



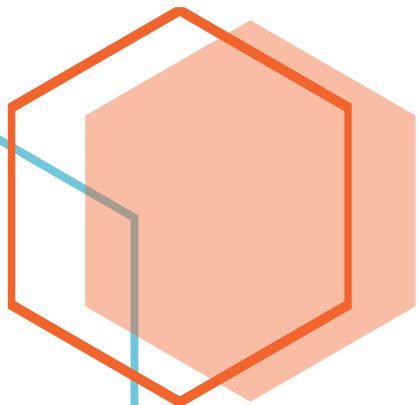
# Bonsai README

---

Goncalo Lopes | Sahil Suresh

Covers: [Sippy Lab GitHub Repository](#)

As of 11 November 2020



# Bonsai Overview

## Installation

### Install from Setup Cmd (Recommended)

Download the Zip file from the GitHub repository. Unzip the folder and find the Bonsai folder inside. There should be a Windows Command Script named "setup"; run this script which will download and install a local Bonsai with all the necessary packages already installed. In that Bonsai folder, you should now see a Bonsai executable file. Run Bonsai and open your .Bonsai project.

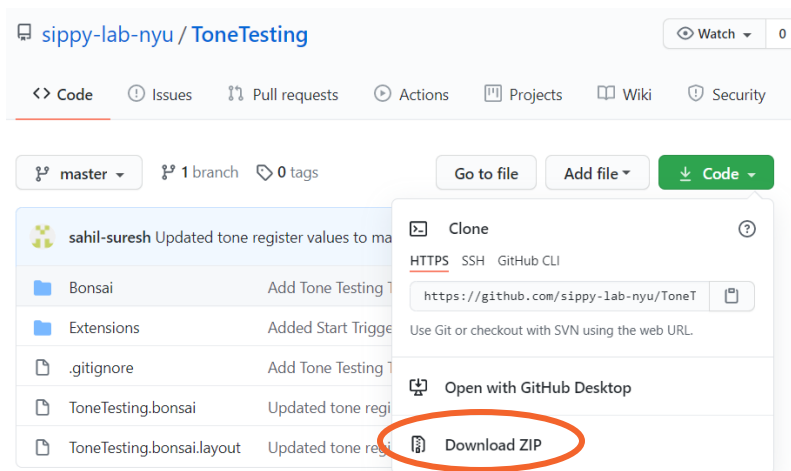
### Install from Site

#### [Installation Page](#)

Requires .NET Framework for installation (Windows only). Download scripts Zip file from the GitHub repository. Run Bonsai and select manage packages. Install the following packages:

- Bonsai Audio Library
- Bonsai Champalimaud Hardware Library
- Bonsai Design Library
- Bonsai DAQmx Library
- Bonsai Harp Library
- Bonsai Harp CF Library
- Bonsai Visualizers Library
- Bonsai Numerics Library
- Bonsai Scripting Library
- Bonsai Spinnaker Library
- Bonsai Starter Pack

Once all packages are installed, open the .bonsai file under Open Projects. Note that extensions must be on path for Bonsai to be able to run them.



**Download full ZIP file from repository**

## Troubleshooting

When extensions are not properly loaded, they will appear as a red node. When the red node is selected, Bonsai will identify the error in the bottom left of the workflow window. When packages are missing, the nodes will appear grayed out.

Check the Bonsai command prompt when errors occur for more information. The most common error of extensions or nodes not functioning properly is usually due to missing packages or extensions not being located on path.

The extensions can be drag and dropped into the appropriate Extensions folder within the unzipped workflow folder. The path can also be set within the Bonsai workflow window in the taskbar next to the Start button

# Bonsai Overview

The screenshot shows the Bonsai software interface with several annotations:

- Manage packages under Tools menu option**: Points to the **Tools** menu in the top menu bar.
- Set directory/path**: Points to the file path in the address bar: `C:\Users\sahil\Desktop\ModifiedGoNoGo`.
- Properly functioning extensions will appear yellow**: Points to the **TwoToneExperiment** node in the workflow area, which has a yellow icon.
- Node description**: Points to the description of the **CameraVisualizer (GroupWorkflow)** node in the Properties panel.
- Node properties**: Points to the **Name** property of the **CameraVisualizer** node in the Properties panel.
- Property description**: Points to the description of the **Name** property in the Properties panel.
- Errors will be reported here**: Points to the status bar at the bottom left, which shows **Ready**.

The interface includes a **Toolbox** on the left with categories like **Source**, **Transform**, **Sink**, **Combinator**, **Workflow**, and **Subject**. The **Workflow** area in the center displays a sequence of nodes: **TwoToneExperiment**, **TrialStatistics**, **Joystick Visualizer**, **Response Statistics**, and **Camera Visualizer**. The **Properties** panel on the right shows details for the selected **CameraVisualizer** node, including its description and properties like **Name** (set to **CameraVisualizer**) and **Misc** (set to **VisualizerFramerate: 20**).

# Harp Dependencies

## Installation (in order)

### Harp Software

#### [Soundcard/Drivers](#)

#### [Harp Download Repository](#)

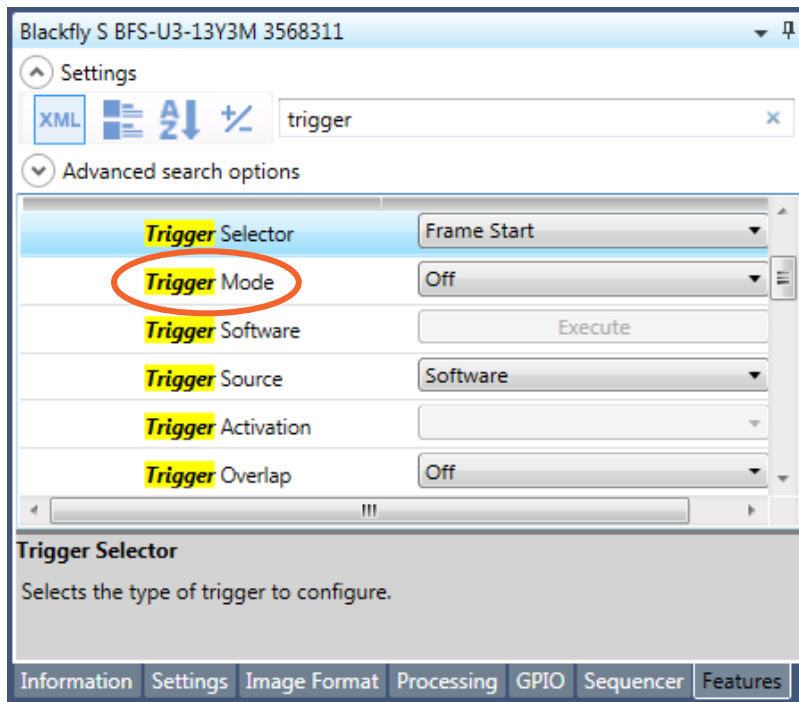
After downloading the Soundcard and drivers from the first link, download Harp Behavior (latest version) zip file from the repository. Install Harp Behavior and then download and install Harp Convert to CSV (latest version) as well. This program will help in converting any binary files that Bonsai can output into a CSV format than can then be used in MATLAB or Python.

### FLIR Camera Setup

#### [Spinnaker Download](#)

#### [Blackfly USB3 Documentation](#)

From the above repository, download and install Spinnaker version 1.29.0.5. This will be named SpinnakerSDK\_FULL\_1.29.0.5 followed by your system specifications. Run SpinView and you should be able to see your camera under the Devices pane.

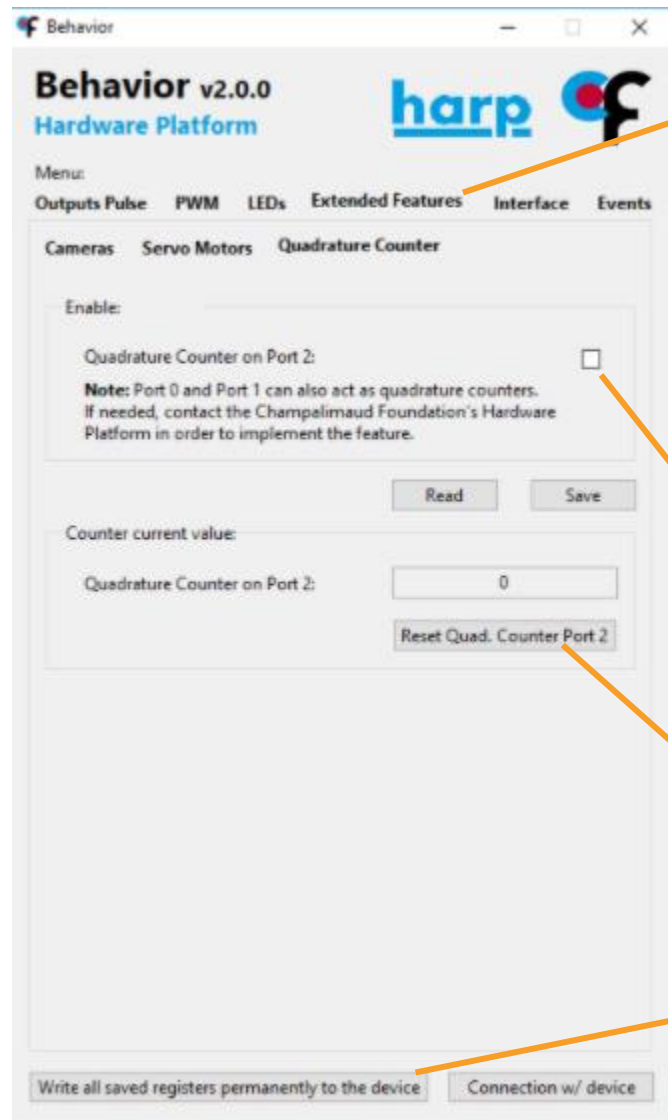


Turning trigger mode on in SpinView

## Troubleshooting

- ❖ Note that the Soundcard's drivers may not be compatible with all computer builds. Within the Zadig window, the instructions mention to **Install WCID Driver** however you may need to select **Install Driver** instead if the Soundcard still does not work.
- ❖ In order to access the **BitBucket** site you may need to create an account.
- ❖ **.NET framework 3.5** is required for the Soundcard driver/software to work.
- ❖ In order to run the camera from Bonsai, make sure that **trigger mode** is turned on in SpinView.

# Harp Dependencies



The screenshot shows the 'Behavior v2.0.0 Hardware Platform' window. The 'Extended Features' tab is selected in the top menu. Under this tab, the 'Quadrature Counter' sub-tab is active. In the 'Enable:' section, the checkbox for 'Quadrature Counter on Port 2:' is unchecked. A note below states: 'Note: Port 0 and Port 1 can also act as quadrature counters. If needed, contact the Champalimaud Foundation's Hardware Platform in order to implement the feature.' Below this, the 'Counter current value:' section shows 'Quadrature Counter on Port 2:' with a value of 0. At the bottom, there are buttons for 'Read', 'Save', and 'Reset Quad. Counter Port 2'. At the very bottom of the window, there are buttons for 'Write all saved registers permanently to the device' and 'Connection w/ device'.

Under Extended Features  
> Quadrature Counter

Check  
this box

Reset to zero at  
the index position

Save registers to  
device

Configuring the quadrature encoder in Harp Behavior

# Hardware Setup

## Harp Behavior Board

The Harp Behavior board contains **three ethernet ports** that can be hooked up to breakout boards or simple poke ports. Quadrature encoders should be connected through a breakout board to ethernet Port 2 on the Behavior board. The Behavior board also contains **4 digital outputs** with **DO0 and DO1** being primarily for camera controlling. The Harp behavior can also be utilized for analog readings as well with **a 5V ADC screw terminal** located next to the ethernet ports.

## Simple Pokeport

The simple pokeport has a solenoid that can be opened via an **ethernet port or a digital input** connected to the respective port on the Behavior board. You can utilize any solenoid in place of this board however the ethernet port makes this easy to use with the Behavior board outputs to trigger valve openings.

## Breakout Board

The breakout board contains **2 digital inputs and 2 digital outputs**. For a quadrature encoder, pin both A and B channels of the rotary encoder to **DI and DIO** on the breakout board which gets connected to **Port 2** of the Harp Behavior ethernet port. Make sure to pin the ground and power as well on the breakout board.

## Harp Soundcard/Amplifier

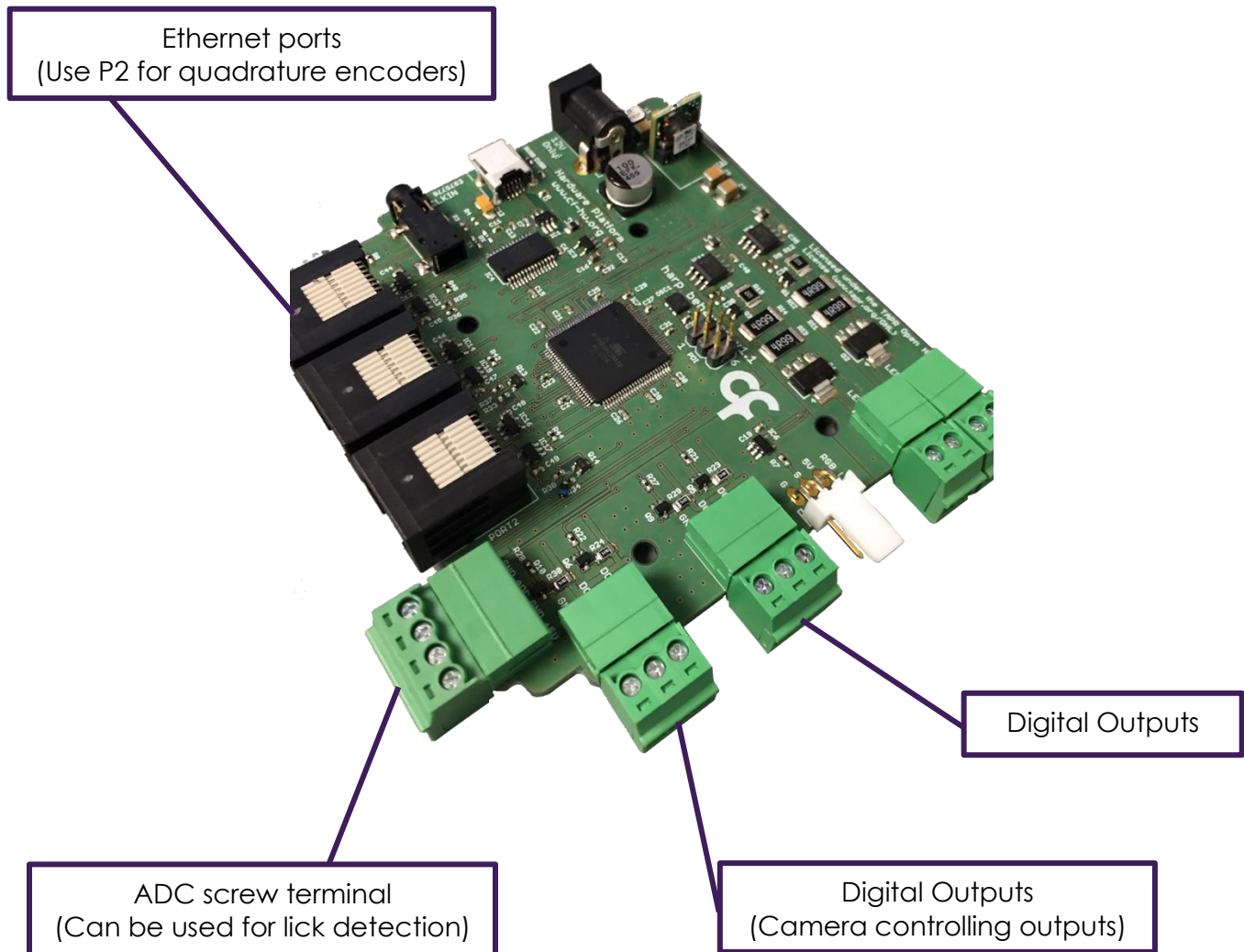
The Harp Soundcard is connected to the Amplifier via **2 RCA cables** which are then hooked to the speaker via **2 banana plug-alligator clip cables** (left and right channels).

## List of components

The following is a full list of Harp-related devices for one rig:

1. 1 Harp Behavior Board
2. 1 Harp Soundcard
3. 1 Harp Amplifier
4. 2 RCA cables
5. 2 banana plug-alligator clip cables
6. 1 Simple pokeport or equivalent solenoid
7. 2 12V power supply
8. 1 Amplifier power supply
9. Jumper cables for the digital outputs and breakout boards (4 needed just for the encoder)
10. 3 ethernet cables
11. 1 USB to micro USB-B cable
12. 2 USB to mini USB-B cables

## Hardware Setup



## Bonsai Resources and Documentation

[Understanding the workflow editor](#)

[Description of Bonsai Nodes](#)

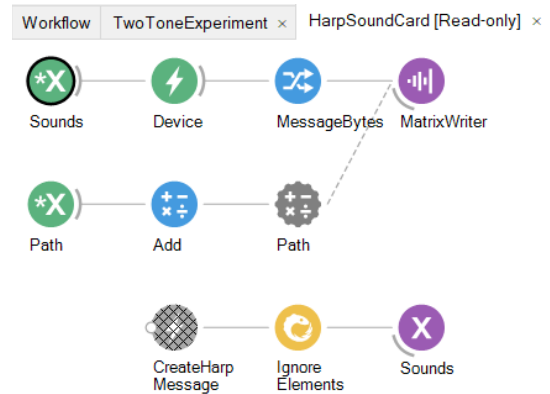
[Subjects and subscribing to sequences](#)

[Resources and Video Tutorials](#)

[Google Groups Forum](#)

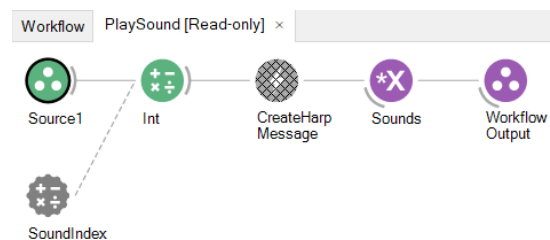
# Harp Extensions

## HarpSoundCard.bonsai



This extension is required to interface with the **Harp Soundcard**. The first row sends all data that is sent and received from the Soundcard and stores it into a **Matrix** that is output by the code once the workflow ends. Make sure to set the Device node to the proper **COM** that the Harp Soundcard is connected to. The .bin data file is saved to the path identified in the second row. To edit the path, edit the value of the **greyed out** externalized property labeled "Path". The third sequence defines the subject, **Sounds**, so that whenever it is called, it sends the sequence to all subscribed observers such as the first row.

## PlaySound.bonsai

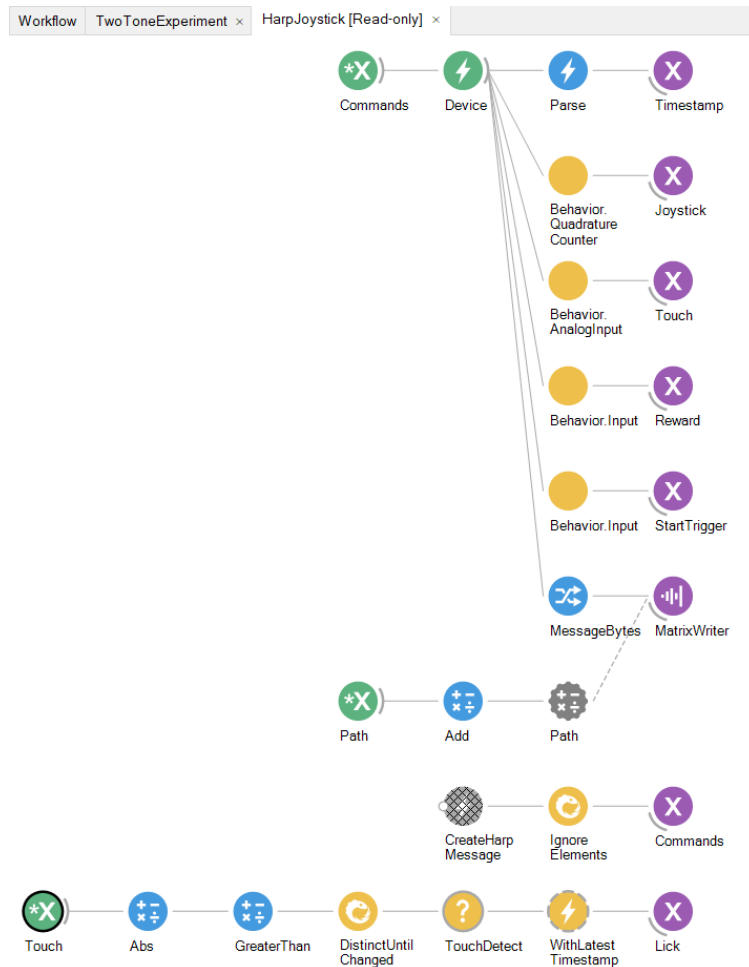


Place this extension anywhere in your workflow where you wish to call a tone from the Soundcard. The value stored in the **Int node** is the index in which the tone is stored on the Harp Soundcard device.



# Harp Extensions

## HarpJoystick.bonsai



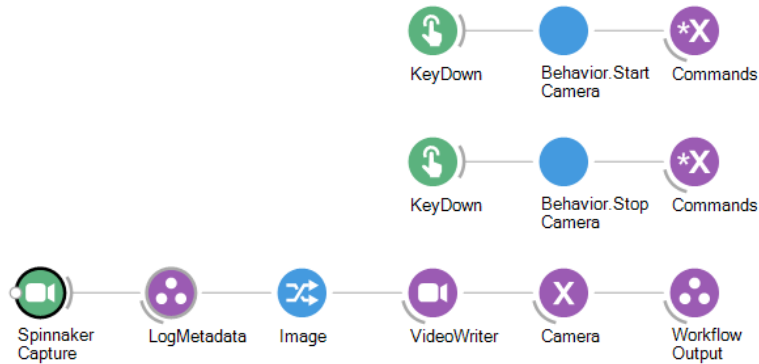
This extension is necessary for interfacing with the **Behavior board**. Like the Soundcard extension, the second to last row simply defines the subject and where it receives an input sequence from so that it can broadcast that sequence to all **subscribed observers**. The first large branch is an observer that then reads from a series of inputs on the Behavior board and syncs that data to new subjects. For example, readings from Port 2's quadrature encoder are stored in the subject, Joystick.

## Yellow Input Nodes

- There are a few yellow **Behavior.X nodes** in the workflow. These are a type of **combinator node** that essentially synchronizes and takes in parallel inputs. These nodes interface with specific inputs on the Behavior board and output or broadcast them to specific **Subjects** (the purple nodes).
- Depending on your setup, you can **comment out/disable** certain branches that are not needed such as Joystick and Touch.
- The last line is a good example of a **subscriber** and **sink** interaction. The code takes the raw analog input from the Behavior board and broadcasts that information to the Touch subject. The last line then subscribes and reads from this subject, defines an absolute threshold and defines any moment at which that threshold is crossed as a lick.

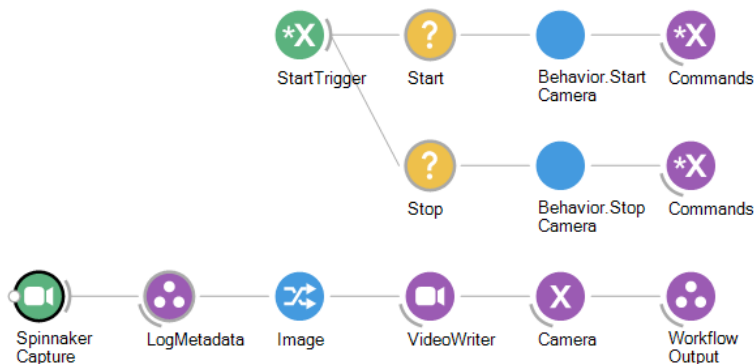
# Harp Extensions

## ManualCamera.bonsai



This extension is necessary for interfacing with the **Spinnaker camera**. Note that the SpinnakerCapture node can be replaced with a CameraCapture node for other cameras such as USB cameras. The **KeyDown nodes** initialize the start trigger and stop trigger for the camera.

## HarpCamera.bonsai



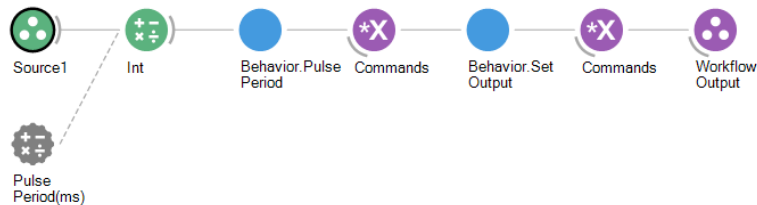
This extension also initiates the FLIR camera however it utilizes a start trigger from the Behavior board. This code can be initialized by a **digital signal** received on any of the digital inputs on the Behavior board and uses that signal to trigger the camera. If you look at the HarpJoystick extension, you will notice that the subject, **StartTrigger** is defined there.

## SINK NODES

- Similar to broadcasting to Subjects, the output of nodes can also be written to different file types such as the VideoWriter node writing to a specific file in video format.
- Sink nodes can be placed in a sequence as seen in both extensions on the left. Right click on any node to see the type of output it is sending out so that you can see what each data type each sink node is receiving.
- Left clicking on the sink node such as VideoWriter will allow you to set the file name and the path to which the file will be saved.

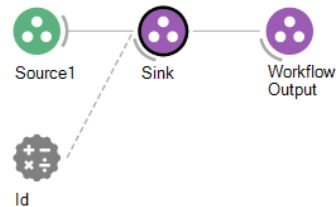
# Harp Extensions

## PulseValve.bonsai



This extension triggers a pulse to a specific port on the Behavior board. This node is most commonly used for triggering a **solenoid** valve opening. The **pulse period** can be changed to set the duration of the valve opening and the port that the solenoid is connected to needs to be identified in the properties of the **Behavior.PulsePeriod** and **Behavior.SetOutput** nodes.

## LogState



Opening up the **CreateObservable**, you will notice a few **LogState** nodes throughout the trial workflow. Selecting any of these will display an **ID** in the properties panel that indicates a change in state of the trial. You will notice that there is a state for the ITI, Go, NoGo and Response states. These Logstate IDs can be changed and moved anywhere throughout the workflow to indicate a transition into a new trial state. These IDs are an important part of the C# scripts that the StateStatistics.bonsai file utilizes to run the GUI.

## Visual Studio

In order to edit the scripts that statestatistics interface with, you need to install [Visual Studio Code](#).

Once Visual Studio Code has been installed, be sure to set the path to where the extensions are located and then you should be able to open the scripts from both your Bonsai workflow and from your file explorer.

Without Visual Studio you may notice that the C# nodes in statestatistics appear grey. Once Visual Studio Code is properly set up, these nodes should appear blue as below. Double clicking on these nodes should open them in Visual Studio Code.



# C# Scripts

## StateId

```
Extensions > StateId.cs
1 public enum StateId
2 {
3     ITI,
4     Go,
5     NoGo,
6     Response,
7     Timeout,
8
9     Annotation,
10
11    Joystick,
12    Lick,
13    Blink,
14
15    Hit,
16    Miss,
17    FalseAlarm,
18    CorrectRejection
19 }
```

In the **StateId** script, this is where you identify your states and responses. Note the first grouping corresponds to specific **states** in the code. The second group below **Annotation** are **responses**. The third and final grouping corresponds to the **trial result**. To add states, responses or results, you simply add another comma separated value to the respective grouping. Then in Bonsai, you can call the State using the **LogState** extension.

## StateVisualizer

```
Extensions > StateVisualizer.cs > ...
63
64 2 references
65 public static class ColorMap
66 {
67     2 references
68     public static readonly Dictionary<StateId, Color> Default = new Dictionary<StateId, Color>
69     {
70         { StateId.ITI, Color.Gray },
71         { StateId.Go, Color.Orange },
72         { StateId.NoGo, Color.Yellow },
73         { StateId.Response, Color.Green },
74         { StateId.Timeout, Color.Red },
75         { StateId.Annotation, Color.Black },
76         { StateId.Joystick, Color.IndianRed },
77         { StateId.Lick, Color.LemonChiffon },
78         { StateId.Blink, Color.Violet },
79         { StateId.Hit, Color.Transparent },
80         { StateId.Miss, Color.Transparent },
81         { StateId.FalseAlarm, Color.Transparent },
82         { StateId.CorrectRejection, Color.Transparent },
83     };
84 }
```

In **StateVisualizer** you need to now add or modify any changes you made in StateId to these specific lines. This will tell the Visualizer how to display these states and responses. Add lines for additional states/responses and remove any that have been deleted from StateId.

## ResponseStatistics

```
Extensions > ResponseStatistics.cs > ...
25
26 {
27     stats.Epoch++;
28     switch (response)
29     {
30         case ResponseId.Hit: stats.Hits++; break;
31         case ResponseId.Miss: stats.Misses++; break;
32         case ResponseId.FalseAlarm: stats.FalseAlarms++; break;
33         case ResponseId.CorrectRejection: stats.CorrectRejections++; break;
34     }
35     return stats;
36 }
37
38 }
```

In **ResponseStatistics**, you will also need to similarly update the Results as this is what Bonsai utilizes to report the result of each trial. Make sure the results here match with the results you defined in StateId.

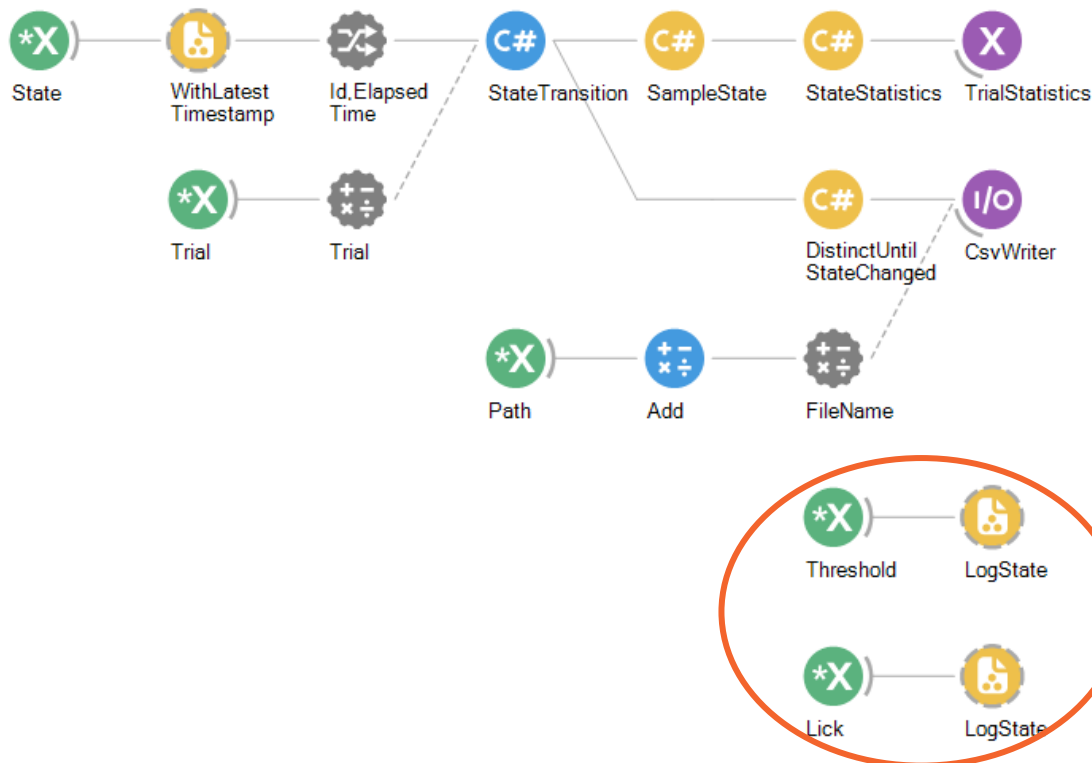
## C# Scripts

### ResponseVisualizer

```
Extensions > ResponseVisualizer.cs > ...
1 reference
21 public static class ColorMap
22 {
23     1 reference
24     public static readonly Dictionary<ResponseId, Color> Default = new Dictionary<ResponseId, Color>
25     {
26         { ResponseId.Hit, Color.Green },
27         { ResponseId.Miss, Color.Orange },
28         { ResponseId.FalseAlarm, Color.IndianRed },
29         { ResponseId.CorrectRejection, Color.LemonChiffon }
30     };
31 }
```

Like with StateVisualizer, add your results to this section of the **ResponseVisualizer** script and this will define to the visualizer how to depict these results.

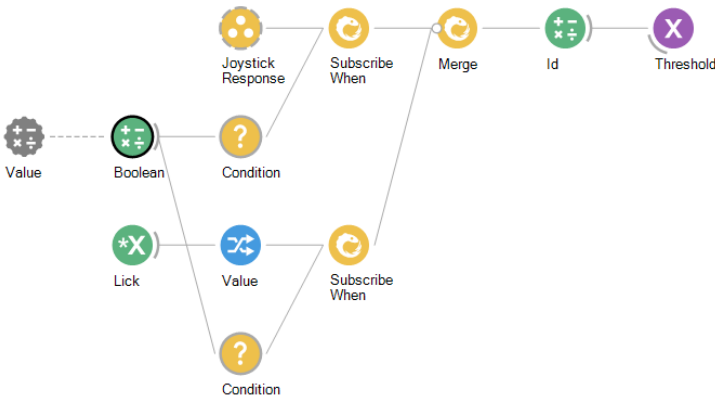
### Generating the GUI in Bonsai



This portion of the code is located in the **top workflow**. In the first line all states are automatically included under the subject States which is what the C# scripts are subscribed to. As long as your states are edited accordingly in the C# scripts, this should not require any changes. On the other hand, responses have to be specifically labeled. In this example, **threshold** (joystick) and **lick** are the two responses being sent to the **logstate** extension. For additional responses, you will need to copy and paste the circled code and add them accordingly to **StateVisualizer** and **StateId**. NOTE that for removing annotations or states, remove from your Bonsai code **FIRST** and then remove from the C# scripts or you may break the code and have to pull the code from the repository again.

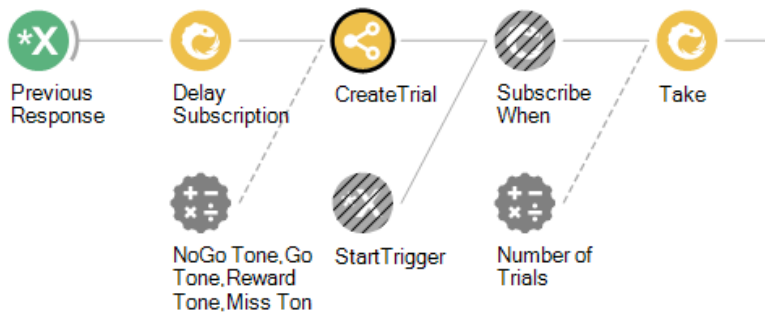
# Trial Workflow

## Response Type



In the top workflow, you can alter the type of response by editing the value of the **Boolean** in the code above. **False** will set the code to a **lick** response task while **True** will set the code to a **joystick** response task.

## Start Trigger



By default, the **StartTrigger** will be disabled however this can be enabled to initialize a Trigger to start your code. Make sure that you set up which port on the behavior board will act as the trigger line in the **HarpJoystick** extension.

## Editing Extensions

In order to edit extensions, you must select the node and press CTRL + G. You should notice the node changes as the following:



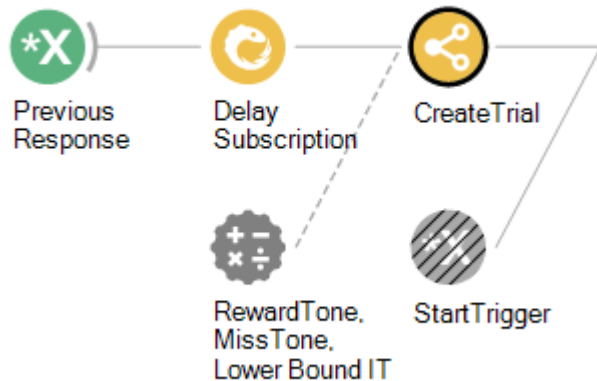
You can save as workflow afterwards however note that this will change this extension in every Bonsai code you use that shares that extension within that folder.

In order to test the Joystick and Soundcard without the equipment handy, replace the HarpJoystick and HarpSoundcard with the SoftJoystick and SoftSoundcard nodes.

Remember that next to the Start button, there is a space to change the current working directory in order to make sure your workflow is accessing the appropriate extensions.

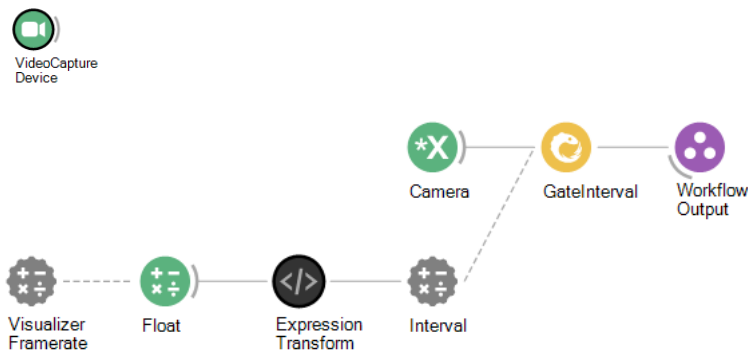
# Trial Workflow

## ExternalizedMapping



In large, grouped workflows/extensions you can select the node and type in toolbox “**ExternalizedMapping**”. Double click on the externalized mapping node and select add all to add the properties stored in that workflow to the top level rather than adding each individual property manually.

## VideoCaptureDevice



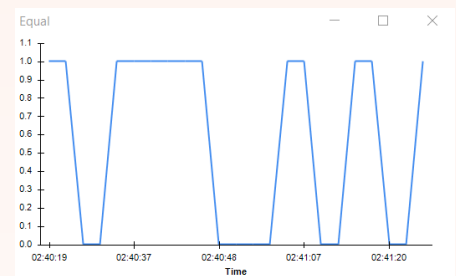
This node requires the **Bonsai video library** package. Place this node into your workflow and when running your code, double click on this node to visualize the acquired sequence in real time. Under properties for this node, you may need to test different indices in order to find your specific **USB camera**. Adding a **GateInterval** with the line below will allow you to edit the rate of frames captured.

## Visualizing Nodes

When the program is running, **double clicking** on any particular node will open up a visualizer indicating the sequence of events the node receives and outputs. For example, in the top workflow on StateStatistics, ResponseStatistics, JoystickVisualizer and CameraVisualizer can all be selected to open up the visualizer for these extensions/grouped workflows.

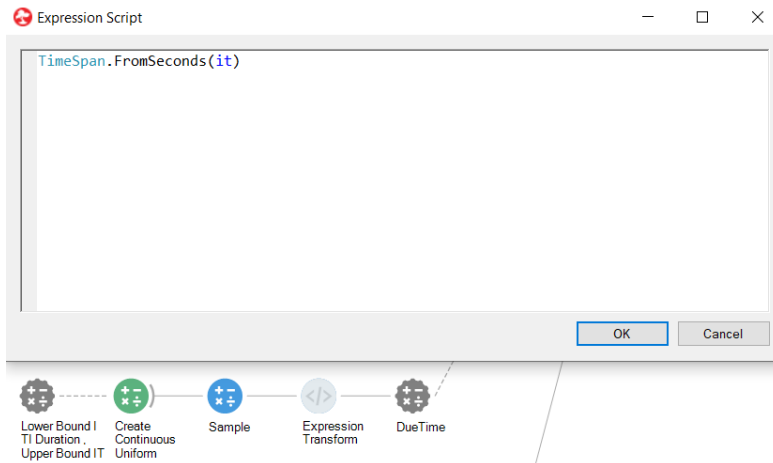
This function is especially useful for **debugging** as you can see what information each node is receiving and outputting and see if certain nodes are receiving values that they should not be receiving.

For example, double-clicking the **Equals node within CreateObservable** opens up the below visualizer which indicates when the node has received values of 0 or 1 which correspond to Go or NoGo Trials within the StateStatistics workflow.



# Trial Workflow

## ExpressionTransform



**ExpressionTransform** nodes are useful for incorporating **C#/Python scripts** into your code. In **StateStatistics**, you will notice that the ITI is generated from a script that pulls a random number within a given range and outputs it as a timespan in seconds.

## CreateTrial



This node stores the main workflow of the experiment. The **Concat** node generates a number of sequences of the observable created by **CreateTrial**, as denoted by the **Take** node. For that reason, the value of the **Take** node corresponds to the number of trials. Note that within **CreateTrial** you can see a condition for two different paths for the trial to follow. The number of paths can be set in the **CreateDiscreteUniform** node in the top workflow as shown below. Changing this range from 0-3 will allow for four paths that you can set up in the **CreateTrial** workflow for many tones.



## TRIAL RESPONSES



At the end of the **CreateTrial** workflow you should notice four possible outcomes. The boolean output from the **yellow SelectMany nodes** are what decide what outcome the trial results in. Notice that for the **Move SelectMany** node, only a joystick push leads to a **True Boolean** which results in the **Hit outcome** while a pull or no response results in a **Miss**.

See the **VisualCode** scripts above for how to incorporate new results into the **ResponseId** script. This is necessary for the result to appear in your **Bonsai** workflow.



# Data Output

## Raw File Output

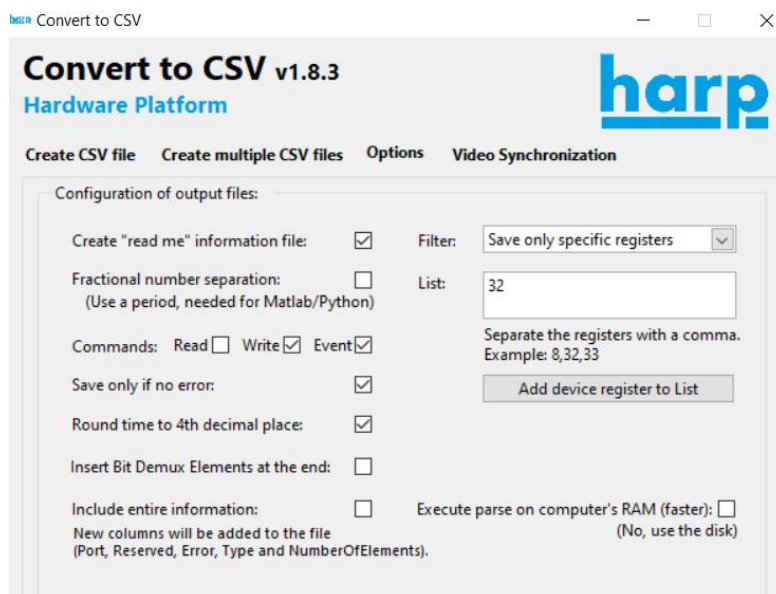
Name	Date modified	Type	Size
Joystick.bin	12/4/2020 9:34 AM	BIN File	0 KB
soundcard.bin	12/4/2020 9:34 AM	BIN File	0 KB
sounds	12/4/2020 9:34 AM	CSV File	0 KB
StateTransitions	12/4/2020 9:34 AM	CSV File	1 KB

In the **Data folder** under where the workflow is located, the Data should be stored within a folder labeled by the date and time of the trial run. You should see a collection of CSV and BIN files depending on the sink nodes you have created in your workflow.

**ResponseStats** gives you the results of each trial while **StateTransitions** gives the timestamps of every state change in the workflow. Joystick and soundcard BIN files give the raw output from the Behavior and Soundcard.

## Harp ConvertToCSV

The BIN files can be converted using the Harp program **Convert to CSV** which can be downloaded [here](#). In order to properly read the Soundcard.BIN file copy the Options window as below. You can also select relevant additional registers by selecting **Add device register to List**.



## Overall Summary of Current Workflows

### ToneTesting.Bonsai

This workflow is simply for playing a modifiable number of tones with a specified ITI in a random order.

### StateStatistics.Bonsai

A full workflow for a modifiable Go No-Go Task with either lick response or joystick responses with a tone stimulus.

### SolenoidTest.Bonsai

For use to test solenoid pulse outputs and running lines through.

### PreTraining.Bonsai

Workflow for the PreTraining task for the modified Go No-Go task with a reward delivery for joystick movements without a tone.

Special thanks to Goncalo Lopes from Neurogears

