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

BLACKROCK® MICROSYSTEMS

Python Offline Utilities (brPy) Instructions for Use

Table of Contents

Introduction

Blackrock Microsystems provides a set of Python utilities to extract and plot data saved in NEV and NSx datafiles. These utilities were built using the Anaconda and PyCharm distributions of Python 2.7, 3.5, and 3.8. Dependencies are noted throughout this manual.

Blackrock Microsystems data are saved in two types of files:

- NEV: Short for Neural EVents, 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 from 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 Blackrock Microsystems website for details on what each file format contains. Blackrock Microsystems recommends using the latest file format to record your data.

Getting Started

The brPY.zip folder contains folders that contain the Windows versions and OSx versions of the brPY libraries. The loader consists of two libraries: brMiscFxns.py and brpylib.py.

To get started using brPY, move brMiscFxns.py and brpylib.py into the libraries folder of your Python distribution. Please see the documentation of your specific Python distribution to find the correct filepath to move the libraries into.
The Python loader consists of two classes: NsxFile and NevFile, which open and extract continuous neural data and neural event data, respectively, as well as experimental setup information in the headers. Both the NsxFile and NevFile classes contain a getdata() function,

which is used to extract data from the .nsx or .nev file. The following examples demonstrate the use of NsxFile's getdata() function to extract continuous data, then writes it to a .txt file.

```
from brpylib             import NsxFile

datafile = 'C:/Users/MaxPower/Desktop/sampledata/sampleData/' \
           'sampleData.ns3'

elec_ids     = [1, 15, 2, 20, 5 ]  # 'all' is default for all (1-indexed)
start_time_s = 1                     # 0 is default for all

nsx_file = NsxFile(datafile)
```

Before beginning to load data, you must import the relevant functions from brpylib. In this example, we have specified the exact filepath in the command to open a file. If you do not specify a filepath or specify an invalid filepath, you will be prompted in the console to either enter a complete file path or hit enter to bring up a file selection window:

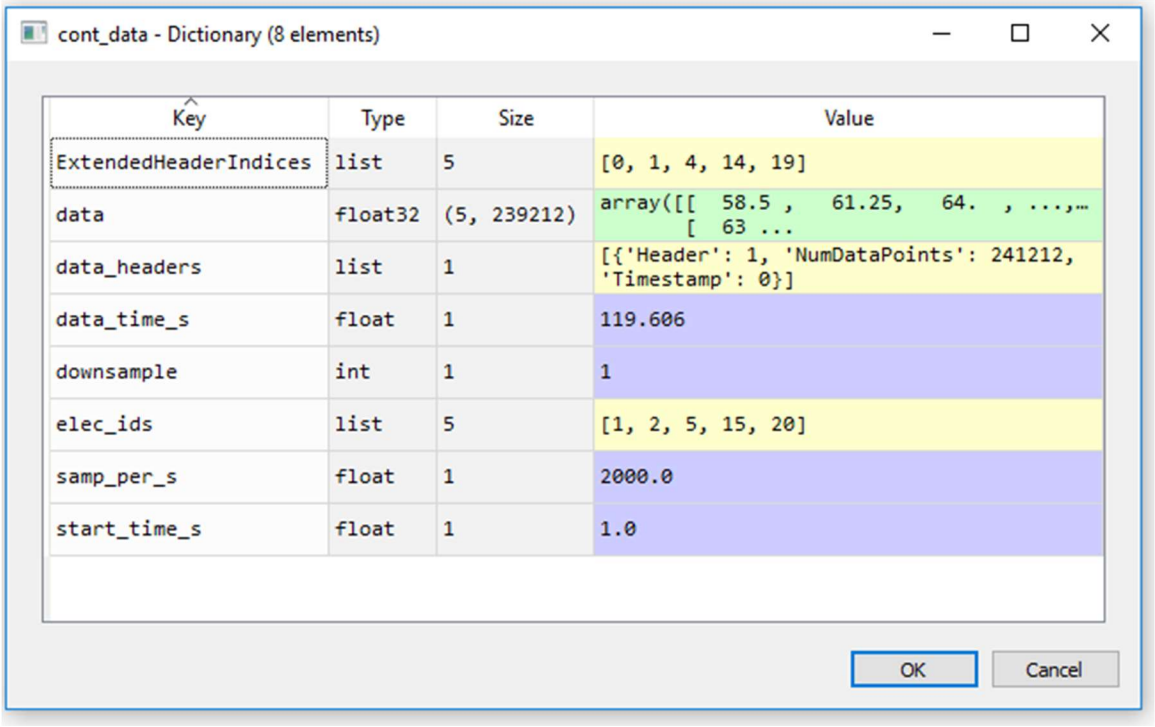
```
Enter complete .ns* file path or hit enter to browse:
```

After the .ns5 file has been opened, data we want can be extracted from the file. Previously, two of getdata's optional parameters (the electrodes we want data from, and the time in seconds we want to start data extraction from) were defined and we will use them here to narrow the scope of the data we will be extracting. Then, after we have finished extracting the data, the .nsx file is closed.

```
cont_data = nsx_file.getdata(elec_ids, start_time_s)

nsx_file.close()
```

getdata() creates an ordered dictionary with data parameters and the data array. More information on the fields created by getdata() and the optional parameters that can be passed are found in the section on getdata() later in the IFU. By default, `cont_data['data']` will be an N-by-M numpy array, where N is the number of channels and M is the number of samples.



To export this data to a text file, we open a text file in write-mode and then iterate over the channel and sample indices of `cont_data['data']`, and then close the text file once we are done:

```
data_txt = open("output.txt","w")

for j in range(0,cont_data['data'].shape[1]):
    for i in range(0,cont_data['data'].shape[0]):
        data_txt.write("%f\t" % cont_data['data'][i][j])
        if i==cont_data['data'].shape[0]-1:
            data_txt.write("\n")

data_txt.close()
```

This results in a text file named output.txt in your Python distribution's working path. Each column of the text file corresponds to a specific channel's 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: 1.2.0

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: *iterator*: {list}, {dict} iterator object

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 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: 2.0.0

Dependencies

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

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 extendedheaders

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 is 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])

wave_read -[optional] {str} Specifies whether or not to read the spike waveforms. Default is to read the waveforms. (wave_read='noread', wave_read='read')

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

spike_data: Dictionary of neural spike events containing lists of ChannelID, TimeStamps, Unit, Channel, and Waveforms. TimeStamps, Unit, and Waveforms will be 2D array where indexing into the array will give the data

for ChannelID[index]. If wave_read = 'noread' is passed, waveforms will not be written to the data object.
Note: All voltage waveforms are in bits, which are represented as 0.25 µV/bit.

dig_events: Dictionary of digital events containing Reason = 1 (digital input) or 2 (serial input), 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.

comments: Dictionary of comment event data containing lists of TimeStamps, TimeStampsStarted, Data, and CharSet. Data contains the comments in character format. TimeStamps indicates when the comment was entered into the file, whereas TimeStampsStarted indicates when the comment writing was started by a user.

tracking_events: Dictionary of tracking events pertaining to Regions of Interest (ROIs) containing lists of TimeStamps, ROIName, ROINumber, Event, and Frame.
Note: Usage of processroicomments() is not needed to parse this data if using brPy v2.0.0+.

video_sync_events: Dictionary of video sync event data containing lists of TimeStamps, FileNumber, FrameNumber, ElapsedTime (in ms), and SourceID.

tracking: Dictionary of tracking event data containing TimeStamps, ParentID, NodeCount, MarkerCount, and tracking points (X and Y). This dictionary concludes with TrackerObjs, which contains the type of the tracked objects. If several types are present, they are enumerated with base-0 counting.

PatientTrigger: Dictionary of event data containing TimeStamps and TriggerType.

reconfig: Dictionary of configuration event data containing TimeStamps and ChangeType.

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 a Dictionary of Regions, Enter event timestamps, and Exit event timestamps.

Usage:

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

Inputs: *comments* – Dictionary returned from getdata().

Dependencies: function getdata()

Note: this function does not need to be used to extract comments for brPy version 2.0.0+

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 is 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: {float} Starting time for data extraction into file.

data_time_s: {float} Length of time of all data extracted.

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

samp_per_s: {float} The samples per second of the returned data

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

ExtendedHeaderIndices: List containing the extended header indices of the electrodes specified in elec_ids. If elec_ids is not passed, this will be [0]

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_writeNSX_toTxt.py

This example shows how to extract data from NSx files and write it to a text file. Current version: 1.0

Dependencies

- brpylib
- numpy

example_extract_spike_data.py

This example shows how to extract and plot spike waveforms saved in the NEV files. Current version: 1.1.2

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: 1.1.1

Dependencies

- brpylib

.

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