cbMEX

Matlab Extension
User's Manual

Blackrock Microsystems®, LLC

630 Komas Drive • Suite 200
Salt Lake City UT • 84108

T: +1 • 801 • 582 • 5533
www.blackrockmicro.com
support@blackrockmicro.com

1.0 Table of Contents

2.0 Introduction

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.

3.0 Conventions

< >	is used to denote optional parameters.
<key, value>	pairs are optional, some pairs do not require values and from left to right parameters will override previous ones or combine with them if possible.

4.0 Commands

4.1 OPEN

Description

This command must be used before calling any of the other commands. It initializes cbMEX, connects to the system 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.

A number of other optional parameters can be used to allow the 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 to which the instrument will send its data using central-addr and central-port.

Format

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

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 instruments ipv4 address. The default is 192.168.137.128.
'inst-port'	Control port number. The Default is 51001.
'central-addr'	String containing the ipv4 address of. The default is 192.168.137.1 through 16.
'central-port'	Broadcast port number. The 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. 3 - Connected to nPlay running on a separate PC (Broadcasting).

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);
```

4.2 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.

Format

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

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);
```

4.3 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.

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

Format

time = cbmex('time', <key, value>);

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');
```

4.4 FILECONFIG

Description

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

Format

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

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.

Outputs

None

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);
```

4.5 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.

Format

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

Inputs

channel	153 (dout1) 154 (dout2) 155 (dout3) 156 (dout4)
value	1 sets dout to ttl high 0 sets dout to ttl low
<key, value>	
'instance'	Library instance, numbered 0 (default) to 3.

Outputs

None

Examples

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

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

4.6 TRIALCONFIG

Description

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 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*.

Note: 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.

Format

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

Inputs

active	1 flushes the data cache and starts buffering data 0 stops buffering data
config_vector_in	Vector [begchan begmask begval endchan endmask endval]. This vector can be used to configure inputs from the digital input or serial input port to begin a trial and to end a trial.
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*. 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'	*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 if 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'	*No value required*. 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'	*No value required*. 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.
'comment'	0 or 1 indicating disabled (0) or enabled (1).
'tracking'	0 or 1 indicating disabled (0) or enabled (1).

Outputs

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)
```

4.7 TRIALDATA

Description

Read the data in the buffer. The buffer contains data since the trial 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. If the buffer fills up, no more event data will be added.

It is your responsibility 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), you get 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 'line up' both because of the data reset and the different sampling rates.

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 by 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.

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.

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).

Format

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

Inputs

active 0 (default) leave buffer intact (don't clear the buffer)
1 to 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 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]

Examples

```
% read some event data, do not reset time
event_data = cbmex('trialdata', 0);

% 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);
```

4.8 CHANLABEL

Description

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

Format

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

Inputs

<channels_vector>	A vector of all the channel numbers to change. If not specified all 156 channels are to 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);
```

4.9 MASK

Description

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

Format

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

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)
```

4.10 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.

Format

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

Inputs

rgba	Color coding or custom event number. For colors, the common colors are: black – 0 white – 16777215 red – 255 green – 65280 blue – 16711680 yellow – 65535 magenta – 16711935 cyan – 16776960
charset	0 for ASCII 1 for 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, 'my comment');
```

4.11 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.

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.

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.

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.

Format

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

Inputs

channel	The channel number
<key, value>	
'instance'	Library instance, numbered 0 (default) to 3.
'userflags'	User supplied information about the channel.
'smpfilter'	
'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'

Outputs

config_cell_aray: Previous parameters. Each row in this matrix contains: <key> [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 +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
cbmex('config', 5, 'amplrejpos', 1000, 'amplrejneg', -1000)
```

4.12 ANALOGOUT

Description

Set properties for given analog output channel. The channel can monitor the continuous or spike stream of a channel or can generate a user defined waveform.

Format

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

Inputs

channel:	The channel number
<key, value>	
'instance'	Library instance, numbered 0 (default) to 3.
'userflags':	If specified, the channel stores user information about the channel: 145 (aout1), 146 (aout2), 147 (aout3), 148 (aout4), 149 (audout1), 150 (audout2)
'pulses':	Waveform is vector containing [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':	Type can be any of 'spike', 'continuous' 'spike' means spikes on 'input' channel are monitored 'continuous' means continuous 'input' channel is monitored
'track':	Monitor the last tracked channel
'disable':	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 'instant' (default), 'off', 'dinrise', 'dinfall', 'spike', 'cmtcolor', 'softreset' 'off' means this trigger is not used 'instant' means immediate trigger (immediate analog output waveform) 'dinrise' is for digital input rising edge, 'dinfall' is for digital input falling edge 'spike' is the spike event on given input channel 'cmtcolor' is the trigger based on colored comment 'softreset' is the trigger based on software reset (e.g. result of file recording)
'input':	Input depends on 'trigger' or 'monitor' 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 'cmtcolor' then 'input' is the high word (two bytes) of the comment color If monitor is 'spike' then 'input' is input channel with spike processing If monitor is 'continuous' then 'input' is 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':	If specified, voltages are considered in milli volts instead of raw integer value
'ms':	If specified, intervals are considered in milli seconds 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', 145, 'sequence', [15000,-32767,30000,32767,60000,0],
'repeats', 5);
```

4.13 TRIALCOMMENT

Description

Grab comments configured by 'trialconfig'.

Format

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

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 Comments colors
For colors, the common colors are:
black – 0
white – 16777215
red – 255
green – 65280
blue – 16711680
yellow – 65535
magenta – 16711935
cyan – 16776960
charset_vector Characterset vector for comments:
0 for ASCII
1 for 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)
```

4.14 TRIALTRACKING

Description

Obtain NeuroMotive tracking data configured by 'trialconfig'.

Format

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

Inputs

active: 0 (default) to leave 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

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
Here is the description 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)
```

4.15 CCF

Description

Read, write and send CCF configuration file.

Format

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

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');
```

4.16 SYSTEM

Description

Perform given cbMEX system command.

Format

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

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');
```

5.0 Example Scripts

5.1 REALTIME 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
```

5.2 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');
```