abstract _stop()

> **Important: Abstract method:** This method must be implemented by a sub-class.

Stop partitioning events froma time-series

abstract _eventsegment()

> **Important: Abstract method:** This method must be implemented by a sub-class.

An implementation of this function should separate individual events of interest from a time-series of ionic current recordings. The data pertaining to each event is then passed to an instance of metaEventProcessor for detailed analysis. The function will collect the results of this analysis.

PartitionEvents()

Partition events within a time-series.

Stop()

Stop processing data.

abstract formatoutputfiles()

> **Important: Abstract method:** This method must be implemented by a sub-class.

Return a formatted string of output files.

abstract formatsettings()

> **Important: Abstract method:** This method must be implemented by a sub-class.

Return a formatted string of settings for display

abstract formatstats()

> **Important: Abstract method:** This method must be implemented by a sub-class.

Return a formatted string of statistics for display

mosaic.process.metaEventProcessor module

class `mosaic.process.metaEventProcessor.`**`metaEventProcessor`**(*icurr*, *icurrU*, *Fs*, ***kwargs*)

Bases: `object`

> **Warning:** This metaclass must be sub-classed. All abstract methods within this metaclass must be implemented.

Defines the interface for specific event processing algorithms. Each event processing algorithm must sub-class metaEventProcessor and implement the following abstract functions:

- ***processEvent*** [process raw event data and populate event meta-data. Store each] piece of processed event data in a class attribute starting with 'md'. For example, the blockade depth meta-data can be defined as 'mdBlockadeDepth'
- *printMetadata* : print meta-data set by event processing in a human readable format.

Parameters

- *icurr* : ionic current in pA
- *icurrU*: ionic current in pA with unfiltered event padding
- *Fs* : sampling frequency in Hz

Keyword Args

- *eventstart* : the event start point
- *eventend* : the event end point
- *baselinestats* : baseline conductance statistics: a list of [mean, sd, slope] for the baseline current
- *algosettingsdict* : settings for event processing algorithm as a dictionary
- *absdatidx* : index of data start. This arg can allow arrival time estimation.
- *datafilehnd* : reference to an metaMDIO object for meta-data IO

`__mdformat`(*dat*)

Round a float to 3 decimal places. Leave ints and strings unchanged

abstract `_init`(***kwargs*)

> **Important: Abstract method:** This method must be implemented by a sub-class.

abstract `_mdHeadingDataType`()

> **Important: Abstract method:** This method must be implemented by a sub-class.

Return a list of meta-data tags data types.

abstract _mdHeadings()

Important: Abstract method: This method must be implemented by a sub-class.

Return a list of meta-data tags for display purposes.

abstract _mdList()

Important: Abstract method: This method must be implemented by a sub-class.

Return a list of meta-data set by event processing.

abstract _processEvent()

Important: Abstract method: This method must be implemented by a sub-class.

---

flagEvent(*status*)
Set a warning status that starts with ‘w’ for non-critical errors. Metadata for these events is preserved and the user can be warned (e.g., in the GUI.)

abstract mdAveragePropertiesList()

Important: Abstract method: This method must be implemented by a sub-class.

Return a list of meta-data properties that will be averaged and displayed at the end of a run. This function must be overridden by sub-classes of metaEventProcessor. As a failsafe, an empty list is returned.

mdHeadingDataType()
Return a list of meta-data tags data types.

mdHeadings()
Return a list of meta-data tags for display purposes.

processEvent()
This is the equivalent of a pure virtual function in C++.

rejectEvent(*status*)
Set an event as rejected if it doesn’t pass tests in processing. The status is assigned to mdProcessingStatus.

writeEvent()
Write event meta data to a metaMDIO object.

mosaic.filters.metaIOFilter module

`class mosaic.filters.metaIOFilter.`**`metaIOFilter`**`(**kwargs)`

Bases: `object`

> **Warning:** This metaclass must be sub-classed. All abstract methods within this metaclass must be implemented.

Defines the interface for specific filter implementations. Each filtering algorithm must sub-class metaIOFilter and implement the following abstract function:

- *filterData* : apply a filter to self.eventData

Parameters

- *decimate* : sets the downsampling ratio of the filtered data (default:1, no decimation).

Properties

- *filteredData* : list of filtered and decimated data
- *filterFs* : sampling frequency after filtering and decimation

`abstract _init``(**kwargs)`

> **Important:** **Abstract method:** This method must be implemented by a sub-class.

`abstract filterData``(icurr, Fs)`

> **Important:** **Abstract method:** This method must be implemented by a sub-class.

This is the equivalent of a pure virtual function in C++.

Implementations of this method MUST store (1) a ref to the raw event data in self.eventData AND (2) the sampling frequency in self.Fs.

Parameters

- *icurr* : ionic current in pA
- *Fs* : original sampling frequency in Hz

`filterFs`

Return the sampling frequency of filtered data.

`filteredData`

Return filtered data

`abstract formatsettings``()`

> **Important:** **Abstract method:** This method must be implemented by a sub-class.

Return a formatted string of filter settings

mosaic.mdio.metaMDIO module

class `mosaic.mdio.metaMDIO.`**`metaMDIO`**

Bases: `object`

> **Warning:** This metaclass must be sub-classed. All abstract methods within this metaclass must be implemented.

This class provides the skeleton for storing metadata generated by algorithms. It also provides an interface to query metadata, for example in a SQL database.

Properties

- *dbColumnNames* : a list of database column names

abstract **`_opendb`**(*dbname*, ***kwargs*)

> **Important:** **Abstract method:** This method must be implemented by a sub-class.

abstract **`_initdb`**(***kwargs*)

> **Important:** **Abstract method:** This method must be implemented by a sub-class.

abstract **`_colnames`**(*table=None*)

> **Important:** **Abstract method:** This method must be implemented by a sub-class.

abstract **`closeDB`**()

> **Important:** **Abstract method:** This method must be implemented by a sub-class.

`initDB`(***kwargs*)

Initialize a new database file.

Parameters

The arguments passed to init change based on the method of file IO selected, in addition to the common args below:

- *dbPath* : directory to store the MD database ('<full path to data directory>')
- *colNames* : list of text names for the columns in the tables
- *colNames_t* : list of data types for each column.

`openDB`(*dbname*, ***kwargs*)

Open an existing database file.

Parameters

  - *dbname* : directory to store the MD database ('<full path to data directory>')

See also:

The arguments passed to init change based on the method of file IO selected, in addition to the common args.

`abstract queryDB(query)`

Important: Abstract method: This method must be implemented by a sub-class.

Query a database. :Parameters:

- *query* : query string

See also:

See specific implementations of metaMDIO for query syntax.

`abstract readAnalysisInfo()`

Important: Abstract method: This method must be implemented by a sub-class.

Read analysis information from the database.

`abstract readAnalysisLog()`

Important: Abstract method: This method must be implemented by a sub-class.

Read the analysis log from the database.

`abstract readSettings()`

Important: Abstract method: This method must be implemented by a sub-class.

Read JSON settings from the database.

`abstract writeAnalysisInfo(infolist)`

Important: Abstract method: This method must be implemented by a sub-class.

Write analysis information to the database. Note that subsequent calls to this method will overwrite the analysis inoformation entry in the table.

Args

- ***infolist*** [A list of strings in the following order [datPath, dataType, partitionAlgorithm, processingAlgorithm, filteringAlgorithm].] *datPath* : full path to the data directory

  dataType : type of data processed (e.g. ABF, QDF, etc.)

partitionAlgorithm : name of partition algorithm (e.g. eventSegment)

processingAlgorithm : name of event processing algorithm (e.g. multStateAnalysis)

filteringAlgorithm : name of filtering algorithm (e.g. waveletDenoiseFilter) or None if no filtering was performed.

abstract writeAnalysisLog(*analysislog*)

Important: Abstract method: This method must be implemented by a sub-class.

Write the analysis log string to the database. Note that subsequent calls to this method will overwrite the analysis log entry.

Args

- *analysislog* : analysis log string to save

abstract writeRecord(*data*, *table=None*)

Important: Abstract method: This method must be implemented by a sub-class.

Write data to a specified table. By default table is None. In this case sub-classes should fall back to writing data to a default table.

abstract writeSettings(*settingsstring*)

Important: Abstract method: This method must be implemented by a sub-class.

Write the settings JSON object to the database.

Args

- *settingsstring* : a **JSON_** formatted settings string.

mosaic.trajio.metaTrajIO module

class mosaic.trajio.metaTrajIO.**metaTrajIO**(***kwargs*)

Bases: `object`

Warning: This metaclass must be sub-classed. All abstract methods within this metaclass must be implemented.

Initialize a TrajIO object. The object can load all the data in a directory, N files from a directory or from an explicit list of filenames. In addition to the arguments defined below, implementations of this meta class may require the definition of additional arguments. See the documentation of those classes for what those may be. For example, the qdfTrajIO implementation of metaTrajIO also requires the feedback resistance (Rfb) and feedback capacitance (Cfb) to be passed at initialization.

Parameters

- *dirname* : all files from a directory ('<full path to data directory>')

- *nfiles* : if requesting N files (in addition to dirname) from a specified directory
- *fnames* : explicit list of filenames ([file1, file2,. . .]). This argument cannot be used in conjuction with dirname/nfiles. The filter argument is ignored when used in combination with fnames.
- *filter* : '<wildcard filter>' (optional, filter is '*' if not specified)
- *start* : Data start point in seconds.
- *end* : Data end point in seconds.
- *datafilter* : Handle to the algorithm to use to filter the data. If no algorithm is specified, datafilter is None and no filtering is performed.
- *dcOffset* : Subtract a DC offset from the ionic current data.
- *filtersettings*: Dict containing low pass filter settings (optional: if not provided filter settings will be loaded from the settings file. If no settings are found, *datafilter* will be turned off.)

Properties

- *FsHz* : sampling frequency in Hz. If the data was decimated, this property will hold the sampling frequency after decimation.
- *LastFileProcessed* : return the data file that was last processed.
- *ElapsedTimeSeconds* : return the analysis time in sec.

Errors

- *IncompatibleArgumentsError* : when conflicting arguments are used.
- *EmptyDataPipeError* : when out of data.
- *FileNotFoundError* : when data files do not exist in the specified path.
- *InsufficientArgumentsError* : when incompatible arguments are passed

abstract _init(***kwargs*)

Important: Abstract method: This method must be implemented by a sub-class.

This function is called at the end of the class constructor to perform additional initialization specific to the algorithm being implemented. The arguments to this function are identical to those passed to the class constructor.

abstract _formatsettings()

Log settings strings

DataLengthSec

Important: Property

Return the approximate length of data that will be processed. If the data are in multiple files, this property assumes that each file contains an equal amount of data.

ElapsedTimeSeconds

Important: Property

Return the elapsed time in the time-series in seconds.

`property FsHz`

Important: Property

Return the sampling frequency in Hz.

`LastFileProcessed`

Important: Property

Return the last data file that was processed

`ProcessedFiles`

Important: Property

Return a list of processed data filenames.

`formatsettings``()`

Return a formatted string of settings for display

`popfnames``()`

Pop a single filename from the start of `self.dataFiles`. If `self.dataFiles` is empty, raise an `EmptyDataPipeError` error.

Parameters

- None

Returns A single filename if successful.

Errors

- *EmptyDataPipeError* : when the filename list is empty.

`previewdata``(`*n*`)`

Preview data points in self.currDataPipe. This function is identical in behavior to popdata, except it does not remove data point from the queue. Like popdata, it uses recursion to automatically read data files when the queue length is shorter than the requested data points. When all data files are read, an EmptyDataPipeError is thrown.

Parameters *n* : number of requested data points

Returns

- Numpy array with requested data

Errors

- *EmptyDataPipeError* : if the queue has fewer data points than requested.

abstract readdata(*fname*)

Important: Abstract method: This method must be implemented by a sub-class.

Return raw data from a single data file. Set a class attribute Fs with the sampling frequency in Hz.

Parameters

- *fname* : fileame to read

Returns An array object that holds raw (unscaled) data from *fname*

Errors None

scaleData(*data*)

Important: Abstract method: This optional interface method can be overridden by a sub-class to modify functionality.

Scale the raw data loaded with `readdata()`. Note this function will not necessarily receive the entire data array loaded with `readdata()`. Transformations must be able to process partial data chunks.

Parameters

- *data* : partial chunk of raw data loaded using `readdata()`.

Returns

- Array containing scaled data.

Default Behavior

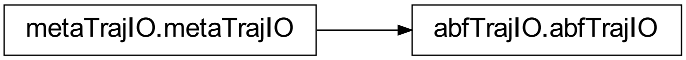
- If not implemented by a sub-class, the default behavior is to return `data` to the calling function without modifications.

Example Assuming the amplifier scale and offset values are stored in the class variables `AmplifierScale` and `AmplifierOffset`, the raw data read using `readdata()` can be transformed by `scaleData()`. We can also use this function to change the array data type.

```
def scaleData(self, data):
        return np.array(data*self.AmplifierScale-self.AmplifierOffset, dtype=
↪'f8')
```

14.1.3 Time-Series IO

mosaic.trajio.abfTrajIO module



A TrajIO class that supports ABF1 and ABF2 file formats via abf/abf.py. Currently, only gap-free mode and single channel recordings are supported.

Created 5/23/2013

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

License See LICENSE.TXT

ChangeLog

9/13/15 AB Updated logging to use mosaicLogFormat class
3/28/15 AB Updated file read code to match new metaTrajIO API.
5/23/13 AB Initial version

class `mosaic.trajio.abfTrajIO.`**`abfTrajIO`**`(`***kwargs*`)`

Bases: *`mosaic.trajio.metaTrajIO.metaTrajIO`*

Read ABF1 and ABF2 file formats. Currently, only gap-free mode and single channel recordings are supported.

A typical settings section to read ABF files is shown below.

```
"abfTrajIO" : {
"filter"                        : "*.abf",
"start"                         : 0.0,
"dcOffset"                      : 0.0
}
```

Parameters

In addition to `metaTrajIO` args, None

`readdata``(`*fname*`)`

Read one or more files and append their data to the data pipeline. Set a class attribute Fs with the sampling frequency in Hz.

Parameters

- *fname* : fileame to read

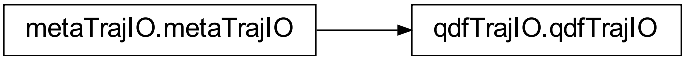
Returns

- An array object that holds raw (unscaled) data from *fname*

Errors

- *SamplingRateChangedError* : if the sampling rate for any data file differs from previous

mosaic.trajio.qdfTrajIO module



QDF implementation of metaTrajIO. Uses the readqdf module from EBS to read individual qdf files.

Created 7/18/2012

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

License See LICENSE.TXT

ChangeLog

9/13/15 AB Updated logging to use mosaicLogFormat class
3/28/15 AB Updated file read code to match new metaTrajIO API.
7/18/12 AB Initial version
2/11/14 AB Support qdf files that save the current in pA. This needs
format='pA' argument.

class `mosaic.trajio.qdfTrajIO.`**`qdfTrajIO`**`(**kwargs)`

Bases: *`mosaic.trajio.metaTrajIO.metaTrajIO`*

Use the readqdf module from EBS to read individual QDF files.

In addition to `metaTrajIO` args, check if the feedback resistance (`Rfb`) and feedback capacitance (`Cfb`) are defined to convert qdf binary data into pA.

A typical settings section to read QDF files is shown below. Note, that the values for `Rfb` and `Cfb` are specific to the amplifier used.

```
"qdfTrajIO": {
"Rfb"                           : 9.1e+12,
"Cfb"                           : 1.07e-12,
"dcOffset"                      : 0.0,
"filter"                        : "*.qdf",
"start"                         : 0.0
}
```

Parameters

In addition to metaTrajIO.__init__ args,

- *Rfb* : feedback resistance of amplifier
- *Cfb* : feedback capacitance of amplifier
- *format* : 'V' for voltage or 'pA' for current. Default is 'V'

Returns None

Errors

- *InsufficientArgumentsError* : if the mandatory arguments `Rfb` and `Cfb` are not set.

readdata (*fname*)

Read one or more files and append their data to the data pipeline. Set a class attribute Fs with the sampling frequency in Hz.

Parameters

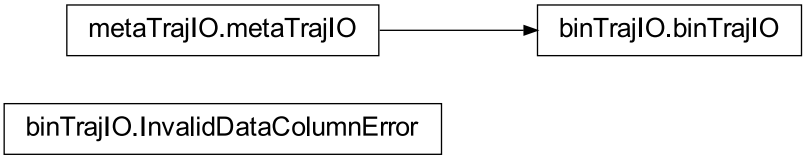
- *fname* : list of data files to read

Returns None

Errors

- *SamplingRateChangedError* : if the sampling rate for any data file differs from previous

mosaic.trajio.binTrajIO module



Binary file implementation of metaTrajIO. Read raw binary files with specified record sizes

Created 4/22/2013

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

License See LICENSE.TXT

ChangeLog

9/13/15 AB Updated logging to use mosaicLogFormat class
3/28/15 AB Updated file read code to match new metaTrajIO API.
1/27/15 AB Memory map files on read.
1/26/15 AB Refactored code to read interleaved binary data.
7/27/14 AB Update interface to specify python PythonStructCode instead of
RecordSize. This will allow any binary file to be decoded
The AmplifierScale and AmplifierOffset are set to 1 and 0
respectively if PythonStructCode is an integer or short.
4/22/13 AB Initial version

exception `mosaic.trajio.binTrajIO.`**`InvalidDataColumnError`**

Bases: `Exception`

class `mosaic.trajio.binTrajIO.`**`binTrajIO`**`(`***kwargs*`)`

Bases: *`mosaic.trajio.metaTrajIO.metaTrajIO`*

Read a file that contains interleaved binary data, ordered by column. Only a single column that holds ionic current data is read. The current in pA is returned after scaling by the amplifier scale factor (`AmplifierScale`) and removing any offsets (`AmplifierOffset`) if provided.

Usage and Assumptions Binary data is interleaved by column. For three columns (*a*, *b*, and *c*) and *N* rows, binary data is assumed to be of the form:

[a_1, b_1, c_1, a_2, b_2, c_2, … … …, a_N, b_N, c_N]

The column layout is specified with the `ColumnTypes` parameter, which accepts a list of tuples. For the example above, if column **a** is the ionic current in a 64-bit floating point format, column **b** is the ionic current representation in 16-bit integer format and column **c** is an index in 16-bit integer format, the `ColumnTypes` paramter is a list with three tuples, one for each column, as shown below:

[('curr_pA', 'float64'), ('AD_V', 'int16'), ('index', 'int16')]

The first element of each tuple is an arbitrary text label and the second element is a valid Numpy type.

Finally, the `IonicCurrentColumn` parameter holds the name (text label defined above) of the column that holds the ionic current time-series. Note that if an integer column is selected, the `AmplifierScale` and `AmplifierOffset` parameters can be used to convert the voltage from the A/D to a current.

Assuming that we use a floating point representation of the ionic current, and a sampling rate of 50 kHz, a settings section that will read the binary file format defined above is:

```
"binTrajIO": {
        "AmplifierScale" : "1",
        "AmplifierOffset" : "0",
        "SamplingFrequency" : "50000",
        "ColumnTypes" : "[('curr_pA', 'float64'), ('AD_V', 'int16'), (
↪'index', 'int16')]",
        "IonicCurrentColumn" : "curr_pA",
        "dcOffset": "0.0",
        "filter": "*.bin",
        "start": "0.0",
        "HeaderOffset": 0
}
```

Settings Examples Read 16-bit signed integers (big endian) with a 512 byte header offset. Set the amplifier scale to 400 pA, sampling rate to 200 kHz.

```
"binTrajIO": {
        "AmplifierOffset": "0.0",
        "SamplingFrequency": 200000,
        "AmplifierScale": "400./2**16",
        "ColumnTypes": "[('curr_pA', '>i2')]",
        "dcOffset": 0.0,
        "filter": "*.dat",
        "start": 0.0,
        "HeaderOffset": 512,
        "IonicCurrentColumn": "curr_pA"
}
```

Read a two-column file: 64-bit floating point and 64-bit integers, and no header offset. Set the amplifier scale to 1 and sampling rate to 200 kHz.

```
"binTrajIO": {
        "AmplifierOffset": "0.0",
        "SamplingFrequency": 200000,
        "AmplifierScale": "1.0",
        "ColumnTypes" : "[('curr_pA', 'float64'), ('AD_V',
↪'int64')]",
        "dcOffset": 0.0,
        "filter": "*.bin",
        "start": 0.0,
        "HeaderOffset": 0,
        "IonicCurrentColumn": "curr_pA"
}
```

Parameters

In addition to `metaTrajIO` args,

- *AmplifierScale* : Full scale of amplifier (pA/2^nbits) that varies with the gain (default: 1.0).
- *AmplifierOffset* : Current offset in the recorded data in pA (default: 0.0).
- *SamplingFrequency* : Sampling rate of data in the file in Hz.
- *HeaderOffset* : Ignore first *n* bytes of the file for header (default: 0 bytes).
- *ColumnTypes* : A list of tuples with column names and types (see Numpy types). Note only integer and floating point numbers are supported.
- *IonicCurrentColumn* : Column name that holds ionic current data.

Returns None

Errors None

`readdata` (*fname*)

Return raw data from a single data file. Set a class attribute Fs with the sampling frequency in Hz.

Parameters

- *fname* : fileame to read

Returns

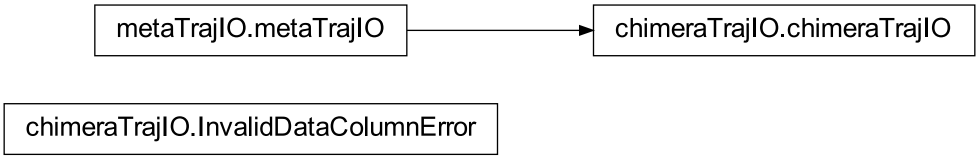
- An array object that holds raw (unscaled) data from *fname*

Errors None

`scaleData` (*data*)

See `mosaic.metaTrajIO.metaTrajIO.scaleData()`.

mosaic.trajio.chimeraTrajIO module



Chimera VC100 concatenated file format implementation of metaTrajIO. Read concatenated chimera files with specified amplifier settings.

Created 7/11/2016

Author Kyle Briggs <kbrig035@uottawa.ca>

License See LICENSE.TXT

ChangeLog

5/12/21 AB Update Chimera settings from MAT files generated during data collection.

7/29/16 KB Miscelleneous bugfixes

7/11/16 KB Initial version

exception `mosaic.trajio.chimeraTrajIO.`**`InvalidDataColumnError`**

Bases: `Exception`

class `mosaic.trajio.chimeraTrajIO.`**`chimeraTrajIO`**`(`*`**kwargs`*`)`

Bases: *`mosaic.trajio.metaTrajIO.metaTrajIO`*

Read a file generated by the Chimera VC100. The current in pA is returned after scaling by the amplifier scale factors.

Usage and Assumptions Binary data is in a single column of unsigned 16 bit integers:

The column layout is specified with the `ColumnTypes` parameter, which accepts a list of tuples.

[('curr_pA', '<u2')]

The option is left in in case of future changes to the platform, but can be left alone in the settings file for now. The first element of each tuple is an arbitrary text label and the second element is a valid Numpy type.

Chimera gain settings are used to convert the integers stored by the ADC to current values. These values are automatically read in from matched MAT files generated by the Chimera software.

```
"chimeraTrajIO": {
        "filter": "*.log",
        "start": "0.0",
        "HeaderOffset": "0"
}
```

Parameters In addition to `metaTrajIO` args,

- *HeaderOffset* : Ignore first *n* bytes of the file for header (currently fixed at: 0 bytes).

Returns None

Errors None

`readdata` (*fname*)

Return raw data from a single data file. Set a class attribute Fs with the sampling frequency in Hz.

Parameters

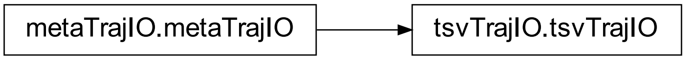
- *fname* : fileame to read

Returns

- An array object that holds raw (unscaled) data from *fname*

Errors None

mosaic.trajio.tsvTrajIO module



An implementation of metaTrajIO that reads tab separated valued (TSV) files

Created 7/31/2012

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

License See LICENSE.TXT

ChangeLog

11/30/15 AB Assumes `timeCol` is specified in seconds.
11/30/15 AB Added a new keyword `scale` to allow scaling TSV data.
3/28/15 AB Updated file read code to match new metaTrajIO API.
6/30/13 AB Added the ‘seprator’ kwarg to the class initializer to allow any delimited
files to be read. e.g. ‘”"t’ (default), ‘,’, etc.
7/31/12 AB Initial version

class `mosaic.trajio.tsvTrajIO.`**`tsvTrajIO`**(***kwargs*)

Bases: *`mosaic.trajio.metaTrajIO.metaTrajIO`*

Read tab separated valued (TSV) files.

Parameters

In addition to `metaTrajIO` args,

- *headers* : If True, the first row is ignored (default: True)

- *separator* : set the data separator (defualt: ‘”"t’)
- *scale* : set the data scale (default: 1). For example to convert from to pA set `scale=1e12`.

Either:

- *Fs* : Sampling frequency in Hz. If set, all other options are ignored and the first column in the file is assumed to be the current in pA.

Or:

- *nCols* : number of columns in TSV file (default:2, first column is time in ms and second is current in pA)
- *timeCol* : explicitly set the time column (default: 0, first col)
- *currCol* : explicitly set the position of the current column (default: 1)

If neither `Fs` nor {`nCols`, `timeCol`, `currCol`} are set then the latter is assumed with the listed default values.

`readdata` (*fname*)

Read a single TSV file and return raw (unscaled) data contained within it. Set/update a class attribute Fs with the sampling frequency in Hz.

Parameters

- *fname* : fileame to read

Returns

- An array object that holds raw (unscaled) data from *fname*

Errors

- *SamplingRateChangedError* : if the sampling rate for any data file differs from previous

`scaleData` (*data*)

Important: Abstract method: This optional interface method can be overridden by a sub-class to modify functionality.

Scale the raw data loaded with `readdata()`. Note this function will not necessarily receive the entire data array loaded with `readdata()`. Transformations must be able to process partial data chunks.

Parameters

- *data* : partial chunk of raw data loaded using `readdata()`.

Returns

- Array containing scaled data.

Default Behavior

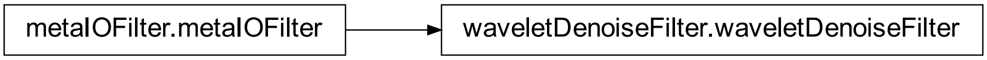
- If not implemented by a sub-class, the default behavior is to return `data` to the calling function without modifications.

Example Assuming the amplifier scale and offset values are stored in the class variables `AmplifierScale` and `AmplifierOffset`, the raw data read using `readdata()` can be transformed by `scaleData()`. We can also use this function to change the array data type.

```
def scaleData(self, data):
        return np.array(data*self.AmplifierScale-self.AmplifierOffset, dtype=
↪'f8')
```

14.1.4 Time-Series Filters

mosaic.filters.waveletDenoiseFilter module



Implementation of a wavelet based denoising filter

Created 8/31/2014

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

License See LICENSE.TXT

Author Arvind Balijepalli

ChangeLog

9/13/15 AB Updated logging to use mosaicLogFormat class
8/31/14 AB Initial version

`class mosaic.filters.waveletDenoiseFilter.`**`waveletDenoiseFilter`**`(`*`**kwargs`*`)`

Bases: *`mosaic.filters.metaIOFilter.metaIOFilter`*

Keyword Args

In addition to metaIOFilter args,

- *wavelet* : the type of wavelet
- *level* : wavelet level
- *threshold* : threshold type

`filterData``(`*`icurr, Fs`*`)`

Denoise an ionic current time-series and store it in self.eventData

Parameters

- *icurr* : ionic current in pA
- *Fs* : original sampling frequency in Hz

`formatsettings``()`

Return a formatted string of filter settings

mosaic.filters.besselLowpassFilter module

Implementation of an 'N' order Bessel filter

Created 7/1/2013

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

License See LICENSE.TXT

ChangeLog

11/2/16 KB changed Bessel filter implementation to match expected rise time
9/27/16 AB Control phase delay
9/13/15 AB Updated logging to use mosaicLogFormat class
7/1/13 AB Initial version

class `mosaic.filters.besselLowpassFilter.`**`besselLowpassFilter`**`(`*`**kwargs`*`)`

Bases: *`mosaic.filters.metaIOFilter.metaIOFilter`*

Keyword Args

In addition to metaIOFilter.__init__ args,

- *filterOrder* : the filter order
- *filterCutoff* : filter cutoff frequency in Hz

`filterData``(`*icurr*, *Fs*`)`

Denoise an ionic current time-series and store it in self.eventData

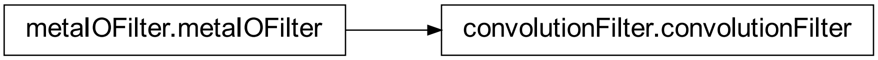
Parameters

- *icurr* : ionic current in pA
- *Fs* : original sampling frequency in Hz

`formatsettings``()`

Populate *logObject* with settings strings for display

mosaic.filters.convolutionFilter module



Implementation of a weighted moving average (tap delay line) filter

Created 8/16/2013

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

License See LICENSE.TXT

ChangeLog

9/13/15 AB Updated logging to use mosaicLogFormat class
8/16/13 AB Initial version

class `mosaic.filters.convolutionFilter.`**`convolutionFilter`**(***kwargs*)

Bases: `mosaic.filters.metaIOFilter.metaIOFilter`

Keyword Args

In addition to metaIOFilter.__init__ args,

- *filterCoeff* : filter coefficients (default is a 10 point uniform moving average)

`filterData`(*icurr*, *Fs*)

Denoise an ionic current time-series and store it in self.eventData

Parameters

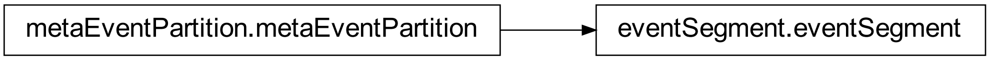
- *icurr* : ionic current in pA
- *Fs* : original sampling frequency in Hz

`formatsettings`()

Return a formatted string of filter settings

14.1.5 Event Partition and Segment

mosaic.partition.eventSegment module



Partition a trajectory into individual events and pass each event to an implementation of eventProcessor

Created 7/17/2012

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

License See LICENSE.TXT

ChangeLog

10/28/18 AB Fix spurious shallow events as outlined in https://github.com/usnistgov/mosaic/issues/102
9/25/17 AB Save unfiltered event padding by default.
1/18/17 AB Fix pre event baseline.
6/17/16 AB Log function timing in developer mode.
5/17/14 AB Delete plotting support
5/17/14 AB Add metaMDIO support for meta-data and time-series storage
2/14/14 AB Pass absdatidx argument to event processing to track absolute time of
    event start for capture rate estimation.
6/22/13 AB Use plotting hooks in metaEventPartition to plot blockade depth histogram in
    real-time using matplotlib.
4/22/13 AB Rewrote this class as an implementation of the base class metaEventPartition.
    Included event processing parallelization using ZMQ.
9/26/12 AB Allowed automatic open channel state calculation to be overridden.
    To do this the settings “meanOpenCurr”,”sdOpenCurr” and “slopeOpenCurr”
    must be set manually. If all three settings are absent or
    set to 01, they are autuomatically estimated.
    Added “writeEventTS” boolean setting to control whether raw
    events are written to file. Default is ON (1)
8/24/12 AB Settings are now read from a settings file that
    is located either with the data or in the working directory
    that the program is run from. Each class that relies on the
    settings file will fallback to default values if the file
    is not found.
7/17/12 AB Initial version

class `mosaic.partition.eventSegment.`**`eventSegment`**(*trajDataObj*, *eventProcHnd*, *eventPartitionSettings*, *eventProcSettings*, *settingsString*, ***kwargs*)

Bases: `mosaic.partition.metaEventPartition.metaEventPartition`

Implement an event partitioning algorithm by sub-classing the metaEventPartition class

Settings In addition to the parameters described in `metaEventPartition`, the following parameters from are read from the settings file (.settings in the data path or current working directory):

- ***blockSizeSec*** [Functions that perform block processing use this value to set the size of] their windows in seconds. For example, open channel conductance is processed for windows with a size specified by this parameter. (default: 1 second)
- *eventPad* : Number of points to include before and after a detected event. (default: 500)
- *minEventLength* : Minimum number points in the blocked state to qualify as an event (default: 5)
- ***eventThreshold*** [Threshold, number of SD away from the open channel mean. If the abs(curr) is less] than 'abs(mean)-(eventThreshold*SD)' a new event is registered (default: 6)
- ***meanOpenCurr*** [Explicitly set mean open channel current. (pA) (default: -1, to] calculate automatically)
- ***sdOpenCurr*** [Explicitly set open channel current SD. (pA) (default: -1, to] calculate automatically)
- ***slopeOpenCurr*** [Explicitly set open channel current slope. (default: -1, to] calculate automatically)

`formatoutputfiles``()`

Important: **Abstract method:** This method must be implemented by a sub-class.

Return a formatted string of output files.

`formatsettings``()`

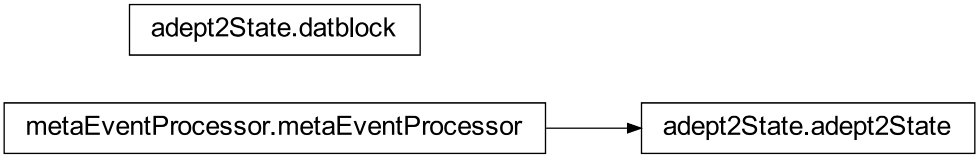
Return a formatted string of settings for display in the output log.

`formatstats``()`

Return a formatted string of statistics for display in the output log.

14.1.6 Event Processing

mosaic.process.adept2State module



A

class that extends metaEventProcessing to implement the step response algorithm from [Balijepalli:2014]

Created 4/18/2013

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

License See LICENSE.TXT

ChangeLog

06/28/16 AB Upgrade lmfit to > 0.9 (https://lmfit.github.io/lmfit-py/whatsnew.html#whatsnew-090-label)
03/30/16 AB Change UnlinkRCConst to LinkRCConst to avoid double negatives.
12/09/15 KB Added Windows specific optimizations
8/24/15 AB Rename algorithm to ADEPT 2 State.
7/23/15 JF Added a new test to reject RC Constants <=0
6/24/15 AB Added an option to unlink the RC constants in stepResponseAnalysis.
11/7/14 AB Error codes describing event rejection are now more specific.
11/5/14 AB Fixed a bug in the event fitting logic that prevented
    long events from being correctly analyzed.
5/17/14 AB Modified md interface functions for metaMDIO support
2/16/14 AB Added new metadata field, 'AbsEventStart' to track
    global time of event start to allow capture rate estimation.
6/20/13 AB Added an additional check to reject events
    with blockade depths > BlockRejectRatio (default: 0.8)
4/18/13 AB Initial version

class `mosaic.process.adept2State.`**`adept2State`**(*icurr*, *icurrU*, *Fs*, ***kwargs*)

Bases: *`mosaic.process.metaEventProcessor.metaEventProcessor`*

Analyze an event that is characteristic of PEG blockades. This method includes system information in the analysis, specifically the filtering effects (throught the RC constant) of either amplifiers or the membrane/nanopore complex. The analysis generates several parameters that are stored as metadata including:

1. Blockade depth: the ratio of the open channel current to the blocked current
2. Residence time: the time the molecule spends inside the pore
3. Tau: the RC constant of the response to a step input (e.g. the entry or exit of the molecule into or out of the nanopore).

Keyword Args

In addition to `metaEventProcessor` args,

- *FitTol* : Tolerance value for the least squares algorithm that controls the convergence of the fit (Default: *1e-7*).
- *FitIters* : Maximum number of iterations before terminating the fit (Default: *50000*).
- *LinkRCConst* : When True, the RC constants associated with each state in the fit function are varied together. (Default: *True*)

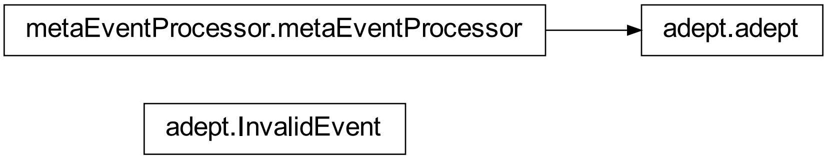
Errors When an event cannot be analyzed, the blockade depth, residence time and rise time are set to -1.

`formatsettings`()

Return a formatted string of settings for display

`mdAveragePropertiesList()`
Return a list of meta-data properties that will be averaged and displayed at the end of a run.

mosaic.process.adept module



Analyze a multi-step event

Created 4/18/2013

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

License See LICENSE.TXT

ChangeLog

06/28/16 AB Upgrade lmfit to > 0.9 (https://lmfit.github.io/lmfit-py/whatsnew.html#whatsnew-090-label)
05/27/16 AB Added warnings when the reduced chi squared is not a number and if the fit parameters are unchanged from the initial guess values.
05/22/16 JF Added new test to reject BD < 0 or BD > 1, improved readability of error tests.
03/30/16 AB Change UnlinkRCConst to LinkRCConst to avoid double negatives.
3/16/16 AB Migrate InitThreshold setting to CUSUM StepSize.
2/22/16 AB Use CUSUM to estimate intial guesses in ADEPT for long events.
2/20/16 AB Format settings log.
12/09/15 KB Added Windows specific optimizations
8/24/15 AB Rename algorithm to ADEPT.
8/02/15 JF Added a new test to reject RC Constants <=0
4/12/15 AB Refactored code to improve reusability.
3/20/15 AB Added a maximum event length setting (MaxEventLength) that automatically rejects events longer than the specified value.
3/20/15 AB Added a new metadata column (mdStateResTime) that saves the residence time of each state to the database.
3/6/15 AB Added a new test for negative event delays
3/6/15 JF Added MinStateLength to output log
3/5/15 AB Updated initial state determination to include a minumum state length parameter (MinStateLength).
    Initial state estimates now utilize gradient information for improved state identification.
1/7/15 AB Save the number of states in an event to the DB using the mdNStates column
12/31/14 AB Changed multi-state function to include a separate tau for
    each state following Balijepalli et al, ACS Nano 2014.
12/30/14 JF Removed min/max constraint on tau

11/7/14 AB Error codes describing event rejection are now more specific.
11/6/14 AB Fixed a bug in the event fitting logic that prevents the
analysis of long states.
8/21/14 AB Added AbsEventStart and BlockDepth (constructed from mdCurrentStep
and mdOpenChCurrent) metadata.
5/17/14 AB Modified md interface functions for metaMDIO support
9/26/13 AB Initial version

exception `mosaic.process.adept.`**`InvalidEvent`**

Bases: `Exception`

class `mosaic.process.adept.`**`adept`**(*icurr*, *icurrU*, *Fs*, ***kwargs*)

Bases: *`mosaic.process.metaEventProcessor.metaEventProcessor`*

Analyze a multi-step event that contains two or more states. This method includes system information in the analysis, specifically the filtering effects (through the RC constant) of either amplifiers or the membrane/nanopore complex. The analysis generates several parameters that are stored as metadata including:

1. Blockade depth: the ratio of the open channel current to the blocked current
2. Residence time: the time the molecule spends inside the pore
3. Tau: the RC constant of the response to a step input (e.g. the entry or exit of the molecule into or out of the nanopore).

Keyword Args

In addition to `metaEventProcessor` args,

- *StepSize* : The multiple of the standard deviations considered significant to dtecting an event (default: 3.0).
- *MinStateLength* : minimum number of data points required to assign a state within an event (default: 4)
- *MaxEventLength* : maximum length (in data points) of events that will be processed (default: 10000)
- *FitTol* : fit tolerance for convergence (default: 1.e-7)
- *FitIters* : maximum fit iterations (default: 5000)
- *LinkRCConst* : When True, the RC constants associated with each state in the fit function are varied together. (Default: *True*)

Errors When an event cannot be analyzed, all metadata are set to -1.

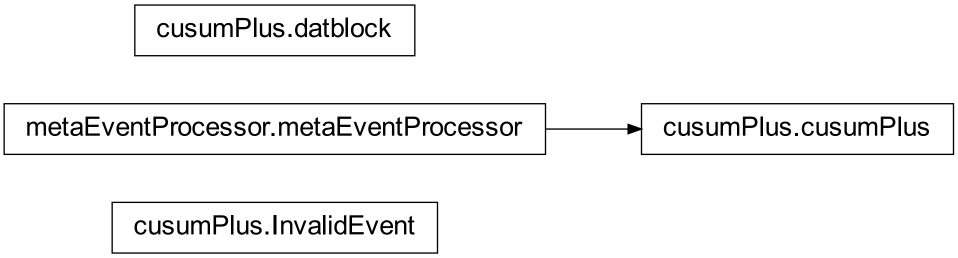
`formatsettings`()

Return a formatted string of settings for display

`mdAveragePropertiesList`()

Return a list of meta-data properties that will be averaged and displayed at the end of a run.

mosaic.process.cusumPlus module



Analyze a multi-step event with the CUSUM+ algorithm. Implements a modified version of the CUSUM algorithm (used by OpenNanopore for example) in MOSAIC. This approach sacrifices including system information in the analysis in favor of much faster fitting of single- and multi-level events.

Created 2/10/2015

Author Kyle Briggs <kbrig035@uottawa.ca>

License See LICENSE.TXT

ChangeLog

11/15/19 JR Updated output: calculates adn includes blockade standard deviation; simplified tab in code

6/3/17 AB Updated docstring.

8/24/15 AB Rename algorithm to CUSUM+

3/20/15 AB Added a new metadata column (mdStateResTime) that saves the residence time of each state to the database.

3/18/15 KB Implemented rise time skipping

3/17/15 KB Implemented adaptive threshold

2/12/15 AB Updated metadata representation to be consistent with stepResponseAnalysis and multiStateAnalysis

2/10/15 KB Initial version

exception `mosaic.process.cusumPlus.`**`InvalidEvent`**

Bases: `Exception`

class `mosaic.process.cusumPlus.`**`cusumPlus`**(*icurr*, *icurrU*, *Fs*, ***kwargs*)

Bases: *`mosaic.process.metaEventProcessor.metaEventProcessor`*

CUSUM+ will detect jumps that are smaller than *StepSize*, but they will have to be sustained longer. Threshold can be thought of, very roughly, as proportional to the length of time a subevent must be sustained for it to be detected. The algorithm will adjust the actual threshold used on a per-event basis in order to minimize false positive detection of current jumps This algorithm is based on code used in OpenNanopore, which you can read about here: http://pubs.rsc.org/en/Content/ArticleLanding/2012/NR/c2nr30951c#!divAbstract

Some known issues with CUSUM+:

1. If the duration of a sub-event is shorter than than the MinLength parameter, CUSUM+ will be unable to detect it. CUSUM+ will not detect events within MinLength of a previous event.

2. CUSUM assumes an instantaneous transition between current states. As a result, if the RC rise time of the system is large, CUSUM+ can trigger and detect intermediate states during the change time. This can be avoided by choosing a number of samples to skip equal to about 2-5RC.
3. As a consequence of using a statistical t-test, CUSUM can have false positives. The algorithm has an adaptive threshold that tries to minimize the chances of this happening while maintaining good sensitivity (expected number of false positives within an event is less than 1).

Keyword Args

In addition to `metaEventProcessor` args,

- *StepSize* : The number of baseline standard deviations are considered significant (3 is usually a good starting point).
- *MinThreshold* : One of two sensitivity parameters (lower is more sensitive). A good starting point is to set *MinThreshold* equal to *StepSize*.
- *MaxThreshold* : One of two sensitivity parameters (lower is more sensitive). Set *MaxThreshold* about 3x higher than *MinThreshold*.
- *MinLength* : The number of samples to skip after detecting a jump, in order to avoid triggering during the rise time and returning an artificially high number of states. This number of points is also skipped when averaging levels. About 4 times the RC constant of the system is a good starting value.

Errors When an event cannot be analyzed, all metadata are set to -1.

To use it requires four settings:

```
"cusumPlus": {
        "StepSize": 3.0,
        "MinThreshold": 3.0,
        "MaxThreshold": 10.0,
        "MinLength" : 10,
}
```

`formatsettings`()

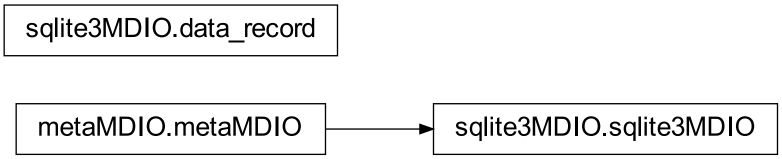
Return a formatted string of settings for display

`mdAveragePropertiesList`()

Return a list of meta-data properties that will be averaged and displayed at the end of a run.

14.1.7 Data Output

mosaic.mdio.sqlite3MDIO module



A class that extends metaMDIO to implement SQLite support for metadata storage.

Created 9/28/2014

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

License See LICENSE.TXT

ChangeLog

3/25/17 AB Allow an optional argument to pass a database name.
12/6/15 AB Add sampling frequency to analysis info table
8/5/15 AB Added a function to export database tables to CSV
8/5/15 AB Misc bug fixes
4/1/15 AB Added an estimate of data length to the DB
3/23/15 AB Added a raw query function that does not automatically decode column data.
11/9/14 AB Implemented the analysis log I/O interface for sqlite3 databases.
9/28/14 AB Initial version

class `mosaic.mdio.sqlite3MDIO.`**`data_record`**(*data_label*, *data*, *data_t*)

Bases: `dict`

Smart data record structure that automatically encodes/decodes data for storage in a sqlite3 DB.

`update`([*E*], ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

class `mosaic.mdio.sqlite3MDIO.`**`sqlite3MDIO`**

Bases: *`mosaic.mdio.metaMDIO.metaMDIO`*

`closeDB`()

Important: Abstract method: This method must be implemented by a sub-class.

`csvString`(*query*)

Return database records that match the specified query as a CSV formatted string.

`exportToCSV`(*query*)

Export database records that match the specified query to a CSV flat file.

`queryDB`(*query*)

Important: Abstract method: This method must be implemented by a sub-class.

Query a database. :Parameters:

- *query* : query string

See also:

See specific implementations of metaMDIO for query syntax.

`readAnalysisInfo`()

Important: Abstract method: This method must be implemented by a sub-class.

Read analysis information from the database.

`readAnalysisLog`()

Important: Abstract method: This method must be implemented by a sub-class.

Read the analysis log from the database.

`readSettings`()

Important: Abstract method: This method must be implemented by a sub-class.

Read JSON settings from the database.

`writeAnalysisInfo`(*infolist*)

Important: Abstract method: This method must be implemented by a sub-class.

Write analysis information to the database. Note that subsequent calls to this method will overwrite the analysis inoformation entry in the table.

Args

- ***infolist*** [A list of strings in the following order [datPath, dataType, partitionAlgorithm, processingAlgorithm, filteringAlgorithm].] *datPath* : full path to the data directory

  dataType : type of data processed (e.g. ABF, QDF, etc.)

  partitionAlgorithm : name of partition algorithm (e.g. eventSegment)

  processingAlgorithm : name of event processing algorithm (e.g. multStateAnalysis)

  filteringAlgorithm : name of filtering algorithm (e.g. waveletDenoiseFilter) or None if no filtering was performed.

writeAnalysisLog(*analysislog*)

Important: Abstract method: This method must be implemented by a sub-class.

Write the analysis log string to the database. Note that subsequent calls to this method will overwrite the analysis log entry.

Args

- *analysislog* : analysis log string to save

writeRecord(*data*, *table=None*)

Important: Abstract method: This method must be implemented by a sub-class.

Write data to a specified table. By default table is None. In this case sub-classes should fall back to writing data to a default table.

writeSettings(*settingsstring*)

Important: Abstract method: This method must be implemented by a sub-class.

Write the settings JSON object to the database.

Args

- *settingsstring* : a **JSON_** formatted settings string.

14.1.8 Miscellaneous

mosaic.settings module

Load analysis settings from a JSON file.

Created 8/24/2012

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

License See LICENSE.TXT

ChangeLog

5/13/21 AB Include chimeraTrajio in default settings.

4/19/19 AB Added an option (trackBaseline) to continuously track the opent channel baseline current during an analysis.

9/22/17 AB Removed the filterEventPadding option to eventSegment.

7/15/17 AB Add the filterEventPadding option to eventSegment.

3/16/16 AB Replaced InitThreshold with StepSize in default settings for ADEPT and warn users when InitThreshold is used.

8/24/15 AB Updated algorithm names.

6/24/15 AB Added an option to unlink the RC constants in stepResponseAnalysis.
3/20/15 AB Added MaxEventLength to multiStateAnalysis settings
3/6/15 JF Corrected formatting on cusumLevelAnalysis and multiStateAnalysis dictionary file
3/6/15 AB Added MinStateLength parameter for multiStateAnalysis to dictionary
2/14/15 AB Added default settings for cusumLevelAnalysis.
8/20/14 AB Changed precedence of settings file search to datpath/.settings,
    datpath/settings, coderoot/.settings and coderoot/settings
8/6/14 AB Add a function to parse a settings string.
9/5/13 AB Check for either .settings or settings in data directory
    and code root. Warn when using default settings
8/24/12 AB Initial version

class `mosaic.settings.`**`settings`**(*datpath*, *defaultwarn=True*)

Bases: `object`

Initialize a settings object.

Args

- *datpath* : Specify the location of the settings file. If a settings file is not found, return default settings.
- *defaultwarn* : If *True* warn the user if a settings file was not found in the path specified by *datpath*.

`getSettings`(*section*)

Return settings for a specified section as a Python dict.

Args

- *section* : specifies the section for which settings are requested. Returns an empty dictionary if the settings file doesn't exist the section is not found.

mosaic.utilities.ionic_current_stats module

Created 10/30/2014

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

License See LICENSE.TXT

ChangeLog

5/24/19 AB Python 3.7 port
7/29/16 KB Added weights to histogram fitting
15/12/15 KB Added error checking and limits to baseline calculations
10/30/14 AB Initial version

`mosaic.utilities.ionic_current_stats.`**`OpenCurrentDist`**(*dat*, *limit*, *minBaseline=- 1*, *maxBaseline=- 1*)

Calculate the mean and standard deviation of a time-series.

Args

- *dat* : time-series data

- *limit* : limit the calculation to the top 50% (+0.5) of the range, bottom 50% (-0.5) or the entire range (0). Any other value of *limit* will cause it to be reset to 0 (i.e. full range).

mosaic.utilities.util module

A collection of utility functions

`mosaic.utilities.util.`**`avg`**`(`*dat*`)`
Calculate the average of a list of reals

`mosaic.utilities.util.`**`commonest`**`(`*dat*`)`
Return the most common element in a list.

`mosaic.utilities.util.`**`decimate`**`(`*dat*, *size*`)`
Decimate dat for a specified window size.

`mosaic.utilities.util.`**`filter`**`(`*dat*, *windowSz*`)`
Filter the data using a convolution. Returns an array of size len(dat)-windowSz+1 if dat is longer than windowSz. If len(dat) < windowSz, raise WindowSizeError

`mosaic.utilities.util.`**`flat2`**`(`*dat*`)`
Flatten a 2D array to a list

`mosaic.utilities.util.`**`partition`**`(`*dat*, *size*`)`
Partition a list into sub-lists, each of length size. If the number of elements in dat does not partition evenly, the last sub-list will have fewer elements.

`mosaic.utilities.util.`**`sd`**`(`*dat*`)`
Wrapper for numpy std

`mosaic.utilities.util.`**`selectS`**`(`*dat*, *nSigma*, *mu*, *sd*`)`
Select and return data from a list that lie within nSigma * SD of the mean.

mosaic.utilities.mosaicLogFormat module

An object that allows arbitrary formatting of log text.

Created 09/12/2015

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

License See LICENSE.TXT

ChangeLog

06/14/16 AB Added a new Python property class mimic to add log property set, get and del.

06/13/16 AB Remove mosaicLogFormat class

09/12/15 AB Initial version

`mosaic.utilities.mosaicLogFormat.`**`_d`**`(`*msg*, **args*`)`
Format a debug log message. This function will automatically append calling function name and file/line number.

Parameters

- *msg* : Log message formatted using the Python formatter class.
- *args* : Message arguments.

Usage Log an integer or a float variable.

```
_d("Var x has value {0}", x)

_d("Var y is a float with value {0:0.2f} to 2 decimal places.", y)
```

`mosaic.utilities.mosaicLogFormat.`**`_dprop`**(*msg*, **args*)

Format a debug log message for a class property. This function will automatically append calling function name and file/line number.

Parameters

- *msg* : Log message formatted using the Python formatter class.
- *args* : Message arguments.

Usage Log a property that returns an integer or a float.

```
_dprop("Var x has value {0}", x)

_dprop("Var y is a float with value {0:0.2f} to 2 decimal places.", y)
```

`class` `mosaic.utilities.mosaicLogFormat.`**`mosaic_property`**(*fget=None*, *fset=None*, *fdel=None*, *doc=None*)

Bases: `object`

Emulate Python property. Add support to the getter and setter methods to automatically log properties in debug mode. The new class can be used exactly as the built-in Python property class, for example as a decorator

```
class foo:
        def __init__(self):
                self.x=100

        @mosaic_property
        def x(self):
                return self.x

        @x.setter
        def x(self, val):
                self.x=val
```

Adapted from: https://docs.python.org/2/howto/descriptor.html#properties.

mosaic.utilities.mosaicLogging module

An implementation of Python logging heavily adapted from http://stackoverflow.com/questions/15727420/using-python-logging-in-multiple-modules.

Created 5/29/2016

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

License See LICENSE.TXT

ChangeLog

5/30/16 AB Strip whitespace from rotating file handler messages.
5/29/16 AB Initial version

class `mosaic.utilities.mosaicLogging.`**`MessageFormatter`**(*fmt=None*, *datefmt=None*, *style='%'*)

Bases: `logging.Formatter`

`format`(*record*)

Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

class `mosaic.utilities.mosaicLogging.`**`mosaicLogging`**(**args*, ***kwargs*)

Bases: `object`

A custom logging class that uses the Python logging facility. Logs are automatically saved to a metaMDIO instance, and to a file log when DeveloperMode is active.

static **`getLogger`**(*name=None*, *dbHnd=None*)

Get a logger instance.

Parameters

- *name* : Logger name
- *dbHnd* : MetaMDIO handle to allow logs to be saved to the database.

Usage In this example, we get an instance of a logger with the module name and log a debug message.

```
logger=mosaicLogging().getLogger(__name__)

logger.debug("Test debug message")
```

mosaic.utilities.mosaicTiming module

A class that provides platform independent timing and function profiling utilities.

Created 4/10/2016

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

License See LICENSE.TXT

ChangeLog

4/14/21 AB Windows fixes
6/17/16 AB Only profile functions in DeveloperMode. Log timing output.
4/10/16 AB Initial version

class `mosaic.utilities.mosaicTiming.`**`mosaicTiming`**

Bases: `object`

Profile code by attaching an instance of this class to any function. All the methods in this class are valid for the function being profiled.

`FunctionTiming` (*func*)

Pass the function to be profiled as an argument. Alternatively with python 2.4+, attach a decorator to the function being profiled.

Parameters

- *func* : function to be profiled

Usage

```
funcTimer=mosaicTiming.mosaicTiming()

@funcTimer.FunctionTiming
def someFunc():
                print 'doing something'

# summarize the profiling results for someFunc
funcTimer.PrintStatistics()
```

`PrintCurrentTime` ()

Print timing results of the most recent function call

`PrintStatistics` ()

Print average timing results of the function call

`Reset` (*funcname=None*)

Reset all profiling data collected for a specified function or all stored functions.

`time` ()

A platform independent timing function.

mosaic.utilities.fit_funcs module

Fit functions used in processing algorithms.

Created 10/30/2014

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

License See LICENSE.TXT

ChangeLog

12/09/15 KB Added a wrapper for multiStateFunc
6/24/15 AB Relaxed stepResponseFunc to include different RC constants
    for up and down states.
12/31/14 AB Changed multi-state function to include a separate tau for
    each state following Balijepalli et al, ACS Nano 2014.
11/19/14 AB Initial version

14.1.9 MOSAIC Script Repository

mosaicscripts.plots.timeseries module

Plot an ionic current time-series.

Created 11/19/2015

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

ChangeLog

12/12/15 AB Generalized plot function to allow different data types
11/19/15 AB Initial version

`mosaicscripts.plots.timeseries.`**`PlotTimeseries`**(*dir*, *data_type*, *t0*, *t1*, *Fs*, ***kwargs*)

Generate publication quality time-series plots.

Args

- *dir* : directory containing data files
- *data_type* : One of "abf", "qdf", "bin" or "tsv".
- *t0* : start time.
- *t1* : end time.
- *Fs* : Sampling rate in Hz.
- *labels* : Axes text labels. For example `` `["t (s)", "-i (pA)"]` `` for a current vs. time plot.

Keyword Args

- *data_args* : (optional) For "qdf", "bin" or "tsv", settings to read in data. See *Settings File* for details.
- *axes* : (optional) Show axes (Default: True)
- ***highlights*** [(optional) Highlight segments of the time-series with a different style (Default: None). For example:]

  highlights=[[[0.282, 0.293], {'color' : '#3F50A0', 'marker' : '.', 'markersize' : 0.1}], [[0.584, 0.597], {'color' : '#D42324', 'marker' : '.', 'markersize' : 0.1}], [[0.685, 0.695], {'color' : '#EB751A', 'marker' : '.', 'markersize' : 0.1}]

  ]

Highlight three events at specied location (arg 1: start, end) with specified styles. - *plotopts* : (optional) Specify plot style. See http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot for details. - *figname* : (optional) figure name if saving an image. File extension determines format. - *dpi* : (optional) figure resolution

mosaicscripts.plots.histogram module

1-D Histogram plot.

Created 12/13/2015

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

License See LICENSE.TXT

ChangeLog

12/13/15 AB Initial version

`mosaicscripts.plots.histogram.`**`histogram_plot`**(*dat*, *nbins*, *x_range*, ***kwargs*)

Generate publication quality contour plots using the `` `contour_plot` `` function. The function expects a two-dimensional array of data (typically blockade depth and residence time) and several options as listed below:

Args

- *dat* : 2-D array with format [[x1,y1], [x2,y2], … … … [xn,yn]]
- *nbinx* : number of bins.
- *x_range* : list with min and max in X. If *None*, min and max values of the data set the range.

Keyword Args

- *density* : (optional) If True, display the probability density function. Default is *False*
- *color* : (optional) Plot color. Default is *#4155A3*.
- *fill_alpha* : (optional) Fill transparency. 0 turns off fill. Default is 0.25.
- *xticks* : (optional) specify ticks for the X-axis. List of format [(tick, label), …]
- *yticks* : (optional) specify ticks for the X-axis. List of format [(tick, label), …]
- *figname* : (optional) figure name if saving an image. File extension determines format.
- *dpi* : (optional) figure resolution.
- *show* : (optional) if True (default) call the show() function to display the plot.
- *return_histogram* : (optional) if True, return the histogram values and bins. Default is False.
- *advanced_opts* : (optional) a Python dictionary that supplies advanced plotting options. See `Matplotlib plot documentation

<http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot>`_ for details.

mosaicscripts.plots.contour module

Contour plot overlaid on top of an image.

Created 11/11/2015

Author Arvind Balijepalli <arvind.balijepalli@nist.gov>

License See LICENSE.TXT

ChangeLog

02/05/16 AB Add options to scale z-axis
01/10/15 AB Rename custom colormaps
11/11/15 AB Initial version

`mosaicscripts.plots.contour.`**`contour_plot`** (*dat2d*, *x_range*, *y_range*, *bin_size*, *contours*, *colormap*, *img_interpolation*, ***kwargs*)

Generate publication quality contour plots using the `` `contour_plot` `` function. The function expects a two-dimensional array of data (typically blockade depth and residence time) and several options as listed below:

Args

- *dat2d* : 2-D array with format [[x1,y1], [x2,y2], … … … [xn,yn]]
- *x_range* : list with min and max in X
- *y_range* : list with min and max in Y
- *bin_size* : bin size
- *contours* : number of contours
- *colormap* : Colormap to use. Expects a colormap object. See http://matplotlib.org/examples/color/colormaps_reference.html.
- *img_interpolation* : interpolation to use for image

Keyword Args

- *zscale* : (optional) plot the probability density if set to *density* or scale to the max count if set to *unity*.
- *xticks* : (optional) specify ticks for the X-axis. List of format [(tick, label), …]
- *yticks* : (optional) specify ticks for the X-axis. List of format [(tick, label), …]
- *figname* : (optional) figure name if saving an image. File extension determines format.
- *dpi* : (optional) figure resolution
- *colorbar_num_ticks* : (optional) number of ticks in the colorbar
- *cb_round_digits* : (optional) round colorbar ticks to multiple of cb_round_digits. For example, -2 rounds to 100. See python docs.
- *min_count_pct* : (optional) set bins with < min_count_pct of the maximum to 0
- *axes_type* : (optional) set linear or log axis. Expects a list for X and Y. For example [‘linear’, ‘log’].

Python Module Index

m

`mosaic.apps.ConvertTrajIO`, 101
`mosaic.apps.SingleChannelAnalysis`, 100
`mosaic.filters.besselLowpassFilter`, 122
`mosaic.filters.convolutionFilter`, 123
`mosaic.filters.waveletDenoiseFilter`, 121
`mosaic.mdio.sqlite3MDIO`, 131
`mosaic.partition.eventSegment`, 124
`mosaic.process.adept`, 127
`mosaic.process.adept2State`, 125
`mosaic.process.cusumPlus`, 129
`mosaic.settings`, 133
`mosaic.trajio.abfTrajIO`, 113
`mosaic.trajio.binTrajIO`, 115
`mosaic.trajio.chimeraTrajIO`, 118
`mosaic.trajio.qdfTrajIO`, 114
`mosaic.trajio.tsvTrajIO`, 119
`mosaic.utilities.fit_funcs`, 138
`mosaic.utilities.ionic_current_stats`, 134
`mosaic.utilities.mosaicLogFormat`, 135
`mosaic.utilities.mosaicLogging`, 136
`mosaic.utilities.mosaicTiming`, 137
`mosaic.utilities.util`, 135
`mosaicscripts.plots.contour`, 141
`mosaicscripts.plots.histogram`, 140
`mosaicscripts.plots.timeseries`, 139

Index

Symbols

__mdformat() (*mosaic.process.metaEventProcessor.metaEventProcessor method*), 104

_colnames() (*mosaic.mdio.metaMDIO.metaMDIO method*), 107

_creategenerator() (*mosaic.apps.ConvertTrajIO.ConvertTrajIO method*), 101

_d() (*in module mosaic.utilities.mosaicLogFormat*), 135

_dprop() (*in module mosaic.utilities.mosaicLogFormat*), 136

_eventsegment() (*mosaic.partition.metaEventPartition.metaEventPartition method*), 103

_filename() (*mosaic.apps.ConvertTrajIO.ConvertTrajIO method*), 101

_formatsettings() (*mosaic.trajio.metaTrajIO.metaTrajIO method*), 110

_init() (*mosaic.filters.metaIOFilter.metaIOFilter method*), 106

_init() (*mosaic.partition.metaEventPartition.metaEventPartition method*), 102

_init() (*mosaic.process.metaEventProcessor.metaEventProcessor method*), 104

_init() (*mosaic.trajio.metaTrajIO.metaTrajIO method*), 110

_initdb() (*mosaic.mdio.metaMDIO.metaMDIO method*), 107

_mdHeadingDataType() (*mosaic.process.metaEventProcessor.metaEventProcessor method*), 104

_mdHeadings() (*mosaic.process.metaEventProcessor.metaEventProcessor method*), 104

_mdList() (*mosaic.process.metaEventProcessor.metaEventProcessor method*), 105

_opendb() (*mosaic.mdio.metaMDIO.metaMDIO method*), 107

_processEvent() (*mosaic.process.metaEventProcessor.metaEventProcessor method*), 105

_stop() (*mosaic.partition.metaEventPartition.metaEventPartition method*), 103

A

abfTrajIO (*class in mosaic.trajio.abfTrajIO*), 37, 113

adept (*class in mosaic.process.adept*), 128

adept2State (*class in mosaic.process.adept2State*), 126

avg() (*in module mosaic.utilities.util*), 135

B

besselLowpassFilter (*class in mosaic.filters.besselLowpassFilter*), 122

binTrajIO (*class in mosaic.trajio.binTrajIO*), 38, 115

C

chimeraTrajIO (*class in mosaic.trajio.chimeraTrajIO*), 39, 118

closeDB() (*mosaic.mdio.metaMDIO.metaMDIO method*), 107

closeDB() (*mosaic.mdio.sqlite3MDIO.sqlite3MDIO method*), 131

commonest() (*in module mosaic.utilities.util*), 135

contour_plot() (*in module mosaic-scripts.plots.contour*), 141

Convert() (*mosaic.apps.ConvertTrajIO.ConvertTrajIO method*), 102

ConvertTrajIO (*class in mosaic.apps.ConvertTrajIO*), 101

convolutionFilter (*class in mosaic.filters.convolutionFilter*), 123

csvString() (*mosaic.mdio.sqlite3MDIO.sqlite3MDIO method*), 131

cusumPlus (*class in mosaic.process.cusumPlus*), 129

D

data_record (*class in mosaic.mdio.sqlite3MDIO*), 131

DataLengthSec (*mosaic.trajio.metaTrajIO.metaTrajIO attribute*), 110

decimate() (*in module mosaic.utilities.util*), 135

E

ElapsedTimeSeconds (*mosaic.trajio.metaTrajIO.metaTrajIO attribute*), 110

eventSegment (*class in mosaic.partition.eventSegment*), 124

exportToCSV() (*mosaic.mdio.sqlite3MDIO.sqlite3MDIO method*), 131

F

filter() (*in module mosaic.utilities.util*), 135

filterData() (*mosaic.filters.besselLowpassFilter.besselLowpassFilter method*), 122

filterData() (*mosaic.filters.convolutionFilter.convolutionFilter method*), 123

filterData() (*mosaic.filters.metaIOFilter.metaIOFilter method*), 106

filterData() (*mosaic.filters.waveletDenoiseFilter.waveletDenoiseFilter method*), 121

filteredData (*mosaic.filters.metaIOFilter.metaIOFilter attribute*), 106

filterFs (*mosaic.filters.metaIOFilter.metaIOFilter attribute*), 106

flagEvent() (*mosaic.process.metaEventProcessor.metaEventProcessor method*), 105

flat2() (*in module mosaic.utilities.util*), 135

format() (*mosaic.utilities.mosaicLogging.MessageFormatter method*), 137

formatoutputfiles() (*mosaic.partition.eventSegment.eventSegment method*), 125

formatoutputfiles() (*mosaic.partition.metaEventPartition.metaEventPartition method*), 103

formatsettings() (*mosaic.filters.besselLowpassFilter.besselLowpassFilter method*), 122

formatsettings() (*mosaic.filters.convolutionFilter.convolutionFilter method*), 123

formatsettings() (*mosaic.filters.metaIOFilter.metaIOFilter method*), 106

formatsettings() (*mosaic.filters.waveletDenoiseFilter.waveletDenoiseFilter method*), 121

formatsettings() (*mosaic.partition.eventSegment.eventSegment method*), 125

formatsettings() (*mosaic.partition.metaEventPartition.metaEventPartition method*), 103

formatsettings() (*mosaic.process.adept.adept method*), 128

formatsettings() (*mosaic.process.adept2State.adept2State method*), 126

formatsettings() (*mosaic.process.cusumPlus.cusumPlus method*), 130

formatsettings() (*mosaic.trajio.metaTrajIO.metaTrajIO method*), 111

formatstats() (*mosaic.partition.eventSegment.eventSegment method*), 125

formatstats() (*mosaic.partition.metaEventPartition.metaEventPartition method*), 103

FsHz() (*mosaic.trajio.metaTrajIO.metaTrajIO property*), 111

FunctionTiming() (*mosaic.utilities.mosaicTiming.mosaicTiming method*), 137

G

getLogger() (*mosaic.utilities.mosaicLogging.mosaicLogging static method*), 137

getSettings() (*mosaic.settings.settings method*), 134

H

histogram_plot() (*in module mosaicscripts.plots.histogram*), 140

I

initDB() (*mosaic.mdio.metaMDIO.metaMDIO method*), 107

InvalidDataColumnError, 115, 118

InvalidEvent, 128, 129

L

LastFileProcessed (*mosaic.trajio.metaTrajIO.metaTrajIO attribute*), 111

M

mdAveragePropertiesList() (mosaic.process.adept.adept method), 128
mdAveragePropertiesList() (mosaic.process.adept2State.adept2State method), 126
mdAveragePropertiesList() (mosaic.process.cusumPlus.cusumPlus method), 130
mdAveragePropertiesList() (mosaic.process.metaEventProcessor.metaEventProcessor method), 105
mdHeadingDataType() (mosaic.process.metaEventProcessor.metaEventProcessor method), 105
mdHeadings() (mosaic.process.metaEventProcessor.metaEventProcessor method), 105
MessageFormatter (class in mosaic.utilities.mosaicLogging), 137
metaEventPartition (class in mosaic.partition.metaEventPartition), 102
metaEventProcessor (class in mosaic.process.metaEventProcessor), 104
metaIOFilter (class in mosaic.filters.metaIOFilter), 106
metaMDIO (class in mosaic.mdio.metaMDIO), 107
metaTrajIO (class in mosaic.trajio.metaTrajIO), 36, 109
module
mosaic.apps.ConvertTrajIO, 101
mosaic.apps.SingleChannelAnalysis, 100
mosaic.filters.besselLowpassFilter, 122
mosaic.filters.convolutionFilter, 123
mosaic.filters.waveletDenoiseFilter, 121
mosaic.mdio.sqlite3MDIO, 131
mosaic.partition.eventSegment, 124
mosaic.process.adept, 127
mosaic.process.adept2State, 125
mosaic.process.cusumPlus, 129
mosaic.settings, 133
mosaic.trajio.abfTrajIO, 113
mosaic.trajio.binTrajIO, 115
mosaic.trajio.chimeraTrajIO, 118
mosaic.trajio.qdfTrajIO, 114
mosaic.trajio.tsvTrajIO, 119
mosaic.utilities.fit_funcs, 138
mosaic.utilities.ionic_current_stats, 134
mosaic.utilities.mosaicLogFormat, 135
mosaic.utilities.mosaicLogging, 136
mosaic.utilities.mosaicTiming, 137
mosaic.utilities.util, 135
mosaicscripts.plots.contour, 141
mosaicscripts.plots.histogram, 140
mosaicscripts.plots.timeseries, 139
mosaic.apps.ConvertTrajIO
module, 101
mosaic.apps.SingleChannelAnalysis
module, 100
mosaic.filters.besselLowpassFilter
module, 122
mosaic.filters.convolutionFilter
module, 123
mosaic.filters.waveletDenoiseFilter
module, 121
mosaic.mdio.sqlite3MDIO
module, 131
mosaic.partition.eventSegment
module, 124
mosaic.process.adept
module, 127
mosaic.process.adept2State
module, 125
mosaic.process.cusumPlus
module, 129
mosaic.settings
module, 133
mosaic.trajio.abfTrajIO
module, 113
mosaic.trajio.binTrajIO
module, 115
mosaic.trajio.chimeraTrajIO
module, 118
mosaic.trajio.qdfTrajIO
module, 114
mosaic.trajio.tsvTrajIO
module, 119
mosaic.utilities.fit_funcs
module, 138
mosaic.utilities.ionic_current_stats
module, 134
mosaic.utilities.mosaicLogFormat
module, 135
mosaic.utilities.mosaicLogging
module, 136
mosaic.utilities.mosaicTiming
module, 137
mosaic.utilities.util
module, 135

mosaic_property (*class in mosaic.utilities.mosaicLogFormat*), 136
mosaicLogging (*class in mosaic.utilities.mosaicLogging*), 137
mosaicscripts.plots.contour
    module, 141
mosaicscripts.plots.histogram
    module, 140
mosaicscripts.plots.timeseries
    module, 139
mosaicTiming (*class in mosaic.utilities.mosaicTiming*), 137

O

OpenCurrentDist() (*in module mosaic.utilities.ionic_current_stats*), 134
openDB() (*mosaic.mdio.metaMDIO.metaMDIO method*), 107

P

partition() (*in module mosaic.utilities.util*), 135
PartitionEvents() (*mosaic.partition.metaEventPartition.metaEventPartition method*), 103
PlotTimeseries() (*in module mosaicscripts.plots.timeseries*), 139
popfnames() (*mosaic.trajio.metaTrajIO.metaTrajIO method*), 111
previewdata() (*mosaic.trajio.metaTrajIO.metaTrajIO method*), 111
PrintCurrentTime() (*mosaic.utilities.mosaicTiming.mosaicTiming method*), 138
PrintStatistics() (*mosaic.utilities.mosaicTiming.mosaicTiming method*), 138
ProcessedFiles (*mosaic.trajio.metaTrajIO.metaTrajIO attribute*), 111
processEvent() (*mosaic.process.metaEventProcessor.metaEventProcessor method*), 105

Q

qdfTrajIO (*class in mosaic.trajio.qdfTrajIO*), 37, 114
queryDB() (*mosaic.mdio.metaMDIO.metaMDIO method*), 108
queryDB() (*mosaic.mdio.sqlite3MDIO.sqlite3MDIO method*), 132

R

readAnalysisInfo() (*mosaic.mdio.metaMDIO.metaMDIO method*), 108
readAnalysisInfo() (*mosaic.mdio.sqlite3MDIO.sqlite3MDIO method*), 132
readAnalysisLog() (*mosaic.mdio.metaMDIO.metaMDIO method*), 108
readAnalysisLog() (*mosaic.mdio.sqlite3MDIO.sqlite3MDIO method*), 132
readdata() (*mosaic.trajio.abfTrajIO.abfTrajIO method*), 113
readdata() (*mosaic.trajio.binTrajIO.binTrajIO method*), 117
readdata() (*mosaic.trajio.chimeraTrajIO.chimeraTrajIO method*), 119
readdata() (*mosaic.trajio.metaTrajIO.metaTrajIO method*), 111
readdata() (*mosaic.trajio.qdfTrajIO.qdfTrajIO method*), 115
readdata() (*mosaic.trajio.tsvTrajIO.tsvTrajIO method*), 120
readSettings() (*mosaic.mdio.metaMDIO.metaMDIO method*), 108
readSettings() (*mosaic.mdio.sqlite3MDIO.sqlite3MDIO method*), 132
rejectEvent() (*mosaic.process.metaEventProcessor.metaEventProcessor method*), 105
Reset() (*mosaic.utilities.mosaicTiming.mosaicTiming method*), 138
Run() (*mosaic.apps.SingleChannelAnalysis.SingleChannelAnalysis method*), 101

S

scaleData() (*mosaic.trajio.binTrajIO.binTrajIO method*), 117
scaleData() (*mosaic.trajio.metaTrajIO.metaTrajIO method*), 112
scaleData() (*mosaic.trajio.tsvTrajIO.tsvTrajIO method*), 120
sd() (*in module mosaic.utilities.util*), 135
selectS() (*in module mosaic.utilities.util*), 135
settings (*class in mosaic.settings*), 134
SingleChannelAnalysis (*class in mosaic.apps.SingleChannelAnalysis*), 100
sqlite3MDIO (*class in mosaic.mdio.sqlite3MDIO*), 131
Stop() (*mosaic.apps.SingleChannelAnalysis.SingleChannelAnalysis method*), 101
Stop() (*mosaic.partition.metaEventPartition.metaEventPartition method*), 103

T

`time()` (*mosaic.utilities.mosaicTiming.mosaicTiming method*), 138

`tsvTrajIO` (*class in mosaic.trajio.tsvTrajIO*), 40, 119

U

`update()` (*mosaic.mdio.sqlite3MDIO.data_record method*), 131

W

`waveletDenoiseFilter` (*class in mosaic.filters.waveletDenoiseFilter*), 121

`writeAnalysisInfo()` (*mosaic.mdio.metaMDIO.metaMDIO method*), 108

`writeAnalysisInfo()` (*mosaic.mdio.sqlite3MDIO.sqlite3MDIO method*), 132

`writeAnalysisLog()` (*mosaic.mdio.metaMDIO.metaMDIO method*), 109

`writeAnalysisLog()` (*mosaic.mdio.sqlite3MDIO.sqlite3MDIO method*), 132

`writeEvent()` (*mosaic.process.metaEventProcessor.metaEventProcessor method*), 105

`writeRecord()` (*mosaic.mdio.metaMDIO.metaMDIO method*), 109

`writeRecord()` (*mosaic.mdio.sqlite3MDIO.sqlite3MDIO method*), 133

`writeSettings()` (*mosaic.mdio.metaMDIO.metaMDIO method*), 109

`writeSettings()` (*mosaic.mdio.sqlite3MDIO.sqlite3MDIO method*), 133