

Manufacturer  
630 Komas Drive | Suite 200  
Salt Lake City | UT 84108 | USA  
P +1 (801) 582-5533 | F +1 (801) 582-1509  
[www.blackrockneurotech.com](http://www.blackrockneurotech.com)



# CereStim API

## *Instructions for Use*

# Table of Contents

What This Manual Covers .....	4
System Requirements .....	4
Installation .....	5
<i>MATLAB API (Stimmex) Installation</i> .....	5
<i>C++ API Installation</i> .....	5
Use Cases .....	5
Getting Started .....	6
Conventions and Terminology .....	7
Commands .....	8
<i>cerestim96</i> .....	8
<i>scanForDevices</i> .....	9
<i>selectDevice</i> .....	10
<i>connect</i> .....	11
<i>isConnected</i> .....	12
<i>disconnect</i> .....	13
<i>libraryVersion</i> .....	14
<i>deviceInfo</i> .....	15
<i>getHardwareValues</i> .....	16
<i>getInterface</i> .....	17
<i>usbAddress</i> .....	18
<i>isSafetyDisabled</i> .....	19
<i>isLocked</i> .....	20

<i>getMinMaxAmplitude</i> .....	21
<i>testModules</i> .....	22
<i>testElectrodes</i> .....	23
<i>measureOutputVoltage</i> .....	24
<i>disableModule</i> .....	25
<i>enableModule</i> .....	26
<i>maxOutputVoltage</i> .....	27
<i>stimulusMaxValue</i> .....	28
<i>updateMap</i> .....	29
<i>setStimPattern</i> .....	30
<i>getStimPattern</i> .....	32
<i>disableStimulus</i> .....	33
<i>manualStim</i> .....	34
<i>beginSequence</i> .....	35
<i>endSequence</i> .....	36
<i>autoStim</i> .....	37
<i>wait</i> .....	38
<i>beginGroup</i> .....	39
<i>endGroup</i> .....	40
<i>play</i> .....	41
<i>trigger</i> .....	42
<i>disableTrigger</i> .....	43
<i>getSequenceStatus</i> .....	44
<i>groupStimulus</i> .....	45
<i>stop</i> .....	46

<i>pause</i> .....	47
Troubleshooting .....	48
Return Merchandise Authorization .....	50
Warranty .....	51
Support .....	51
<i>Complaints</i> .....	51

## What This Manual Covers

Inside this manual, you will find information on the CereStim API. This API is offered as a way to interface with the CereStim hardware using your own custom code so that it can be controlled programmatically within existing experimental designs or new custom applications. Since the Stim Manager software is built on top of this API, users can expect that every feature that the Graphical User Interface is capable of is also possible using the methods in this manual.

## 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
- USB 2.0 or 3.0 Port

# Installation

To use the CereStim API, it must first be downloaded from [www.blackrockneurotech.com](http://www.blackrockneurotech.com). The downloaded file will include both the C++ and Matlab versions of the API.

## *MATLAB API (Stimmex) Installation*

The MATLAB API requires one of the following files, depending on whether you are using a 32-bit or 64-bit MATLAB version, as well as the “cerestim96.m” file:

Stimmex.mexw32

Stimmex.mexw64

After placing the file, or the downloaded Stimmex API folder, in the location of your choice, add the file/folder to MATLAB’s search path to ensure that MATLAB can always find the file when called upon.

The exact method of adding to MATLAB’s search path may differ between versions of MATLAB – refer to your specific version’s documents.

## *C++ API Installation*

We suggest using Microsoft Visual Studio for working with the C++ API. While the steps will differ significantly depending on how you plan to integrate the application, it is suggested to choose Empty Project as the project type and then follow along with example source files downloaded from [www.blackrockmicro.com](http://www.blackrockmicro.com). The BStimAPI libraries should be loaded into your project as existing resources, and the “BStimulator.h” header as an existing header file.

# Use Cases

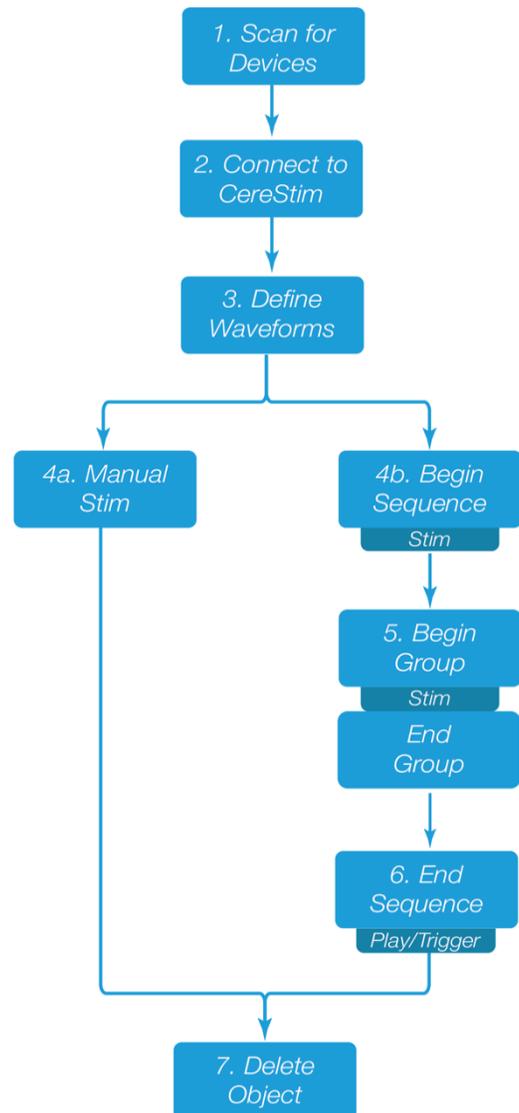
The CereStim API has the following typical use cases:

- Update stimulation waveforms or programs during the course of an experiment
- Integrate stimulation into an existing experimental control scheme
- Create custom applications that can control the CereStim hardware

# Getting Started

It is easiest to start with an example file downloaded from [www.blackrockneurotech.com](http://www.blackrockneurotech.com) and modify it to suit your needs. An understanding of the basic paradigm of the API is useful. A generic description of API usage is shown below without language-specific examples. This can be used as a guide for the order in which API commands are used in your own programs.

1. Every stimulation program always begins with opening connection to the stimulator(s). You must first scan for the available devices and select the one that you would like to connect to.
2. Connect to the available CereStim and make an object.
3. Every program or manual stimulus will require defining at least one waveform, also referred to as stim pattern. These get saved to the CereStim's internal memory to get used at the user's discretion.
4. At this point, you can either a) manually execute this Stim Pattern using the manualStim method, or b) you can create a more complex program for later execution. This is done using the beginSequence and endSequence methods.
5. Stimulation calls between the beginSequence and endSequence methods are executed in that order. If you want your saved waveforms to be delivered all at once, use the beginGroup and endGroup methods between the beginSequence and endSequence methods. Stimulation calls between the beginGroup and endGroup methods are executed simultaneously.
6. When you are ready to execute the program, you can use the play or trigger methods. The play method will execute your program, while the trigger method will enable trigger mode on the CereStim, causing the program to execute when the device receives a TTL pulse on its trigger input.
7. To properly end your stimulation program, always clear the stimulator object from memory using the API.



To learn more about the available methods, type 'help cerestim96' in MATLAB, or look through the header files in C++.

## Conventions and Terminology

<b>Stim Pattern</b>	A biphasic pulse repeated a specified number of times. Sometimes referred to as a stim “waveform.”
<b>CereStim</b>	This is the name of the CereStim object in the API workspace for this IFU. The name of this object is arbitrary and can be named to the user’s liking.
<b>Program</b>	A set of stim patterns executed in a specified order.
<b>&lt; &gt;</b>	Used to denote optional parameters.
<b>&lt;key, value&gt;</b>	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

## *cerestim96*

This command is needed to create a stimulator object in the workspace.

### USE

```
[CereStim_object] = cerestim96()
```

### INPUTS

None	None
------	------

### OUTPUTS

CereStim_object	A stimulator object to be used to control the stimulator through the API.
-----------------	---

### EXAMPLES

```
% Create a stimulator object named "cerestim." This naming  
% convention is used throughout the document  
cerestim = cerestim96();
```

## *scanForDevices*

The `scanForDevices` method scans USB ports to look for attached stimulators and returns a list of devices. It must be called before `selectDevice`, and both methods must be called before connecting to the CereStim.

### USE

```
[Serials] = cerestim.scanForDevices()
```

### INPUTS

None	None
------	------

### OUTPUTS

Serials	A list of serial numbers of stimulators plugged into the computer, or a single serial number if only one CereStim is attached
---------	---

### EXAMPLES

```
% Return list of connected serials  
SerialNumbers = cerestim.scanForDevices();
```

## *selectDevice*

Selects a stimulator, starting from 0 (the first device in DeviceList) and incrementing by 1 for each subsequent stimulator. This selected device is associated with the stimulator object and will be connected to when the connect method is executed.

### **USE**

```
cerestim.selectDevice(SerialNumber)
```

### **INPUTS**

Serial	The CereStim selected (integer indexing from 0 in the order displayed in DeviceList)
--------	--

### **OUTPUTS**

None	None
------	------

### **EXAMPLES**

#### ***MATLAB***

```
% Connect to the first stimulator in the array 'Serials'  
cerestim.selectDevice(Serials(1));
```

## *connect*

Connects to the selected CereStim stimulator in selectDevice. This function has multiple optional parameters that, generally, do not need to be accessed. These are primarily used for debugging purposes by Blackrock Support and Engineering, so these can be left to their default values.

### **USE**

cerestim.connect(<Interface>,<USBParams>)

### **INPUTS**

<Interface>            0: USB (Default)  
                          1: USB (Reserved)

<USBParams>            A three element array [pid timeout vid] where:  
                          pid = Product ID  
                          timeout = Time (ms) to connect before timeout  
                          vid = Vendor ID

### **OUTPUTS**

None                    None

### **EXAMPLES**

#### ***MATLAB***

```
% Connect through the default method  
cerestim.connect(0);
```

## *isConnected*

Tests whether the API is connected to a physical CereStim device. This can be used to ensure that there is no connection failures or disconnections that have occurred. Useful for ensuring that no commands are issued to the interface after a connection failure, to prevent program crashes.

### **USE**

Status = cerestim.isConnected()

### **INPUTS**

None                      None

### **OUTPUTS**

Status                      0: Not Connected  
                                  1: Connected

### **EXAMPLES**

```
% Check the connection status of the CereStim
status = cerestim.isConnected();
```

## *disconnect*

Disconnects the connected stimulator. This function should be called whenever you are finished using the stimulator and should be called before connecting to the stimulator. Failing to disconnect and then creating a new a stimulator object will prevent the new stimulator object from being able to connect to the stimulator.

### **USE**

```
cerestim.disconnect()
```

### **INPUTS**

None	None
------	------

### **OUTPUTS**

None	None
------	------

### **EXAMPLES**

```
% Disconnects the CereStim  
cerestim.disconnect();
```

## *libraryVersion*

Prints the current version of the API that is being used. This is useful for ensuring that libraries are up to date and knowing which version of the library the code is depending on. Since this device prints the libraryVersion, it does not require an output.

### **USE**

```
cerestim.libraryVersion()
```

### **INPUTS**

None	None
------	------

### **OUTPUTS**

None	None
------	------

### **EXAMPLES**

```
% Print the current library version  
cerestim.libraryVersion();
```

## *deviceInfo*

Returns information about the connected device. Will return a structure containing the serial number, firmware version of the motherboard, protocol version that the motherboard is using with the current modules, the status of the modules, and firmware version of the current modules.

### **USE**

```
DeviceInfo = cerestim.deviceInfo()
```

### **INPUTS**

None	None
------	------

### **OUTPUTS**

DeviceInfo	A struct containing the serial number, firmware version of the motherboard, protocol version that the motherboard is using with the current modules, module status, and current module firmware version
------------	---

### **EXAMPLES**

```
% Get information about the device  
DeviceInfo = cerestim.DeviceInfo();
```

## *getHardwareValues*

Reads the hardware values that are set in the stimulator. For safety and reliability reasons, values are stored in the CereStim.

### **USE**

Values = cerestim.getHardwareValues()

### **INPUTS**

None                      None

### **OUTPUTS**

Values                      A structure containing max phase amplitude (uA), max charge (pC), max interphase(uS), max compliance voltage (V), max frequency (Hz), minimum compliance voltage (V), minimum frequency (Hz), number of modules installed, and max phase width (uS)

### **EXAMPLES**

```
% Get information about current hardware and limits  
Values = cerestim.getHardwareValues();
```

## *getInterface*

Checks what interface type is being used for the connection to the CereStim96.  
Currently USB is the only available interface.

### **USE**

Type = cerestim.getInterface()

### **INPUTS**

None                      None

### **OUTPUTS**

Type                      0: Default  
                                 1: USB

### **EXAMPLES**

```
% Check which interface type the CereStim is using  
Type = cerestim.getInterface();
```

## *usbAddress*

Returns the USB address of the connected device.

### **USE**

Address = cerestim.usbAddress()

### **INPUTS**

None                      None

### **OUTPUTS**

Address                      The USB address that the stimulator is attached to.

### **EXAMPLES**

```
% Check which interface type the CereStim is using
Type = cerestim.usbAddress();
```

## *isSafetyDisabled*

Reports whether safety limits in the CereStim firmware are disabled. Safety limits can only be disabled by Blackrock personnel. If you have a device with disabled safety limits, please contact [support@blackrockneuro.com](mailto:support@blackrockneuro.com).

### **USE**

SafetyStatus = cerestim.isSafetyDisabled()

### **INPUTS**

None                      None

### **OUTPUTS**

SafetyStatus            0: Safety limits are enabled.  
                             1: Safety limits are disabled.

### **EXAMPLES**

```
% Check to see if the device has safety limits enabled  
SafetyStatus = cerestim.isSafetyDisabled();
```

## *isLocked*

Reports the lock status of the device. The device may be locked down for two reasons: if the number of detected current modules doesn't match the hardware configuration or if the hardware has not been configured. The first situation can occur if the device was programmed incorrectly or if a board in the device has become loose. The second situation can occur if the device was reprogrammed, but programming was not finished. If your device is locked down, please contact [support@blackrockneuro.com](mailto:support@blackrockneuro.com).

### **USE**

LockedStatus = cerestim.isLocked()

### **INPUTS**

None                      None

### **OUTPUTS**

LockedStatus            0: The device is not locked  
                             1: The device is locked

### **EXAMPLES**

```
% Check whether the CereStim hardware is locked
LockedStatus = cerestim.isLocked();
```

## *getMinMaxAmplitude*

Returns the minimum and maximum amplitudes allowed for stimulation.

### USE

[Minimum, Maximum] = cerestim.getMinMaxAmplitude()

### INPUTS

None                      None

### OUTPUTS

Minimum	The minimum amplitude allowed for stimulation
Maximum	The maximum amplitude allowed for stimulation

### EXAMPLES

```
% Check the allowed amplitude ranges for stimulation  
[MinAmp MaxAmp] = cerestim.getMinMaxAmplitude();
```

## *testModules*

Reports the status of each module and the voltage measured during a known stimulation. This serves as a diagnostic tool to identify bad output voltage levels of the stimulator. The voltage levels are taken during a biphasic waveform (config\_0): before stimulus, during amplitude 1, during interphase, during amplitude 2, and during interpulse over an internal test circuit.

### **USE**

```
ModuleStruct = cerestim.testModules()
```

### **INPUTS**

None                      None

### **OUTPUTS**

ModuleStruct            A struct containing the status, according to the chart below, as well as a 2D array with 5 voltage measurements, in millivolts, for each module:

0: Unavailable

1: Enabled

2: Disabled

3: Normal Voltage Levels

4: Voltage Levels Below Normal

### **EXAMPLES**

```
% Check module status  
ModuleStatus = cerestim.testModules();
```

## *testElectrodes*

Returns the impedance of each electrode at 1kHz. It also returns the voltage measured at five locations during a known stimulation. The voltage levels are taken during a biphasic waveform (config\_0): before stimulus, during amplitude 1, during interphase, during amplitude 2, and during interpulse.

### **USE**

```
ElectrodeStruct = cerestim.testElectrodes()
```

### **INPUTS**

None                      None

### **OUTPUTS**

ElectrodeStruct        A struct containing the estimated impedance values as well as a 2D array containing the five voltage measurements, in millivolts, for each electrode

### **EXAMPLES**

```
% Test electrode impedance  
ElectrodeStruct = cerestim.testElectrodes();
```

## *measureOutputVoltage*

Returns the voltage measured at five locations during a known stimulation. The voltage levels are taken during a biphasic waveform (config\_0): before stimulus, during amplitude 1, during interphase, during amplitude 2, and during interpulse.

### **USE**

Measurements = cerestim.measureOutputVoltage(Module, Electrode)

### **INPUTS**

Module	The current module (0-15) that should send the stimulation
Electrode	The electrode to which the stimulation should be sent (1-96)

### **OUTPUTS**

Measurements	A five member array of voltage measurements, in millivolts, at the five locations in the known stimulus waveform
--------------	--

### **EXAMPLES**

```
% Check voltages on a given electrode  
VoltageMeasures = cerestim.measureOutputVoltage();
```

## *disableModule*

Disables the selected modules for stimulation. This can be used by Blackrock Support for troubleshooting, or it may be used to limit the possible number of simultaneous stimulations.

### **USE**

cerestim.disableModule(ModuleList)

### **INPUTS**

ModuleList                    An array of module numbers (1-16) to be disabled

### **OUTPUTS**

None                            None

### **EXAMPLES**

```
% Disable the first, fourth, and sixteenth modules  
cerestim.disableModule([1 4 16]);
```

## *enableModule*

Enables the selected modules for stimulation.

### **USE**

cerestim.enableModule(ModuleList)

### **INPUTS**

ModuleList            An array of module numbers (1-16) to be enabled

### **OUTPUTS**

None                    None

### **EXAMPLES**

```
% Enable the first, fourth, and sixteenth modules  
cerestim.enableModule([1 4 16]);
```

## *maxOutputVoltage*

Limits the maximum output voltage that can be delivered during stimulation. Reads the specified maximum output voltage if no inputs are used.

### **USE**

<MaxOutput> = cerestim.maxOutputVoltage(<VoltageIndex>)

### **INPUTS**

VoltageIndex            Voltage level selection index, based on the following list:

7:	4.7 V
8:	5.3 V
9:	5.9 V
10:	6.5 V
11:	7.1 V
12:	7.7 V
13:	8.3 V
14:	8.9 V
15:	9.5 V

### **OUTPUTS**

<MaxOutput>            The currently set maximum output voltage in millivolts

### **EXAMPLES**

```
% Limit the maximum output voltage to 7.1V
cerestim.maxOutputVoltage(11);

% Check the maximum output voltage
MaxOutputVoltage = cerestim.maxOutputVoltage();
```

## *stimulusMaxValue*

Sets upper limits for stimulation parameters and reads current limits. If no inputs are provided, the current limits are read and returned as an output.

### USE

<LimitsStructure> =  
cerestim.stimulusMaxValue(<Voltage,Amplitude,PhaseCharge,Frequency>)

### INPUTS

<Voltage>	The maximum voltage, using the index in maxOutputVoltage, allowed during stimulation
<Amplitude>	The maximum current, in microamps, allowed during stimulation
<PhaseCharge>	The maximum charge, in picocoulombs, allowed per phase
<Frequency>	The maximum allowed stimulation frequency, in hertz

### OUTPUTS

<LimitsStructure> A structure containing the information described under inputs

### EXAMPLES

```
% Limit the maximum outputs to 9.5V, 9000 uA, 1000000  
% pC, and 1000 Hz  
cerestim.stimulusMaxValue(15,9000,1000000,1000);  
  
% Check the maximum outputs  
MaxOutputVoltage = cerestim.stimulusMaxValue();
```

## *updateMap*

Maps the connection of each channel to its corresponding electrode number.

### USE

```
cerestim.updateMap(BankA, BankB, BankC)
```

### INPUTS

BankA	Array of 32 elements where the index represents the channel (1-32), and the value at each position is the electrode number
BankB	Array of 32 elements where the index represents the channel (33-64), and the value at each position is the electrode number
BankC	Array of 32 elements where the index represents the channel (65-96), and the value at each position is the electrode number

### OUTPUTS

None	None
------	------

### EXAMPLES

```
% Odd and even channel electrode pairings on BankA
BankA = [2 1 4 3 6 5 8 7 10 9 12 11 14 13 16 15 18 17 20 19
22 21 24 23 26 25 28 27 30 29 32 31];
BankB = [33:64];
BankC = [65:96];
cerestim.updateMap(BankA, BankB, BankC);
```

## *setStimPattern*

Creates a custom biphasic stimulation waveform. Up to 15 waveforms can be defined, even if the CereStim has less current modules. Current modules limit the number of electrodes that can be stimulated simultaneously, not the number of defined waveforms.

### **USE**

`cerestim.setStimPattern(WaveformID, Polarity, Pulses, Amp1, Amp2, Width1, Width2, Interphase, Frequency)`

### **INPUTS**

WaveformID	The stimulation waveform (1-15) that is being configured
Polarity	The polarity of the first phase of the biphasic waveform: 0: Anodic 1: Cathodic
Pulses	The number of times to play the biphasic pulse (1-255)
Amp1	The amplitude of the first phase, in microamps. The values this can take depend on the stimulator model Microstimulator: 1-215 uA Macrostimulator: 100-10000 uA
Amp2	The amplitude of the second phase, in microamps.
Width1	The width of the first phase in microseconds (44-65535)
Width2	The width of the second phase in microseconds (44-65535)
Interphase	Period of time between the first and second phases in microseconds (53-65535)
Frequency	Rate at which biphasic pulses will be repeated in Hz (4-5000); could also be called interpulse, as it defines the tail end of the waveform

## OUTPUTS

None

None

## EXAMPLES

```
% Set waveform 3 to a waveform with an anodic first  
% phase, 100 repeating pulses, 45 uA amplitude, with  
% 100 us phases, a 55 us interphase, at 100 hz. Since  
% this is 100 pulses at 1000 hz, it will take  
% approximately 0.11 second  
cerestim.setStimPattern(3,0,100,45,45,100,100,55,1000)
```

## *getStimPattern*

Returns the configuration of a specific stimulation waveform.

### **USE**

WaveformStruct = cerestim.getStimPattern(Waveform ID)

### **INPUTS**

Waveform ID            The stimulation waveform to read (1-15)

### **OUTPUTS**

WaveformStruct        A structure containing the information specified in the input of setStimPattern

### **EXAMPLES**

```
% Obtain the parameters for stim pattern 4  
WaveformStruct = cerestim.getStimPattern(4)
```

## *disableStimulus*

Disables a stimulation waveform that was configured with setStimPattern.

### **USE**

cerestim.disableStimulus(Waveform ID)

### **INPUTS**

<Waveform ID>      The stimulation waveform that is being disabled (1-15)

### **OUTPUTS**

None                      None

### **EXAMPLES**

```
% Clear stimPattern 2  
cerestim.disableStimulus(2)
```

## *manualStim*

Sends a previously configured stim pattern to one electrode.

### **USE**

cerestim.manualStim(Electrode, Waveform ID)

### **INPUTS**

Electrode	The electrode that should be stimulated (1-96)
Waveform ID	The stimulation waveform to be used (1-15)

### **OUTPUTS**

None	None
------	------

### **EXAMPLES**

```
% Send stimulus 3 on channel 2  
cerestim.manualStim(2,3);
```

## *beginSequence*

Defines the beginning of a stimulation sequence (also known as a stimulation program). A stimulation sequence is a set of stimulation patterns to be executed. These can be setup to be done sequentially, simultaneously, or as a combination of both. Every stimulation program begins with the `beginSequence` command and is defined in the lines between the `beginSequence` and `endSequence` commands. Valid stimulation program lines are 'wait', 'autoStim', 'begofgroup', and 'endofgroup' commands. A stimulation script can have up to 128 commands in between its `beginSequence` and `endSequence` lines.

### **USE**

```
cerestim.beginSequence()
```

### **INPUTS**

None	None
------	------

### **OUTPUTS**

None	None
------	------

### **EXAMPLES**

```
% Small stim pattern
cerestim.beginSequence();
cerestim.autoStim(2,2);
cerestim.endSequence();
```

## *endSequence*

The end of a stimulation sequence (also known as a program). See `beginSequence` for more information about this pair of commands.

### **USE**

`cerestim.endSequence()`

### **INPUTS**

None                      None

### **OUTPUTS**

None                      None

### **EXAMPLES**

```
% Small stim pattern
cerestim.beginSequence();
cerestim.autoStim(2,2);
cerestim.endSequence();
```

## *autoStim*

Defines a stimulus to an electrode in a stimulation script.

### **USE**

cerestim.autoStim(Electrode, Waveform ID)

### **INPUTS**

Electrode	The electrode that should be stimulated (1-96).
Waveform ID	The stimulation waveform that should be used for stimulation (1-15)

### **OUTPUTS**

None	None
------	------

### **EXAMPLES**

```
% Small stim program
cerestim.beginSequence();
cerestim.autoStim(2,2);
cerestim.endSequence();
```

## *wait*

Tells the CereStim to wait a specified amount of time before executing the next command in a stimulation sequence (program). The maximum wait time is 65535 ms.

### **USE**

cerestim.wait(Milliseconds)

### **INPUTS**

Milliseconds                      The amount of time to wait (ms)

### **OUTPUTS**

None                                      None

### **EXAMPLES**

```
% Small stim program
cerestim.beginSequence();
cerestim.autoStim(2,2);
wait(10);
cerestim.autoStim(2,2);
cerestim.endSequence();
```

## *beginGroup*

Defines the beginning of a set of stimulations to occur simultaneously in a stim sequence (also called a stim program). Only autoStim commands may be used between beginGroup and endGroup commands. The number of simultaneous stimulations is determined by the number of modules installed and enabled in the stimulator.

### **USE**

cerestim.beginGroup()

### **INPUTS**

None                      None

### **OUTPUTS**

None                      None

### **EXAMPLES**

```
% Small stim program
cerestim.beginSequence();
cerestim.autoStim(3,3);
cerestim.beginGroup();
cerestim.autoStim(2,2);
cerestim.autoStim(3,3);
cerestim.endGroup();
cerestim.autoStim(4,4);
cerestim.endSequence();
```

## *endGroup*

Defines the end of a set of stimulations to occur simultaneously in a stim sequence (also known as stim program). Refer to the beginGroup command for more information.

### **Use**

cerestim.endGroup()

### **Inputs**

None                      None

### **Outputs**

None                      None

### **Examples**

```
% Small stim program
cerestim.beginSequence();
cerestim.autoStim(3,3);
cerestim.beginGroup();
cerestim.autoStim(2,2);
cerestim.autoStim(3,3);
cerestim.endGroup();
cerestim.autoStim(4,4);
cerestim.endSequence();
```

## *play*

Run a stimulation sequence (stim program) a specified number of times. Using this command requires having created a program using `beginSequence` and `endSequence` previous to this.

### **USE**

`cerestim.play(Repetitions)`

### **INPUTS**

Repetitions	The number of times to execute the stimulation sequence. Passing a zero will repeat the stimulation indefinitely until stopped
-------------	---

### **OUTPUTS**

None	None
------	------

### **EXAMPLES**

```
% Play a program 3 times  
cerestim.play(3);
```

## *trigger*

Sets the stimulator to trigger mode. When in trigger mode, the stimulator is waiting for signal on the hardware trigger port. When triggered, the CereStim will play the stim sequence that was previously defined using `beginSequence` and `endSequence` commands. Latency between edge and program start is approximately 3 microseconds.

### USE

`cerestim.trigger(Mode)`

### INPUTS

Mode	The type of event to trigger stimulation:
0:	Disable Trigger Mode
1:	Rising (low to high)
2:	Falling (high to low)
3:	Either rising or falling edges

### OUTPUTS

None	None
------	------

### EXAMPLES

```
% Wait for a rising edge TTL on the trigger port  
cerestim.trigger(1);
```

## *disableTrigger*

Takes the stimulator out of trigger mode. Refer to trigger for more information about this mode.

### **USE**

cerestim.disableTrigger()

### **INPUTS**

None                      None

### **OUTPUTS**

None                      None

### **EXAMPLES**

```
% Disable the waiting trigger  
cerestim.disableTrigger();
```

## *getSequenceStatus*

Reads the current state of the stimulator. The stimulator can only receive certain commands in certain states, so this function is useful for identifying which state it is in. Generally, the stimulator can only receive most commands while it is stopped, but some commands are made to interact with a given state (stop, for example, interacts with the playing state). This command is important in ensuring that commands are not issued to the CereStim when it is busy.

### **USE**

```
Status = cerestim.getSequenceStatus()
```

### **INPUTS**

None                      None

### **OUTPUTS**

Status                      The stimulator's status:

- 0:      Stopped
- 1:      Paused
- 2:      Playing
- 3:      Writing
- 4:      Waiting for Trigger

### **EXAMPLES**

```
% Check the current status of the stimulator  
Status = cerestim.getSequenceStatus();
```

## *groupStimulus*

Performs simultaneous stimulations on different electrodes with different waveforms. This command can be used in place of creating a stimulation sequence to improve latency times for creating and executing simple simultaneous stimulations.

### USE

cerestim.groupStimulus(BeginSeq, Play, Times, Number, Electrodes, Patterns)

### INPUTS

BeginSeq	Boolean expression to signal whether this is the beginning of a sequence
Play	Boolean expression to indicate whether to play the waveforms immediately or to wait for another command
Times	Number of times to play stimulation
Number	Number of stimuli that will occur simultaneously
Electrodes	Array (with length equal to number of modules) with each entry containing an electrode to be stimulated.
Patterns	Array (with length equal to number of modules) with each entry containing the Waveform ID to be used for stimulation on the corresponding electrode set in the previous parameter

### OUTPUTS

None                      None

### EXAMPLES

```
% Send a stimulus to play three times on channels 33-48  
% immediately with stim patterns 1 and 2  
cerestim.groupStimulus(0, 1, 3,16,[33 34 35 36 37 38 39 40  
41 42 43 44 45 46 47 48],[1 2 1 2 1 2 1 2 1 2 1 2 1 2 1  
2]);
```

Further examples of groupStimulus can be found on the Blackrock Support Knowledge Base, linked here: <https://support.blackrockneurotech.com/portal/en/kb/support>

## *stop*

Stops the currently running stimulation sequence (stim pattern) and resets it. When played again, it will begin from the first command. It can only be called while the stimulator has a status of 'stimulating' or 'paused'.

### **USE**

cerestim.stop()

### **INPUTS**

None                      None

### **OUTPUTS**

None                      None

### **EXAMPLES**

```
% Stop the stimulator from executing the current stim.  
Cerestim.stop();
```

## *pause*

Pauses the currently running stimulation sequence (program). Will begin at the next command in the sequence when the stimulator is set to play again.

### **USE**

cerestim.pause()

### **INPUTS**

None                      None

### **OUTPUTS**

None                      None

### **EXAMPLES**

```
% Pause the current stimulus, then continue again.  
cerestim.pause();  
wait(3);  
cerestim.play(1);
```

# Troubleshooting

Because the CereStim API relies on a programming language, such as MATLAB or C++, there are many issues that can be encountered that may be related to use of the language itself and not of the CereStim API; these types of errors, for the most part, are not described below.

Problem	Symptom	Failure	Potential fix
API cannot detect a CereStim	The scanForDevices Method Returns Empty	CereStim is not powered on, not connected by USB, or is already connected to by another object instance or by Stim Manager	Turn the CereStim off and back on again. Close Stim Manager if it is open and connected to the CereStim
API cannot execute a programmed sequence	You get an error stating there is a "Sequence Error"	Stimulator is busy with another sequence Sequence commands were programmed in the wrong order, like if an endSequence comes before its prerequisite beginSequence	See if there are other sequences running by calling getSequenceStatus(). If this does not return 0, you may need to wait or disable the trigger to get it to run again.  Change code to properly program your sequence, and rerun the script
Code stops mid script even though all arguments are valid	You get an error stating "Incorrect Number of Output Parameters"	Many functions in the CereStim API require output arguments. Failing to include a variable to receive these outputs will result in this error	Change code at point where it breaks to have API commands fill output variables, even if these output variables are never used later in your script
Desired number of stimuli are not generated in the timeframe you expect	You get an error stating "Insufficient Number of Modules"	Your system does not have enough functioning modules to create the desired stimulation  Your modules are malfunctioning	Contact Blackrock Support



# Return Merchandise Authorization

In the event of a returned material authorization (RMA) or complaint, please provide the product description, product number, lot number, person requesting the RMA or complaint and address, and the nature of the RMA and complaint.

In the unlikely event that your device needs to be returned to Blackrock for repair or maintenance, do not send any equipment back without a Return Merchandise Authorization Number (RMA). An RMA number will be issued to you by a Blackrock representative. If you need to obtain an RMA number, you may contact a product support representative at +1 (801) 582 5533 or by emailing [support@blackrockneuro.com](mailto:support@blackrockneuro.com).

Once an RMA number has been issued, it is important to safely pack the returned item for shipping back to Blackrock. It is preferred that you save the original boxes and packing materials that your system arrived in for return shipment. Please address the package as follows:

**Blackrock Microsystems, LLC**

ATTN: RMA#

630 S. Komas Dr., Suite 200

Salt Lake City, UT 84108 USA

Tel: +1 (801) 582-5533

# 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**

<https://blackrockneurotech.com/research/support/>

## *Complaints*

When filing a complaint, please provide the product description, product number, software version, lot number, complainant’s name and address, and the nature of the complaint.

## **Issues or Questions**

<https://blackrockneurotech.com/research/support/>

[support@blackrockneuro.com](mailto:support@blackrockneuro.com)

U.S.: +1 (801) 582-5533