CBMEX
Matlab Extension

User's Manual

391 Chipeta Way, Suite G
Salt Lake City, UT 84108
866-806-3692
www.blackrockmicro.com
support@blackrockmicro.com

Description

The cbmex library provides an online MATLAB interface to connect to and control Blackrock Microsystems Neural Signal Processor (NSP). The library facilitates reading spike timestamps and continuous sample data as well as other data coming through the analog and digital inputs and the optional video tracking system (NeuroMotive). It can also start and stop file recording programmatically based on analysis of the data. Additionally, channel configuration, channel labels, comments, and digital outputs can be controlled programmatically through this interface.

The interface is initialized by using the *open* command. In order to begin buffering data from the NSP in cbmex, the *trialconfig* command is executed with the parameter set to 1. Buffered data can be read into Matlab and analyzed which can be used to control file recording and the digital output ports. User defined channels can be masked so that they are not buffered nor returned. The interface can be closed with the *close* command.

The cbmex interface can run on the same computer running Central or can run on a separate computer that is connected through the instrument network through a business-class network switch. Up to 16 computers can have access to the data streamed by the NSP.

Commands

OPEN

Description:

This command must be used before calling any of the other commands. It initializes cbmex, connects to the NSP or nPlay through Central or UDP, and prints out the current cbmex version, protocol version and NSP version numbers. The optional *interface* parameter below tells cbmex to access the data stream either through Central software or through the UDP data stream. If used on the same computer as Central, the Central interface must be used. 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.

Note: You can also specify a number of other optional parameters that allow your application to connect to multiple instruments at once (up to 4) using *instance*; the address and port of those instruments using *inst-addr* and *inst-port*; and the address and port of your application or Central (whichever you use to receive data) to which the instrument will send its data using *central-addr* and *central-port*.

Usage:

```
[connection instrument] = cbmex('open', [interface = 0], [<parameter>, [value]]...)
```

Parameters:

```
interface (optional): 0 (Default), 1 (Central), 2 (UDP)
<parameter>, value pairs are optional, some parameters do not have values and from left to right parameters will override previous ones or combine with them if possible.
<parameter>, value can be any of the following:
'instance', value: value is the library instance to open with instances numbered 0-3 (default is 0)
'inst-addr', value: value is a string containing the instrument's ipv4 address
'inst-port', value: value is the port number the instrument is using
'central-addr', value: value is a string containing the ipv4 address of where the instrument will send its data. This is useful when your instrument resides in a different network from your computer running Central or cbmex application.
'central-port', value: value is the port number your instrument is sending its data to
```

Return Value:

```
connection (optional): 1 (Central), 2 (UDP)
instrument (optional): 0 (NSP), 1 (Local nPlay), 2 (Local NSP), 3 (Remote nPlay)
```

Examples:

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

```
% Try to connect automatically, return what is used
[connection instrument] = cbmex('open')

% Only try Central
connection = cbmex('open', 1)

% Only try UDP
cbmex('open', 2)

% Try default, return assigned connection type
connection = cbmex('open')

% Open a new connection to a specific NSP and return the connection type
connection = cbmex('open', 2, 'instance', 1, 'inst-addr', '192.167.14.2',
'inst-port', 5001)
```

CLOSE

Description:

This command is used to close an instance of the library. You can have up to 4 unique instances open at once in an application using the optional *instance* parameter in the *open* command with *instance* defaulting to 0 when not specified. With *close* you can also specify an instance to close or have *instance* default to 0.

Usage:

```
cbmex('close', [<parameter>, [value]]...);
```

Parameters:

```
<parameter>, value pairs are optional, some parameters do not have values
and from left to right parameters will override previous ones or combine
with them if possible.
<parameter>, value can be any of the following:
'instance', value: value is the library instance to close with instances
numbered 0-3 (default is 0)
```

Return Value:

None

Examples:

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

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

TIME

Description:

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.

Warning: The timer starts over if Reset is pressed on Central and when recording starts

Usage:

```
Time = cbmex('time', [<parameter>, [value]]...);
```

Parameters:

```
<parameter>, value pairs are optional, some parameters do not have values
and from left to right parameters will override previous ones or combine
with them if possible.
<parameter>, value can be any of the following:
'instance', value: value is the library instance (instrument) from which
the time is reported with instances numbered 0-3 (default is 0)
'samples': if specified then the sample number is returned instead of the
time in seconds
```

Return Value:

```
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')
```

FILECONFIG

Description:

Starts or stops file recording to the filename specified including the comments provided. Filename contains the full path to the filename.

Note: If the file application (the app *File Storage* in Central) is not running, usually when first starting up the system, you will need to call *fileconfig* with *action* set to 0 to open the application. After this subsequent calls to *fileconfig* will start and stop the recording of a file depending on the value of *action*. This function requires connection to a running instance of Central.

Usage:

```
cbmex('fileconfig', filename, comments, action)
```

Parameters:

```
filename: file name string (255 character maximum)
comments: file comment string (255 character maximum)
action: set to 1 to start recording, 0 to stop recording
```

Return Value:

```
None
```

Examples:

```
% Start file storage app if not already started, stop recording otherwise
cbmex('fileconfig', 'c:\data\20120420', '', 0)
% Start 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)
```

DIGITALOUT

Description:

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. For more detailed information please see the Cerebus User's Manual.

Usage:

```
cbmex('digitalout', channel, value)
```

Parameters:

```
channel: 153 (dout1), 154 (dout2), 155 (dout3), 156 (dout4)
value: 1 sets dout to ttl high and 0 sets dout to ttl low
```

Return Value:

```
None
```

Examples:

```
% Sets Digital Out 1 to TTL high
cbmex('digitalout', 153, 1)
% Sets Digital Out 1 to TTL low
cbmex('digitalout', 153, 0)
```

TRIALCONFIG

Description:

Initialize the trial and set cbmex to start or stop buffering data.

Warning: The default behavior has changed from previous versions of cbmex. This version will return 16-bit values instead of double-precision values; also the timestamps are returned as 32-bit integers. You can specify *double* parameter to get the data types used by default in the previous versions.

Note: 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 and you will be unable to retrieve the flushed data. Additionally there is no method for determining if there is new data waiting to be collected. For further information please see the description for *trialdata*.

Usage:

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

Parameters:

```
active: set to 1 to flush data cache and start buffering data immediately,
set to 0 to stop collecting data immediately
config_vector_in (optional):
       vector [begchan begmask begval endchan endmask endval]
              begchan: specifies the first channel in the trial's range
              begmask: used in conjunction with begval to determine when
              a particular event occurs that can signal the start of a
              trial
              begval: used with begmask to determine in event data if a
              particular event has occurred which can be used to signal
              the start of a trial and buffering of trial data
              endchan: specifies the last channel in the trial's range
              endmask: used in conjunction with endval to determine when
              a particular event occurs that can signify the end of a
              trial
              endval: used with endmask to determine in event data if a
              particular event has occurred that would signal the end of
              a trial and halt buffering of trial/event data
  <parameter>, <value> pairs are optional, Some parameters do not have
values and from left-to-right parameters will override previous ones or
combine them if possible.
<parameter> can be any of:
'double': if specified, the data is in double precision format (old
default behaviour), this includes setting the timestamps to be represented
in a seconds format rather than a sample number format.
'absolute': if specified, event timing is absolute (setting 'active' to 1
will not reset time for events) and so time will not be relative to the
start if the trial
'noevent': if specified, an event data cache is not created nor configured
(same as 'event', 0)
```

'nocontinuous': if specified, a continuous data cache is not created nor configured (same as 'continuous', 0)
'continuous', value: set the number of continuous data points to be cached/buffered for each channel in the trial. This will default to 102400 sample points per channel if not set or 'nocontinuous' is not used.
'event', value: set the number of events to be cached/buffered in a trial. This will default to 2097152 events encompassing all channels if not set or 'noevent' is not used.

Return Value:

active_state: return 1 if data collection is active, 0 otherwise
config_vector_out: vector [begchan begmask begval endchan endmask endval double waveform continuous event comment tracking] 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)

Examples:

```
% Configure to get events
cbmex('trialconfig', 1)

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

% Configure to get double data and timestamps (in seconds)
cbmex('trialconfig', 1, 'double')

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

% Configure to buffer only continuous data
cbmex('trialconfig', 1, 'noevent')

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

TRIALDATA

Description:

Read the data in the buffer. The buffer contains data since collection started or the last time the data buffer was cleared/flushed. Data buffering starts only after the *trialconfig* command is executed with the *active* parameter set to 1. Calls to *trialdata* return all data since the start of collection or the last time the buffer was flushed, this includes both old data and new data added since the last call to *trialdata* (if the buffer was not flushed).
The buffer will default to 16384 events per channel. The continuous sample buffer will default to 102400 samples per channel (about 3 seconds). MATLAB may slow down if you let it get larger than that, so exercise caution when choosing the appropriate read delay. In addition, if trial is configured with the *double* parameter larger memory allocation is required and data transfer to MATLAB is slower.

Note: It is your responsibility to flush the data cache frequently, by calling

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

or

```
cbmex('trialdata', 1)
```

If the buffer fills up, no more event data will be added.

Note: Current syntax has changed from previous versions of cbmex so that if two output arguments are specified, the first output is the *time*, and the second one is *continuous_cell_array*. This new syntax is useful when only continuous data is wanted (or buffered).

Usage:

```
[timestamps_cell_array, time, continuous_cell_array] = cbmex('trialdata', [active = 0])
```

or

```
[time, continuous_cell_array] = cbmex('trialdata',  [active = 0])
```

or

```
[timestamps_cell_array] = cbmex('trialdata',  [active = 0])
```

Parameters:

```
active (optional): set to 0 (default) to leave buffer intact (no clearing the buffer)
                   set to 1 to clear all the data and reset its recording time to the current time
```

Return Value:

```
timestamps_cell_array: Timestamps for events of 152 channels consisting of 128 front end amp channels, 16 analog input channels, 4 analog output, 2 audio output, digital input, and serial input. 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: Continuous sample data, variable number of  rows.
Each row in this matrix contains:
[channel number] [sample rate (in samples / s)] [values_vector]
```

Note: If this call to *trialdata* cleared the buffer (*active* set to 1), you get the time at which the previous buffer started. The time is set for the next call to *trialdata*.

Note: 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 'line up' both because of the data reset and the different sampling rates.

Note: By default any *timestamps_vector* is made up of unsigned 32bit integers (UINT32) representing the sample number of a data point sampled by the NSP at 30kHz. In order to convert integer timestamps to seconds, they should be divided to 30000. In addition, if the trial is configured with the *double* parameter then timestamps will be returned as seconds since the beginning of the trial or the last time *trialdata* was called with *active* set to 1.

Note: By default every continuous data *values_vector* is made up of signed 16bit integers (INT16), and any digital *values_vector* is unsigned 16bit integers (UINT16). Some MATLAB functions cannot handle integer data types or may need the fixed point toolbox. MATLAB *double* function can be used to convert returned data to double format (as shown in the example script). Alternatively *trialconfig* command can be configured with *double* parameter.

Examples:

```
% read some event data, do not reset time
event_data = cbmex('trialdata');

% read some event data, reset time for the next trialdata
event_data = cbmex('trialdata', 1);

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

% read some continuous data, then reset time
[t, continuous_data] = cbmex('trialdata', 1);
```

CHANLABEL

Description:

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

Usage:

```
[label_cell_array] = cbmex('chanlabel', [channels_vector],
[new_label_cell_array])
```

Parameters:

```
channels_vector (optional): a vector of all the channel numbers, if not
specified means all the 156 channels
new_label_cell_array (optional): cell array of new labels (each a string
of maximum 16 characters) for each channel in channels_vector
```

Note: *new_label_cell_array* must be of the same length as *channels_vector*.

Return Value:

```
label_cell_array (optional): Each row in this matrix looks like:
  For spike channels:
  'channel label' [spike_enabled] [unit1_valid] [unit2_valid]
[unit3_valid] [unit4_valid] [unit5_valid]
  For digital input channels:
  '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
cbmex('chanlabel', 5, 'ch5')

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

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

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

% Set labels for channels 5 and 6, and return previous labels
chan_labels = cbmex('chanlabel', [5 6], {'chan5' 'chan6'})
```

MASK

Description:

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

Usage:

```
cbmex('mask', channel, [active = 1])
```

Parameters:

```
channel: the channel number to mask
  channel 0 means all channels
active (optional): set to 1 (default) to activate, 0 to deactivate
```

Return Value:

```
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

Description:

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.

Usage:

```
cbmex('comment', rgba, [charset = 0], [comment])
```

Parameters:

```
rgba: color coding or custom event number
charset (optional): character-set 0 (default) for ASCII or 1 for UTF16
comment (optional): comment string (maximum 127 characters)
```

Return Value:

```
None
```

Examples:

```
% Add black ASCII comment
cbmex('comment', 0, 0, 'my comment')

% Add colored ASCII comment
cbmex('comment', 255, 0, 'my comment')

% Add custom event number 1
cbmex('comment', 1)

% Add custom event 1 with character set UTF16
cbmex('comment', 1, 1)

% Add comment with color
cbmex('comment', 2, 'my comment')

% Add colored ASCII comment
cbmex('comment', 255, 'my comment')
```

CONFIG

Description:

Get channel configuration and optionally set new channel configuration for a given channel, only the specified parameters are changed. Parameter values are in raw format.

Note: Although the firmware rejects most of the invalid parameters, setting a wrong configuration for a wrong channel might cause unpredicted behavior. It is strongly recommended to leave the parameters that are not relevant unaffected.

Note: Please take extra care while changing a parameter during recording, for example if a new channel is added to a sampling group the data file becomes corrupted because the channels will no longer be aligned.

Note: Some parameters (such as threshold) are only recorded once at beginning of the file, changing these parameters might not be visible to data playback tools. Custom events or comments may be used to record such changes and customize the data playback tool.

Usage:

```
[config_cell_aray] = cbmex('config', channel, [<parameter>, value], ...)
```

Parameters:

```
channel: The channel number
<parameter>, <value> pairs are optional . Each parameter must have a
value.
<parameter> can be any of:
'userflags': if specified, the channel stores user information about the
state of the channel. Note: this is for future expansion.
'doutopts': this is a 32-bit value composed of flags setting or indicating
various digital output options listed as follows:
                'dinpopts', 'aoutopts', 'ainpopts', 'smpfilter',
'smpgroup', 'spkfilter', 'spkopts', 'spkthrlevel', 'spkgroup',
'amplrejpos', 'amplrejneg', 'refelecchan'
```

Return Value:

```
config_cell_aray (optional): Previous parameters. Each row in this matrix
looks like:
<parameter> [value]
```

Note: New parameters may be added in the future, therefore *config_cell_array* might have different number of rows.

Examples:

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

% Set threshold of the channel 5 to -125uV
cbmex('config', 5, 'spkthrlevel', -125*4)
```

```
% Get full configuration of channel 6, and set its new threshold to -65uV
config = cbmex('config', 6, 'spkthrlevel', -65*4)
```

Example Scripts:

REALTIME SPECTRUM DISPLAY

Example 1:

```
% 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 = 156;  % dout 1 = 153, 2 = 154, 3 = 155, 4 = 156

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');
```