630 Komas Drive | Suite 200
Salt Lake City | UT 84108 | USA
P +1 801.582.5533 | F +1 801.582.1509
www.blackrockmicro.com

Python Offline Utilities

Table of Contents

Introduction 3

Modules.. 4

brMiscFxns.py 4

function openfilecheck 4

function checkequal 4

brpylib.py 4

class NevFile 5

class NsxFile.. 6

Example Scripts 9

example_extract_continous_data.py.................................. 9

example_extract_spike_data.py.. 9

example_save_subset_nsx.py 9

Warranty.. 10

Support 10

Version History.. 11

Introduction

Blackrock Microsystems provides a set of Python utilities to extract and plot data saved in NEV and NSx datafiles. These utilities require python 3.4 or newer, were built using the Anaconda distribution of Python 3.5, and have some dependencies as noted throughout this manual.

Blackrock Microsystems data are saved in two types of files:

- NEV: Short for Neural EVents, it contains events recorded during an experimental session. These events may include information on threshold crossed spike waveforms, information received through the digital input and the serial input, tracking information recorded in NeuroMotive Video Tracking System, text comments, or other custom comments sent to the NSP during an experiment.
- NSx: Short for Neural Stream X, where X indicates the sampling frequency of the continuous file. This data type contains continuous streamed data to the NSP. Depending on the sampling frequency, a second of data may contain up to 30,000 samples in this file.
    - NS1: Data sampled at 500 Hz
    - NS2: Data sampled at 1 kHz
    - NS3: Data sampled at 2 kHz
    - NS4: Data sampled at 10 kHz
    - NS5: Data sampled at 30 kHz

File Specifications (Format)
The data files can be recorded in various file specifications. Refer to LB-0023 NEV File Format available on the website for details on what each file format contains. Blackrock Microsystems recommends using the latest file format to record your data.

Modules

brMiscFxns.py

This library contains miscellaneous functions that are useful in many classes and scripts, and may be required by other modules (e.g., brpylib).   Current version: 1.00

Dependencies

- os
- qtpy

function **openfilecheck()**

Used to open a file in a specified manner.

Usage:
```
openedFile = openfilecheck(open_mode, file_name='',
                           file_ext='', file_type='')
```

Inputs:

open_mode:	{str} how to open the file. (*open_mode*='rb' for read binary)
file_name:	[optional] {str} complete path and name of file to open
file_ext:	[optional] {str} extension for file type to open. (*file_ext*='.pdf')
file_type:	[optional] {str} type of file to open. *file_type* = 'PDF Files')

Return: opened File object

function **checkequal()**

Used to check if all values within an iterator (often a list) are equal.

Usage: `checkequal(iterator)`
Inputs: an iterator object, often a list
Return: Boolean

brpylib.py

This library contains classes and functions for extracting information contained in NEV and NSx files saved using Blackrock Microsystems data acquisition systems. For more detailed information on all of the basic and extended header fields contained in the NsxFile and NevFile objects, refer to LB-0023 NEV File Format available on the website.   Current version: 1.10

Dependencies

- brMiscFxns
- collections
- datetime
- math
- numpy
- os
- struct
- time

class NevFile()

Object representing all experimental setup information (headers) and data stored in the NEV file. If no file is passed using the *datafile* parameter, a prompt will ask for a file name or request to browse to an NEV file. Basic and extended headers will be extracted during initialization.

Usage: `NevFileObj = NevFile(datafile)`

Inputs: *datafile* – [optional] {str} complete path to NEV file

Return: *NevFileObj* containing opened datafile, extracted basic header and extended headers

basic_header

The basic timing, creation, and comment information of the file.

datafile

The currently opened NEV file. It is recommended to run `NevFileObj.datafile.close()` after all data extracted.

extended_headers

Extended headers provide specific experimental information for the different data types stored in NEV files. Some data, such as for neural events, will have multiple extended headers for each channel (waveform and label information for neural data).

function getdata()

Returns event data for all the different possible data contained in NEV files.

Usage: `output = NevFileObj.getdata(elec_ids='all')`

Inputs: *elec_ids* – [optional] [list] of neural channel ids to extract. (elec_ids=[1, 2])

Return: *output* – Dictionary with one or more of the following ordered dictionaries. All dictionaries will include a list of Timestamps.

digital_data: Ordered dictionary of digital events containing Reason = ‘parallel’ or ‘serial’, lists of timestamps, and lists of data values, where indexing to the lists is based on the Reason.
Note: Serial data will already be in a single byte format with the upper byte of the 16-bit stored digital value having been stripped off.
Note: For file spec 2.2 and below, AnalogData and AnalogDataUnits will be also be included.

spike_data: Ordered dictionary of neural spike events containing Units=’nV’ and lists of ChannelID, TimeStamps, ExtendedHeaderIndices, Classification, and Waveforms. TimeStamps, Classifications, and Waveforms will be 2D array where indexing into the array will give the data for ChannelID[index].

comments: Ordered dictionary of comment event data containing lists of TimeStamps, CharSet, Flag, and comment strings.

video_sync_events: Ordered dictionary of video sync event data containing lists of TimeStamps, VideoFileNum, VideoFrameNum, VideoElapsedTime_ms, and VideoSource.

tracking_events: Ordered dictionary of tracking event data containing Parent and Node ID, TimeStamps, Node and Point Count, and tracking points.

button_trigger_events: Ordered dictionary of event data containing TimeStamps and TriggerType.

configuration_events: Ordered dictionary of configuration event data containing TimeStamps, ConfigChangeType, and the string of what configuration change occurred.

Dependencies: none

Note: when passing *elec_ids*, all digital events and other data contained in NEV (e.g., tracking data) will still be extracted. Only neural waveforms for channels not in *elec_ids* will be excluded.

function processroicomments()

Returns region of interest (ROI) data processed into an Ordered Dictionary of Regions, Enter event timestamps, and Exit event timestamps.

Usage:

```
roi_events = NevFileObj.processroicomments(comments)
```

Inputs: *comments* – Ordered dictionary returned from getdata().

Dependencies:

function getdata()()

class NsxFile()

Object representing all experimental setup information (headers) and data stored in the various NSx files. If no filename is passed using the file parameter, *datafile*, a prompt will ask for a file name or request to browse to an NSx file.

Usage: `NsxFileObj = NsxFile(datafile)`

Inputs: *datafile* – [optional] {str} complete path to NSx file

Return: *NsxFileObj* containing opened datafile, extracted basic header and extended headers

basic_header

The basic timing, creation, and comment information of the file.

datafile

The currently opened NSx file. It is recommended to run `NsxFile.datafile.close()` after all data extracted.

extended_headers

Each channel will have its own extended header with additional information such as electrode labeling, digitization factor, and filtering specs.

function **getdata**()

Returns a dictionary containing data parameters and a 2D array of data.

Usage:
```
output = NsxFileObj.getdata(elec_ids='all',
                            start_time_s=0, data_time_s='all',
                            downsample=1)
```

Inputs: *elec_ids* – [optional] [list] of neural channel ids to extract. (elec_ids=[1, 2]).

If specific elec_ids do not exist in the data, only those that do will be returned, along with a warning.

start_time_s – [optional] {float} Starting time for data extraction in seconds. (start_time_s = 1.0)

data_time_s – [optional] {float} Length of time of data to return. (data_time_s = 30.0)

downsample – [optional] {int} Downsampling factor (downsample = 2)

Return: *output* – Dictionary containing the following:

data_headers: The timestamp and number of data points for each data packet saved in the file. There will only be more than one data header when pausing is used, which will result in 0-padded data during pauses.

elec_ids: List of extracted electrode ids (sorted).

start_time_s: Starting time for data extraction into file.

data_time_s: Length of time of all data extracted.

downsample: The downsampling factor of the data, used to reduce data extraction size for large data files.

bytesize: Number of bytes per data sample.

data: 2D numpy array of continuous data. Indexing into the data will return the data for elec_ids[index].

Dependencies: none

Example: `NSxFile.getdata([5, 6, 7, 8, 9, 10], 1, 30, 2)` – returns 30 seconds of data starting from one second into the data file for channels 5-10 with a downsample factor of two.

Note: If bad parameters are passed, such as an invalid start_time_s or non-existant elec_ids, a warning will be displayed and parameters may be altered accordingly.

function **savesubsetnsx**()

Used to save a subset of data based on electrode IDs, file sizing, or file data time. If both *file_time_s* and *file_size* are passed, it will default to *file_time_s* and determine sizing accordingly.

Usage:
```
NsxFileObj.savesubsetsx(elec_ids='all',
   file_size=None, file_time_s=None, file_suffix='')
```

Inputs: *elec_ids* – [optional] [list] of neural channel ids to extract. (elec_ids=[1, 2]).

If specific elec_ids do not exist in the data, only those that do will be returned, along with a warning.

file_size – [optional] {in t} Byte size of each new file to save. If nothing is passed, file_size will be all data points. (e.g., 1024**3 = 1 Gb).

file_time_s – [optional] {float} Time length of data for each new file, in seconds. If nothing is passed, file_size will be used as default.

file_suffix – [optional] {str} Suffix to append to NSx datafile name for new files. If nothing is passed, default will be "_subset".

Return: None - None of the electrodes requested exist in the data
SUCCESS - All file subsets extracted and saved

Dependencies: none

Example: `NsxFileObj.savesubsetnsx(elec_ids=[1, 2, 20, 200], file_time_s=30, file_suffix='elecAndTime_subset')`
saves a set of data files that each have 30 seconds of data extracted sequentially for electrode IDs 1, 2, 20, and 200, if they exist in the data.

Note: If bad parameters are passed, a warning will be displayed and parameters may be altered accordingly.

Note: If the file name of the subset file already exists, an overwrite warning and request is presented.

Example Scripts

example_extract_continous_data.py

This example shows how to extract and plot continuous data saved in NSx files. Current version: 1.00

Dependencies

- brpylib
- matplotlib
- numpy
- sys
- time

example_extract_spike_data.py

This example shows how to extract and plot spike waveforms saved in the NEV files. Current version: 1.00

Dependencies

- brpylib
- matplotlib
- numpy

example_save_subset_nsx.py

This example shows how to extract and save a subset of data in a new set of NSx files from data saved in an NSx file. This can be useful when very large data files are created. Current version: 1.00

Dependencies

- brpylib

.

Warranty

Blackrock Microsystems (“Blackrock”) warrants its products are free from defects in materials and manufacturing for a period of one year from the date of shipment. At its option, Blackrock will repair or replace any product that does not comply with this warranty. This warranty is voided by: (1) any modification or attempted modification to the product done by anyone other than an authorized Blackrock employee; (2) any abuse, negligent handling or misapplication of the product; or (3) any sale or other transfer of the product by the original purchaser.

Except for the warranty set forth in the preceding paragraph, Blackrock provides no warranties of any kind, either express or implied, by fact or law, and hereby disclaims all other warranties, including without limitation the implied warranties of merchantability, fitness for a particular purpose, and non-infringement of third-party patent or other intellectual property rights.

Blackrock shall not be liable for special, indirect, incidental, punitive, exemplary or consequential damages (including without limitation, damages resulting from loss of use, loss of profits, interruption or loss of business or other economic loss) arising out of non-compliance with any warranty. Blackrock’s entire liability shall be limited to providing the remedy set forth in the previous paragraph.

Support

Blackrock prides itself in its customer support. For additional information on this product or any of our products, you can contact our Support team through the contact information below:

Manuals, Software Downloads, and Application Notes
www.blackrockmicro.com/technical-support

Issues or Questions
www.blackrockmicro.com/technical-support
support@blackrockmicro.com
U.S. - +1.801.839.1062
Europe - +49 (0)511.132.211.10

Version History

Version 1.0	Initial Release
Version 1.1	Inclusion of example example_save_subset_nsx to brpy directory
	Inclusion of function NsxFile.savesubsetnsx to file brpylib.py
Version 1.11	Updates to function NsxFile.savesubsetnsx wrt overwriting files