BOTTANGO NETWORK DRIVER DOCUMENTATION

Documentation rev 1

-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.

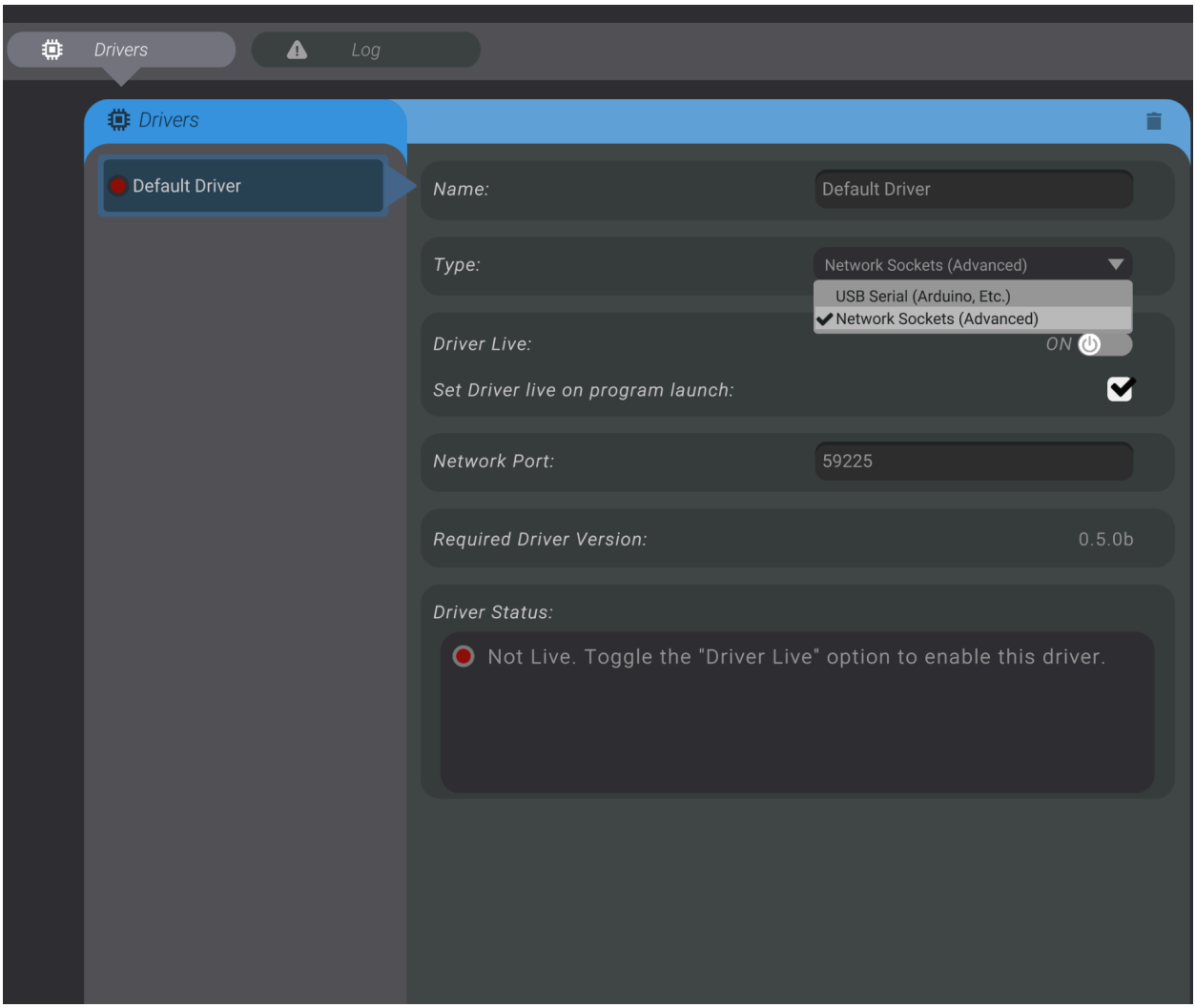
YOU MUST REVIEW AND AGREE TO THE BOTTANGO BETA SOFTWARE END USER LICENSE AGREEMENT BEFORE USING BOTTANGO: http://www.Bottango.com/EULA

-2-
Using the Bottango Networked Driver

What is the Bottango Networked Driver?

The Bottango Networked Driver is an advanced tool for users comfortable writing their own code, and looking for flexibility beyond what is offered by USB serial connections or saving animations to baked code. It is made of two parts:

First, in Bottango you can set a hardware driver to operate as a sockets based networked server, instead of communicating via USB serial.



Second, there is example code, written in Python, of a socket based client that can connect to the server and respond to commands from Bottango. You are not limited to using Python, that is just the fully functional example code.

When is the Bottango Networked Driver the right choice for me?

If your desired use case is more advanced than what can be done over USB serial or exported to code, then the most flexible alternative is the Bottango Networked Driver. Some example use cases where the Bottango Networked Driver makes sense:

- I want to communicate from the Bottango app to my own code running on a computer.
- I want to control a motor that requires a connection to a laptop/desktop computer instead of a microcontroller.
- I want to control a robot with Bottango wirelessly, and I'm able to use a Python compatible microcontroller, or can rewrite the C++ code to create and maintain a socket connection for streamed data (providing a networked variant of the C++ code is on the Bottango roadmap, but not yet delivered)

Starting a Bottango Network Server

In Bottango, as shown in the above image, you can set a hardware driver to operate as a sockets based networked server, instead of communicating via USB serial.

When you do so, and set that driver live, you will be creating a sockets based server on your network, at the port you indicated. That server will listen for incoming connections, and stream commands to the connected clients for you.

That server is located at the ip address of the computer running the Bottango application. Connecting to that server with a client on a local network should use the local network address of the computer, and the port you entered into Bottango (the default port is 59225). In order to communicate over the public facing internet, you would need to have the client connect to the public IP address of the computer, and you would be responsible for opening/forwarding the port as needed by your local network configuration.

Running a Network Client

The client that connects to the created server can be anything that can open and communicate in network connection via sockets. The API that communicates commands back and forth in the C++ USB serial microcontroller code and over the network is identical, and fully documented in the next chapter. You are welcome to build your own network client if needed, using the API documentation provided.

However, Bottango also provides a fully functional example implementation of a client that can connect to the network server in Python 3. If you're trying to just get up and going fast, the example Python code is fully functional and will be supported for future features.

Modifying the example Python Code

The most important file, if you want to quickly modify the Python code to meet your needs is the "CallbacksAndConfiguration.py" file, located in the following directory in the Bottango installation .zip downloaded from the Bottango website: Bottango/AdvancedFeatures/Driver_Networked/src/CallbacksAndConfiguration.py

In this file you can set the address and port that you want to connect to. As well, there are callbacks for registering, deregistering, and setting signal on effectors, with documentation of what parameters are passed to those callbacks.

Make sure to set these configuration fields to match your server's setup:

```
address = '127.0.0.1'          # The server's hostname or IP address
port = 59225                   # The port used by the server
```

These other configuration fields are less likely to be needed:

```
log = True                     # enable logging
roundSignalToInt = True        # treat signal as an int (true) or as a float (false)
apiVersion = "0.5.0b"          # api version to send in handshake response
```

Add your custom code as needed to the following lifecycle callbacks:

This callback is called whenever an effector is registered
handleEffectorRegistered(effectorType, identifier, minSignal, maxSignal, startingSignal):

This callback is called whenever an effector is deregistered
handleEffectorDeregistered(identifier):

This callback is called whenever an effector is changes its target signal for any reason
handleEffectorSetSignal(effectorType, identifier, signal):

This callback is called whenever an on/off custom event changes its target on/off state for any reason
handleEffectorSetOnOff(effectorType, identifier, on):

This callback is called whenever an trigger custom event fires for any reason
handleEffectorSetTrigger(effectorType, identifier):

Error handling and connection lifecycle

The Bottango desktop app expects a standard implementation of a socket connection client, and uses the common signals of a network connection to determine the connection state between the server and client.

Gracefully exiting the socket connection when the client disconnects itself is encouraged. If the server reads from the socket connection and receives an empty string, it will interpret that as a closed connection per socket connection standards.

There is no “heartbeat” signal between the server and client. The closest to that currently is the time synchronization command every 30 seconds. If your use case requires a more frequent or intentional heartbeat signal, reach out to Bottango devs! We can probably help!