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

cbMEX

Instructions for Use

Revision 3.00 / LB-590

Contents

Welcome!

Thank you for choosing Blackrock Microsystems!

Inside this manual, you will find information on the cbMEX library, a utility that reads Blackrock Microsystems Neural Signal Processor (NSP) data packets into MATLAB and enables researchers to perform online processing and analysis of their data as well as programmatically control and communicate with their NSP using MATLAB.

This manual will cover the system requirements, associated hardware, and an overview of the most important commands within the library.

For questions on this product or any other Blackrock products, contact our 24/7 support service: support@blackrockmicro.com

System Requirements

The specifications listed below are the minimum required for the software to run as intended.

- Microsoft Windows 7 (32-bit or 64-bit)
- AMD or Intel 2.0GHz Quad Core CPU
- 4 GB of RAM
- Gigabit (1 Ghz) Ethernet Adapter or USB 2.0 Port (for communicating with Blackrock Microsystems' Data Acquisition Systems)

Installation

cbMEX is automatically installed alongside the Central Software Suite. The required files will be in Central's installation directory. Please add Central's installation directory to MATLAB's search path to ensure full functionality: instructions on modifying the MATLAB search pathway can be found at https://www.mathworks.com/help/matlab/matlab_env/add-remove-or-reorder-folders-on-the-search-path.html. By default, this directory is: C:\Program Files\Blackrock Microsystems\XXXX Windows Suite (XXXX may be either NeuroPort, Cerebus, or CerePlex Direct depending on your device).

The exact method of adding to MATLAB's search path may differ between versions of MATLAB—refer to your specific version's documentation if the above instructions do not work.

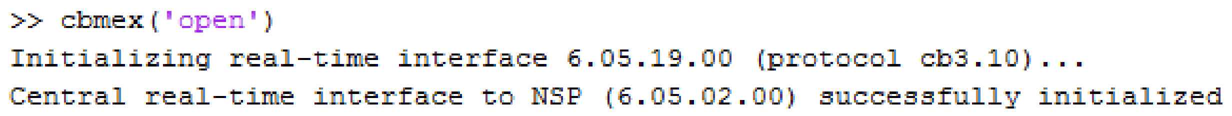
Use Cases

The cbMEX library has the following typical use cases:

- Read spike timestamps and continuous sample data as well as other data coming through the analog and digital inputs. For Cerebus and CerePlex Direct systems, it can also read data from the optional video tracking system (NeuroMotive).
- Start and stop file recording programmatically based on analysis of the data
- Programmatically control channel configuration, channel labels, comments, and digital outputs

Getting Started

Use the *open* command to initialize the Matlab interface:



```
>> cbmex('open')
Initializing real-time interface 6.05.19.00 (protocol cb3.10)...
Central real-time interface to NSP (6.05.02.00) successfully initialized
```

The above output is shown when cbmex(‘open’) has successfully connected through Central. See the section on ‘open’ for more detail on active parameters and output. For cbMEX to connect through Central successfully, Central must be open and running when cbmex('open') is called.

When ‘trialconfig’ is executed with the active parameter set to 1, cbMEX begins to read data into a buffer. By default, the event data buffer is 16384 events per channel and the continuous sampling buffer defaults to 102400 samples per channel (about 3 seconds at the Neural Signal Processor’s sample rate of 30khz). The buffer will not accept any new data beyond this point, so choose an appropriate read delay to clear the buffer frequently enough.

Calling ‘trialdata’ reads the data currently contained in the buffer and can optionally return that data as a MATLAB variable.

```
%set channels 3-7 to sample continuously at different sampling rates
cbmex('config',3,'smpgroup',1);
cbmex('config',4,'smpgroup',2);
cbmex('config',5,'smpgroup',3);
cbmex('config',6,'smpgroup',4);
cbmex('config',7,'smpgroup',5);

trial_length=5; %How long to activate cbmex/data collection
current_time=tic; %start clock to measure against trial_length

cbmex('trialconfig',1) %initialize and start reading data into buffer

while (trial_length>toc(current_time))
    pause(0.5); %Pause for a defined amount of time;
    [event_data,time,cont]=cbmex('trialdata',1); %Read data & flush buffer
end
```

In the above example, 'trialconfig' is called after channels are configured (see section on 'config' for more detail) and the trial length is set. Upon call of 'trialconfig,' MATLAB initializes the data buffer and begins filling the buffer with data. Data collection is configured to continue for five seconds: within this five seconds, cbMEX pauses for half a second, allowing time for data to fill the buffer. Channel 7 is configured to sample continuously at 30,000 kHz on channel 7), so we can buffer for about 3 seconds before the continuous buffer fills up.

Once the pause is over, 'trialdata' is called to read the previous half-seconds' worth of data, then flush the buffer (see section 'trialdata' for more information). 'trialdata' returns a cell array of event data, the time of call, and a cell array of continuous data (shown below).

```
cont =


    [3]    [  500]    [  260x1 int16]
    [4]    [ 1000]    [  520x1 int16]
    [5]    [ 2000]    [ 1040x1 int16]
    [6]    [10000]    [ 5200x1 int16]
    [7]    [30000]    [15600x1 int16]
```

Basic Online Data Monitoring & Analysis

Data that is read from 'trialdata' can be used for online monitoring and analysis during the recording using some of cbMEX's other features. The following example demonstrates use of cbMEX to conduct basic online data reading/analysis.

```
cbmex('open')
cbmex('trialconfig',1)

trial_length=10;
snippet_length=0.5;
num_cycle=1;
current_time=tic;

while (trial_length>toc(current_time))
    pause(snippet_length);
    [event_data]=cbmex('trialdata',1);
    counter = 0;
    for i=1:length(event_data)
        if ~isempty(event_data{i,2})
            counter = counter+1;
        end
    end
    disp(sprintf('Cycle %d: Spike activity on %d channels',num_cycle,...
        counter));
    num_cycle = num_cycle+1;
end

cbmex('close');
```

During each read-flush cycle begun by 'trialdata', data from the buffer is read and the number of channels that have spike activity is determined. Before initiating the next buffer cycle, a command line output is generated stating the number of channels that displayed spike activity during that cycle.

```
Initializing real-time interface 6.05.19.00 (protocol cb3.10)...
Central real-time interface to NSP (6.05.02.00) successfully initialized
Cycle 1: Spike activity on 54 channels
Cycle 2: Spike activity on 96 channels
Cycle 3: Spike activity on 96 channels
Cycle 4: Spike activity on 96 channels
Cycle 5: Spike activity on 49 channels
Cycle 6: Spike activity on 51 channels
Cycle 7: Spike activity on 51 channels
Cycle 8: Spike activity on 55 channels
Cycle 9: Spike activity on 50 channels
Cycle 10: Spike activity on 55 channels
Cycle 11: Spike activity on 53 channels
Cycle 12: Spike activity on 54 channels
Cycle 13: Spike activity on 51 channels
Cycle 14: Spike activity on 53 channels
Cycle 15: Spike activity on 52 channels
Cycle 16: Spike activity on 50 channels
Cycle 17: Spike activity on 51 channels
Cycle 18: Spike activity on 73 channels
Cycle 19: Spike activity on 52 channels
Cycle 20: Spike activity on 52 channels
```

Conventions

< >	Used to denote optional parameters.
<key, value>	Pairs are optional; some pairs do not require values. From left to right parameters will override previous ones or combine with them if possible.

Commands

OPEN

[<connection> <instrument>] = cbmex('open', <interface>, <key, value>);

Use the 'open' command to initialize cbMEX and connect to the system through Central or UDP. The command line will print out the current cbMEX version, protocol version and NSP version numbers. You may specify which connection method (Central or UDP) to use with the **<interface>** parameter. To connect through Central, Central must be running when 'open' is called. If no parameter or 0 (default) is used, the interface first tries connecting through Central and if that fails, tries connecting using the UDP interface.

The application can connect to multiple instruments at once (up to 4) using 'instance.' You may also connect to an NSP by specifying the address and port of those instruments using 'inst-addr' and 'inst-port,' or specifying the address and port of your application or Central to which the instrument will send its data using 'central-addr' and 'central-port.'

Inputs

<interface>	0: (Default) Try Central 1st then UDP
	1: Connect using Central
	2: Connect using UDP

<key, value>

'instance'	Library instance, numbered 0 (default) to 3
'inst-addr'	String containing the instrument's ipv4 address (default is 192.168.137.128)
'inst-port'	Control port number (default is 51001)
'central-addr'	String containing the ipv4 address of the Central instance (default is 192.168.137.X where X is 1 through 16 and can be found in Network Devices)
'central-port'	Broadcast port number (default is 51002)

Outputs

<connection>	1 - Connected to Central
	2 - Connected via UDP
<instrument>	0 - Connected to an NSP
	1 - Connected to nPlay running on the same PC
	2 - Connected to an NSP running on the NSP hardware (i.e., connected to a CerePlex Direct)
	3 - Connected to nPlay running on a separate PC (Broadcasting)

Examples

```
%Default is to try Central, then UDP
cbmex('open')

%connect through Central only; specify the ipv4 address of Central
cbmex('open', 1, 'central-addr', '192.168.137.1');

%connect to a second NSP on the same computer
cbmex('open', 1, 'central-addr', '192.168.137.17',’instance’,1);

%connect to a third NSP on the same computer
cbmex('open', 1, 'central-addr', '192.168.137.33',’instance’,2);
```

CLOSE

cbmex('close', <key, value>);

This command closes an instance of the library. Up to 4 unique instances may be open at once in an cbMEX and with 'close' you can specify an instance to close or have *instance* default to 0.

Inputs

<key, value>

'instance'	Library instance, numbered 0 (default) to 3.

Outputs

None

Examples

```
% Close the default interface to NSP
cbmex('close');

% Close a specific instance
cbmex('close', 'instance', 1);
```

FILECONFIG

cbmex('fileconfig', filename, comments, action, <key, value>) ;

Starts or stops file recording. The filename and comments are specified. The filename can contain the full path to the filename. If the path doesn't exist, it will be created.

Inputs

filename	File name string (255 character maximum)
comments	File comment string (255 character maximum)
action	1 starts recording 0 stops recording
<key, value>	
'instance'	Library instance, numbered 0 (default) to 3.
'option'	Value can be any of the following: 'close' – closes the File dialog if open 'open' – opens the File dialog if closed, ignoring other parameters 'none' – opens the File dialog if closed, sets parameters given, starts or stops recording

Outputs

<recording>	Unavailable if using any 'option' parameters 1 – recording is in progress 0 – recording is not in progress
<filename>	recording file name; unavailable if using any 'option' parameters
<username>	recording user name; unavailable if using any 'option' parameters

Note: output may have 1 (<recording>) or 3 (<recording> ,<filename>,<username>) values.

Examples

```
% Start file recording the specified file
cbmex('fileconfig', 'c:\data\20120420', '', 1);

% Start file recording the specified file with a comment
cbmex('fileconfig', 'c:\data\20120420', 'First trial with Fred', 1);

% Stop recording
cbmex('fileconfig', 'c:\data\20120420', '', 0);
```

```
% Start file recording, open the File dialog,
cbmex('fileconfig', 'c:\data\20120420', '', 1, 'option', 'open');

% Return recording status, output file and user name
[rec filename username] = cbmex('fileconfig');

% Return recording status on a specific instance
rec = cbmex(‘fileconfig’, ‘instance’, 1);
```

CONFIG

config_cell_array = cbmex('config', channel, <key, value>);

This command is used to return the channel configuration and optionally set new channel configurations for a given channel. Parameter values are in raw format.
Although the firmware will reject most invalid parameter settings, setting an invalid configuration for a channel may cause unpredictable behavior. It is strongly recommended not to modify parameters that are not immediately relevant.
Changing a parameter during recording may lead to unexpected behavior—for example, if a new channel is added to a sampling group the data file will be corrupted because the channels will be misaligned.
Some parameters (such as threshold) are only recorded once at beginning of the file and changing these parameters may not be visible to data playback tools. Custom events or comments may be used to record such changes and customize the data playback tool.

Inputs

channel	channel number
<key, value>	
'instance'	Library instance, numbered 0 (default) to 3.
'userflags'	User supplied information about the channel.
'smpgroup'	Number from 0 to 5 indicating the continuous sampling rate group the channel belongs to. Values correspond to the following sampling rates: 0 – None 1 – 500Hz 2 – 1kHz 3 – 2kHz 4 – 10kHz 5 – 30kHz
'smpfilter'	Number from 0-16 defining the continuous Butterworth filter to apply to the specified channel. Values correspond to the following filter values: 0 – None 1 – 750Hz High pass 2 – 250Hz High pass 3 – 100Hz High pass 4 – 50Hz Low pass 5 – 125Hz Low pass 6 – 250Hz Low pass 7 – 500Hz Low pass 8 – 150Hz Low pass 9 – 10Hz-250Hz Band pass

	10 – 2.5kHz Low pass 11 – 2kHz Low pass 12 – 250Hz-5kHz Band pass 13 to 16 are reserved for user defined filters created using the Digital Filter tool and loaded through Central.
'spkfilter'	Number from 0 – 16 defining the spike butterworth filter to apply to the specified channel. Input values and their corresponding filter values are the same as above in 'smpfilter.'
'spkgroup'	Number indicating the nTrode group to which the channels belong.
'spkthrlevel'	String containing the voltage requested. E.g. '-100uV'
'amplrejpos'	String indicating the positive value over which spikes will be rejected.
'amplrejneg'	String indicating the negative value under which spikes will be rejected.
'refelecchan'	Channel number to subtract from this channel for reference.

Outputs

config_cell_array:	Previous parameters. Each row in this matrix contains: <key> [value]

Examples

```
% Get full configuration of channel 4
config = cbmex('config', 4)

% Set threshold of the channel 5 to +500mV
cbmex('config', 5, 'spkthrlevel', '500mV')

% Get full configuration of channel 6, and set its new threshold to -
65uV
config = cbmex('config', 6, 'spkthrlevel', '-65uV')

% Set amplitude reject to +-1V for channel 5
cbmex('config', 5, 'amplrejpos', 1000, 'amplrejneg', -1000)
```

TRIALCONFIG

[<active_state>, <config_vector_out>] = cbmex('trialconfig', active, <config_vector_in>, <key, value>)

Use this command to initialize the trial and set cbMEX to start or stop buffering data. The initial call to 'trialconfig' with *active* set to 1 will begin buffering data for collection while subsequent calls with *active* set to 1 will flush the buffer. Flushed data cannot be retrieved. Before you call 'trialconfig,' be aware of whether there is new data waiting in the buffer to be collected. For further information please see the description for 'trialdata.'

Inputs

active	1 - flushes the data cache and starts buffering data 0 - stops buffering data
config_vector_in:	Vector that can be used to configure inputs from the digital input or serial input port to begin a trial and to end a trial.
[begchan begmask begval endchan endmask endval]	
begchan	Specifies the channel used to start a trial and start buffering data
begmask	Specifies the hex mask to apply to the digital data to start a trial and start buffering data
begval	Specifies the hex value the digital data must match to start a trial and start buffering data
endchan	Specifies the channel used to end a trial and stop buffering data
endmask	Specifies the hex mask to apply to the digital data to end a trial and stop buffering data
endval	Specifies the hex value the digital data must match to end a trial and stop buffering data
<key, value>	
'instance'	Library instance, numbered 0 (default) to 3
'double'	No value required. When specified, the data is in double precision format and timestamps are represented in seconds rather than as sample numbers
'absolute'	No value required. Event timing is absolute (setting 'active' to 1 will not reset time for events) and so time will not be relative to the start of the trial
'noevent'	No value required. The event data cache is not created nor configured (same as 'event', 0)
'nocontinuous'	No value required. If specified, a continuous data cache is not created nor configured (same as 'continuous', 0)
'continuous'	Set the number of continuous data points to be cached/buffered for each channel in the trial. Defaults to

	102400 sample points per channel if not set and 'nocontinuous' is not used.
'event'	Set the number of events to be cached/buffered in a trial. Defaults to 2097152 events encompassing all channels if not set and 'noevent' is unused.
‘comment’	Number indicating the number of comments to store in the buffer.
‘tracking’	0 or 1 indicating disabled (0) or enabled (1)

Outputs

active_state	Return 1 if data collection is active at the time that ‘trialconfig’ is called, 0 otherwise (i.e., the initial call of ‘trialconfig’ after starting cbMEX results in active_state = 0).
config_vector_out:	Vector specifying the configuration state with the values either entered or their defaults if not (e.g. 'continuous' will be the number of sample points per channel to be buffered or 0 if 'nocontinuous' was specified) **[begchan begmask begval endchan endmask endval double waveform continuous event comment tracking]**

Examples

```
% Stop data collection
cbmex('trialconfig', 0)

% Configure to return continuous data as double type and timestamps in
seconds
cbmex('trialconfig', 1, 'double')

% Configure to buffer only event data
cbmex('trialconfig', 1, 'nocontinuous')

% Configure to buffer 200000 continuous data points in the double format
cbmex('trialconfig', 1, 'double', 'noevent', 'continuous', 200000)
```

TRIALDATA

[timestamps_cell_array, <time>, <continuous_cell_array>] = cbmex('trialdata', active, <key, value>)

Reads the data currently stored in the buffer and optionally return it as MATLAB variables. The buffer contains data stored since the trial started or the last time the data buffer was flushed, including old data and new data added since the last call to 'trialconfig' if the data buffer was not flushed.

It is advisable to flush the data cache frequently, by calling either cbmex('trialconfig', 1) or cbmex('trialdata', 1). If this call to 'trialdata' cleared the buffer (active set to 1),<time> is the time at which the previous buffer started. The time is set for the next call to 'trialdata.'

If you change the sampling rate on a given channel, its values are erased so the next time you call 'trialdata' be aware that sample values on different channels may not be in alignment because of the data reset and the different sampling rates.

Inputs

Active	0 - (default) leave buffer intact (do not clear the buffer) 1 - clear all the data and reset its recording time to the current time
<key, value>	
'instance'	Library instance, numbered 0 (default) to 3.

Outputs

timestamps_cell_array	Timestamps for events of all of the channels consisting of the front end amp channels, analog input channels, analog output, audio output, digital input, and serial input. By default, timestamps are returned as UINT32 representing a sample number at a sampling rate of 30 kHz. If 'double' parameter is passed in 'trialconfig,' timestamps are returned as seconds elapsed since trial start or since last call of 'trialconfig' with active set to 1.

Each row in this matrix contains:
For spike channels:
'channel name' [unclassified timestamps_vector] [u1_timestamps_vector] [u2_timestamps_vector] [u3_timestamps_vector] [u4_timestamps_vector] [u5_timestamps_vector]
...u1-u5 are the spike sorting units per channel while unclassified is any spike not sorted into a unit...

For digital input channels
'channel name' [timestamps_vector] [values_vector] ...remaining columns are empty...

time	Time (in seconds) that the data buffer was most recently cleared.
continuous_cell_array	An n x 3 cell array containing continuous sample data, n being the number of channels configured to collect continuous data (i.e., each row is a channel). By default, continuous data values are returned as signed 16bit integers (INT16), and any digital values are unsigned 16bit integers (UINT16). If ‘double’ parameter was passed in ‘trialconfig,’ values are returned as double type.

Each row in this cell contains:
[channel number] [sample rate (in samples / s)] [values_vector]

Examples

```
% read event data, do not clear the buffer
event_data = cbmex('trialdata', 0);

% read event data, clear the buffer for the next trialdata
event_data = cbmex('trialdata', 1);

% read continuous data, time, and events, then clear the buffer
[event_data, t, continuous_data] = cbmex('trialdata', 1);

% read continuous data, then clear the buffer
[t, continuous_data] = cbmex('trialdata', 1);
```

TIME

Time = cbmex(‘time’, <key, value>);

Returns the current NSP time in seconds. This command can optionally specify the time for different instances (instruments) and report the time in terms of the number of samples that have occurred.

The timer starts over when Reset is pressed in Central and also when recording starts.

Inputs

<key, value>

'instance'	Library instance, numbered 0 (default) to 3.
'samples'	No value required. Return the number of samples instead of the time in seconds.

Outputs

time	The time for the current instance (default instance is 0) in either seconds or samples (default is seconds) since the instrument was last reset.

Examples

```
% Get the current time for the default instance (instrument)
time = cbmex('time');

% Get the time for a specific instance in samples
time = cbmex('time', 'instance', 1, 'samples');

% Get the time in samples
time = cbmex('time', 'samples');
```

DIGITALOUT

cbmex('digitalout', channel, value, <key, value>);
cbmex('digitalout', channel, <key, value>);

Sends a value to one of the Digital Out ports on the NSP as long as the port is not configured to monitor a channel or set for timed output.

Inputs

channel		1 - (dout1) 2 - (dout2) 3 - (dout3) 4 - (dout4**)**
Value		1 - sets digital output to TTL high 0 - sets digital output to TTL low
<key, value>		
	'instance'	Library instance, numbered 0 (default) to 3.
	'monitor'	Can be any combination of the following: 'unclass' - monitor unclassified spikes 'unit1' - monitor unit 1 'unit2' - monitor unit 2 'unit3' - monitor unit 3 'unit4' - monitor unit 4 'unit5' - monitor unit 5 'all' - monitor all units including unclassified
	'track'	No value required. Monitor the last tracked channel
	'disable'	No value required. Disable digital output
	'timed'	Can be any of the following: 'frequency' – input and value parameters are frequency and offset 'samples' - input and value parameters are # of samples and offset
	'trigger'	Trigger can be any of the following: 'off' - this trigger is not used 'instant' - Immediate trigger (immediate digital output waveform) 'dinrise' - digital input rising edge 'dinfall' - digital input falling edge 'spike' - spike event on given input channel 'roi' - trigger based on neuromotive ROI (region of interest)

	'softreset' - trigger based on software reset (e.g. result of file recording)
'input'	Input depends on 'trigger' or 'timed' If trigger is 'dinrise' or 'dinfall' then 'input' is bit number of 1 to 16 for first digital input If trigger is 'spike' then 'input' is input channel with spike data If trigger is 'roi' then 'input' is the region of interest 1-4 If timed is 'frequency' then 'input' is timed frequency If timed is 'samples' then 'input' is number of samples high
'value'	Trigger value depends on 'trigger' If trigger is 'roi' then 'value' is 1=enter, 2=exit If trigger is 'spike' then 'value' is spike unit number (0 for unclassified, 1-5 for first to fifth unit and 254 for any unit) If timed is 'frequency' then 'value' is timed duty cycle If timed is 'samples' then 'value' is number of samples low
'offset'	offset of where to start timed output

Outputs

None

Examples

```
% Sets dout1 to TTL high
cbmex('digitalout', 1, 1);

% Sets dout1 to TTL low
cbmex('digitalout', 1, 0);

% Sets dout3 to Monitor unit 1, 3, and 5 of channel 8 and track channels
cbmex('digitalout',  3,  'monitor',  'unit1  unit3  unit5',  'input',  8,
track');

%  Sets  dout2  to  generate  a  1000Hz  waveform  with  a  20%  duty  cycle
cbmex('digitalout', 2, 'timed', 'frequency', 'input', 1000, 'value', 20);

% Sets dout2 to generate a waveform with 15 samples high and 30 low with
an offset of 30 samples
cbmex('digitalout',  2,  'timed',  'samples',  'input',  15,  'value',  30,
'offset', 30);
```

CHANLABEL

label_cell_array = cbmex('chanlabel', <channels_vector>, <new_label_cell_array>, <key, value>)

Get channel label(s) and optionally set new channel labels for given channel(s).

Inputs

<channels_vector>	A vector of all the channel numbers to change. If not specified, then all channels will be changed. The *new_label_cell_array* must be of the same length as *channels_vector*.
<new_label_cell_array>	Cell array of new labels for each channel in the channels_vector. Each string can be a maximum of 16 characters.

<key, value>

'instance'	Library instance, numbered 0 (default) to 3.

Outputs

label_cell_array

For spike channels, each row in this matrix contains: 'channel label' [spike_enabled] [unit1_valid] [unit2_valid] [unit3_valid] [unit4_valid] [unit5_valid]
For digital input channels, each row in this matrix contains: 'channel label' [digin_enabled] ...remaining columns are empty...

Note: In this version of cbmex, only Hoops unit validity is returned.

Examples

```
% Get all the channel labels
chan_labels = cbmex('chanlabel');

% Get channel label of channel 156
chan_label = cbmex('chanlabel', 156);

% Get channel label of digital and serial channels
chan_labels = cbmex('chanlabel', [151 152]);

% Set channel label of channel 5 to ch5
chan_labels = cbmex('chanlabel', 5, 'ch5');

% Set channel label of channel 6 to name
chan_labels = cbmex('chanlabel', 6, {'name'});

% Set labels for channels 5 and 6
chan_labels = cbmex('chanlabel', [5 6], {'e5' 'e6'});

% Get labels for channels 2 to 6
chan_labels = cbmex('chanlabel', 2:6);
```

MASK

cbmex(‘mask', channel, <active = 1>, <key, value>)

Activate or deactivate data collection for specified channels. The mask is applied both to data buffering (*trialconfig*) and to data retrieval (*trialdata*).

Inputs

<Channel>	The channel number to mask. 0 indicates all channels.
<Active>	1 (default) to activate. 0 to deactivate.
<key, value>	
'instance'	Library instance, numbered 0 (default) to 3.

Outputs

None

Examples

```
% Activate all the channels
cbmex('mask', 0)

% Activate all the channels
cbmex('mask', 0, 1)

% Deactivate all the channels
cbmex('mask', 0, 0)

% Deactivate channel number 5
cbmex('mask', 5, 0)
```

COMMENT

cbmex('comment', rgba, charset, comment, <key, value>);

Generate a comment or custom event. The comment appears on applications displaying comments (such as *Raster*) and is recorded if file recording is active. The color parameter (*rgba*) can be used for custom events.

Inputs

rgba	Color coding or custom event number. Common color codes: black – 0 white – 16777215 red – 255 green – 65280 blue – 16711680 yellow – 65535 magenta – 16711935 cyan – 16776960
charset	0 - ASCII 1 - UTF16
comment	Comment string (maximum 127 characters)
<key, value>	
'instance'	Library instance, numbered 0 (default) to 3.

Outputs

None

Examples

```
% Add white ASCII comment
cbmex('comment', 16777215, 0, 'my comment');

% Add green ASCII comment
cbmex('comment', 65280, 0, 'my comment');

% Add red UTF16 comment
cbmex('comment', 255, 1, 'my comment');
```

ANALOGOUT

cbmex('analogout', channel, <key, value>)

Inputs

Channel	1 - (aout1) 2 - (aout2) 3 - (aout3) 4 - (aout4) 5 - (audout1) 6 - (audout2)

<key, value>

'instance'	Library instance, numbered 0 (default) to 3.
'pulses'	Waveform vector in format: [nPhase1Duration nPhase1Amplitude nInterPhaseDelay nPhase2Duration nPhase2Amplitude nInterPulseDelay]
'sequence'	Waveform is variable length vector of duration and amplitude. Waveform format is [nDuration1 nAmplitude1 nDuration2 nAmplitude2 ...] Waveform must be a nonempty even-numbered vector of double-precision numbers. Each duration must be followed by amplitude for that duration Durations must be positive and indicate number of samples. Amplitudes are given in binary and range from -32767 for -5V and +32767 for 5V. Each duration-amplitude pair is a phase in the waveform
'sinusoid'	Waveform is vector [nFrequency nAmplitude]
'monitor'	Can be either: 'spike', or 'continuous' 'spike' - spikes on 'input' channel are monitored 'continuous' - continuous 'input' channel is monitored
'track'	No value required. Monitor the last tracked channel
'disable'	No value required. Disable analog output
'offset'	Amplitude offset
'repeats'	Number of repeats. 0 (default) means non-stop
'index'	Trigger index (0 to 4) is the per-channel trigger index (default is 0)
'trigger'	Trigger can be any of the following: 'instant' (default), 'dinrise', 'dinfall', 'spike', 'cmtcolor', 'softreset' 'instant' - immediate trigger (immediate analog output waveform) 'dinrise' - digital input rising edge 'dinfall' - digital input falling edge 'spike' - spike event on given input channel

	'cmtcolor' - trigger based on colored comment 'softreset'- trigger based on software reset (e.g. result of file recording)
'input'	Input depends on 'trigger' or 'monitor' If trigger is 'dinrise' or 'dinfall' - 'input' is bit number of 1 to 16 for first digital input If trigger is 'spike' - 'input' is an input channel with spike data If trigger is 'cmtcolor' - 'input' is the high word (two bytes) of the comment color If monitor is 'spike' - 'input' is an input channel with spike processing If monitor is 'continuous' - 'input' is an input channel with continuous data
'value'	Trigger value depends on 'trigger' If trigger is 'cmtcolor' then 'value' is the low word (two bytes) of the comment color If trigger is 'spike' then 'value' is spike unit number (0 for unclassified, 1-5 for first to fifth unit and 254 for any unit)
'mv'	No value required. If specified, voltages are considered in millivolts instead of raw integer value
'ms'	No value required. If specified, intervals are considered in milliseconds instead of samples

Outputs

None

Examples

```
% Output a ½ second -5V output followed by a 1 second +5V output,
followed by 0V for 2 seconds which it will repeat 5 times.
cbmex('analogout',  1,  'sequence',  [15000,-32767,30000,32767,60000,0],
'repeats', 5);

% Output a 10Hz, 5V amplitude sinusoidal signal
cbmex('analogout', 1, 'sinusoid', [10,32767]);

% Output a ½ second -5V output followed by a 1 second +5V output,
followed by 0V for 2 seconds, triggered by spike activity on channel 5
cbmex('analogout',  1,  'sequence',  [15000,-32767,30000,32767,60000,0],
'trigger','spike','input',5);
```

TRIALCOMMENT

[<comments_cell_array>, <timestamps_vector>, <rgba_vector>, <charset_vector>] = cbmex('trialcomment', <active = 0>, <key, value>)

Retrieve comments configured by 'trialconfig'

Inputs

active:	0 - (default) leaves buffer intact 1 - clears all the data and reset the trial time to the current time
<key, value>	
'instance'	Library instance, numbered 0 (default) to 3.

Outputs

comments_cell_array	Cell-array of comments (strings of possibly different sizes)
timestamps_vector	Timestamps of the comments
rgba_vector	Comment colors Common colors: black – 0 white – 16777215 red – 255 green – 65280 blue – 16711680 yellow – 65535 magenta – 16711935 cyan – 16776960
charset_vector	Character set vector for comments: 0 - ASCII 1 - UTF16

Examples

```
% Receive the comments, an equal number of timestamps, a color identity
for each comment, and the character set used for each comment and flush
the buffer.
[Comments, Timestamps, Colors, CharacterSet] = cbmex('trialcomment', 1)
```

TRIALTRACKING

[tracking_cell_array] = cbmex('trialtracking', <active = 0>, <key, value>)

Retrieve NeuroMotive tracking data configured by ‘*trialconfig’*

Inputs

active	0 - leave buffer intact 1 - clear all data in the buffer and reset the trial time to the current time.
<key, value>	
'instance'	Library instance, numbered 0 (default) to 3.

Outputs

tracking_cell_array:	Each row in this matrix contains: ['trackable_name' desc_vector timestamps_vector synch_timestamps_vector synch_frame_numbers_vector rb_cell_array]
Decription of output:	
desc_vector	column vector [type id max_point_count]
type	1 (2DMARKERS), 2 (2DBLOB), 3 (3DMARKERS), 4 (2DBOUNDARY)
id	node unique ID
max_point_count	maximum number of points for this trackable
timestamps_vector	the timestamps of the tracking packets
synch_timestamps_vector	synchronized timestamps of the tracking (in milliseconds)
synch_frame_numbers_vector	synchronized frame numbers of tracking
rb_cell_array	each cell is a matrix of rigid-body, the rows are points, columns are coordinates

Examples

```
%Pull current buffer tracking data and clear the buffer
[Tracking_Data] = cbmex(‘trialtracking’,1)
```

CCF

cbmex('ccf', filename, <key, value>);

Read, write and send CCF configuration file.

Inputs

filename		CCF filename to read. Read from NSP if it is zero length string (i.e. '')
<key, value>		
	'instance'	Library instance, numbered 0 (default) to 3.
	‘load’	Value is a filename string. Specifies the input filename for ‘convert’.
	'send'	Value is a filename string. Read and send CCF file to the NSP.
	'convert'	Value is a filename string. Read and Convert CCF file to a new CCF file in the latest format (must also specify ‘load’ to specify the input filename)
	‘save’	Value is a filename string. Saves the NSP’s current configuration as a new CCF.
	‘threaded’	No value needed. Specifies that a send command is to run in its own thread.

Outputs

None

Examples

```
% Send \ccf-files\mydefault.ccf to the NSP
cbmex('ccf', 'send', '\ccf-files\my-default.ccf');

% Send \ccf-files\mydefault.ccf to the NSP in its own thread
cbmex('ccf', 'send', '\ccf-files\my-default.ccf', 'threaded');

% Convert \ccf-files\old.ccf to \ccf-files\new.ccf
cbmex('ccf', 'load', '\ccf-files\old.ccf', 'convert', '\ccf-
files\new.ccf');

% Save the current settings to \ccf-files\save.ccf
cbmex('ccf', 'save', '\ccf-files\save.ccf');
```

SYSTEM

cbmex('system', command, <key, value>)

Perform given cbMEX system command.

Inputs

command	Can be any of the following: 'reset'- resets instrument 'shutdown' - shuts down instrument 'standby' - sends instrument to standby mode
<key, value>	
'instance'	Library instance, numbered 0 (default) to 3.

Outputs

None

Examples

```
% Shutdown the NSP
cbmex(‘system’, ‘shutdown’);

% Reset the NSP
cbmex(‘system’, ‘reset’);

% Reset the NSP
cbmex(‘system’, ‘standby);
```

Using cbMEX on OSX

cbMEX can function on OSX operating systems using files available for download from the Blackrock Microsystems Helpdesk. This .zip file contains the necessary files and libraries for cbmex to work on OSX. Instructions for use of these files can be found below.

On the OSX computer, configure the instrument port to the following address:
IP: 192.168.137.X (X can be any number between 1-16 not already on the network)
Subnet: 255.255.255.0

Inside the .zip file will be three files.

-cbmex.mexmaci64
-QtCore
-QtXml

cbmex.mexmaci64 is the mex file (cbmex) that will be interfacing with the Neural Signal Processor. QtCore and QtXml are MATLAB libraries that need to be placed in the correct file path.

From finder, navigate to the following folder:

/usr/lib

Place both QtCore and QtXml into this folder. They should not be in there initially, so there is no need to overwrite anything, but you will need administrator access (Note: MacOS 10.11+ has System Integrity Protection protecting the lib folder. SIP will need to be temporarily disabled for installation).

Once these are placed, make sure the /usr/lib folder are in the MATLAB search path and then attempt cbmex('open'). By default, MacOX allows much less memory in these spaces than Windows, so a memory error may be encountered. To address this, open the terminal and enter the following commands:

```
sudo sysctl -w kern.sysv.shmmax=33554432
sudo sysctl -w kern.sysv.shmall=4194304
```

Then, attempt to change max.sockbuf:

```
sudo sysctl -w kern.ipc.maxsockbuf=8388608
```

At this point, cbMEX should be ready to open. Enter the following command in MATLAB:

```
cbmex('open','receive-buffer-size',6291456,'inst-addr', '192.168.137.128',
'inst-port', 51001, 'central-addr', '255.255.255.255', 'central-port', 51002)
```

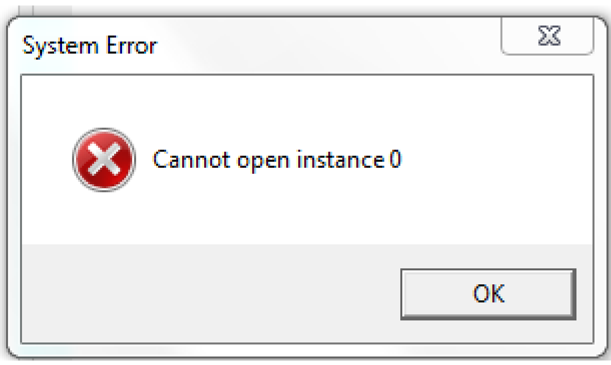
After you have opened the interface, all other commands can be used normally.

If there are any issues with version compatibility, please contact Blackrock Support.

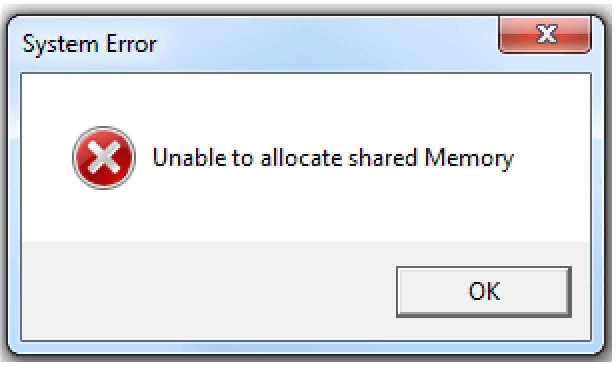
Troubleshooting

Unable to Allocate Shared Memory

Attempting to open Central after connecting cbMEX to the NSP through UDP results in two successive error messages:


System Error

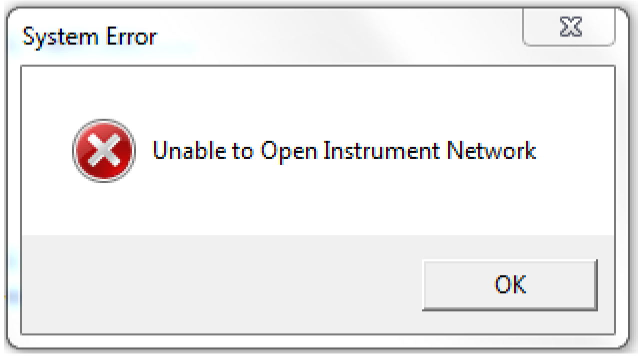
Cannot open instance 0


System Error

Unable to allocate shared Memory

The host PC is unable to open two separate connections through UDP and Central to the NSP simultaneously. Either connect solely through UDP without having Central running or close the UDP connection to the NSP, open Central, then run cbmex('open') to connect cbMEX through Central.

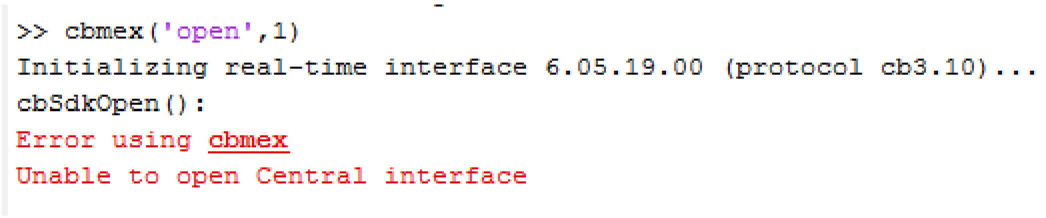
Unable to Open Instrument Network

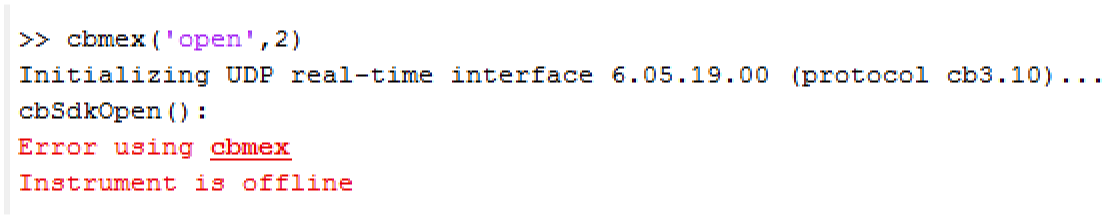

System Error

Unable to Open Instrument Network

After opening non-default cbMEX instances through UDP, the instance will persist and even after closing cbMEX, Central will give the following error message on attempts to open Central with the default instance. Closing MATLAB or executing clear all in the MATLAB command window will allow Central to open normally.

Other Connection Errors


```
>> cbmex('open',1)
Initializing real-time interface 6.05.19.00 (protocol cb3.10)...
cbSdkOpen():
Error using cbmex
Unable to open Central interface
```



```
>> cbmex('open',2)
Initializing UDP real-time interface 6.05.19.00 (protocol cb3.10)...
cbSdkOpen():
Error using cbmex
Instrument is offline
```

If you receive either of these errors, check that your firewall exceptions include Central.exe and MATLAB.

In addition, verify that the real-time interface version (displayed in the MATLAB command line output) and the NSP firmware version (shown on the NSP's display on startup) are compatible. If you have any questions about firmware compatibility or require an older Central/cbMEX version, please email support@blackrockmicro.com.

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

Example Scripts

Display Number of Channels with Spike Activity

```
% Author and Date:  Stephen Hou 21 March 2017
% Copyright:        Blackrock Microsystems
% Workfile:         NumSpikeChans.m
% Purpose:          Count/display number of channels with spike activity

close all;
clear variables;

cbmex('open')   %open library
cbmex('trialconfig',1)  %initialize data buffering

trial_length=10;        %length of time to run the script
snippet_length=0.5;     %length in seconds of data snippet to read during
                        %each read/flush buffer cycle
num_cycle=1;            %start with cycle number 1
current_time=tic;       %set current time

%while elapsed time is smaller than the trial length
while (trial_length>toc(current_time))
    pause(snippet_length);
    [event_data time cont]=cbmex('trialdata',1);      %read/clear data
    counter = 0;        %number of channels displaying spike activity

    %iterate through all channels; check for spike activity
    for i=1:length(event_data)
        if ~isempty(event_data{i,2})
            counter = counter+1;
        end
    end
    disp(sprintf('Cycle %d: Spike activity on %d channels',num_cycle,...
        counter));
    num_cycle = num_cycle+1;
end

cbmex('close');
```

Real-Time Spectrum Display

```
% Author and Date: Ehsan Azar  14 Sept 2009
% Copyright:       Blackrock Microsystems
% Workfile:        RealSpec.m
% Purpose: Realtime spectrum display. All sampled channels are displayed.

close all;
clear variables;

f_disp = 0:0.1:15;       % the range of frequency to show spectrum over.
% Use f_disp = [] if you want the entire spectrum

collect_time = 0.1; % collect samples for this time
display_period = 0.5; % display spectrum every this amount of time

cbmex('open'); % open library

proc_fig = figure; % main display
set(proc_fig, 'Name', 'Close this figure to stop');
xlabel('frequency (Hz)');
ylabel('magnitude (dB)');

cbmex('trialconfig', 1); % empty the buffer

t_disp0 = tic; % display time
t_col0  = tic; % collection time
bCollect = true; % do we need to collect
 % while the figure is open
while (ishandle(proc_fig))

    if (bCollect)
        et_col = toc(t_col0); % elapsed time of collection
        if (et_col >= collect_time)
            [spike_data, t_buf1, continuous_data] = cbmex('trialdata',1); %
read some data
            nGraphs = size(continuous_data,1); % number of graphs
            % if the figure is still open
            if (ishandle(proc_fig))
                % graph all
                for ii=1:nGraphs
                    fs0 = continuous_data{ii,2};
                    % get the ii'th channel data
                    data = continuous_data{ii,3};
                    % number of samples to run through fft
                    collect_size = min(size(data), collect_time * fs0);
                    x = data(1:collect_size);
                    %uncomment to see the full rang
                    if isempty(f_disp)
                        [psd, f] =
periodogram(double(x),[],'onesided',512,fs0);
                    else
                        [psd, f] = periodogram(double(x),[],f_disp,fs0);
                    end
                    subplot(nGraphs,1,ii,'Parent',proc_fig);
                    plot(f, 10*log10(psd), 'b');title(sprintf('fs = %d t =
%f', fs0, t_buf1));
                    xlabel('frequency (Hz)');ylabel('magnitude (dB)');
                end
                drawnow;
            end
            bCollect = false;
        end
    end

    et_disp = toc(t_disp0); % elapsed time since last display
    if (et_disp >= display_period)
        t_col0  = tic; % collection time
        t_disp0 = tic; % restart the period
        bCollect = true; % start collection
    end
end
cbmex('close'); % always close
```

Pulse DigOut on Specific Value from Serial

```
% Author & Date:     Hyrum L. Sessions   14 Sept 2009
% Copyright:         Blackrock Microsystems
% Workfile:          DigInOut.m
% Purpose:           Read serial data from the NSP and compare with a
%                    predefined value.  If it is the same, generate a
%                    pulse on dout4
%
% This script will read data from the NSP for a period of 30 seconds.  It
% is waiting for a character 'd' on the Serial I/O port of the NSP.  If
% received it will generate a 10ms pulse on Digital Output 4

% initialize
close all;
clear variables;

run_time = 30;       % run for time
value = 100;         % value to look for (100 = d)
channel_in = 152;    % serial port = channel 152, digital = 151
channel_out = 4;     % dout 1 = 1, 2 = 2, 3 = 3, 4 = 4

t_col = tic;         % collection time

cbmex('open');               % open library
cbmex('trialconfig',1);      % start library collecting data

start = tic();

while (run_time > toc(t_col))
    pause(0.05);     % check every 50ms
    t_test = toc(t_col);
    spike_data = cbmex('trialdata', 1);  % read data
    found = (value == spike_data{channel_in, 3});
    if (0 ~= sum(found))
        cbmex('digitalout', channel_out, 1);
        pause(0.01);
        cbmex('digitalout', channel_out, 0);
    end
end

% close the app
cbmex('close');
```