



BOTTANGO DRIVER API

Documentation rev 8a

Api Version: 8

Driver Version: 0.7.0a

End User License Agreement	3
ALL USE OF BOTTANGO IS AT YOUR SOLE RISK.	3
Bottango Driver API	4
Create Your Own Bottango Microcontroller Code	4
The Bottango Driver API	4
Change Log	12

-1-

End User License Agreement

ALL USE OF BOTTANGO IS AT YOUR SOLE RISK.

BOTTANGO IS IN BETA TESTING AND MAY CONTAIN ERRORS, DESIGN FLAWS, BUGS, OR DEFECTS. BOTTANGO SHOULD NOT BE USED, ALONE OR IN PART, IN CONNECTION WITH ANY HAZARDOUS ENVIRONMENTS, SYSTEMS, OR APPLICATIONS; ANY SAFETY RESPONSE SYSTEMS; ANY SAFETY-CRITICAL HARDWARE OR APPLICATIONS; OR ANY APPLICATIONS WHERE THE FAILURE OR MALFUNCTION OF THE BETA SOFTWARE MAY REASONABLY AND FORESEEABLY LEAD TO PERSONAL INJURY, PHYSICAL DAMAGE, OR PROPERTY DAMAGE.

YOU MUST REVIEW AND AGREE TO THE BOTTANGO BETA SOFTWARE END USER LICENSE AGREEMENT BEFORE USING BOTTANGO: <http://www.Bottango.com/EULA>

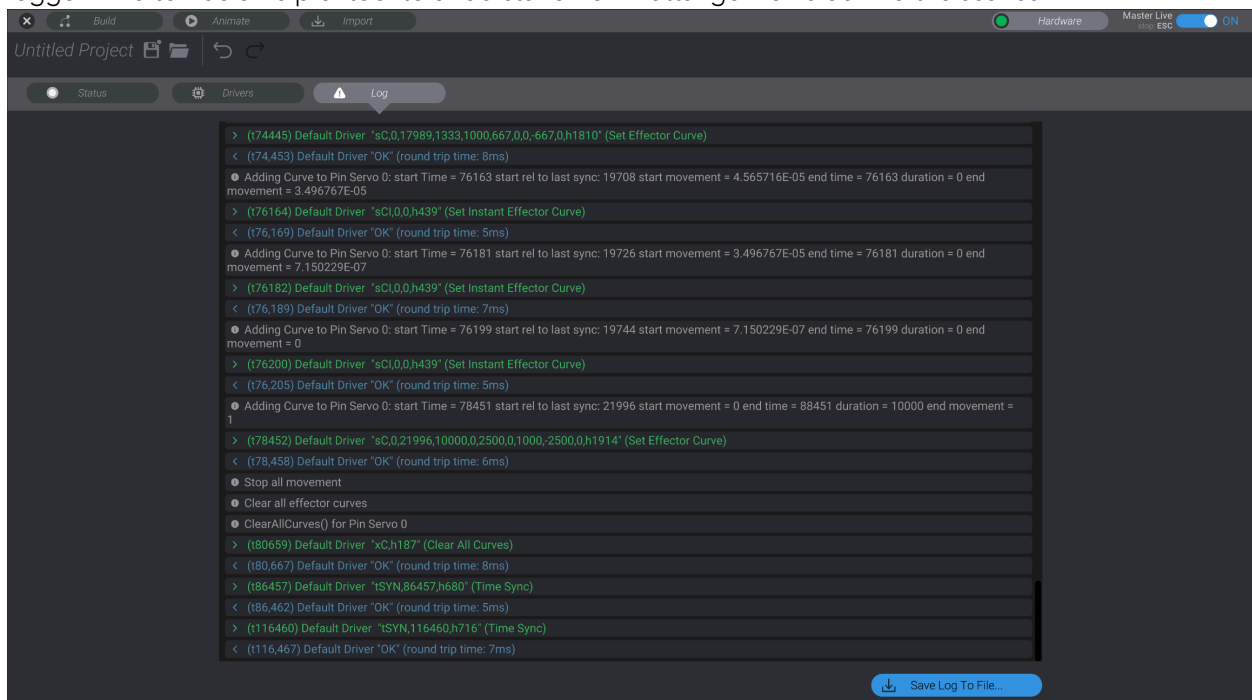
-2- Bottango Driver API

Create Your Own Bottango Microcontroller Code

Bottango provides the included open source Arduino compatible code so you can communicate with the Bottango application using an Arduino compatible microcontroller. Bottango as well provides an example Python implementation of the driver code as well for networked communication. If you were so inclined, you could implement your own microcontroller or networked code to respond to the Bottango driver API on any kind of microcontroller or network capable machine.

If you do so, please let us know! We'd love to see what you come up with!

You can monitor all incoming and outgoing serial commands in the Bottango application using the logger. This can be a helpful tool to understand how Bottango works behind the scenes.



The Bottango Driver API

Commands are terminated with a '\n' character. Parameters are separates with a ',' character.

Every command contains a hash code as the final parameter. This allows you to sanity check that the command string did not get errors introduced over the line. If you would like to use the hash code, see the method `bool checkHash(char *cmdString)` in the C++ file `src/BottangoCore.cpp` to see the hash

check logic. Hash parameters start with an 'h' character. As an example, you might see the command "xUC,4,h368\n"

The Python network driver ignores the hash value. Error rates in network communication are far lower than over serial, as network communication contains much more redundancy and error checking inherently.

As a reminder again, commands are terminated with a '\n' character. Parameters are separated with a ',' character.

After sending a command, Bottango will wait until it gets the "ready for next" signal before sending the next one.

- Outgoing Ready For Next <- \nOK\n

— BASIC LIFECYCLE —

The lifecycle of communication begins with a handshake request from the Bottango application, which should elicit a handshake response from the microcontroller.

- *Outgoing startup, on serial opened*

You should send this message immediately upon serial being opened, or the driver restarting, etc.

<- BOOT\n

- *Incoming Handshake Request after BOOT is received:*

Param 0: random code for verification

-> hRQ,144\n

- *Outgoing Handshake Response:*

Param 0: Current driver version. For the stock Bottango driver, this will be the most recent version of Bottango that had a change to the driver code. You can see the expected driver version in the desktop app when you select a hardware driver. Bottango will refuse to connect to a driver that does not report the correct driver version.

However, In order to facilitate custom driver development, you can override this behavior. The driver code will change frequently, the API of the communication less so. As such, the all caps key "CUSTOM{X}" will also be accepted as a driver version, where {X} (IE "CUSTOM5") is the version of the communication API instead of the version of the driver code.

Param 1: repeat back random code for verification

Param 2: bool (as int) true if accepting commands, false in saved animation playback mode

<- btngoHSK,0.6.2a,144,1\n or <-btngoHSK,CUSTOM5,144,1\n

<- \nOK\n

(Note that like all commands, the outgoing handshake response should be followed by an "\nOK\n" message to indicate that the driver is ready for the next command)

- *Incoming Time Sync*

(see `src/time.cpp` for how Bottango keeps time between the Application and the microcontroller in sync)

Param 0, current time in MS as tracked by the Bottango Desktop app

-> tSYN,32200\n

— START AND STOP —

- *Incoming STOP command. You should stop all movement, deregister all effectors, shutdown, etc.*

-> STOP\n

(Note that after stop, Bottango expects a boot and handshake before it will send commands to the driver again)

- *Incoming Deregister All Effectors Currently registered.*

-> xE\n

- *Incoming Deregister a specific effector .*

Param 0, identifier of effector

-> xUE,3\n

- *Incoming clear all curves currently playing on all registered effectors. This is sent at any time playing or to be played curves are invalidated or soon to be replaced.*

-> xC\n

- *Incoming clear all curves currently playing on a specific effector .*

-> xUC,3\n

— REGISTER EFFECTOR —

Note: There is no update an effector. If a change occurs, the effector is deregistered, and then a new replacement with updated data is registered.

- *Incoming register a servo with pin control*

Param 0, pin

Param 1, minimum PWM signal

Param 2, maximum PWM signal

Param 3, maximum PWM signal change per second

Param 4, starting PWM signal

-> rSVPin,3,800,1900,600,1500\n

- *Incoming register a servo with i2c control*

Param 0, i2c address

Param 1, pin

Param 2, minimum PWM signal

Param 3, maximum PWM signal

Param 4, maximum PWM signal change per second

Param 5, starting PWM signal

-> rSVPin,64,3,800,1900,600,1500\n

- *Incoming register a stepper with 4 pin control*

Param 0, pin 0

Param 1, pin 1

Param 2, pin 2

Param 3, pin 3

Param 4, maximum counter clockwise steps from home (will always be negative)

Param 5, maximum clockwise steps from home (will always be positive)

Param 6, starting steps offsets from home (negative or positive)

-> rSTPin,3,4,5,6,-500,500,0\n

- *Incoming register a stepper with step dir control*

Param 0, step Pin

Param 1, dir Pin

Param 2, bool (as int), true if clockwise on low, false if counterclockwise on low

Param 3, maximum counter clockwise steps from home (will always be negative)

Param 4, maximum clockwise steps from home (will always be positive)

Param 5, starting steps offsets from home (negative or positive)

-> rSTDir,3,4,1,-500,500,0\n

- *Incoming register a curved custom event*

Param 0, identifier

Param 1, max movement per second

Param 2, starting movement

Param 3, hardware pin (255 for none)

-> rECC,myEvent,0.25,0.5,5\n

- *Incoming register an on/off custom event*

Param 0, identifier

Param 1, bool as int, true if should start on.

Param 2, hardware pin (255 for none)

-> rEConOff,myEvent,5\n

- *Incoming register a trigger custom event*

Param 0, identifier

Param 1, hardware pin (255 for none)

Param 2, fire hardware pin HIGH (1) or LOW (0)

-> rECTrig,myEvent,5,0\n

- *Incoming register a color custom event*

Param 0, identifier

Param 1, integer between 0 - 255 for red channel of starting color.

Param 2, integer between 0 - 255 for green channel of starting color.

Param 3, integer between 0 - 255 for blue channel of starting color.

-> `rECColor,255,0,255\n`

- *Incoming register a custom motor*

Param 0, identifier

Param 1, minimum signal

Param 2, maximum signal

Param 3, maximum signal change per second

Param 4, starting signal

-> `rMTR,myMotor,-1000,1000,300,0\n`

- *Incoming register an audio event*

Param 0, identifier

Param 1, pin (if trigger) or index (if i2s).

Param 2, fire is high or low (only used for trigger. HIGH (1) LOW (0))

-> `rAud,myAud,5,0\n`

- *Incoming update the signal bounds on a registered effector*

Param 0, identifier

Param 1, new minimum signal for the effector

Param 2, new maximum signal for the effector

Param 3, new maximum signal change per second

-> `upE,6,1000,2025,500\n`

- *Incoming sync an effectors signal in order to home it*

Param 0, identifier

Param 1, amount of signal to sync by (can be negative or positive). Numbers between -100 and 100 are meant to be manual steps. 101 means auto sync clockwise, -101 means auto sync counter clockwise.

-> `sycM,myMotor,-100\n`

- *Outgoing inform Bottango app that auto sync is complete on an effector with the given identifier*

Param 0, identifier

<- `sycMDone,10\n`

— ANIMATION CURVES —

With the default configuration, Bottango attempts to cache curves on the microcontroller, rather than send them just in time, whenever possible. When a curve is 1 second or less away from being played, Bottango will send the curve to the microcontroller, unless at least 3 curves are currently cached and waiting, in which case it will send the most immediate three, and wait to send the next curve until Bottango predicts that a cached curve has been played and can be removed.

Bottango sends to the microcontroller animation curves, rather than sampled data. This takes the form of a starting and ending keyframe and duration.

Please see `src/BezierCurve.cpp` to see how the incoming curve is translated into animation and sampled over time.

- *Incoming Set Motor Curve*

Param 0, identifier

Param 1, time of start in MS relative to last sync time (could be positive or negative)

Param 2, duration of curve in MS

Param 3, start movement (expressed as an int between 0 - 8192)

Param 4, start control point tangent X (relative to start time in MS)

Param 5, start control point tangent Y (relative to start movement) (expressed as an int between 0 - 8192)

Param 6, end movement (expressed as an int between 0 - 8192)

Param 7, end control point tangent X (relative to start time in MS) (this will be a negative number or 0)

Param 8, end control point tangent Y (relative to start movement) (expressed as an int between 0 - 8192)

-> sC,3,2029,1500,250,0,500,750,-250,0\n

- *Incoming Set On Off Curve*

Param 0, identifier,

Param 1, time of start in MS relative to last sync time

Param 2, bool as int, true if on

-> sCO,myEvent,322,1\n

- *Incoming Set Trigger Curve*

Param 0, identifier,

Param 1, time of start in MS relative to last sync time

-> sCT,myEvent,322\n

- *Incoming Set Color Curve*

Param 0, identifier,

Param 1, time of start in MS relative to last sync time

Param 2, duration of curve in MS

Param 3, red channel of starting color of curve (int between 0 - 255)

Param 4, green channel of starting color of curve (int between 0 - 255)

Param 5, blue channel of starting color of curve (int between 0 - 255)

Param 6, red channel of ending color of curve (int between 0 - 255)

Param 7, green channel of ending color of curve (int between 0 - 255)

Param 8, blue channel of ending color of curve (int between 0 - 255)

-> sCC,myEvent,322,1000,255,0,0,0,0,255\n

!Note that color curves are interpolated linearly!

If the curve that Bottango would send has a duration at or very close to instant and should be executed now, an instant curve is sent instead. This optimizes the size of the serial stream, and allows for more responsive streaming of commands. Behind the scenes, the Bottango Arduino code turns these instant curves into zero length regular curves with a start time at the time the command is received. This is purely a serial size space saving command.

- *Incoming Set Instant Curve*

Param 0, identifier

Param 1, target movement (expressed as an int between 0 - 1000)

-> sCI,3,550\n

- *Incoming Set Instant Color Curve*

Param 0, identifier

Param 1, red channel of color to be instantly set (int between 0 - 255)

Param 2, green channel of color to be instantly set (int between 0 - 255)

Param 3, blue channel of color to be instantly set (int between 0 - 255)

-> sCCI,myEvent,255,0,0\n

— RELAY CONTROLLERS AND SYNCHRONIZED COMMANDS —

A currently in development set of features is relay controllers (which allow one driver to relay commands to another) and synchronized commands (which concatenate multiple commands into one single command). More detail will be provided on these commands as features continue in development.

- *Incoming Command to register a relay controller*

Param 0, identifier

Param 1, relay connection type, additional tokens connection type dependent

-> rCtrl,child\n

- *Incoming Command to deregister a relay controller*

Param 0, identifier

-> xUCtrl,child\n

- *Incoming Command to deregister all relay controllers*

-> xCtrl\n

- *Incoming Command to pass the command to a relay controller*

Param 0, identifier of relay controller, the rest of tokens are the command to be passed

Param 1, relay connection type, additional tokens connection type dependent

-> sR,[command to be passed]\n

- *Incoming Command flagged as a synchronized command that should be split from one string to multiple commands*

Each sub command in the string is delineated with a ; instead of a \n

-> sSY,[command to be split]\n

— APP CONTROL —

- *Outgoing EStop*

Requests stop on the Bottango app. The same as hitting escape in the desktop app, or turning off master.

`<- reqStop\n`

- *Outgoing Request Pause*

Requests stop playing the Bottango app. The same as hitting space bar or pause in the desktop app

`<- reqPause\n`

- *Outgoing Request Play*

Requests start playing in the Bottango app. The same as hitting space bar or pause in the desktop app

`<- reqPlay,2,100\n`

Param 0, index of the desired animation (Pass -1 as the index to use whatever the currently selected animation in the app is)

Param 1, start time to begin playing of the desired animation in milliseconds (Pass -1 as the start time to use whatever the current time in the app is)

As an example, reqPlay,2,100 would start playing animation 2 at 100 ms into the beginning of the animation. reqPlay,-1,-1, is the same as hitting spacebar while paused to start playing at the current point.

-3- Change Log

- API changes in 0.7.0a (**API VERSION 8**):
 - Added extra parameter to trigger events that controls if optional hardware pin should be high or low on fire.
 - Added register audio event command.
 - Added commands for relay controllers and synchronized commands.
- API changes in 0.6.3a (**API VERSION 7**):
 - Added extra parameter to curved, trigger, and on/off events to control pin (0-254 controls the pin, 255 is none)
 - Added outgoing commands to control the desktop app:
 - Estop
 - Request Play
 - Request Pause
- API changes in 0.6.3a (**API VERSION 6**):
 - Added incoming command upE to update the signal bounds on a registered effector.
- API changes in 0.6.2a (**API VERSION 5**):
 - CUSTOM{X} is now an acceptable driver version alternative in handshake, where x is the current api version.
- API changes in 0.6.1a (**API VERSION 4**):
 - Removed register i2c stepper command, as that functionality is depreciated.
 - Added STOP command. Previously xE was sent at stop, now a more explicit STOP command is sent.
 - Compressed movement in curves is sent as an int between 0 - 8192. Previously was between 0 - 1000. This value is now adjustable as well in the desktop app.
 - Bottango requires an "OK" response before sending more commands after receiving a valid handshake response.
 - Sync motor command now can get -101 and 101 for the sync value. These mean auto sync counter clockwise and auto sync clockwise respectively.
 - New outgoing command "sycMDone" informs Bottango that a motor is done auto syncing.
 - Documentation fix: previously time sync was documented to say that time parameter was time since last sync. This was an error. Time sync parameter is current time on the desktop app in MS.