

# Fundamentals of Fiber Photometry Workflows

**NEUROPHOTOMETRICS**

The fiber photometry company



Christopher S. Johnson

October 23rd, 2023

# Table of Contents

---

<b>Preface</b>	<b>4</b>
<b>Introduction to Bonsai</b>	<b>5</b>
<b>Chapter 1: Hardware/Software Environment</b>	<b>9</b>
System Requirements	10
Downloads and Installations	11
Updating Bonsai Software and Packages	16
<b>Chapter 2: Standard Photometry</b>	<b>19</b>
Building the Workflow	20
FP3001 Systems	23
FP3002 System	26
<b>Chapter 3: Synchronization</b>	<b>40</b>
Software	41
Hardware	49
Machine Vision	53
<b>Chapter 4: Data Acquisition</b>	<b>65</b>
Delayed Start	65
Hardware Control	70
Manual Control	72
Basic Periodic Control	79
Periodic Control, Variable Duty Cycle	83
Periodic Control, Delayed Start	96
Second Order Periodic Control	104
<b>Chapter 5: Stimulation</b>	<b>117</b>
Delayed Start	117
Hardware Control	123
Manual Control	127
Basic Periodic Control	139
Periodic Control, Variable Duty Cycle	146
Periodic Control, Delayed Start	158
Second Order Periodic Control	166
<b>Chapter 6: Machine Vision</b>	<b>178</b>
Synchronized Behavioral Camera	179
One Chamber Animal Tracking	185
Three Chamber Animal Tracking	197
Chamber Dependent Data Acquisition	206
Chamber Dependent Stimulation	213
Real Time Analysis	218

<b>Appendix I: Node Glossary</b>	<b>245</b>
Acquisition Control	245
Digital IOs	246
Digital Output	248
FP3001	249
FP3002	251
Photometry Data	267
Photometry Writer	269
Stimulation	273
Temperature	275
<b>Appendix II: Hotkeys</b>	<b>276</b>

# Preface

---

Neurophotometrics systems use an open-source visual programming language called [Bonsai](https://bonsai-rx.org/) (<https://bonsai-rx.org/>) to acquire data, visualize signals, and synchronize with external hardware (behavior hardware, cameras, etc.). Bonsai allows users to record and temporally align parallel data streams with ease. Bonsai files consist of “nodes” that interface with hardware, perform processing steps, or write out files. Nodes are connected to one another to create “workflows”.

This document discusses the fundamentals of developing Bonsai workflows for conducting experiments using Neurophotometrics fiber photometry systems. The material begins with simple workflows for recording FP data, and builds in complexity over the course of each chapter. Each example workflow outlined in this manual can be downloaded from [our website](#).

Chapter 1 of this document details installation and system requirements. From there, we delve into fundamental concepts found within fiber photometry experiments. Chapter 2 presents our standard workflows for acquiring data from FP3001 and FP3002 fiber photometry systems. In Chapter 3, we discuss the synchronization of parallel data streams, where we highlight common techniques for aligning with a fiber photometry data stream. Chapter 4 builds on the idea of controlling data acquisition for the FP3002 system. In Chapter 5, we incorporate the use of laser stimulation during experiments, showcasing different ways to control stimulation. Finally, Chapter 6 explores implementations of machine vision techniques for animal tracking and closed loop experiments.






This document concludes with several appendices including a node glossary, useful hotkeys, and a troubleshooting guide. The [Node Glossary](#) provides details on each of the individual nodes contained within our Neurophotometrics packages. The [Hotkeys](#) section provides useful hotkeys for interfacing with Bonsai and our nodes. Finally, the [Troubleshooting](#) section guides the reader through fixing common errors in fiber photometry workflows.

# Introduction to Bonsai

---

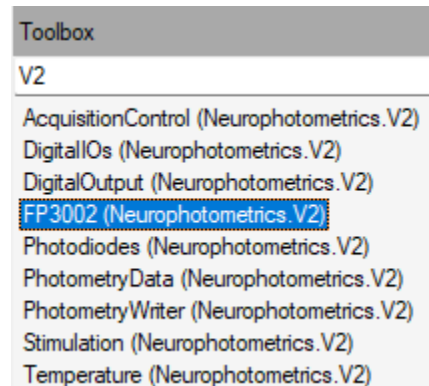
Bonsai is an open-source visual programming language that is great for processing and recording from heterogeneous, asynchronous data streams. In this software, users build workflows for acquiring, processing, and saving data. These workflows are constructed by chaining together workflow elements called “Nodes”.

Nodes are reactive operators used to build all sorts of observable sequences. When first learning Bonsai it is useful to know the color-coded system that Bonsai uses to group together nodes of a certain type. Below is an example of each type of node:

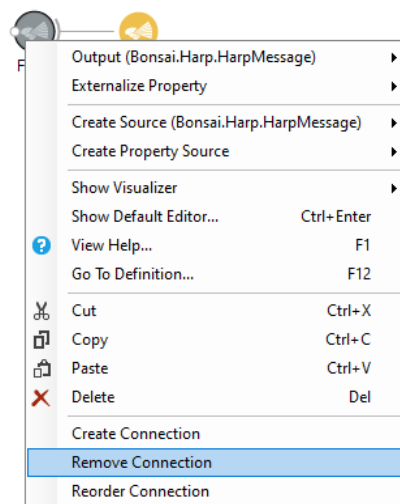
- **Source:**  These nodes generate observable sequences. Often they are used to communicate with external devices or to read data from files saved to the computer.
- **Transform:**  These nodes process data, oftentimes changing the datatype of the elements of the data stream.
- **Condition:**  These nodes can filter the data stream based on specified conditions.
- **Sink:**  These nodes do not alter the observable sequence. Instead they are used to funnel data into other locations, such as saving to a file or publishing to a separate data stream.
- **Combinator:**  These nodes are used to manage how separate data streams are combined together.

In order to construct a workflow, you can add, remove, connect, and disconnect nodes:

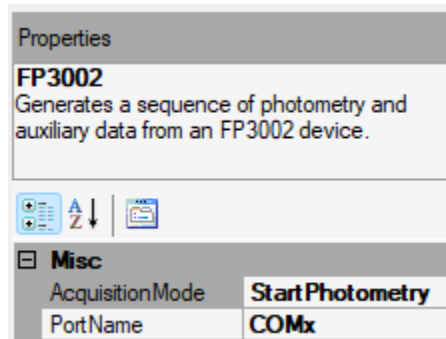
- **Adding:** To add a node to the workflow, search for it in the “Toolbox” on the left of the workflow. Once the desired node is found, double click it to add it to the workflow. Our Neurophotometrics nodes are most easily found by searching “V1” for FP3001 system nodes or “V2” for FP3002 system nodes.



- **Removing:** To remove a node from the workflow, select it and press the “Del” key to delete it.
- **Connecting:** To connect a node to another node, click and drag from the upstream node to the downstream node.
- **Disconnecting:** To disconnect a node from another node, right click one of the nodes, click “Remove Connection”, then left click the other node.

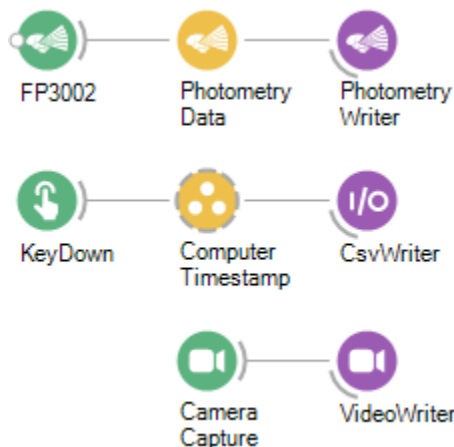


Once a workflow is constructed, it likely needs to be configured. Many nodes have accessible properties contained within the “Properties” panel on the right of the workflow. This panel also provides a node description which is useful when working with a node for the first time.

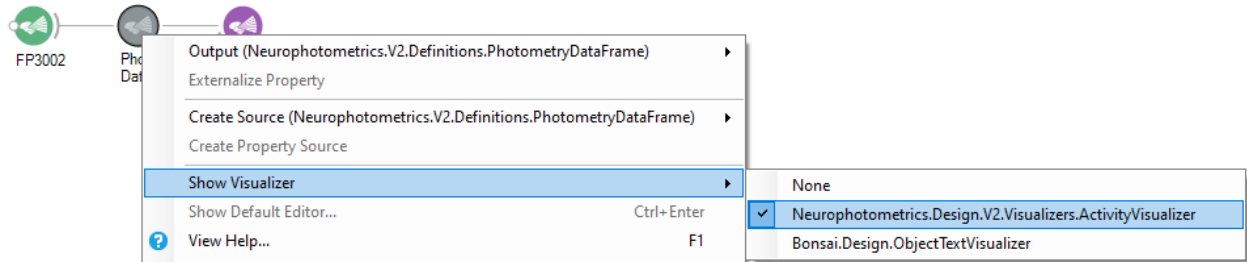


Some nodes that require more complex configuration also have an associated editor. You can access these node editors by double clicking the node while the workflow is not running. For example, the “FP3001” and “FP3002” nodes both have their own editors for configuring the FP3001 and FP3002 systems.

Once a workflow is constructed and configured, press the “Start” button to begin running the workflow. After starting the workflow, data will begin passing through all of the defined data streams.



While running the workflow, you can visualize the data passing through the node by double clicking the node. Some nodes have multiple visualizers associated with them. You can specify which visualizer to use by right clicking the node and selecting “Show Visualizer”



The nodes used to create Bonsai workflows are contained in NuGet packages available for downloading in the “Manage Packages” window. Bonsai publishes its own packages for a wide range of operations. However, third-party packages can also be downloaded from the “Community” Package source. Neurophotometrics has created our own packages for fiber photometry workflows.



# Chapter 1: Hardware/Software Environment

---

This chapter outlines the minimum hardware requirements for conducting fiber photometry experiments within Bonsai. Then, we cover required downloads, installation instructions, and how to manage and update software packages within Bonsai.

# **System Requirements**

## **Operating System**

Windows 10/11

## **Processor**

64-bit, four logical cores, with a clock speed of 1.6GHz.

## **Graphics Card**

None

## **RAM**

8GB

## **Ports**

1 Type A USB 3.0 port (SuperSpeed logo should be visible **SS**  )

NOTE: please keep in mind, your additional hardware (such as behavioral cameras, keyboards, mice, etc) may require USB ports as well. The FP3002 cannot run through a USB hub. So please ensure you have sufficient ports for all necessary hardware.

## **Notes**

- NPM Fiber Photometry systems must be connected directly to Type A USB 3.0 ports.
- FP systems should be connected using the original USB cable shipped with the system to ensure sufficient data transfer speeds.
- NPM Fiber Photometry systems cannot be connected to USB hubs as they require the full bandwidth of the USB 3.0 port.
- Internet access is required for initial downloads and installations as well as for future updates.

## **Downloads and Installations**

In order to set up the software environment for fiber photometry workflow development, three pieces of software need to be downloaded and installed.

1. FTDI's Virtual COM Port (VCP) driver
2. Spinnaker Software Development Kit (SDK)
3. Bonsai

### **FTDI Drivers:**

This allows the Neurophotometrics Fiber Photometry System to communicate over the USB 3 port to Bonsai by treating USB connections as COM ports.

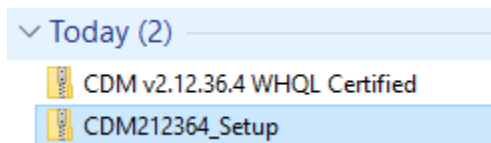
Go to the FTDI VCP Drivers [website \(https://ftdichip.com/drivers/vcp-drivers/\)](https://ftdichip.com/drivers/vcp-drivers/), and navigate to the currently supported "Processor Architecture" table to find the list of downloadable drivers. Find the row that contains the Windows OS on your computer, most likely labeled "Windows (Desktop)\*", and find the column labeled "X64 (64-Bit)". Click the link in that cell to download the .ZIP file for the drivers. Then click the "Setup Executable" link in the rightmost column of that row to download the .exe file that will set up the drivers.

Currently Supported VCP Drivers:

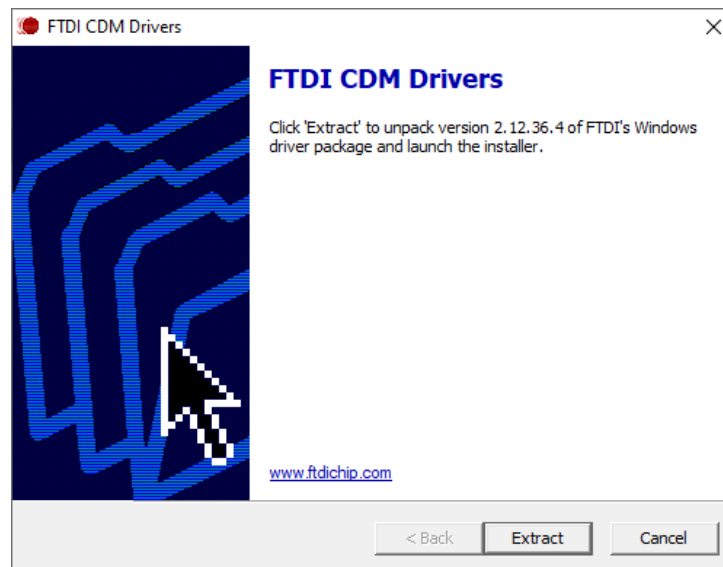
[Subscribe to Our Driver Updates](#)

Operating System	Release Date	Processor Architecture							Comments
		X86 (32-Bit)	X64 (64-Bit)	PPC	ARM	MIPSII	MIPSIV	SH4	
Windows (Desktop)*	2021-07-15	<a href="#">2.12.36.4</a>	<a href="#">2.12.36.4</a>	-	-	-	-	-	WHQL Certified. Includes VCP and D2XX. Available as a <b>setup executable</b> . Please read the <a href="#">Release Notes</a> and <a href="#">Installation Guides</a> .
Windows (Universal)***	2021-11-12	<a href="#">2.12.36.4U</a>	<a href="#">2.12.36.4U</a>	-	-	-	-	-	WHQL Certified. Includes VCP and D2XX.
Linux	-	-	<a href="#">1.5.0</a>	-	-	-	-	-	All FTDI devices now supported in Ubuntu 11.10, kernel 3.0.0-19. Refer to <a href="#">TN-101</a> if you need a custom VCP VID/PID in Linux.

In the downloads folder of the computer, there will be two .zip files with names similar to “CDM v2.12.36.4 WHQL Certified” and “CDM212364\_Setup”. Navigate into the “CDM212364\_Setup.zip” folder and find the “CDM212364\_Setup.exe” file. Double click it to open up the installation wizard.



Follow the steps in the “FTDI CDM Drivers” setup window to complete the installation.

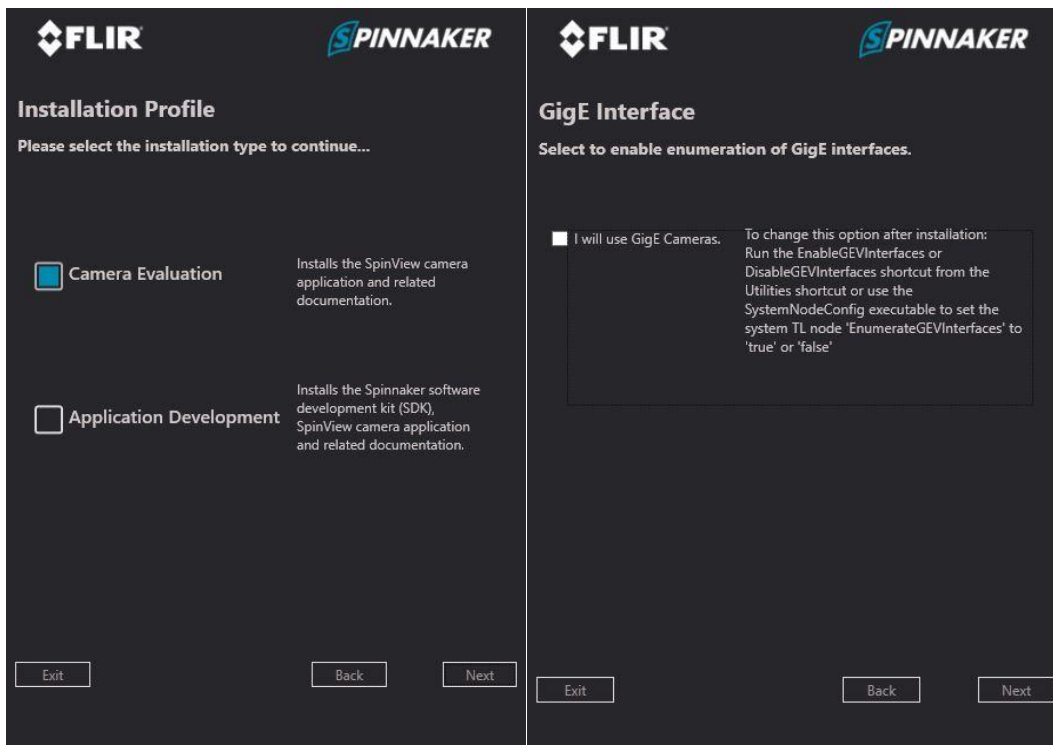


## Spinnaker SDK:

This is used to allow the computer to communicate with the internal CMOS camera on the Fiber Photometry system.

Download the [SpinnakerSDK FULL 1.29.0.5 x64.exe](#)

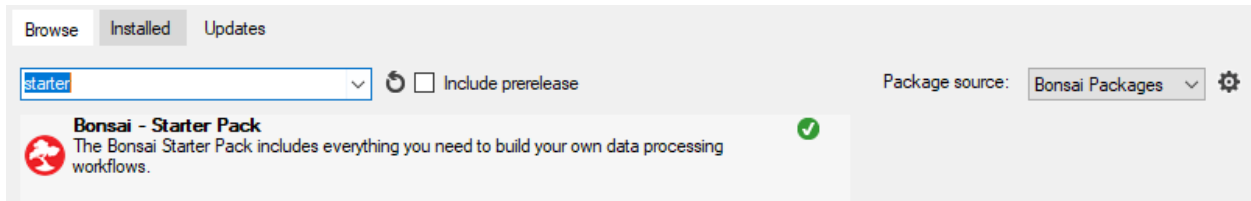
Currently only the 1.29.05 version of the SDK is compatible. Run the .exe file and follow the installation wizard. When choosing between Camera Evaluation or Application Development choose **Camera Evaluation**. When at the “GigE Interface” window, uncheck "I will use GigE Cameras" and complete by clicking “Install.”



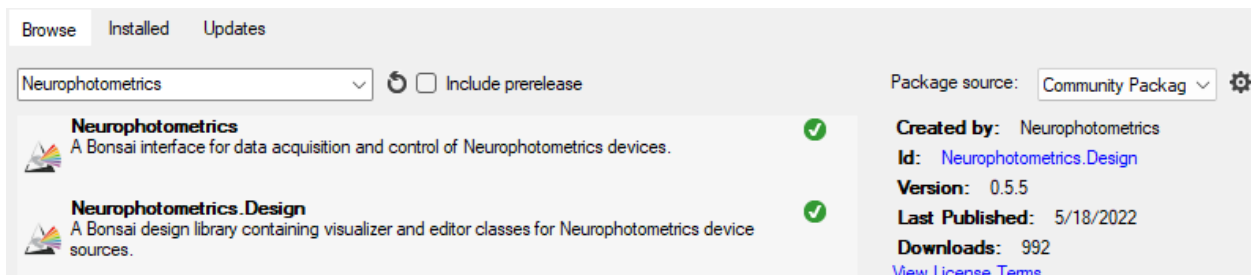
## Bonsai Software:

Bonsai is an open source software that utilizes a visual programming language for creating and synchronizing heterogeneous data streams. This software interfaces with NuGet to allow for easy installation of open source software pages, including our packages for communicating with NPM Fiber Photometry systems.

Download the .exe installer for the latest stable version of [Bonsai](https://bonsai-rx.org/) (<https://bonsai-rx.org/>). When installation is complete, **Launch** Bonsai and it will complete its installation. Internet access is required for the initial launch of Bonsai as it will automatically download and install several necessary software packages off of NuGet. At the start up window, click on **Manage Packages**. Navigate to the **Browse** tab, and set “Bonsai Packages” as the **Package Source**. Now search for the “Bonsai - Starter Pack” and install it by clicking it and clicking the “Install” button that pops up. This will install all of the Bonsai packages used in the majority of workflows. **Note:** Do not check the “Include Prerelease” box because it will allow the installation of packages still in development.



Next, change the **Package Source** to “Community” and search for the “Neurophotometrics” packages. Install the “Neurophotometrics.Design” package. Doing so will also install the “Neurophotometrics” package with all of their dependencies.

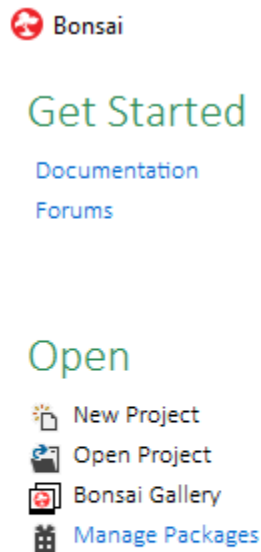


# Updating Bonsai Software and Packages

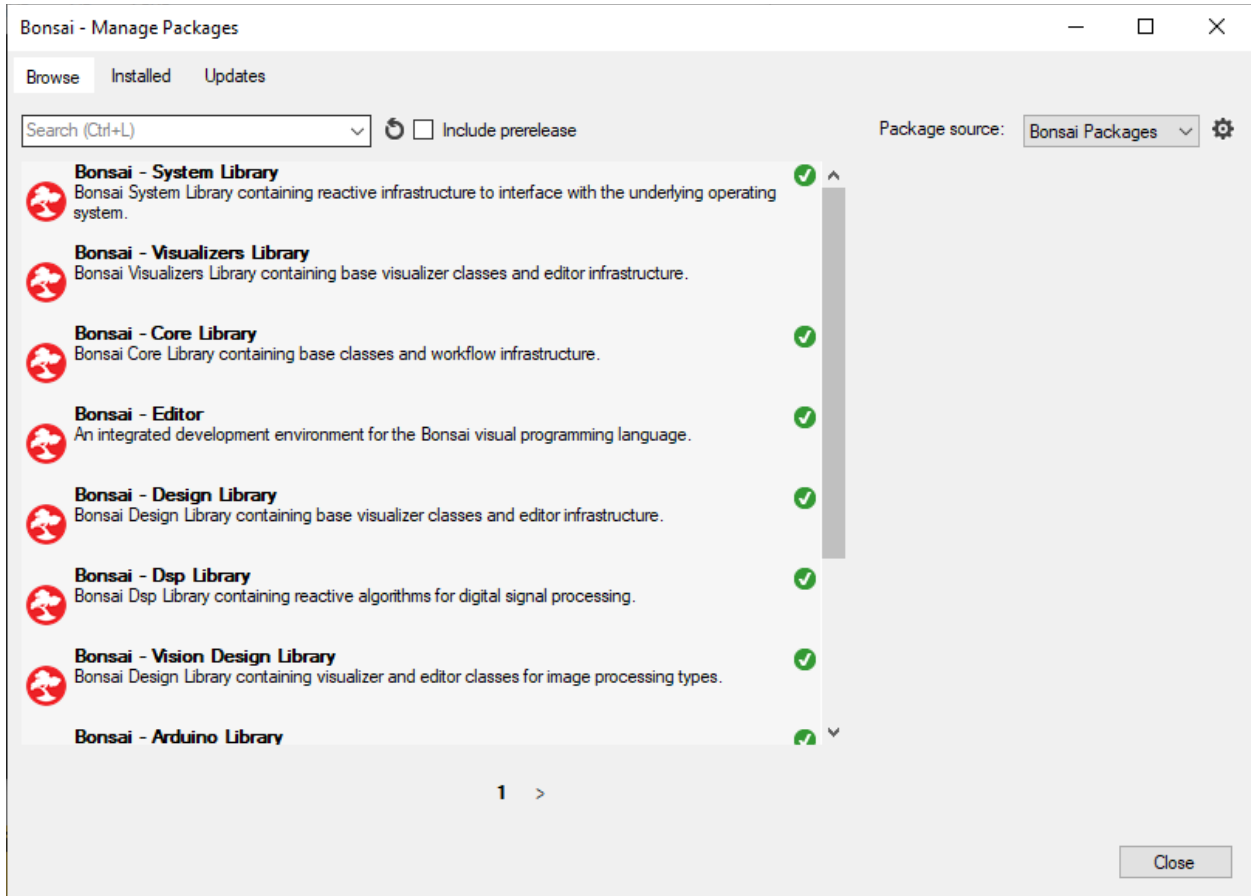
Keeping the Bonsai software and Neurophotometrics packages up to date on the Bonsai Packages and the Bonsai Software is vital to keeping experimental setups running smoothly and getting the most out of the Neurophotometrics Fiber Photometry systems. Below, we outline the processes of updating packages and updating the Bonsai software.

## **Updating Packages:**

Neurophotometrics pushes updates to our Bonsai packages in order to add new features and fix known bugs in the software. Upon release of package updates, users will be notified via an email blast. It is best to update the “Neurophotometrics” package immediately after receiving this email blast. To update, open the Bonsai software and navigate to the “Manage Packages” window.







Once in the “Manage Packages” window, navigate to the “Updates” tab and select “Community Packages” as the “Package Source”. If there are newer versions of the “Neurophotometrics” packages, they will appear here. Click the “Neurophotometrics.Design” Package and click “Update”. This will update both “Neurophotometrics” packages and all of their dependencies.

If you change the “Package Source” to “All”, you will find the available updates for all of your other installed packages. Most packages are safe to update in this way; however, **be cautious** updating the “Bonsai - Spinnaker”, “OpenCV.NET”, and “OpenTK” packages. These packages are not always forward compatible, so if these are updated, then the “Neurophotometrics” packages might not be able to work properly. The safest way to update these packages is to update the “Neurophotometrics.Design” package which will automatically update these packages to the correct versions.

**Updating Bonsai Software:**

As Bonsai Packages are updated to newer versions, occasionally new features are not compatible with older Bonsai Software versions. To update the Bonsai Software, uninstall Bonsai then reinstall the latest stable version. Find “Bonsai” inside of the Windows search bar. Right click it and click “uninstall”. This will open the “Programs and Features” window. Find “Bonsai” in the list of programs, click it, and click “uninstall”. Then, follow the “[Downloads and Installations](#)” Bonsai section on downloading and installing the latest stable version of Bonsai.

# Chapter 2: Standard Photometry

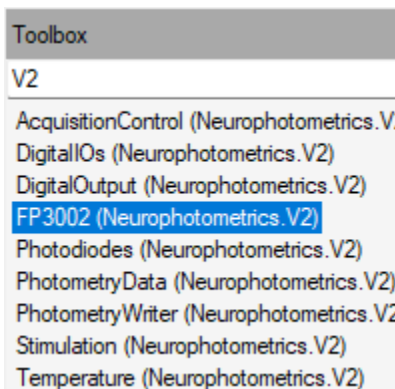
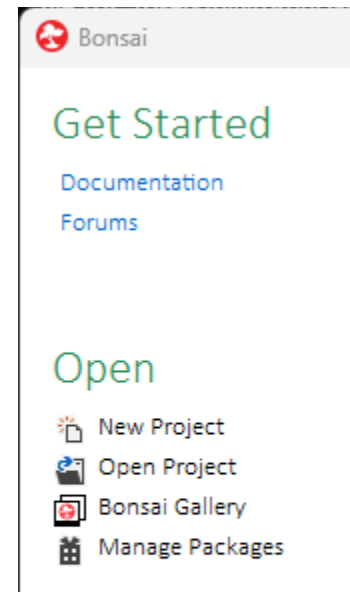
---

In this chapter, we walk through standard fiber photometry workflow setup. This workflow is used to record, visualize, and save interleaved fiber photometry data. The “Standard Photometry” workflow is the minimum needed to record data, and acts as a base for more complex workflows. In the two sections of this chapter, we will cover the construction and configuration of standard workflows for recording from both the FP3002 and FP3001 systems. In later chapters we will build off of the FP3002 workflow developed here.

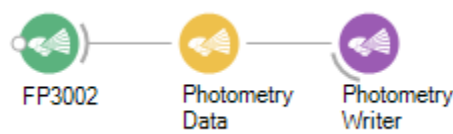
## Building the Workflow

The structure of the Standard Photometry workflow is the same for both FP3001 and FP3002 systems. To build a new workflow, open Bonsai and select New Project. This will open up a blank workflow.

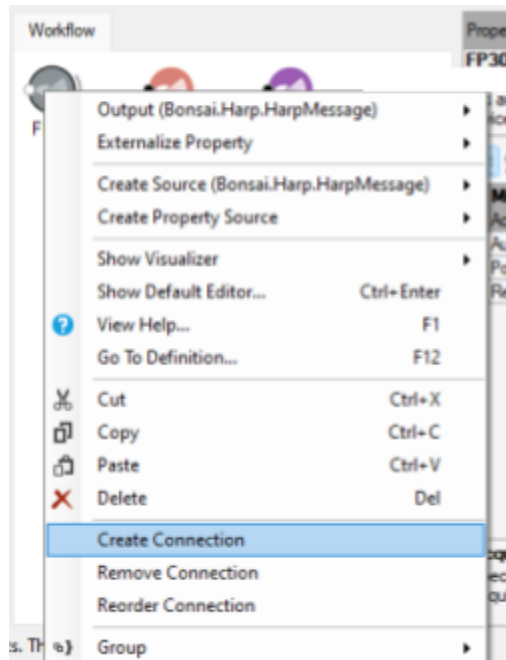
In the toolbox search bar on the left side of the workflow window, search for “V1” to access the nodes for the FP3001 system and “V2” to access the nodes for the FP3002 system. The “FP3001” and “FP3002” nodes are source nodes that allow Bonsai to communicate with FP systems. If your system has a driver box, it is a V1 system that uses the “FP3001” node. If it does not have a driver box, it is a V2 system. Double click to add the node to your workflow.



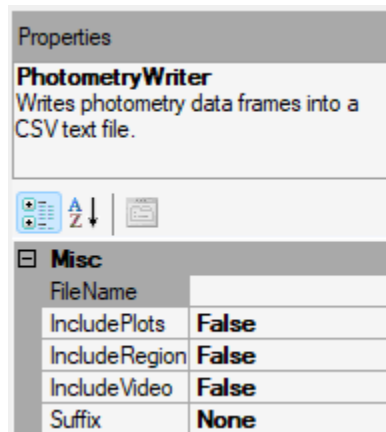
Next, search for “PhotometryData (Neurophotometrics)” and double click to add to workflow. Note: there are 2 options for PhotometryData, one for V1 and one for V2. Be sure to select the correct one for your system. The Photometry Data node processes incoming data and also allows for visualization in real time. This node now encompasses both PhotometryData and VisualizeROI nodes in previous versions of the software. Finally, add the appropriate “PhotometryWriter (Neurophotometrics)” node. These nodes process incoming photometry data and write out data files, respectively.



The nodes should connect in the order that you add them. If the nodes do not connect automatically, you can connect them by right clicking on an upstream node, selecting “Create Connection”, then clicking on the downstream node.



You will need to configure the PhotometryWriter node in order to properly save recorded data. In addition to saving an output .csv file containing photometry data, this node provides additional, optional functionality through the “Include..” properties. If any of the “Include...” properties are set to “True” then the “Photometry Writer” node will generate a folder containing the output .csv as well as the additional files from the “Include...” properties. Below is a description for each property:



**FileName:** The name of the output .csv file. If any of the “Include...” properties are set to “True” this filename will also help generate the name of the output folder. To name

files and specify where they are saved, click into the empty box next to FileName, then click the ellipsis button to navigate to the desired folder.

IncludePlots: When enabled, a “Plots” folder will be created that contains the raw data plot for each region of interest to serve as a quick snapshot of each recording. This property is not recommended for experiments longer than 1 hour. Please note that this is raw data, and we do not recommend using these plots for uses beyond internal reference.

Include Regions: When enabled, a “RegionImages” folder will be created that contains the first and last image taken by the FP3002 system with the user-defined regions of interest overlaid.

Include Video: When enabled, a “VideoFiles” folder will be created that contains multi-page *.tif* files. These *.tif* files are lossless to allow for post-hoc analysis of the raw image data coming from the FP3002 system. This feature saves large amounts of data so be sure to have sufficient write speed and storage capacity before enabling. We do not recommend using this feature unless you specifically plan to analyze the raw images.

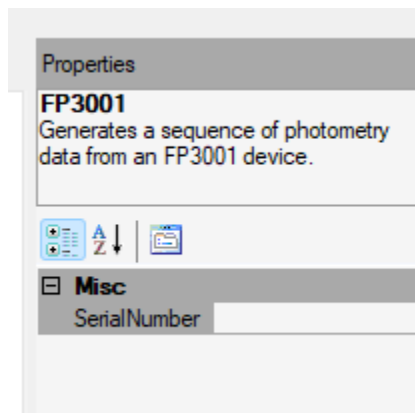
Suffix: Allows the repetition of the same file name for multiple experiments by appending a unique suffix to the filename. The options for unique suffixes are an integer value (count) or a date-time value (timestamp).

Once you have built a Standard Photometry workflow, there are some settings to configure before you can record. These settings differ between FP3001 and FP3002 setups, and are detailed in the following sections.

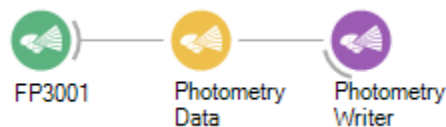
## FP3001 Systems

The “FP3001” node is a source node used to communicate with the FP3001 system. This node processes the information coming from the FP3001 system and generates photometry data frames. Each photometry data frame contains an image, frame counter, system timestamp, computer timestamp, frame flags, and emission data.

To configure the FP3001 node, single click the node to open the properties panel. Click into the empty box next to “SerialNumber”, and select the camera serial number from the dropdown menu. This is the serial number of the internal camera and not the system.

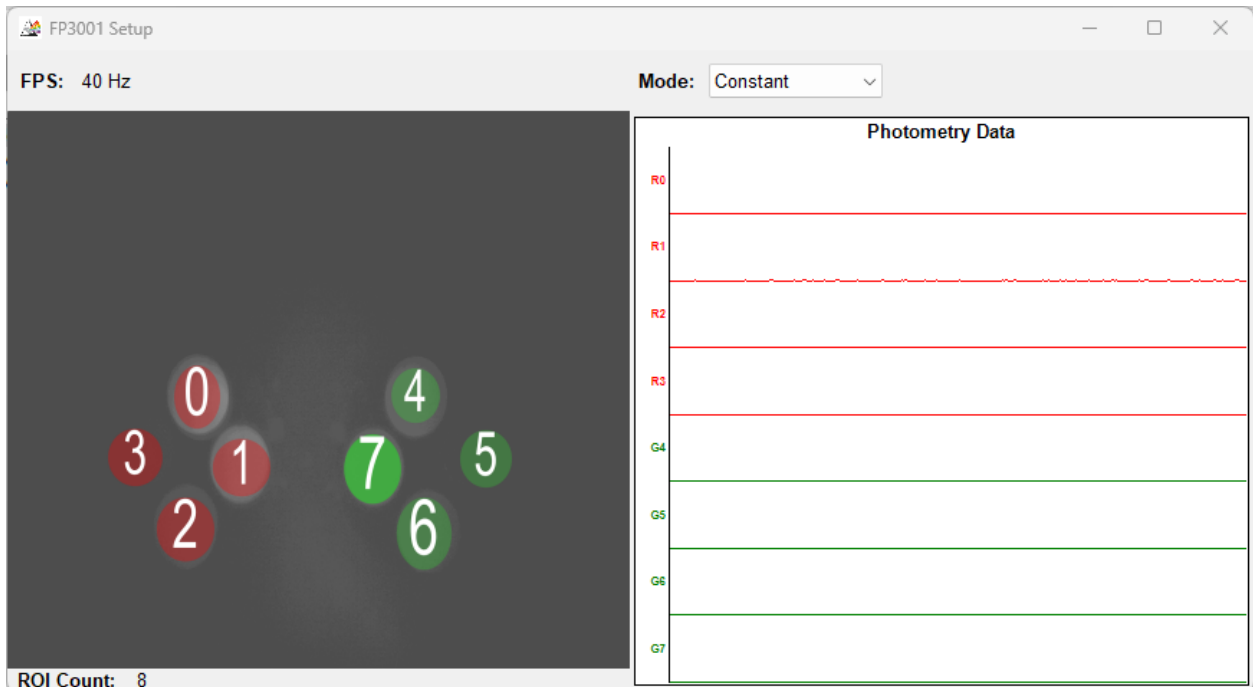


Once the serial number is specified, double click the FP3001 node to open a calibration window.



Begin data acquisition on the driver box by toggling the play/pause switch to play. The calibration window should populate a running plot of photometry data. There will be a signal for each ROI specified, and if none are specified the signal will represent the pixel average of the whole image.

You will notice that there are 2 images of your patch cord reaching the camera. The emission light coming into the system from the patch cord is split by wavelength to yield red and green channels. These two channels are then projected onto opposing sides of the camera, allowing you to record red and green sensors simultaneously.



Below are settings to configure in the FP3001 node. **\*\*\*Please ensure that the parameters entered in the FP3001 node match the parameters on the driver box. A mismatch will result in incorrectly saved data\*\*\*.**

1. **Connect your patch cord** and ensure that it is in focus, such that the ends of the patch cord are visible.
2. **Draw ROIs:** Draw regions of interest in the desired channels and over the desired fibers of the patch cord. It is only necessary to draw ROIs for the recording channels in use (i.e. no need to draw red ROIs if you are only recording GCaMP), and the branches in use. If you are only using 2 or 4 branches, it is only necessary to draw ROIs for those 2 branches. You will want the ROI to be just slightly smaller than the illuminated outline of each patch cord branch. Below are the controls for interacting with the regions of interest in this window:
  - Left-Click + Move: If no region of interest is selected, this will draw a new region of interest. If a region of interest is selected, then this will move the region to the desired location.
  - Right-Click + Move: If a region of interest is selected, this will re-size the region to the desired dimensions.



- Left-Click: Selects a region of interest.
- Tab: Iteratively selects a region of interest.
- Del: Deletes the selected region of interest.
- CTRL: When drawing or resizing a region, this will ensure that the region encompasses a circular region of pixels on the camera.

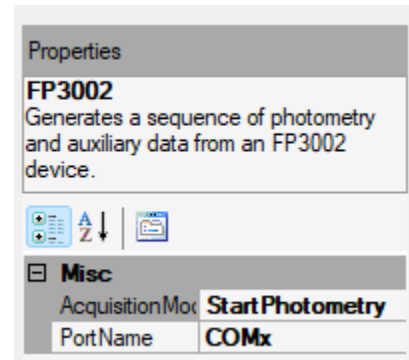
Whenever drawing ROIs, double check that the “ROI Count” in the bottom left of the “FP3001 Setup” window matches the desired number of drawn ROIs. If there are more signals than visible ROIs, then a small ROI has been accidentally drawn. You can correct this by using the “Tab” key within the “Regions” window to cycle through ROIs to select the unintended ROI. Then press the “Del” key to delete the extra ROI. Click and drag to create a circle. Branch.

3. **Specify FPS:** Ensure that the “FPS” property in the “FP3001 Setup” window matches the “FPS” set on the driver box (using the slider on the right hand side). This removes the need to specify the camera’s exposure time in previous versions of the software by automatically calculating the maximum exposure time based on the “FPS”.
4. **Specify Mode:** Ensure that the “Mode” property in the “FP3001 Setup” window matches the mode specified on the driver box. This property handles how frame flags are assigned to new photometry data frames coming from the FP3001 system.

## **FP3002 System**

The “FP3002” node is a source node used to communicate with the FP3002 system. In this node, you will input all of your recording parameters. This node also processes the information coming from the FP3002 system and generates photometry data frames. To configure the settings in this node, the FP system must be connected to the computer and turned on.

To configure the FP3002 node, single click the node. Click the “FP3002” node to open the “Properties” panel on the right side of the Bonsai workflow.



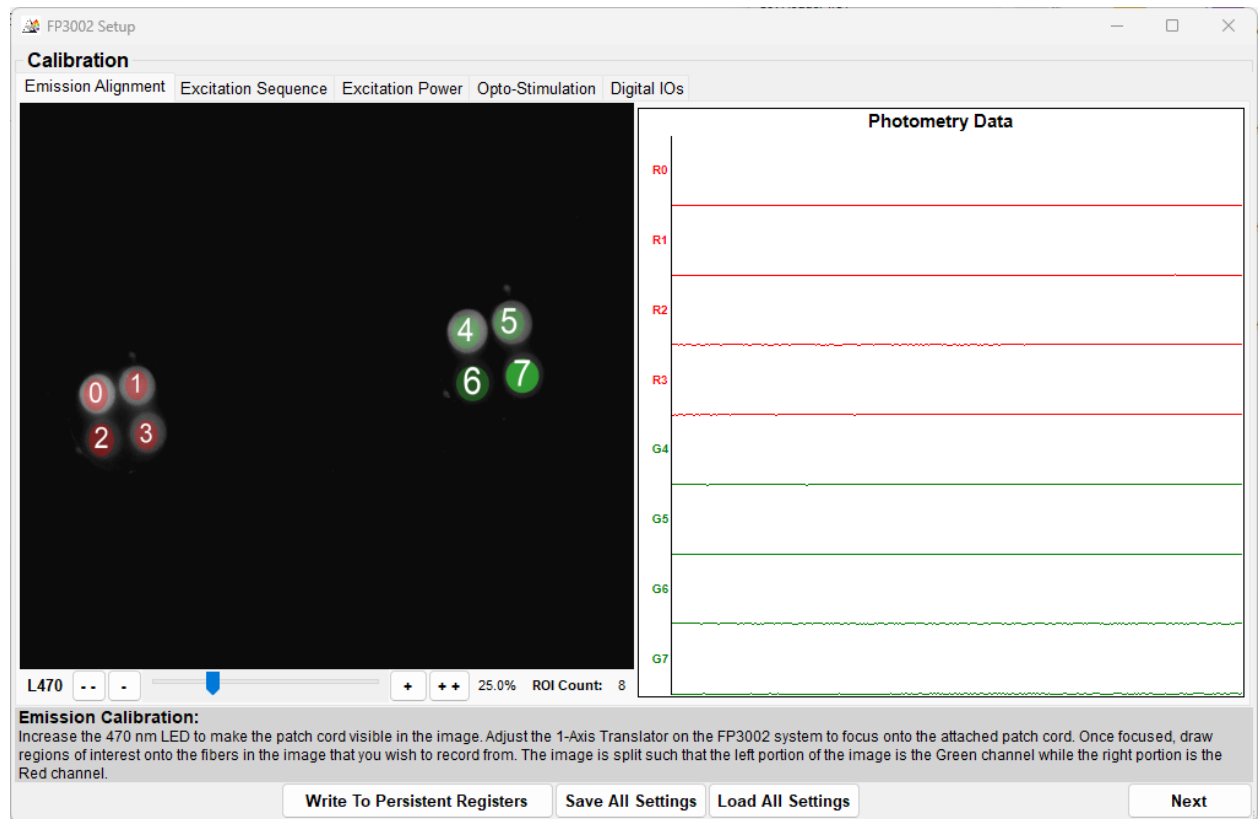
**PortName:** Connect Bonsai to the FP3002 system by setting the value of the “Port Name” property to the correct COM port. Once the system is properly connected, the FP3002 system’s information will populate in Bonsai’s command window. This information will only display if the FP3002 system is powered on and connected via a USB 3.0 port on the computer.

```
Serial Harp device.  
WhoAmI: 2064-ffff  
Hw: 2.0  
Fw: 2.6  
Timestamp (s): 9  
DeviceName: FP3002
```

The information populated here consists of the system’s serial number labeled as “WhoAmI”, the hardware version labeled as “Hw”, the firmware version labeled as “Fw”, and the current system timestamp in seconds since the system powered on.

**AcquisitionMode:** Specify the initial state of the FP3002 system by setting the value of the “Acquisition Mode”. The “StartPhotometry” option will cause the system to be acquiring data when the workflow is started. However, the “StopPhotometry” option will cause the system to await a “Start Acquisition” command to begin data acquisition.

Once the FP3002 node properties are set, double click the node to open the FP3002 setup window. This is where you will configure your recording parameters.



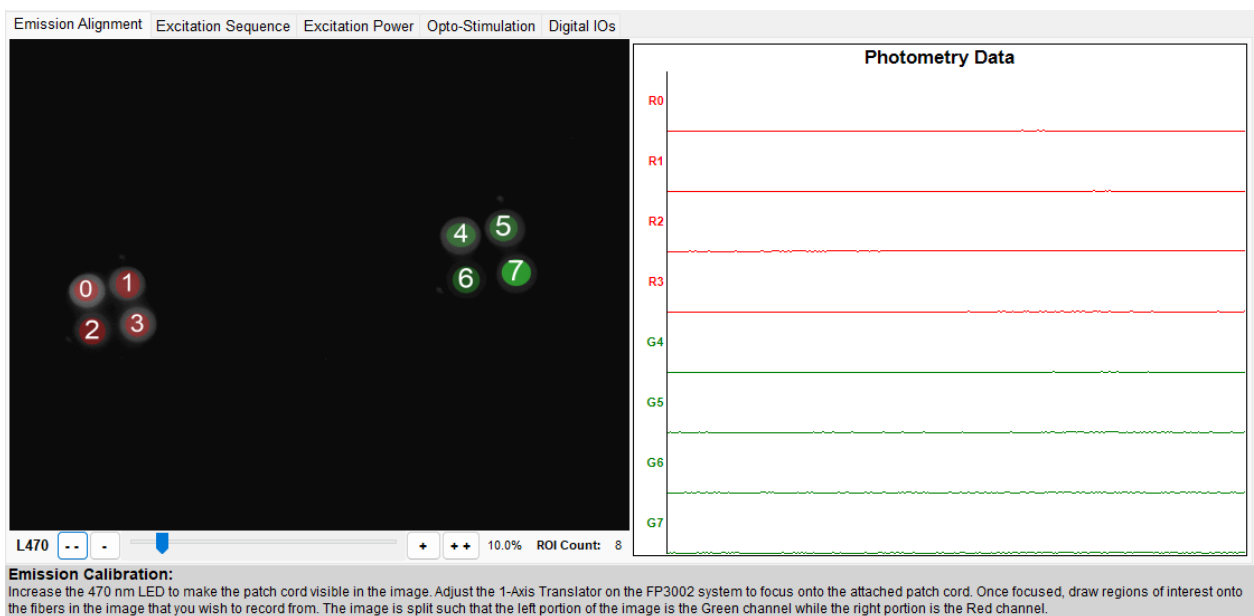
There are five tabs within the “FP3002 Setup” window; “Emission Alignment”, “Excitation Sequence”, “Excitation Power”, “Opto-Stimulation”, and “Digital IOs”. At the bottom of the window are buttons for navigating the setup window, saving/loading settings, and writing settings to persistent registers on the system. Settings will be saved automatically when you navigate between tabs. It is not necessary to click “Save All Settings” on each tab.

Below we walk through configuring settings on each tab.

## **Emission Alignment:**

The “Emission Alignment” tabpage is broken up into two parts. The left side consists of a video feed from the internal camera as well as a slider for controlling the power of the 470nm LED. The right side consists of the photometry data coming from each user-defined region of interest. This tabpage is used to align the patch cord and define regions of interest. These regions of interest will be used to indicate to the “FP3002” node which pixels are grouped together for averaging.

You will notice that there are 2 images of your patch cord reaching the camera. The emission light coming into the system from the patch cord is split by wavelength to yield red and green channels. These two channels are then projected onto opposing sides of the camera, allowing you to record red and green sensors simultaneously. The software will automatically detect which ROIs correspond to green and red emission channels.



To begin the emission alignment, adjust the 470nm LED using the “L470” slider to increase visibility. Then adjust the 1-Axis translator on the FP3002 system, until the branches of the patch cord are visible and in focus. Then use the controls below to draw and adjust the regions of interest. It is only necessary to draw ROIs for the recording channels in use (i.e. no need to draw red ROIs if you are only recording GCaMP), and for the patch cord branches in use. For example, if you are only using 2 of 4 branches, it is only necessary to draw ROIs for 2 branches.

- Left-Click + Move: If no region of interest is selected, this will draw a new region of interest. If a region of interest is selected, then this will move the region to the desired location.
- Right-Click + Move: If a region of interest is selected, this will re-size the region to the desired dimensions.
- Left-Click: Selects a region of interest.
- Tab: Iteratively selects a region of interest.
- Del: Deletes the selected region of interest.
- CTRL: When drawing or resizing a region, this will ensure that the region encompasses a circular region of pixels on the camera.

Whenever drawing regions of interest, double check that the “ROI Count” below the image matches the desired number of drawn regions of interest. If there are more signals than visible ROIs, then a small ROI has been accidentally drawn. You can correct this by using the “Tab” key within the “Regions” window to cycle through ROIs to select the unintended ROI. Then press the “Del” key to delete the extra ROI.

## Excitation Sequence:

The “Excitation Sequence” tabpage is used to specify the frame rate of data acquisition as well as the cycle of excitation LEDs used during data acquisition.

Index	LED	Flag
0	470 nm	2
1	560 nm	4
2	415 nm	1

Excitation Sequence Calibration:  
Set the order of the LEDs in a given trigger sequence. Their associated "Frame Flag" values will appear next to the LED names.

In order to set the excitation sequence, use the table on the left to specify which LEDs will be used and the order of the LEDs in the sequence.

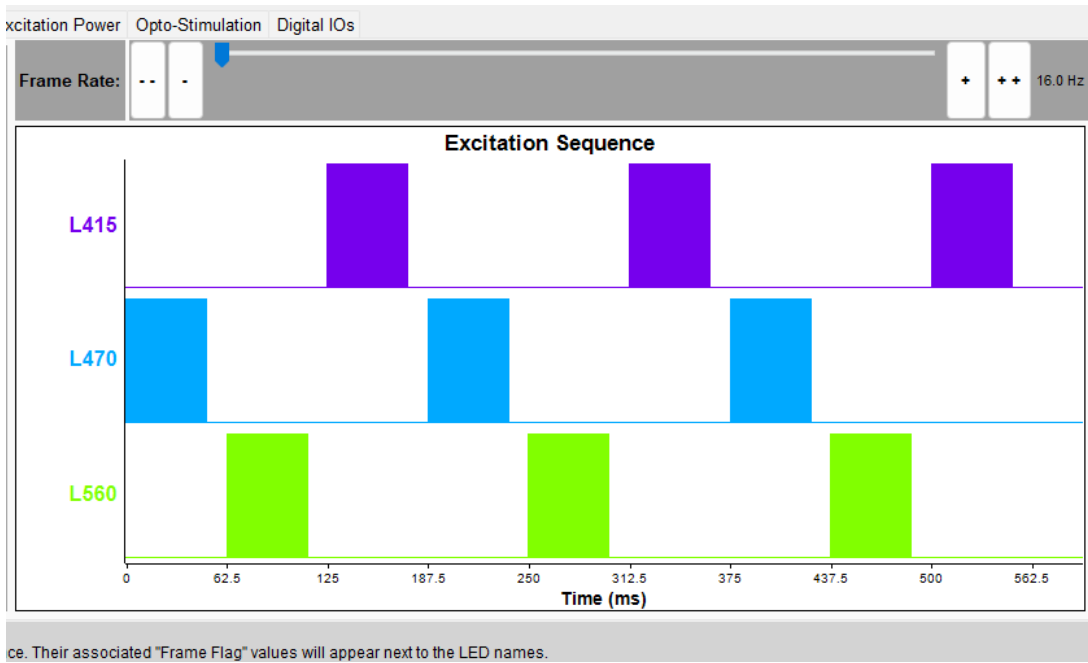
Index	LED	Flag
0	470 nm	2
1	560 nm	4
2	415 nm	1
	None	
	None	
	415 nm	
	470 nm	
	560 nm	

Remove Selected

To change an LED, select it, click the drop-down arrow, and select the new desired LED. To add a new LED, select the bottom empty LED cell, click the drop-down arrow, and select the desired LED. To remove an LED, select it, and click the “Remove Selected” button.

You will notice that “None” is an option in the LED sequence. During none frames, the camera will still capture a frame, but no LEDs will turn on. None frames can be added to a trigger sequence to reduce the duty cycle of the LEDs to reduce photobleaching in longer (greater than a few hours) recordings. Including None frames is not necessary for most experiments.

After configuring the table on the left, specify the frame rate at the top of the tabpage either by using the slider, or typing it in. Whenever the frame rate is changed or an LED is changed, added, or removed, the “Excitation Sequence” plot on the right will be updated. This provides a visual for the timing of the LEDs.

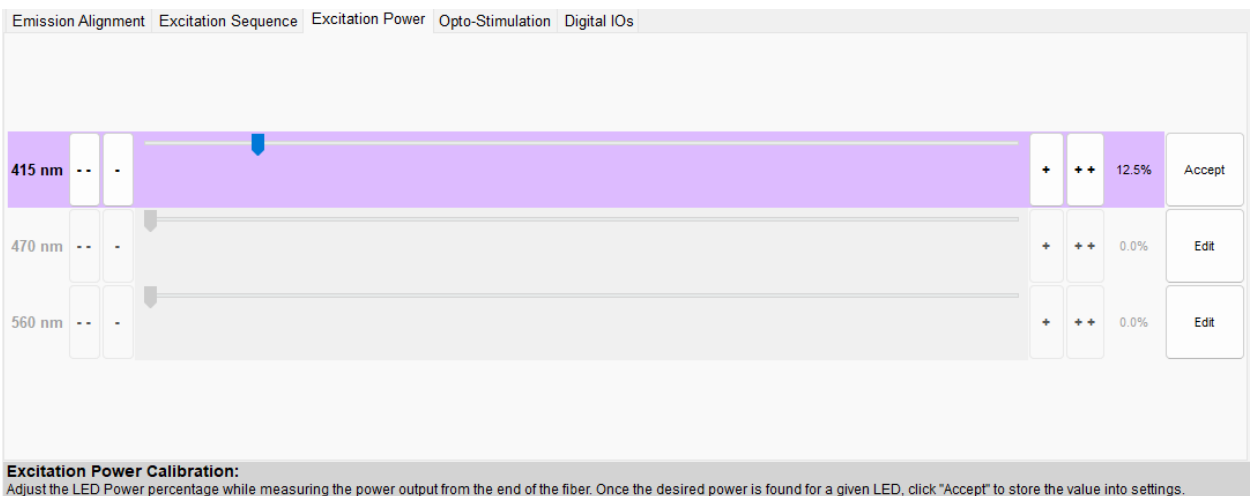


## Excitation Power:

The “Excitation Power” tabpage is used in conjunction with an external power meter to set the appropriate LED power percentage for an experiment. The power coming out of a single fiber of a patch cord should be high enough to record activity and low enough to limit photobleaching of the region of the brain that is being observed. Generally, we recommend  $50\mu\text{W}$  for  $200\mu\text{m}$  fibers and approximately  $120\mu\text{W}$  for  $400\mu\text{m}$  fibers, to start. Whenever possible, use the lowest light powers needed to record a signal. These recommendations are valid for experiments under one hour. For longer experiments, consider lowering the duty cycle of the LEDs and/or lowering the LED powers.



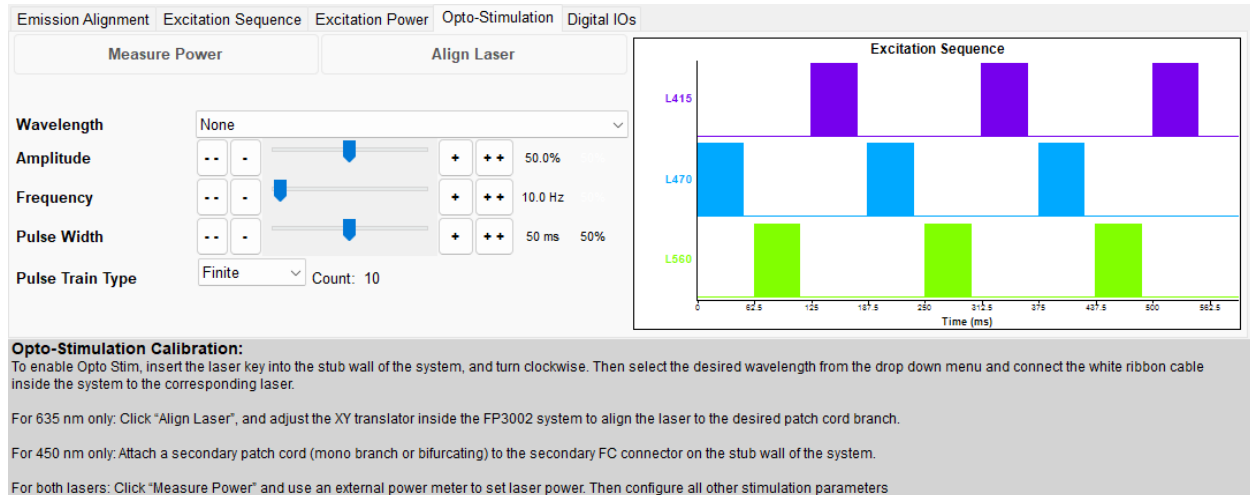
Calibrate each LED by clicking the “Edit” button, adjusting the amplitude until the desired output power is reached, then clicking the “Accept” button to turn off the LED. You can either use the sliders and buttons to adjust LED power, or type in the values. Be sure to change the wavelength on your power meter when switching between LEDs, and that only one LED is on at a time while you are measuring.





## **Opto-Stimulation**

The “Opto-Stimulation” tabpage allows for the configuration of laser parameters for optogenetic stimulation.



The screenshot displays the 'Opto-Stimulation' tab within a software interface. At the top, there are navigation tabs: 'Emission Alignment', 'Excitation Sequence', 'Excitation Power', 'Opto-Stimulation', and 'Digital IOs'. Below these are two buttons: 'Measure Power' and 'Align Laser'. The main configuration area includes a 'Wavelength' dropdown menu set to 'None', and sliders for 'Amplitude' (set to 50.0%), 'Frequency' (set to 10.0 Hz), and 'Pulse Width' (set to 50 ms). The 'Pulse Train Type' is set to 'Finite' with a 'Count' of 10. To the right, a graph titled 'Excitation Sequence' shows a timeline from 0 to 662.5 ms. The graph has three horizontal lines representing different wavelengths: L415 (top, purple), L470 (middle, blue), and L560 (bottom, green). The L415 line shows three purple pulses at approximately 125 ms, 312.5 ms, and 487.5 ms. The L470 line shows three blue pulses at approximately 62.5 ms, 250 ms, and 437.5 ms. The L560 line shows three green pulses at approximately 62.5 ms, 250 ms, and 437.5 ms.

**Opto-Stimulation Calibration:**  
To enable Opto Stim, insert the laser key into the stub wall of the system, and turn clockwise. Then select the desired wavelength from the drop down menu and connect the white ribbon cable inside the system to the corresponding laser.

For 635 nm only: Click “Align Laser”, and adjust the XY translator inside the FP3002 system to align the laser to the desired patch cord branch.

For 450 nm only: Attach a secondary patch cord (mono branch or bifurcating) to the secondary FC connector on the stub wall of the system.

For both lasers: Click “Measure Power” and use an external power meter to set laser power. Then configure all other stimulation parameters

Begin by specifying the “Wavelength” property. If the wavelength is 450 nm, then the “Measure Power” button will enable. If the wavelength is 635 nm, then both the “Measure Power” and “Align Laser” buttons will be enabled. Additionally, once the “Wavelength” property is specified, then the laser pulse train will appear in the “Excitation Sequence”

**Insert and turn the laser key to enable the laser. Please note that if you are using the 635 nm laser, light will come out of the main patch cord attached to the translator. If you are using the 450 nm laser, light will come out of the FC connector on the front of the system.**

Measure Power: Upon clicking the “Measure Power” button, the laser will run in constant mode at the specified amplitude. Adjust the “Amplitude” property until the desired output power is reached, as measured on an external power meter. Once complete, click the “Stop Measuring Power” button.

Emission Alignment   Excitation Sequence   Excitation Power   Opto-Stimulation   Digital IOs

**Stop Measuring Power**   **Align Laser**

Wavelength: 450nm

Amplitude: 50.0%

Frequency: 10.0 Hz

Pulse Width: 50 ms 50%

Pulse Train Type: Finite Count: 10

**Opto-Stimulation Calibration:**  
 To enable Opto Stim, insert the laser key into the stub wall of the system, and turn clockwise. Then select the desired wavelength from the drop down menu and connect the white ribbon cable inside the system to the corresponding laser.

For 635 nm only: Click "Align Laser", and adjust the XY translator inside the FP3002 system to align the laser to the desired patch cord branch.

For 450 nm only: Attach a secondary patch cord (mono branch or bifurcating) to the secondary FC connector on the stub wall of the system.

For both lasers: Click "Measure Power" and use an external power meter to set laser power. Then configure all other stimulation parameters

Align Laser: Upon clicking the "Align Laser" button, a low power, low duty cycle laser pulse train will begin and the image from the camera will appear instead of the "Excitation Sequence" plot. Adjust the 2-Axis translator for the 635nm laser until it is aligned with the desired fiber of the patch cord. You can also adjust the laser amplitude from 0%-30% during this phase, as well as the 470 nm LED to make the fibers and laser pulse train more visible.

Emission Alignment   Excitation Sequence   Excitation Power   Opto-Stimulation   Digital IOs

**Measure Power**   **Stop Aligning Laser**

Wavelength: 635nm

Amplitude: 10.0%

Frequency: 10.0 Hz

Pulse Width: 50 ms 50%

Pulse Train Type: Finite Count: 10

L470 12.5% ROI Count: 8

## Digital IOs

The “Digital IO” tabpage allows users to configure the functionality of the Digital IO ports on the FP3002 system.

**Digital IOs**

Digital Input 0: Event: Rising

Digital Input 1: Event: Rising

Digital Output 0: Software

Digital Output 1: Both

**Miscellaneous**

Screen Brightness: [Slider]

Clock Configuration: This Device

**Digital Inputs**

**Events:** Samples a +5V TTL signal, generating timestamps on the specified type of voltage change.  
**Control:** Allows a +5V TTL signal to control data acquisition of the FP3002 system and/or an external camera.  
**Start Stimulation:** Allows a +5V TTL signal to start a finite or continuous pulse train.

**Digital Output 0**

**Software:** The system will output a +5V TTL signal generated within Bonsai, using the Digital Output Node.  
**Strobe:** Outputs a +5V TTL signal that will be HIGH while the camera is exposing, and LOW between frames.  
**Trigger State:** Outputs a +5V TTL will be HIGH while an LED is ON and LOW while an LED is OFF.

**Digital Output 1**

**Both:** Internal signal used to control the laser will be sent to both the laser and the Digital Output 1 port.  
**BNC:** Internal signal will be sent to the Digital Output 1 port only. Timing can be configured by changing laser settings.  
**Laser:** Internal signal will be sent to the laser only.

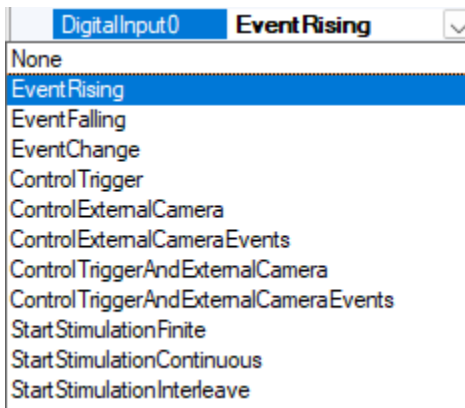
**Digital IOs Calibration:**

Digital IOs- Use drop down menus to configure how the system will send and receive TTL pulses. Note: Digital Output 1 is yoked to laser function and

Screen Brightness- Use slider to change LCD screen brightness.

Clock Configuration- Use drop down menu to select the master clock to use for photometry data. The system will default to the internal clock. If using an external device as the master clock, set clock configuration to 'External Device'.

“Digital Input 0/1”: Specifies how the FP3002 system handles +5V TTL signals on the Digital Input ports. There are a total of 12 options for configuring these input ports.



**None:** Nothing will occur when there is a TTL signal on the Digital Input port.

**Event Rising:** Sends a Harp Message to Bonsai indicating an event occurred whenever the +5V TTL signal changes from LOW to HIGH.

**Event Falling:** Sends a Harp Message to Bonsai indicating an event occurred whenever the +5V TTL signal changes from HIGH to LOW.

**Event Change:** Sends a Harp Message to Bonsai indicating an event occurred whenever the +5V TTL signal changes from LOW to HIGH or HIGH to LOW.

**Control Trigger:** Allows the +5V TTL signal to control data acquisition of the FP3002 system. While the TTL signal is HIGH, the system will be acquiring data frames and while the TTL signal is LOW, the system will stop acquiring data frames.

**Control External Camera:** Allows the +5V TTL signal to control data acquisition of an external camera. While the TTL signal is HIGH, the external camera will be acquiring data frames and while the TTL signal is LOW, the external camera will stop acquiring data frames.

**Control External Camera Events:** Allows the +5V TTL signal to control data acquisition of an external camera. While the TTL signal is HIGH, the external camera will be acquiring data frames and while the TTL signal is LOW, the external camera will stop acquiring data frames. In addition, it causes the FP3002 system to send Harp events to Bonsai with every external camera exposure.

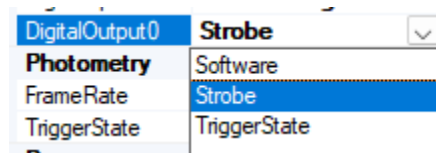
**Control Trigger and External Camera:** Allows the +5V TTL signal to control data acquisition of both the FP3002 system and an external camera. While the TTL signal is HIGH, the FP3002 system and the external camera will be acquiring data frames and while the TTL signal is LOW, the FP3002 system and the external camera will stop acquiring data frames.

**Control Trigger and External Camera Events:** Allows the +5V TTL signal to control data acquisition of both the FP3002 system and an external camera. While the TTL signal is HIGH, the FP3002 system and the external camera will be acquiring data frames and while the TTL signal is LOW, the FP3002 system and the external camera will stop acquiring data frames. In addition, it causes the FP3002 system to send Harp events to Bonsai with every change in camera exposure state.

**Start Stimulation Finite:** While the +5V TTL signal is HIGH, the FP3002 system will begin stimulation with a finite duration pulse train. The laser will pulse the number of times specified with the “Pulse Count” property.

**Start Stimulation Continuous:** While the +5V TTL signal is HIGH, the FP3002 system will begin stimulation with a continuous pulse train. The pulse train will possess all of the parameters specified in the “Stimulation Pulse”, except the pulse count will be infinite. This pulse train will continue until the TTL signal returns LOW.

“Digital Output 0” - Specifies how the FP3002 system outputs +5V TTL signals from the Digital Output 0 port. There are three options for the digital output 0 port.



**Software:** Specifies that the digital output signal will be generated within Bonsai using a software trigger and the “Digital Output” node, then sent to the system through the “FP3002” node.

**Strobe:** Specifies that the camera's strobe will be sent out of the digital output 0 port such that the port will have a HIGH value while the internal camera is exposing, and a LOW value during the internal camera’s dead time.

**Trigger State:** Specifies that the digital output signal will be HIGH while an LED is ON and LOW while an LED is OFF.

“Digital Output 1” - Specifies whether the digital output pin 1 state is routed to the BNC port, internal laser, or both. In experiments involving opto-stimulation, this property should be set to “Both” so that the laser can be triggered and so that the “Digital IOs” node can be used to record and timestamp the laser state. Otherwise, this property should be set to “BNC” so that the Digital Output 1 port can be used to output a TTL without triggering the internal laser.

“Clock Synchronizer”: Specifies whether the FP3002 system outputs its own clock line, or synchronizes to an external clock. In most experiments this property is set to “ThisDevice”.

## Load / Save:

The “Load All Settings” and “Save All Settings” buttons are located at the bottom of the “FP3002 Setup” window. These buttons will load/save the FP3002 settings from/to an XML file. The XML file itself can be referenced during post-hoc analysis to verify the system settings used during a particular experiment.

```
<?xml version="1.0" encoding="utf-8"?>
<Settings>
  <DeviceInformation>
    <Id>2064-00cd</Id>
    <HardwareVersion>2.0</HardwareVersion>
    <FirmwareVersion>2.6</FirmwareVersion>
    <SerialNumber>205</SerialNumber>
  </DeviceInformation>
  <Configuration>
    <ClockSynchronizer>ThisDevice</ClockSynchronizer>
    <ScreenBrightness>0</ScreenBrightness>
  </Configuration>
  <Emission>
    <Region X="128" Y="503" Width="52" Height="64">R0</Region>
    <Region X="216" Y="491" Width="42" Height="54">R1</Region>
    <Region X="867" Y="395" Width="46" Height="42">G2</Region>
    <Region X="952" Y="373" Width="56" Height="66">G3</Region>
  </Emission>
  <Excitation>
    <TriggerPeriod Reg="9900">101.010 Hz</TriggerPeriod>
    <Sequence>
      <LED Index="0" Flag="2">L470</LED>
      <LED Index="1" Flag="4">L560</LED>
      <LED Index="2" Flag="1">L415</LED>
      <LED Index="3" Flag="0">None</LED>
      <LED Index="4" Flag="0">None</LED>
    </Sequence>
    <Power>
      <L415 Reg="10112">2.000%</L415>
      <L470 Reg="10368">3.000%</L470>
      <L560 Reg="10624">4.000%</L560>
    </Power>
  </Excitation>
  <OptoStimulation>
    <LaserWavelength>450</LaserWavelength>
    <LaserAmplitude Reg="11140">16.999%</LaserAmplitude>
    <StimPeriod Reg="32">31.250 Hz</StimPeriod>
    <StimOn Reg="4">4 ms</StimOn>
    <StimReps>34</StimReps>
  </OptoStimulation>
  <DigitalIOs>
    <DigitalOutput0>Strobe</DigitalOutput0>
    <Output1Routing>InternalLaser</Output1Routing>
    <DigitalInput0>EventFalling</DigitalInput0>
    <DigitalInput1>EventFalling</DigitalInput1>
  </DigitalIOs>
</Settings>
```

**Write to Persistent Registers:**

The “Write To Persistent Registers” button is also located at the bottom of the “FP3002 Setup” window. This button is used to store the configured FP3002 settings to the system’s persistent registers. This means that upon powering on the system, these settings will be loaded into the system.

# Chapter 3: Synchronization

---

This chapter explores methods for synchronizing and temporally aligning various data streams to fiber photometry signals. The method used for synchronization depends on the experimental design. In this chapter, we present three techniques for synchronization: Software, Hardware, and Machine Vision.

The software synchronization technique is used to synchronize parallel data streams (i.e behavior cameras, Arduinos) within Bonsai. This is the most commonly used technique, however, it requires all data sources to have support within Bonsai.

When trying to synchronize with data sources without Bonsai support (i.e. behavior hardware running on external software), there are methods of hardware synchronization where TTL signals can either be sent through the FP3002 system into Bonsai or through a DAQ/Arduino into Bonsai. For this technique, external hardware must be capable of outputting +5V TTL signals.

For cases in which a data source is not supported by Bonsai and does not output a +5V TTL signal, we present a method for synchronization utilizing machine vision. Here we use machine vision techniques to record the states of LED indicators that can be found on many devices.

With these three techniques, most devices can be synchronized with the fiber photometry data stream using Bonsai. However, if a device is unsupported by Bonsai, does not have +5V TTL signals, and possesses no LED indicators, then more complex hardware synchronization methods will have to be implemented.



## Software

In many experiments, data is generated by multiple devices. When these devices have support within Bonsai, their data streams can be easily synchronized using built-in Bonsai nodes. Let's begin our discussion with the standard photometry workflow.

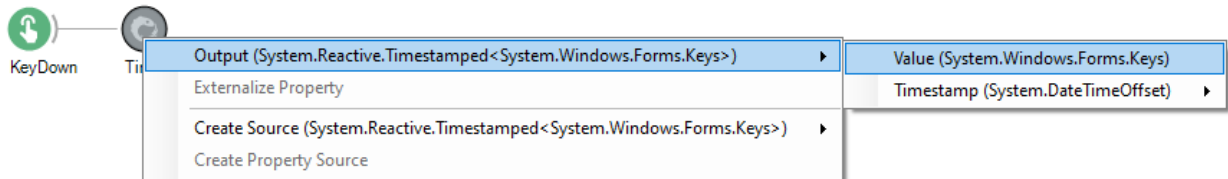


The .csv file generated by the “Photometry Writer” contains timestamps from two separate clocks: the system’s internal clock and the computer’s clock. The system’s internal clock has greater precision than most computer clocks and is unaffected by stress on the computer. However, other data streams do not have immediate access to the system’s internal clock. Meanwhile, the computer’s clock can be shared by multiple asynchronous data streams within Bonsai. The computer timestamp recorded in the output .csv file utilizes the computer’s “Time of Day, Total Milliseconds” timestamp. So by generating these timestamps for all asynchronous data streams within Bonsai, the photometry data can easily be aligned to all other data streams.

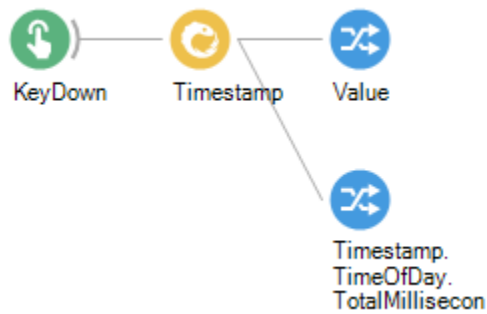
Let's show this process with a keystroke data stream. First, create a keystroke source node using the “KeyDown” node.



Connect the “KeyDown” node to a “Timestamp” node. Then right click the “Timestamp” node and output the incoming value.

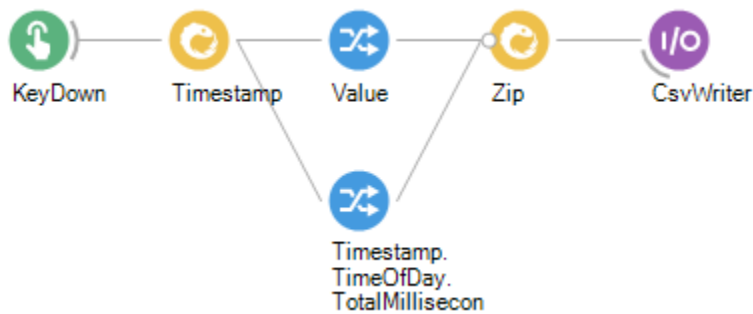


Then output the “TimeOfDay.TotalMilliseconds” timestamp.

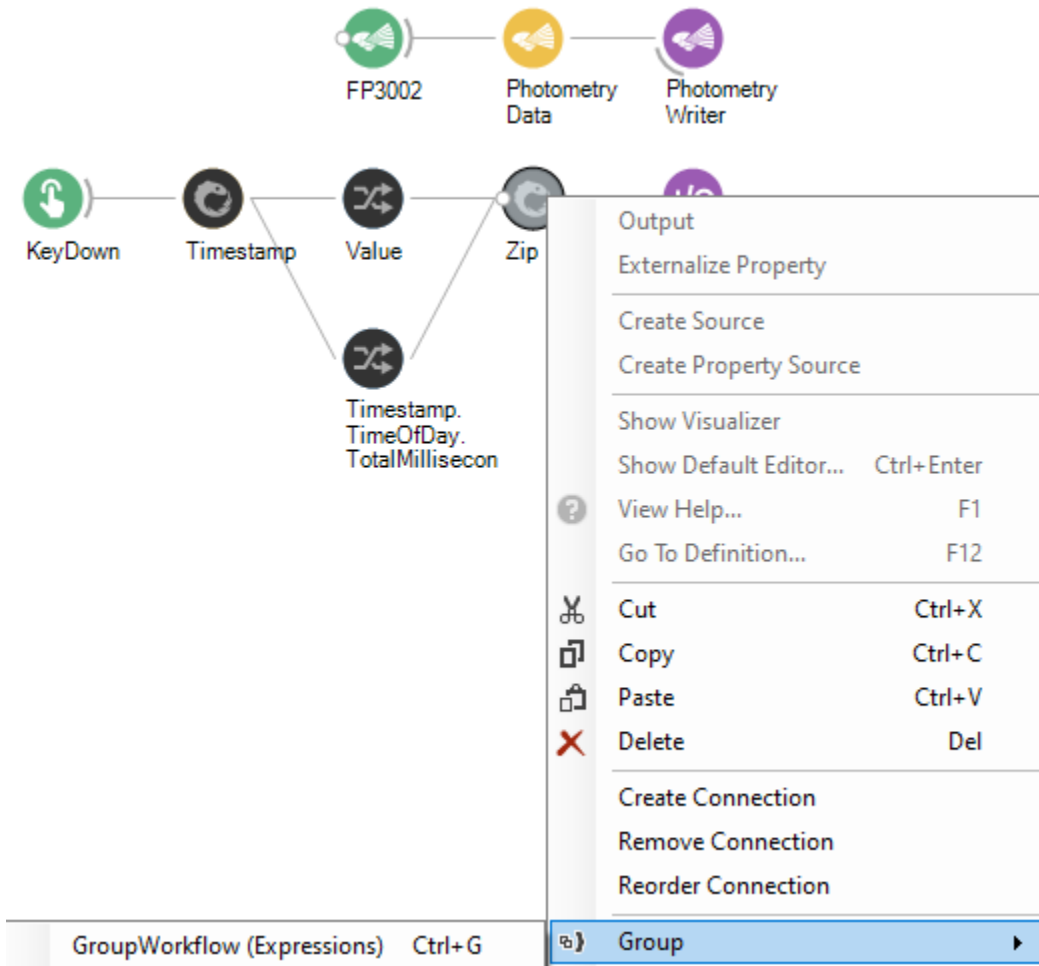


Then use the “Zip” node to combine the two outputs and connect it to a “Csv Writer”.

This will save a column for the key pressed and a column containing the computer’s timestamp which will be comparable to the computer timestamp in the .csv file created by the “Photometry Writer” node.



Here we will group together the computer timestamp nodes again to add a level of organization. Select all of the nodes used to generate a computer timestamp, then right click and select “Group”.



This will group the nodes into a single workflow element that you can provide a name and description for. You can still access the grouped nodes by double clicking the grouped workflow element.

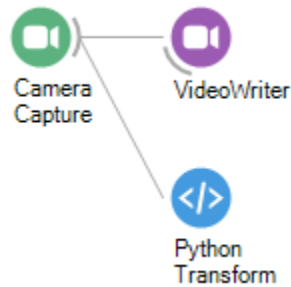
The screenshot shows a workflow editor interface. On the left, under the 'Workflow' tab, there are two rows of nodes connected by lines. The top row consists of 'FP3002' (green icon), 'Photometry Data' (yellow icon), and 'Photometry Writer' (purple icon). The bottom row consists of 'KeyDown' (green icon), 'Computer Timestamp' (grey icon), and 'CsvWriter' (purple icon). On the right, the 'Properties' panel is open, showing the name 'Computer Timestamp (GroupWorkflow)' and the description 'Generates a computer timestamp for keystroke data.' Below this, there are some icons and a 'Design' section with a table:

Design	
Description	Generates a compute
Name	Computer Timestamp

Next, let's add a behavioral camera data stream. This data stream will have to be treated differently because it makes use of the "Video Writer" node, meaning we cannot simply add a timestamp to the image data.



A common way of synchronizing the behavioral camera data stream is to generate a frame number for every behavioral camera frame, combining a computer timestamp to the generated frame number, and saving the timestamps of every behavioral camera frame number to a .csv file. We will begin this process by connecting the camera's source node to a "Python Transform" node in parallel with the "Video Writer" node.



This "Python Transform" node will contain a python script that will count the number of behavior camera frames that have been generated since the start of the workflow. Declare a global variable to keep track of the number of frames that have occurred, then increment it every frame and return the new value.

```
Python Script

# Global variable for keeping track of the number
# of behavioral camera frames that have passed through
# this script.
count = 0

# Returns a value of type "Integer"
@returns(int)
def process(value):
    global count

    # Every time the behavioral camera passes
    # a value through this script, increment the
    # count variable and return the new count.
    count = count + 1
    return count

OK Cancel
```

We can also provide a name for the python script from the “Python Transform” node’s property panel.

The image shows a workflow editor interface with a 'Workflow' tab on the left and a 'Properties' panel on the right. The workflow consists of several nodes connected by lines:

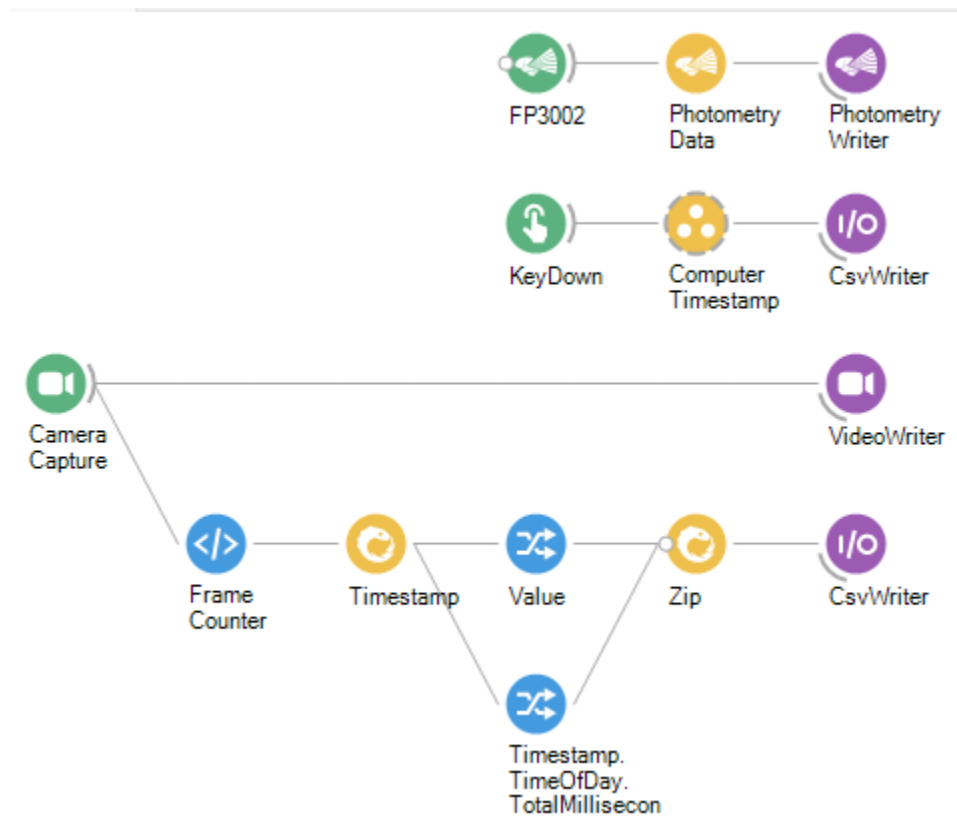
- FP3002 (green icon) connects to Photometry Data (yellow icon), which connects to Photometry Writer (purple icon).
- KeyDown (green icon) connects to Computer Timestamp (yellow icon), which connects to CsvWriter (purple icon).
- Camera Capture (green icon) connects to VideoWriter (purple icon).
- Frame Counter (grey icon with code symbols) is connected to Camera Capture.

The 'Properties' panel for the 'Frame Counter (Python Transform)' node is shown on the right. It includes a description: 'A Python script used to transform each element of the sequence.' Below this, there are sections for 'Design' and 'Misc'.

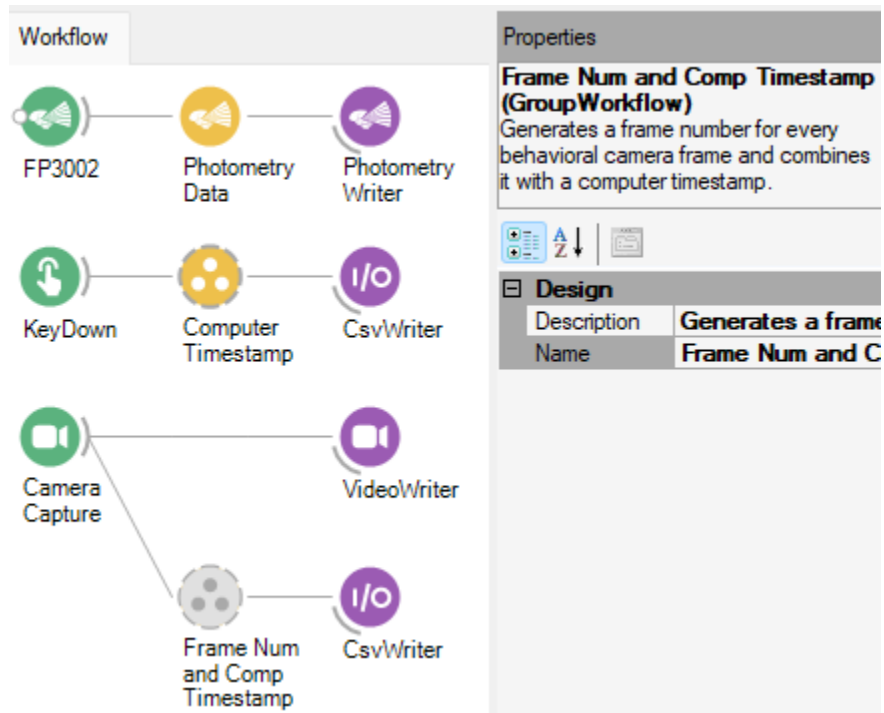
Design	
Description	
Name	Frame Counter

Misc	
Script	# Global variable

To add a timestamp to the behavioral camera's frame number, you can treat it the same as in the keystroke data stream. Add a "Timestamp" node, then output the incoming value and the same timestamp type as the photometry data stream. Then zip the two outputs together and connect to a "Csv Writer" node.



We will add some organization by grouping together the nodes used to produce a behavioral camera frame number and a computer timestamp. Then we will give the grouped workflow a name and a description.



We now have three major data streams. The photometry data stream will save one .csv file containing the photometry data timestamped using both the system's clock and the computer's clock. These two data sets can be aligned by frame number. Then the other two data streams: the behavioral camera and keystroke data stream, are timestamped using the computer's clock, allowing these data sets to be aligned to the timestamped photometry frame number data set.



## Hardware

There are a variety of devices used within fiber photometry experiments that are not currently supported by a NuGet package usable within Bonsai. This adds a level of complexity for synchronizing with the photometry data stream. While designing an experiment there are some key factors that will influence how the hardware will be synchronized. These factors include the following:

- The communication protocol(s) that the external sensors use to communicate with other devices
- The quantity of external sensors used in an experiment.
- Whether or not the external sensors utilize their own software to record data.

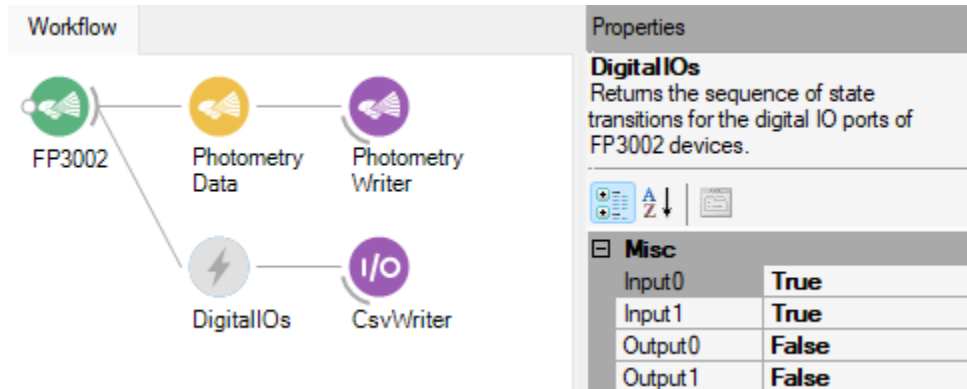
Let's begin with an example experimental design consisting of the following devices:

- One FP3002 system.
- One lever switch that produces a +5V digital voltage when triggered.
- One lickometer that produces a +5V digital voltage when licked.

Here we have two external sensors: the lever switch and the lickometer, both of which communicate with external devices via a +5V digital signal. Finally, neither of these sensors utilize a specific software to record data. In order to synchronize these two external sensors with the FP3002 system, we can utilize the digital input ports. In this case, connect the external sensors to the digital input ports via BNC cables and configure the FP3002 node such that the "Digital Input 0" and "Digital Input 1" settings on the "Digital IOs" tabpage are specified to "Event Change".

Digital IOs	
Digital Input 0	Event: Change
Digital Input 1	Event: Change
Digital Output 0	Software
Digital Output 1	Both

With the digital input ports configured and connected in this way, we can use the “Digital IOs” node to record and timestamp the signals from both external sensors using the system’s internal clock. Be sure to configure the “Digital IOs” node such that both the “Input0” and “Input1” properties are set to “True.”

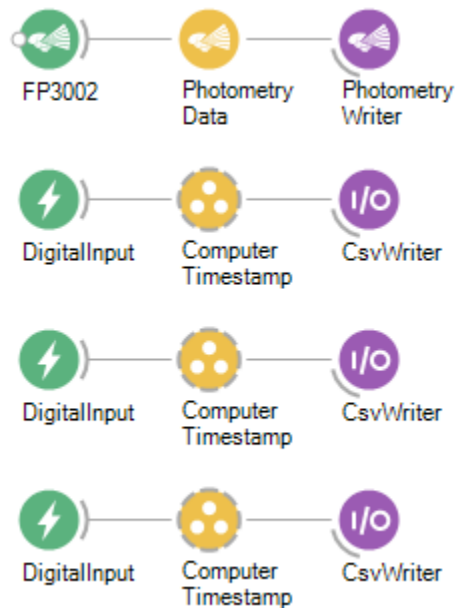


This example showcases that up to two external sensors can be synchronized with the photometry data stream so long as they output +5V digital signals. In the following example, suppose we had an additional level switch. Here we have three external sensors: two lever switches and the lickometer, all of which communicate with external devices via a +5V digital signal. In this case, there are not sufficient digital input ports on the FP3002 system to record the signals from all of the external sensors so our previous solution is no longer valid. For experimental designs consisting of greater than two external sensors that communicate via +5V digital signals, we can utilize an Arduino with the Firmata protocol installed.

Follow the steps outlined below for installing “Standard Firmata” onto an Arduino:

1. Visit the GitHub link: <https://github.com/firmata/arduino>
2. Download Zip (in the "Code" drop down menu in the upper right).
3. Open up Arduino IDE.
4. Click Sketch -> Include Library -> Add .ZIP Library
5. Click on the .ZIP file you just downloaded (should be in the downloads folder)
6. Open the "StandardFirmata.ino" ( Click File -> Examples -> Firmata -> StandardFirmata)
7. Check that the Arduino is connected to the Computer via a USB.
8. Click Tools and double check that the "Board:" is filled with the correct board type (usually Arduino Uno) and that "Port:" is the correct port (should be COMX, where X is a number).
9. Finally, click the Upload button in the Arduino IDE to upload the code onto your Arduino.

This will allow you to connect the external sensors to the Arduino and read their values into Bonsai using the nodes found within the “Bonsai.Arduino” package. With this setup, you can use the “Digital Input (Arduino)” nodes to read in the values from the specified pins on the Arduino then you can synchronize with the photometry data stream using the method described in the “[Synchronize: Software](#)” section.



This method of hardware synchronization is no longer limited to two external sensors. Furthermore, it is no longer limited to digital signals. Arduino's are capable of reading analog signals up to +5V through their analog input pins. If utilizing the analog input pins of an Arduino, be sure to change from a "Digital Input (Arduino)" node to an "Analog Input (Arduino)" node.

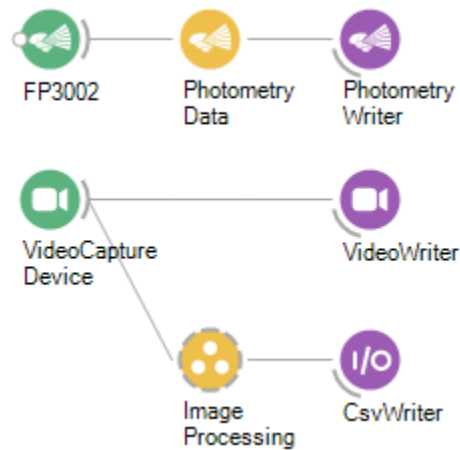
Now let's consider a case where the external sensors do not output a +5V signal such that the FP3002 digital input port and the Arduino are not viable solutions. Suppose we are working with an operant chamber that utilizes a proprietary software for recording the data from its sensors. Many of these operant chambers possess a control box that all of the external sensors connect to, then this control box is connected to the computer to send the data from all of the external sensors to the company's proprietary software. In this case, the data from the external sensors will not be able to be synchronized within Bonsai. However, there is usually a way to synchronize within the proprietary software. Often the control box that connects all of the external sensors also accepts digital input signals. If not, then the company that manufactures the operant chamber and control box combination, also manufactures a DAQ that is compatible with the control box or the proprietary software. In this case, we can output a +5V digital signal from the FP3002 system to either the control box or the DAQ to be synchronized within the proprietary software. The digital output 0 port of the FP3002 system can be configured within the "FP3002 Setup" window to output the camera's strobe signal. To do this, specify the "Digital Output 0" setting to be "Strobe". With this set, the digital output 0 port of the FP3002 system will output a +5V digital signal that is HIGH while the internal camera is exposing and LOW during the internal camera's dead time.

Digital IOs	
Digital Input 0	None
Digital Input 1	None
Digital Output 0	Strobe
Digital Output 1	Both

For more information on the options for configuring the digital input and output ports of the FP3002 system please visit the "[FP3002](#)" entry in the "[Appendix I: Node Glossary](#)"

## Machine Vision

There are times when a device contains no support within Bonsai and is not compatible with +5V TTL signals. Most of the time, these devices have their own proprietary software for data acquisition. This can cause difficulty for synchronizing data sets produced by Bonsai with data sets produced by a different software. These devices tend to have a built-in method for hardware synchronization, however, these hardware designs can quickly become complex. This document discusses a method for synchronizing such devices, relying on the fact that many of these devices have an interface with LED indicators. If a device, not supported by Bonsai, has status LEDs, we can use machine vision techniques within Bonsai to synchronize data streams.

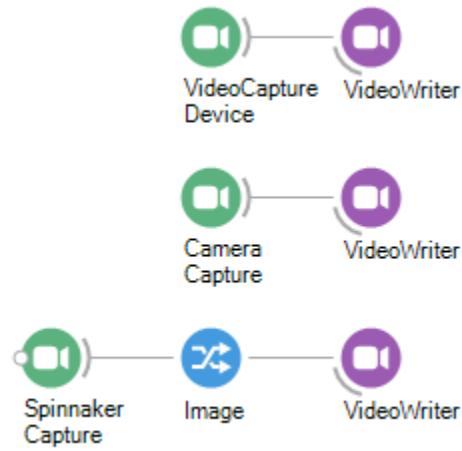


This workflow will timestamp each photometry data frame using the computer's clock, while using basic image processing techniques to read the state of the LED indicators using an external camera. To construct this workflow, begin with the standard photometry workflow.



Now that the photometry data stream is constructed, we can begin constructing the external camera's data stream. There are three commonly used source nodes for connecting to external cameras and producing frames from them. For Spinnaker cameras use the "Spinnaker Capture" node. For DirectShow based capture devices use the "Video Capture Device". Finally for most webcams, the "Camera Capture" node is usable. This workflow will work the exact same way whether using the "Video Capture Device" or the "Camera Capture" nodes. However, the "Spinnaker Capture" node works slightly differently. The output of the "Spinnaker Capture" node is of type "SpinnakerDataFrame" while the other two nodes output elements of type "IplImage". However, the "SpinnakerDataFrame" consists of an "IplImage" and "ChunkData" so the machine vision techniques described in this section can still be used with the "Spinnaker Capture" node if the "IplImage" is selected from the "SpinnakerDataFrame".

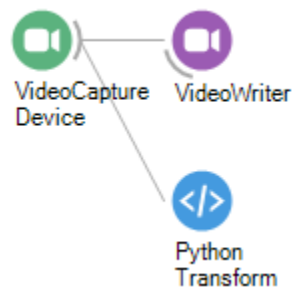
Create an external camera data stream by connecting the desired source node to a “Video Writer” node.



Some configuration is available for the capture nodes and the “Video Writer” node. All three capture nodes have the option to specify the camera’s index. For the “Camera Capture” and “Video Capture Device” nodes, the internal camera on the “FP3002” system will not be recognized, so if only one external camera is connected to the computer, it will appear on index 0. However, the “Spinnaker Capture” node will recognize the internal camera on the FP3002 system, so some care needs to be taken so that the “Spinnaker Capture” and “FP3002” nodes do not try to both access the same camera. The “FP3002” node registers the internal camera when applying a firmware update so we need to only verify that the “Spinnaker Camera” is not trying to access the internal camera. You can do this by specifying the “Index” or “SerialNumber” properties of the “SpinnakerCapture” node.

The “Video Writer” node has a variety of properties that are configurable. Similar to the “Csv Writer” node, be sure to specify the “File Name”, “Overwrite”, and “Suffix” properties. Be sure to include the file extension in the “File Name” and that it matches the “FourCC”. By default “FMP4” will be used as the “FourCC” in order to save an .avi file. Next, set the “Frame Rate” property to the frame rate of the external camera. This will allow the playback of the video to be at the same rate that the camera frames were acquired.

With the source node and the video writer configured, it is time to implement our machine vision algorithm for tracking the state of the LED indicators. We will begin by generating a frame number for each external camera frame. This process will be done in parallel to the “Video Writer” node. Insert a “Python Transform” node and implement a basic counter script that counts the number of camera frames passed through the node.



```
Python Script

# Global variable for keeping track of the number
# of external camera frames that have passed through
# this script.
count = 0

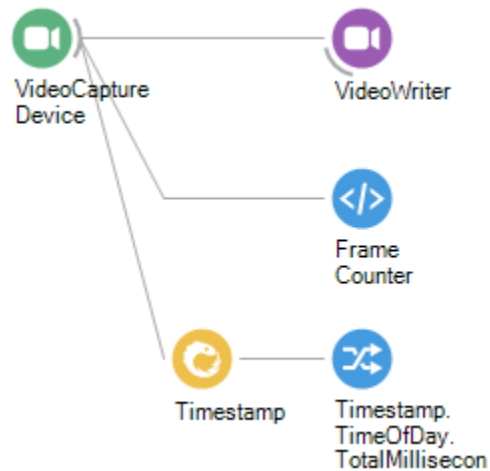
# Returns a value of type "Integer"
@returns(int)
def process(value):
    global count

    # Every time the external camera passes
    # a value through this script, increment the
    # count variable and return the new count
    count = count + 1
    return count

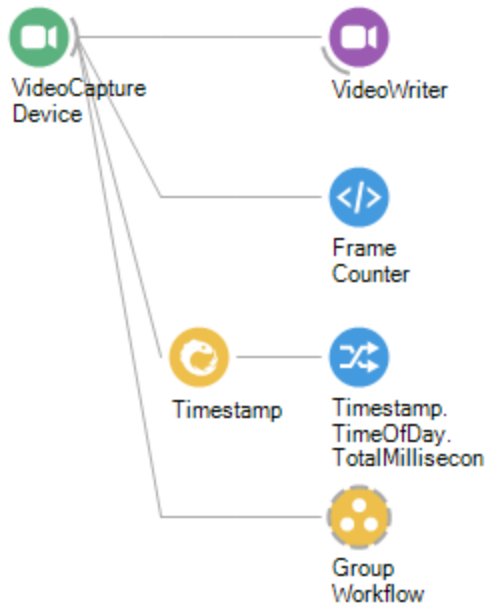
OK Cancel
```



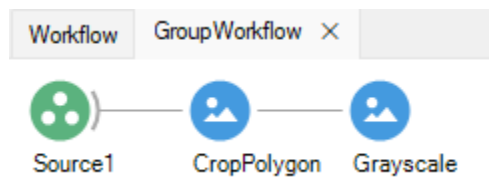
It is useful to rename the “Python Transform” node to indicate its purpose. Here we will rename it to “Frame Counter”. Parallel to the “Frame Counter” node we will implement a computer timestamp so that we can synchronize this data stream with the photometry data stream. Here we will use a “Timestamp” node that only outputs the time of day, total milliseconds timestamp.



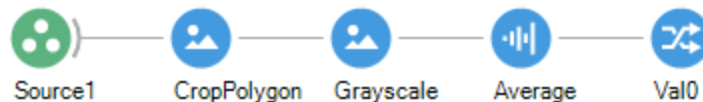
Next we will create the image processing algorithm for tracking the state of a single LED indicator. This algorithm will crop the incoming image to a single LED indicator, convert the cropped image to grayscale, find the average pixel value of the grayscale image, and output a boolean value indicating if the average pixel is greater than a specified value. This concept works because the average pixel value will change significantly when the LED indicator changes state. Begin construction by inserting a “Group Workflow”, opening it by double clicking it, and inserting a “Workflow Input” node. Connect it parallel to the frame counter and the computer timestamp.



Inside of the grouped workflow, begin the image processing algorithm with a “Crop Polygon” node followed by a “Grayscale” node. This will generate a grayscale image, cropped to a single LED indicator.

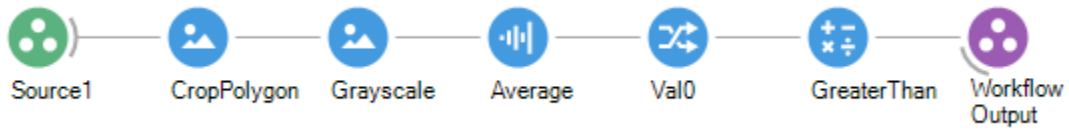


After the “Grayscale” node, add an “Average” node and output the “Val0” element from it. This will average the pixel values and output the blue component. Since the incoming image is grayscale, all color components are equal. Thus, outputting the “Val0” element works to convert the data type from “OpenCV.Net.Scalar” to an integer.



We can complete the image processing algorithm with a “Greater Than” node. This will output True when the average pixel value is greater than the specified value. Otherwise,

it will output False. Be sure to include a “Workflow Output” node at the end of this data stream so that data can exit the grouped workflow.



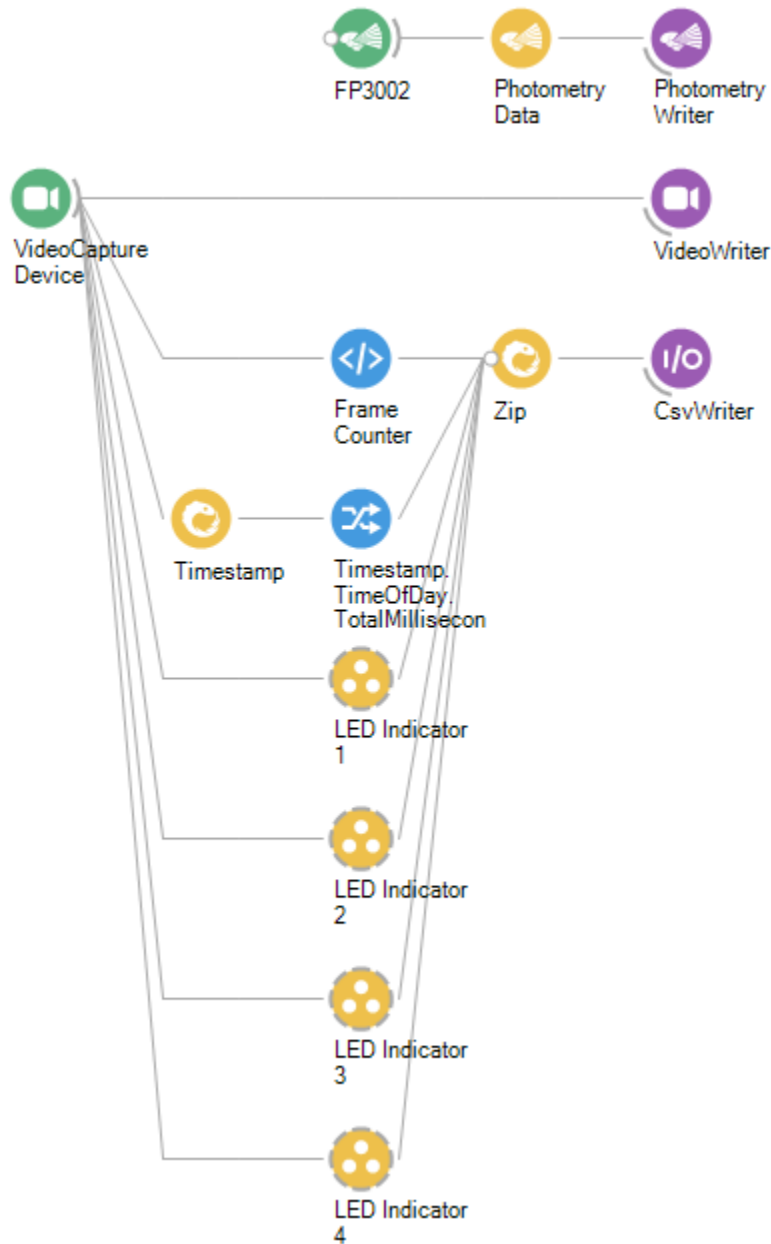
Be sure to give the grouped workflow a unique name and a description.

The screenshot shows a workflow editor interface. On the left, a workflow is visualized with nodes: FP3002, Photometry Data, Photometry Writer, VideoCapture Device, VideoWriter, Frame Counter, Timestamp, and LED Indicator 1. On the right, the Properties panel for 'LED Indicator 1 (GroupWorkflow)' is displayed. It includes a description and a Design table.

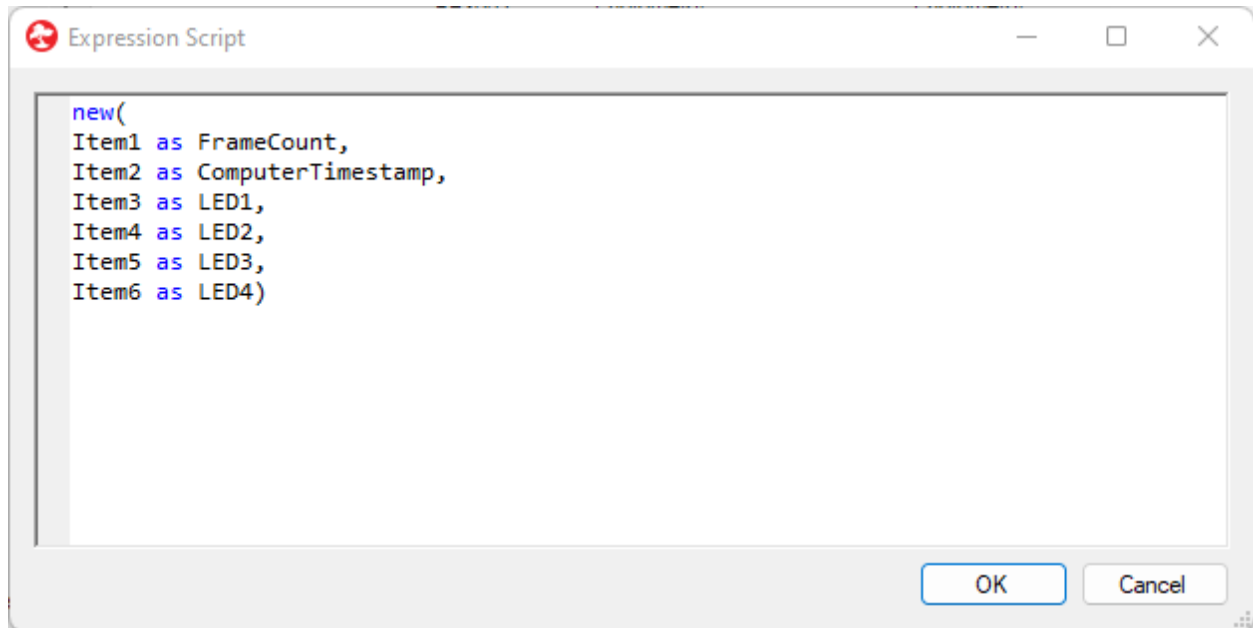
**LED Indicator 1 (GroupWorkflow)**  
Crops the incoming image to a single LED Indicator, converts to grayscale, and finds the average pixel value. Outputs a boolean value indicating if this pixel average is greater than a specified value. Used to find the state of an LED Indicator.

Design	
Description	Crops the incomin
Name	LED Indicator 1

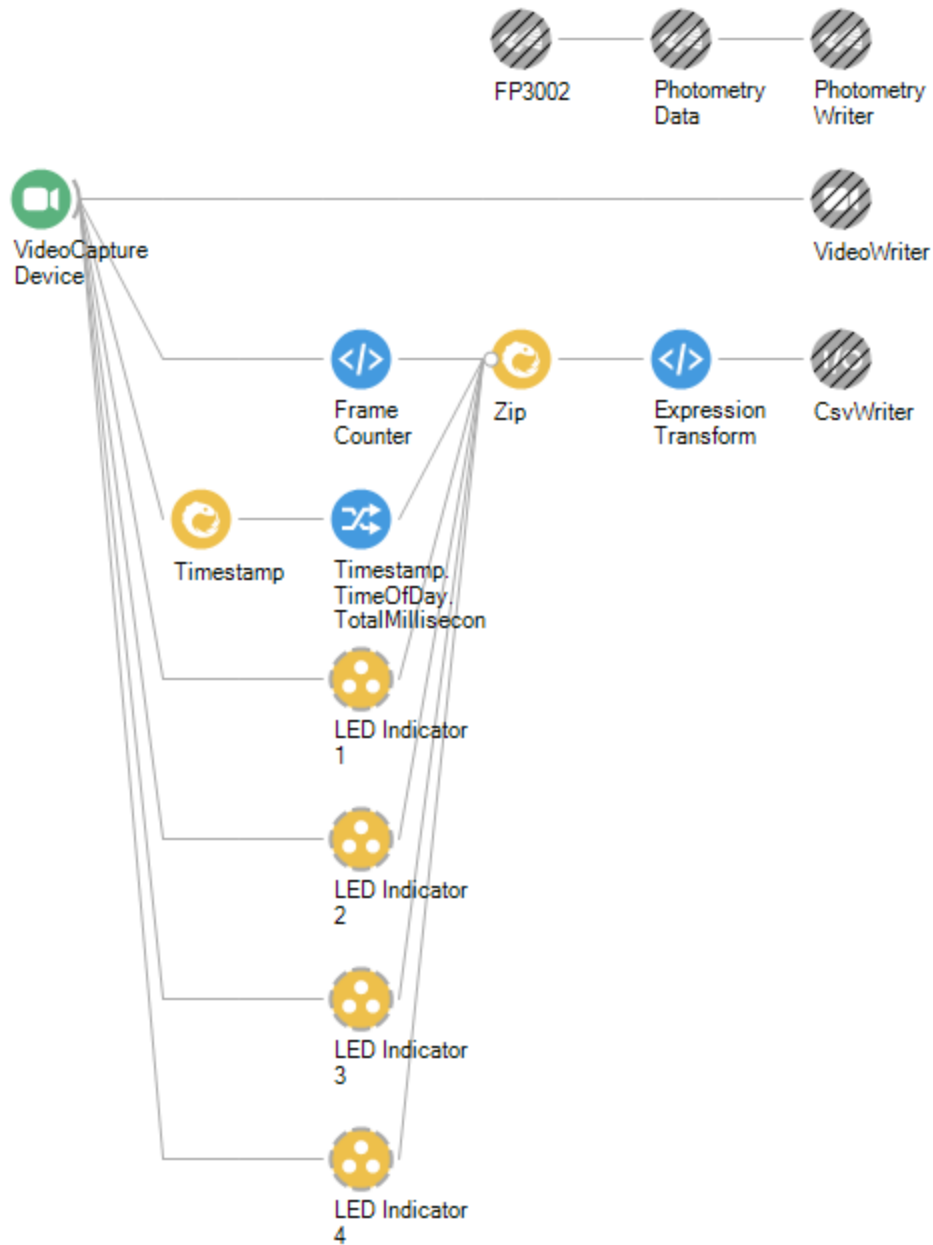
This “LED Indicator” grouped workflow can be copy and pasted for each LED indicator. Be sure that each still has a unique name. Next, the frame counter, the computer timestamp, and all of the image processing grouped workflows need to be combined into a single datastream using the “Zip” node. This “Zip” node can then be connected to a “Csv Writer” node. This way a .csv file will be written with the first column as the external camera’s frame number, the second column number as the computer timestamp, and any subsequent columns as the LED Indicator states.



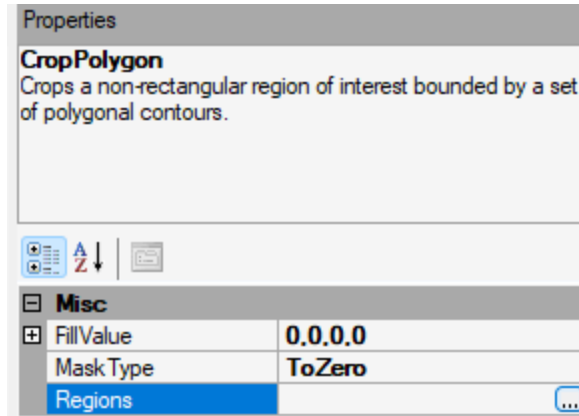
We can also add headers to the .csv file by inserting an “Expression Transform” node immediately after the “Zip” node with the following script. Be sure that the “Include Header” property of the “Csv Writer” node is set to “True”.



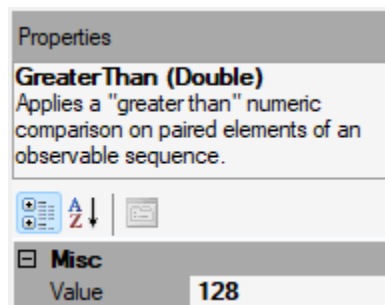
The image processing algorithms must be configured before every experiment. Any movement of the external camera can cause the “Crop Polygon” nodes to no longer be aligned to the LED indicators. Also, changes to lighting conditions can cause the cutoff values for the “Greater Than” nodes to be incorrect. To configure the “Crop Polygon” nodes, disable the photometry data stream and all of the writer nodes by selecting them and pressing “CTRL + D”.



Then open all of the LED indicator grouped workflows so that we can have access to all of the “Crop Polygon” and “Greater Than” nodes while the workflow is running. Start the workflow and configure each “Crop Polygon” node by selecting the node and clicking the “...” that appears in the “Regions” property. This will open a calibration editor where you can click and drag a rectangular region across an individual LED indicator.

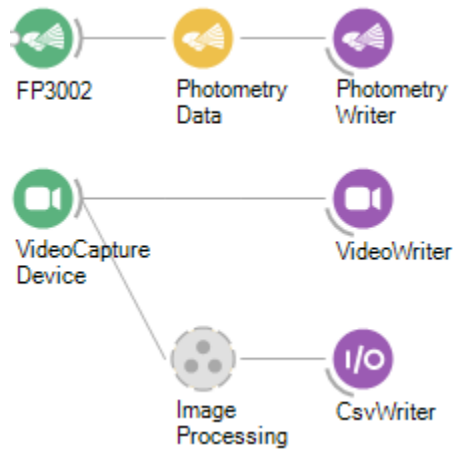


With the “Crop Polygon” nodes configured, adjust the “Value” properties of the “Greater Than” nodes such that it outputs “True” while the LED indicator is ON and “False” while the LED indicator is OFF.



Once all of the “Crop Polygon” and “Greater Than” nodes are configured, stop the workflow and enable all of the nodes by pressing “CTRL + A”, to select everything, followed by “CTRL + Shift + D”, to enable everything selected.

Finally, to organize the workflow, select all of the nodes in the external camera data stream other than the source node and the writer nodes. Then right click and select group workflow. Now you can name and provide a description for this complex workflow logic.





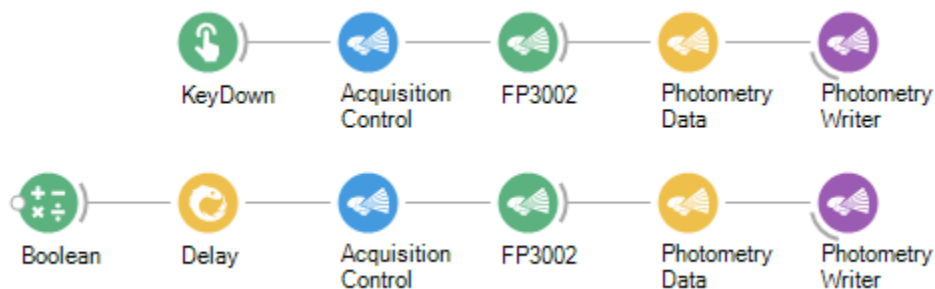
# Chapter 4: Data Acquisition

---

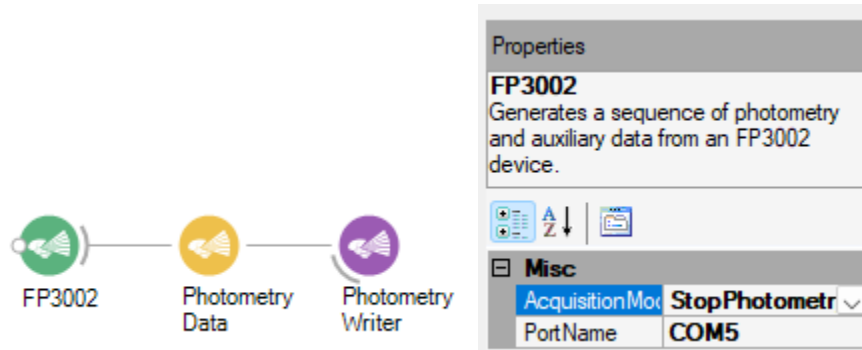
This chapter explores methods for controlling data acquisition of the FP3002 system during fiber photometry experiments. The workflows presented here build in complexity over the course of the chapter and all build off of the “Standard Photometry” workflow. One major theme of this chapter is the difference between automatic and manual control over data acquisition. We explore how to use “Timer” nodes to add a level of automatic control and how to use “Key Down” nodes to add a level of manual control.

## Delayed Start

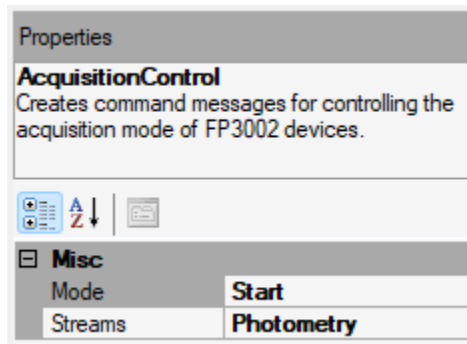
The “Standard Photometry” workflow can be expanded to allow for a delayed start to data acquisition. This example workflow makes use of the “Acquisition Control” node and a software trigger to command the FP3002 system to start data acquisition a period of time after the Bonsai workflow has been started. The software triggered used for this workflow will depend on if the user wishes to automatically or manually trigger the start of data acquisition. For an automated delayed start, we will use a “Boolean” node followed by a “Delay (Reactive)” node. For a manual delayed start, we will use the “KeyDown” node.



To create a workflow that implements a delayed start to data acquisition, begin with the “Standard Photometry” workflow. In the “Standard Photometry” workflow, the “FP3002” node was configured with the “Acquisition Mode” property set to “Start Photometry”. This causes the FP3002 system to begin data acquisition as soon as possible, after the Bonsai workflow is started. However, for a delayed start, the FP3002 system needs to wait to start data acquisition until it is commanded to start. For this case, the “Acquisition Mode” property needs to be set to “Stop Photometry”.

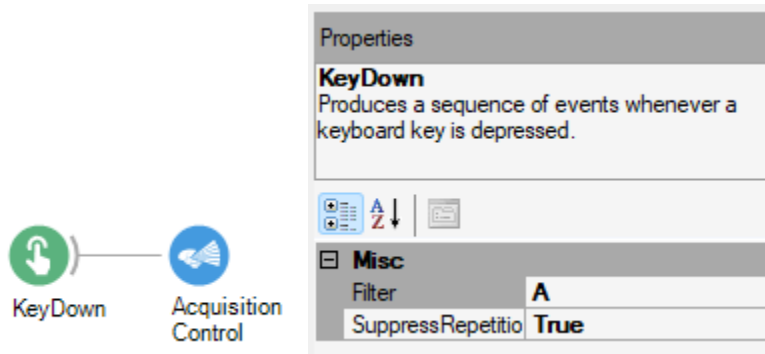


Now that the “Standard Photometry” workflow is configured to wait to start data acquisition until it is commanded, all that is left is to create the “Start Data Acquisition” command from a software trigger and pass it to the “FP3002” node. This is where the use of the “Acquisition Control” node comes into play. When the “Acquisition Control” node is configured such that the “Mode” property is set to “Start” and the “Streams” property is set to “Photometry”, the node will output the “Start Photometry” command whenever a value is passed to it.

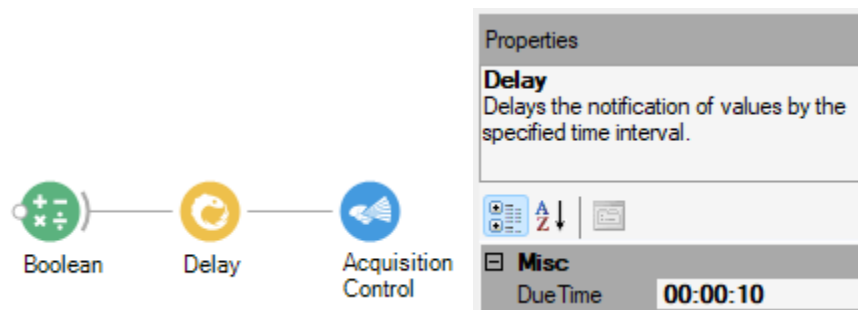


With this in mind, a software trigger can be implemented before the “Acquisition Control” node such that the software trigger sends a value to the “Acquisition Control” node after a delay. If data acquisition is desired to be started manually, the “KeyDown”

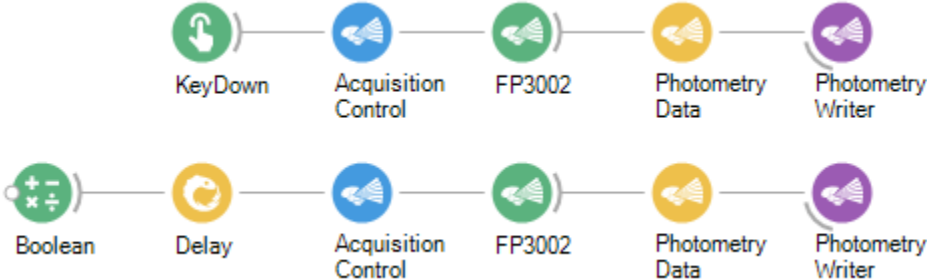
node is a good choice. The “KeyDown” node gives the option to filter by keystroke so that only a single button on the keyboard can be used to trigger data acquisition. There is also the option to “Suppress Repetitions”. This option prevents the “KeyDown” node from sending a lot of the same messages in a row while holding down a key for too long. In the example below, the “KeyDown” node will send a value every time the user presses the “A” button, and will only send the value once if the “A” button is held down for an extended period of time. Once the “A” button is pressed, the “Acquisition Control” node will output the “Start Photometry” command.



If instead, the desired delay to data acquisition is to be a precise amount of time since the workflow has started, a “Boolean” node followed by a “Delay (Reactive)” node as the software trigger would be more appropriate. The “Delay (Reactive)” node delays the notification of values by the specified time interval. Since the “Boolean” node produces a value once at the start of the workflow, the “Acquisition Control” node will not produce its first and only command until the amount of time specified in the “Delay (Reactive)” node has passed since the workflow has started. In the example below, the “Start Data Acquisition” command will be generated 10 seconds after the start of the workflow



Once the desired software trigger is connected to the “Acquisition Control” node, the “Acquisition Control” node must be connected to the input of the “FP3002” node to complete the workflow. This will allow the “Start Photometry” command, created by the “Acquisition Control” node, to be sent to the “FP3002” node, which will send the command to the FP3002 system.



# Hardware Control

In some experiments it is desired to control the data acquisition of the FP3002 system using an external device. The FP3002 system possesses two digital input ports that accept +5V digital signals. These ports can be configured such that a TTL signal can dictate when data acquisition is occurring. Using the standard photometry workflow, open the “FP3002 Setup” window by double clicking the “FP3002” node while the workflow is stopped. Then navigate to the “Digital IOs” tabpage.

The screenshot shows the 'FP3002 Setup' window with the 'Digital IOs' tab selected. The window title is 'FP3002 Setup' and it has standard window controls (minimize, maximize, close). The 'Calibration' section is active, with sub-tabs for 'Emission Alignment', 'Excitation Sequence', 'Excitation Power', 'Opto-Stimulation', and 'Digital IOs'. The 'Digital IOs' section contains four dropdown menus: 'Digital Input 0' (None), 'Digital Input 1' (None), 'Digital Output 0' (Strobe), and 'Digital Output 1' (Both). Below these is a 'Miscellaneous' section with a 'Screen Brightness' slider and a 'Clock Configuration' dropdown set to 'This Device'. To the right, there is a detailed explanation of the digital IOs, including sections for 'Digital Inputs' (Events, Control, Start Stimulation) and 'Digital Output 0' (Software, Strobe, Trigger State). Below that is a section for 'Digital Output 1' (Both, BNC, Laser). At the bottom, there is a 'Digital IOs Calibration' section with instructions for each setting, and four buttons: 'Previous', 'Write To Persistent Registers', 'Save All Settings', and 'Load All Settings'.

**Digital IOs**

Digital Input 0: None

Digital Input 1: None

Digital Output 0: Strobe

Digital Output 1: Both

**Miscellaneous**

Screen Brightness: [Slider]

Clock Configuration: This Device

**Digital Inputs**

**Events:** Samples a +5V TTL signal, generating timestamps on the specified type of voltage change.

**Control:** Allows a +5V TTL signal to control data acquisition of the FP3002 system and/or an external camera.

**Start Stimulation:** Allows a +5V TTL signal to start a finite or continuous pulse train.

**Digital Output 0**

**Software:** The system will output a +5V TTL signal generated within Bonsai, using the Digital Output Node.

**Strobe:** Outputs a +5V TTL signal that will be HIGH while the camera is exposing, and LOW between frames.

**Trigger State:** Outputs a +5V TTL will be HIGH while an LED is ON and LOW while an LED is OFF.

**Digital Output 1**

**Both:** Internal signal used to control the laser will be sent to both the laser and the Digital Output 1 port.

**BNC:** Internal signal will be sent to the Digital Output 1 port only. Timing can be configured by changing laser settings.

**Laser:** Internal signal will be sent to the laser only.

**Digital IOs Calibration:**

Digital IOs- Use drop down menus to configure how the system will send and receive TTL pulses. Note: Digital Output 1 is yoked to laser function and

Screen Brightness- Use slider to change LCD screen brightness.

Clock Configuration- Use drop down menu to select the master clock to use for photometry data. The system will default to the internal clock. If using an external device as the master clock, set clock configuration to 'External Device'.

Previous Write To Persistent Registers Save All Settings Load All Settings

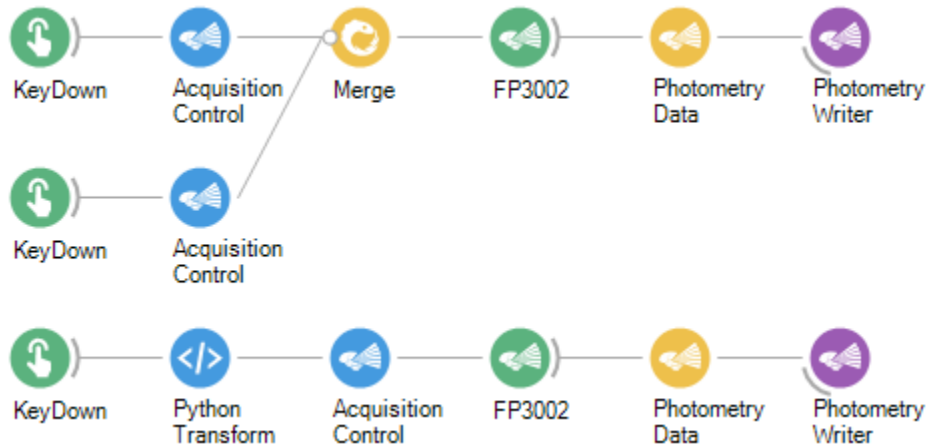
In the “Digital IOs” section, configure either the “Digital Input 0” or “Digital Input 1” setting to be “Control Trigger”.

Digital IOs	
Digital Input 0	Control: Trigger
Digital Input 1	None
Digital Output 0	Strobe
Digital Output 1	Both

This will allow the +5V TTL signal on the specified port to control the data acquisition of the FP3002 system. While the TTL signal is HIGH, the system will be acquiring data frames and while the TTL signal is LOW, the system will not be acquiring data frames.

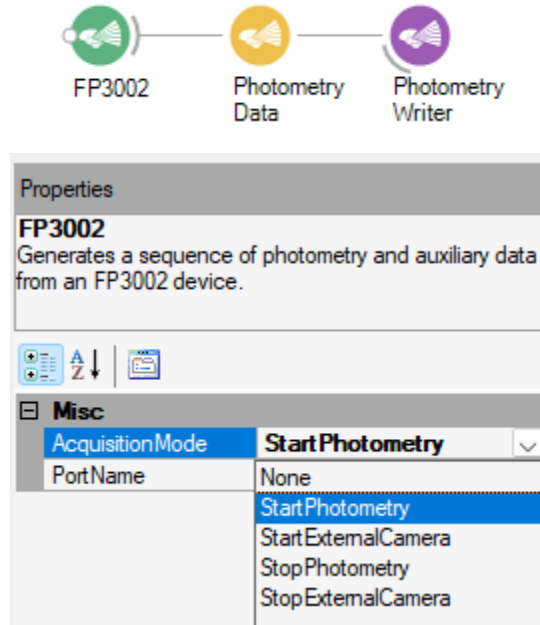
# Manual Control

This section discusses two methods for full manual control over data acquisition. Both cases utilize the “Acquisition Control” node for creating “Stop Photometry” and “Start Photometry” commands to be sent to the “FP3002” node. In one case, we use separate keys for the start and stop commands, and in the other case we use any key to toggle between data acquisition states.

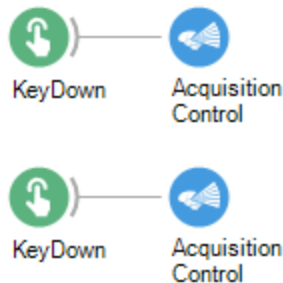




In both cases, begin with the “Standard Photometry” workflow and specify the “Acquisition Mode” property of the “FP3002” node based on the desired initial acquisition state. If the FP3002 system should be acquiring data immediately after the workflow is started, select “Start Photometry”. Otherwise, if the FP3002 system should wait until the first “Start Photometry” command is manually sent, select “Stop Photometry”.



For the case in which separate keys will be used for the “Stop Photometry” and “Start Photometry” commands, pair two “Acquisition Control” nodes with two “KeyDown” nodes. Both “Acquisition Control” nodes should have the “Streams” property set to “Photometry”, meanwhile, one should have a “Mode” property of “Start” and the other should have a “Mode” property of “Stop”.

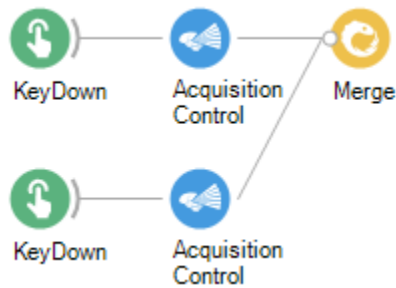


The “Filter” property of each “KeyDown” node should also be set so that only a particular keystroke will trigger the specified command. In this example, the “A” key is used to trigger the “Start Photometry” command, while the “B” key is used to trigger the “Stop Photometry” command.

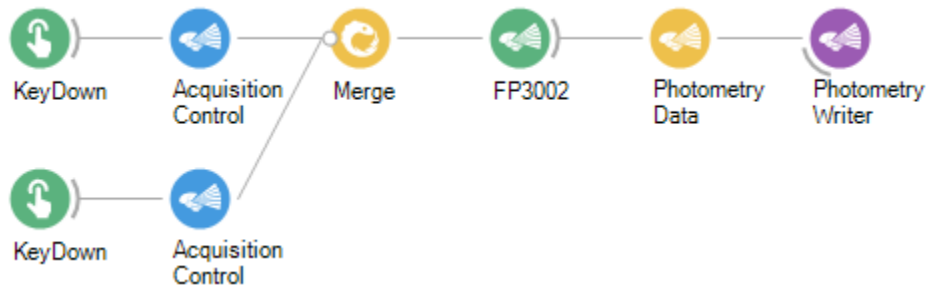
The image displays four screenshots of software property windows, arranged in a 2x2 grid. Each window is divided into a 'Properties' section and a 'Misc' section.

- Top-Left Window:** The 'Properties' section is for a **KeyDown** node, described as 'Produces a sequence of events whenever a keyboard key is depressed.' The 'Misc' section shows the 'Filter' property set to **A** and 'SuppressRepetitio' set to **False**.
- Top-Right Window:** The 'Properties' section is for an **AcquisitionControl** node, described as 'Creates command messages for controlling the acquisition mode of FP3002 devices.' The 'Misc' section shows the 'Mode' property set to **Start** and 'Streams' set to **Photometry**.
- Bottom-Left Window:** The 'Properties' section is for a **KeyDown** node, described as 'Produces a sequence of events whenever a keyboard key is depressed.' The 'Misc' section shows the 'Filter' property set to **B** and 'SuppressRepetitio' set to **False**.
- Bottom-Right Window:** The 'Properties' section is for an **AcquisitionControl** node, described as 'Creates command messages for controlling the acquisition mode of FP3002 devices.' The 'Misc' section shows the 'Mode' property set to **Stop** and 'Streams' set to **Photometry**.

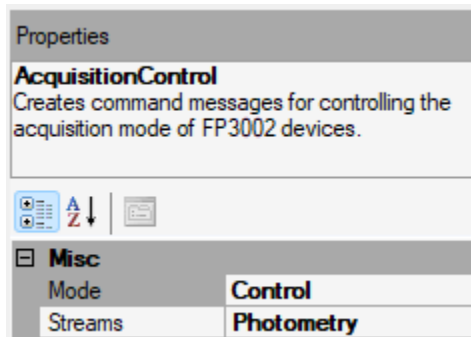
Currently, there are two data streams for producing commands to be sent to the “FP3002” node, one producing the “Start Photometry” command when the “A” key is pressed and the other producing the “Stop Photometry” command when the “B” key is pressed. These two data streams need to be converted to a single data stream without combining elements from both data streams into a different data type. For situations like this, the “Merge” node is ideal for converting from two data streams into one data stream. This node accepts data from multiple data streams that produce elements of the same type and outputs the most recent element from any data stream.



Now that the commands controlling data acquisition are merged together into a single data stream, the merged data stream can be connected to the input of the “FP3002” node so that the commands can be sent to the FP3002 system once the workflow is started.



For the case in which the state of data acquisition is to be toggled by any keystroke, the logic for combining a software trigger with the “Acquisition Control” node must be changed. Beginning with the “Acquisition Control” node, configure the “Mode” Property to “Control”. This will change the “Acquisition Control” node to only accept boolean values (True/False). In particular, when a “True” value is passed to the “Acquisition Control” node, it will output the “Start Photometry” command. When a “False” value is passed to the “Acquisition Control” node, it will output the “Stop Photometry” command.



With the “Acquisition Control” node configured in this way, the software trigger logic must be reconfigured to alternate between outputting True and False values. One way to set up the software trigger logic is to connect a “KeyDown” node to a “Python Transform” node. “Python Transform” nodes allow users to implement python scripts within Bonsai workflows, and in this case we want to create a python counter that will increment a value and output a True or False value depending on if that counter is even or odd. This sort of logic will allow the user to easily toggle the software trigger with the press of a key.



```
Python Script
# Global 'count' Variable incremented every time an element
# is passed through this node. If this value is
# initialized to be even, then the first output will be
# False (the 'Stop Photometry' trigger). Otherwise,
# the first output will be True (the 'Start Photometry'
# trigger).
count = -1

# This node returns a boolean value (True or False).
@returns(bool)
def process(value):
    global count    # Allow global access to the 'count' variable.

    # Increment the 'count' variable every time an element is passed through this node.
    count = count + 1

    # Return True if the count is Even and False if the count is Odd.
    return count % 2 == 0

OK Cancel
```

This python script initializes the ‘count’ variable to ‘-1’ when the Bonsai workflow is started. Then every time a value is passed through this node, the count variable is incremented. The script outputs `count % 2 == 0` which is True if the count is Even and False if the count is Odd. This script can be configured to specify whether True or False should be the first output value. To have the first output be True, leave the `count = -1` line of code as is. To have the first output be False, change the `count = -1` to `count = 0`. Previously, we set the “Acquisition Mode” of the “FP3002” node to specify whether the FP3002 system should begin data acquisition on the start of the Bonsai workflow, or to wait until the first “Start Photometry”

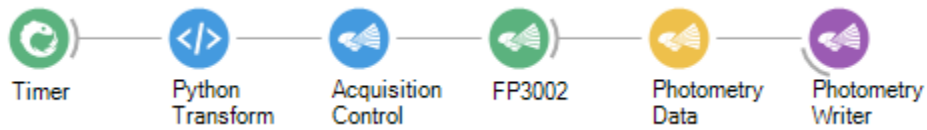
command. The “Acquisition Mode” property and the initialization of the software trigger should agree such that the first output of the software trigger is True when the FP3002 system is not acquiring at the start of the Bonsai workflow, and False when the system is acquiring when the Bonsai workflow is started. This prevents a “Start Photometry” command from being sent while the system is already acquiring and prevents the “Stop Photometry” command from being sent while the system is already not acquiring.

Once the software trigger is configured to alternate between True and False so that the “Acquisition Control” node will alternate between “Start Photometry” and “Stop Photometry” commands, this logic can be connected to the input of the “FP3002” node of the “Standard Photometry” workflow. This will allow the user to manually toggle between periods of data acquisition with any keystroke.



# Basic Periodic Control

Previously we discussed how to manually control data acquisition in fiber photometry experiments. Here we discuss a method to automatically control data acquisition in a periodic fashion with a 50% duty cycle. This means that the system will alternate between two states, acquiring and not acquiring, spending an equal amount of time in each state. For cases where periodic control is desired, but equal time spent between acquiring and not acquiring is not desired, please see the “Data Acquisition: Period Control, Variable Duty Cycle”



To construct this workflow, begin with the “Standard Photometry” workflow, setting the “Acquisition Mode” property of the “FP3002” node to the desired initial acquisition state. If data acquisition should begin with the start of the Bonsai workflow, select “Start Photometry”. Otherwise, if data acquisition should wait until the first start command, select “Stop Photometry”.



Properties

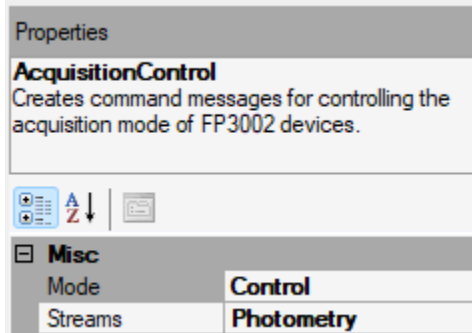
**FP3002**  
Generates a sequence of photometry and auxiliary data from an FP3002 device.

Misc

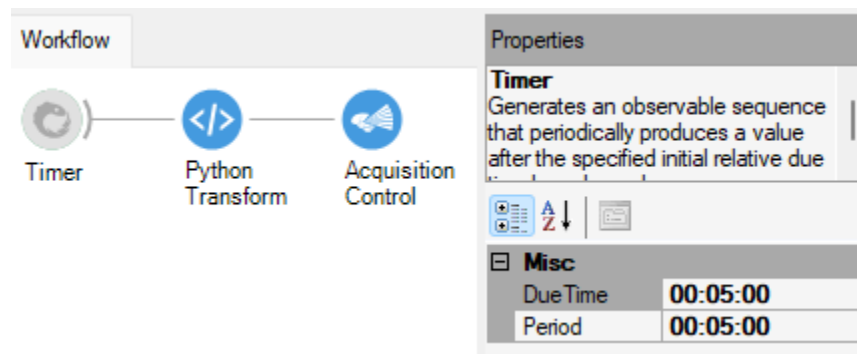
AcquisitionMode	StartPhotometry
PortName	None

- StartPhotometry
- StartExternalCamera
- StopPhotometry
- StopExternalCamera

The rest of the workflow consists of a software trigger section connected to the input of an “Acquisition Control” node. Beginning with the “Acquisition Control” node, configure the “Mode” Property to “Control”. This will change the “Acquisition Control” node to only accept boolean values (True/False). In particular, when a “True” value is passed to the “Acquisition Control” node, it will output the “Start Photometry” command. Also, when a “False” value is passed to the “Acquisition Control” node, it will output the “Stop Photometry” command.



With the “Acquisition Control” node configured in this way, the software trigger logic must be configured to alternate between outputting True and False values over a specified time interval. One way to construct the software trigger logic is to connect a “Timer” node to a “Python Transform” node. The “Timer” node can be configured to periodically output an incremented value after a specified time interval has elapsed. Then the “Python Transform” node can accept the integer from the “Timer” node and output True or False based on whether the value is even or odd. This setup will output a periodic stream of boolean values with a 50% duty cycle as desired.

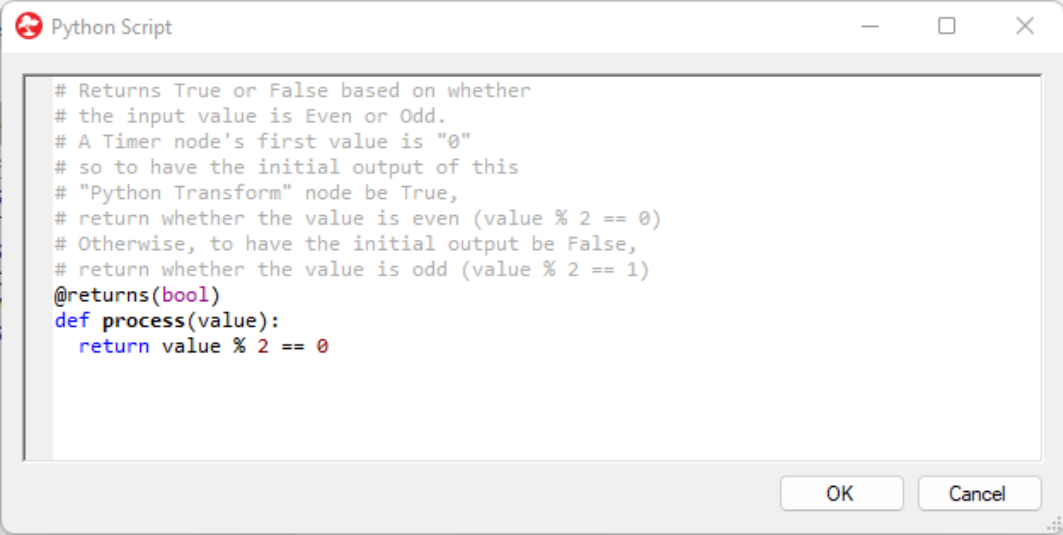


To properly configure the “Timer” node, set the “Period” property to be the duration of time for data acquisition to occur each cycle. Since the duty cycle of this particular



workflow is 50%, this will also be the period of time that data is not being acquired during each cycle. Then, set the “Due Time” property to be the same as the “Period” property. This is because the “FP3002” node sends the command specified in its “Acquisition Mode” property at the start of the workflow. So in order to prevent sending multiple acquisition control commands at the start of the workflow, we need to provide a delay before the first command from the “Acquisition Control” node is generated.

The script within the “Python Transform” node simply checks whether or not the value is even or odd. The script does this by outputting `count % 2 == 0`, which is True if the value is Even and False if the value is Odd. This script can be configured to specify whether True or False should be the first output value. Since the first value that the “Timer” node outputs is “0”, True will be the first value output by the “Python Transform” node when checking for Even values: `count % 2 == 0`. However, if the first value from the “Python Transform” node should be false, then check for Odd values: `count % 2 == 1`. This should agree with the “Acquisition Mode” property of the “FP3002” node such that the first command generated by the “Acquisition Control” node is opposite as the command specified in the “Acquisition Mode” property. In particular, if the initial acquisition state is desired to be ON then the “Acquisition Mode” property should be set to “Start Photometry” and the python transform node should check for Odd values.



```
# Returns True or False based on whether
# the input value is Even or Odd.
# A Timer node's first value is "0"
# so to have the initial output of this
# "Python Transform" node be True,
# return whether the value is even (value % 2 == 0)
# Otherwise, to have the initial output be False,
# return whether the value is odd (value % 2 == 1)
@returns(bool)
def process(value):
    return value % 2 == 0
```

With the software trigger and “Acquisition Control” node configured and connected, the “Acquisition Control” node can now be connected to the input of the “FP3002” node. When the workflow is run, the “Timer” node will increment its output value after a specified amount of time has elapsed. Then the “Python Transform” will output a True or False value depending on whether or not the value from the “Timer” node is even or Odd. The boolean output from the “Python Transform” will travel to the “Acquisition Control” node which will convert the boolean value to a “Start Photometry” or “Stop Photometry” command that the “FP3002” node will be able to send to the FP3002 system.



# Periodic Control, Variable Duty Cycle

In this section we generalize the concept of periodic control over data acquisition to allow for a variable duty cycle. In the periodic control workflow presented below, the times spent acquiring versus not acquiring are no longer dependent on each other. The user has full control over the duration of data acquisition and the duration of no data acquisition.



To construct this workflow, begin with the standard photometry workflow, setting the “Acquisition Mode” property of the “FP3002” node to the desired initial acquisition state. If data acquisition should begin with the start of the Bonsai workflow, select “Start Photometry”. Otherwise, if data acquisition should wait until the first start command, select “Stop Photometry”.



Properties

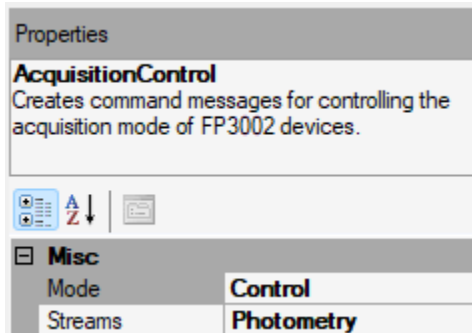
**FP3002**  
Generates a sequence of photometry and auxiliary data from an FP3002 device.

Misc

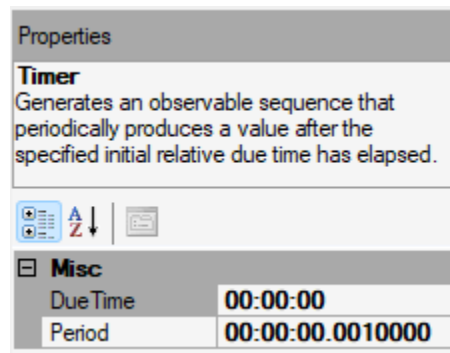
AcquisitionMode	StartPhotometry
PortName	None

- StartPhotometry
- StartExternalCamera
- StopPhotometry
- StopExternalCamera

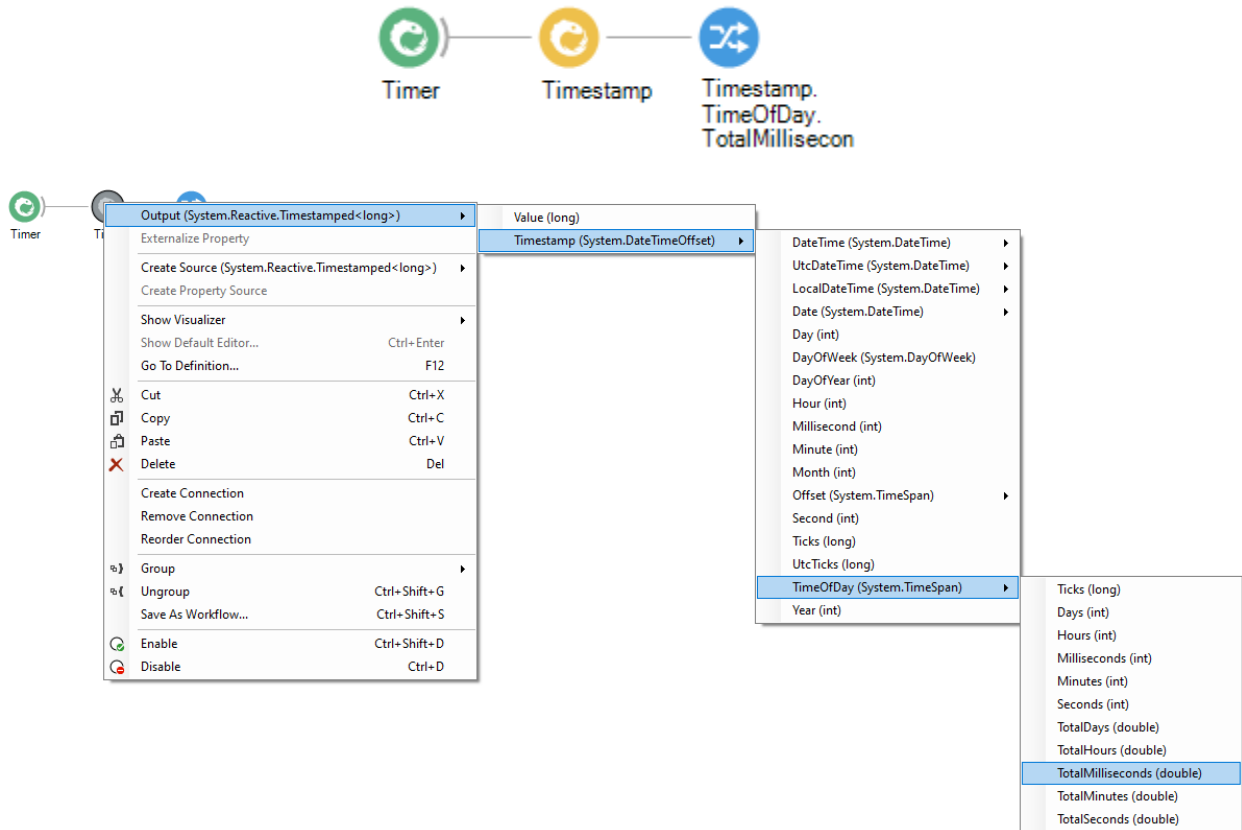
Similar to the previous workflows in this chapter, the rest of the workflow consists of a software trigger connected to the input of an “Acquisition Control” node. Beginning with the “Acquisition Control” node, configure the “Mode” Property to “Control”. This will change the “Acquisition Control” node to only accept boolean values (True/False). In particular, when a “True” value is passed to the “Acquisition Control” node, the node will output the “Start Photometry” command. When a “False” value is passed to the “Acquisition Control” node, the node will output the “Stop Photometry” command.



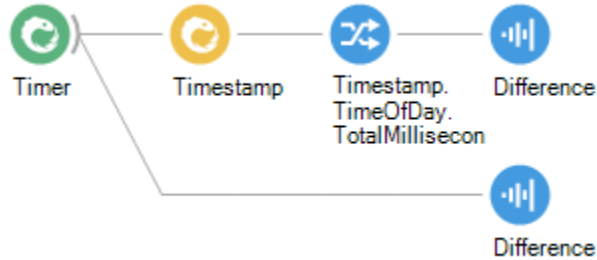
With the “Acquisition Control” node configured in this way, the software trigger logic must be reconfigured to alternate between outputting True and False values with a specified duty cycle. To begin construction of this software trigger, start with a “Timer” node configured with a “Period” of 0.001 seconds. This will force the “Timer” node to output a value as fast as possible.



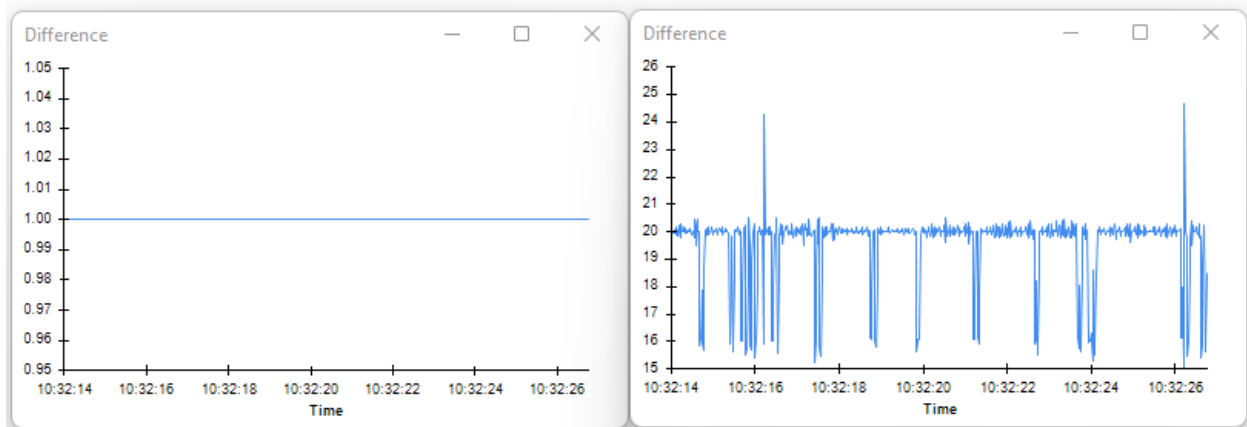
The “Timer” node does not have the precision to output a value every 1ms so it is advised to actually timestamp the values coming from this node using the computer’s timestamp. To do this, connect a “Timestamp” node after the “Timer” node. Then select the output of the “Timestamp” node to be “Time Of Day, Total Milliseconds” by right clicking the “Timestamp” node and selecting “Output → Timestamp → TimeOfDay → TotalMilliseconds”.



To highlight the importance of actually timestamping the output of the “Timer” node that has a low “Period”, let us use a “Difference” node on the “Timer” and on the “Timestamp.TimeOfDay.TotalMilliseconds” nodes.

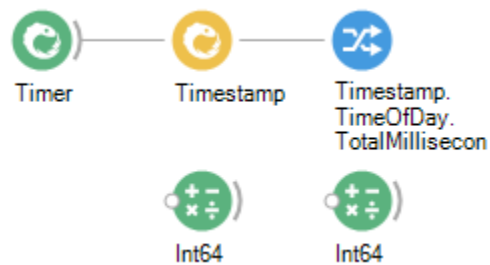


When running the above code, the “Difference” node connected to the “Timer” will show the theoretical amount of milliseconds between outputs of the “Timer”. Meanwhile, the “Difference” node connected to the “Timestamp.TimeOfDay.TotalMilliseconds” node will show the observed amount of milliseconds between outputs of the “Timer”.

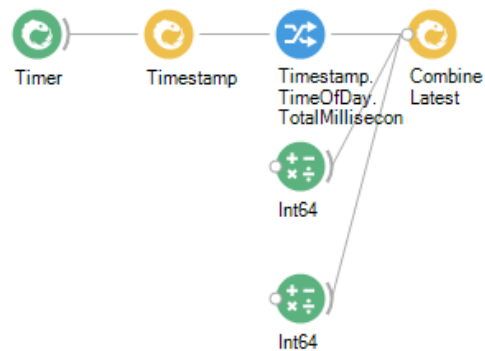


The left plot shows that theoretically, only one millisecond passed between outputs of the “Timer” node, but the right plot shows the actual time between outputs is closer to 20ms. The actual rate at which values are output from the “Timer” configured with a 1ms period is dependent on many factors, so it is often best to timestamp these values when working with small periods.

Transitioning back to the workflow at hand, remove the “Difference” nodes, we will not actually need them to complete this workflow. The goal for this software trigger is to alternate between outputting True and False values, where the user can specify how long to stay in each state. With this in mind, we need a way to store the duration spent acquiring versus not acquiring. This is easily done by inserting two “Int64” nodes into the workflow, parallel to the timestamped timer data stream. The “Int64” nodes will output the specified integer value once at the start of the Bonsai workflow.

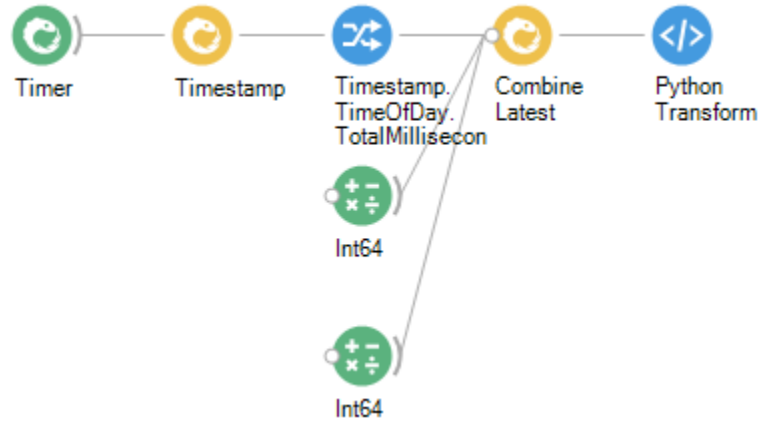


With these three data streams, we have all the information we need to pass through a python script to periodically output boolean values with the desired duty cycle. However, to get this information into a “Python Transform” node, these three data streams must be combined. In this case, connecting the three data streams to a “Combine Latest” node is appropriate. This node will accept inputs from each data stream and output a Tuple containing the latest values from each stream every time any stream has produced a value. Since the “Int64” nodes only produce a value once, at the start of the Bonsai workflow, the output of the “Combine Latest” node will only produce a value when the timer’s data stream produces a value. This output will contain the current time of day in milliseconds and the ON/OFF times specified in the “Int64” nodes.





Now that we have one data stream containing all the information we need, we can connect a “Python Transform” node after the “Combine Latest” node to process the information, outputting the appropriate boolean value. In this python script, we will keep track of each time a new acquisition cycle starts, find the amount of time spent in the current cycle, and output a boolean value accordingly. This script will be configurable to specify its initial output value.





Python Script



```
# Global 'startTime' variable that keeps track
# of the time that the current acquisition cycle
# started.
startTime = -1

@returns(bool)
def process(value):
    global startTime

    # Local variables for containing the information
    # coming into the script. The 'value' variable
    # is the Tuple coming from the "Combine Latest"
    # node. To access elements of a tuple you use the
    # ".itemN" notation. The order of the elements of the
    # Tuple follows from the input connections into the "Combine Latest"
    # node from Top to Bottom.
    currentTime = value.Item1
    onTime = value.Item2
    offTime = value.Item3

    # Only called for the first element passed through
    # the script. The only time this value is negative is
    # when it is initialized.
    if startTime < 0:
        # Set the start time of the cycle to the current time.
        startTime = currentTime

        # In this case the first output is True.
        return True

    # Calculate the amount of milliseconds spent in the current cycle.
    dt = currentTime - startTime

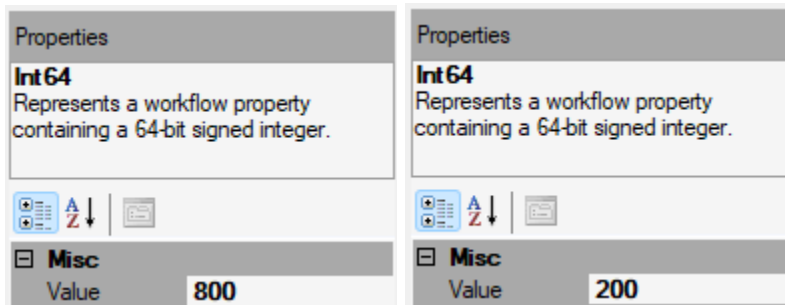
    # Account for date change
    if startTime > currentTime:
        dt = dt + (1000.0 * 3600.0 * 24.0)

    # If currently in the ON time of the cycle, return True.
    if dt < onTime:
        return True
    # Else if currently in the OFF time of the cycle, return False.
    elif dt < onTime + offTime:
        return False
    # Otherwise the full cycle has passed, update the start time and return True.
    else:
        startTime = currentTime
        return True
```

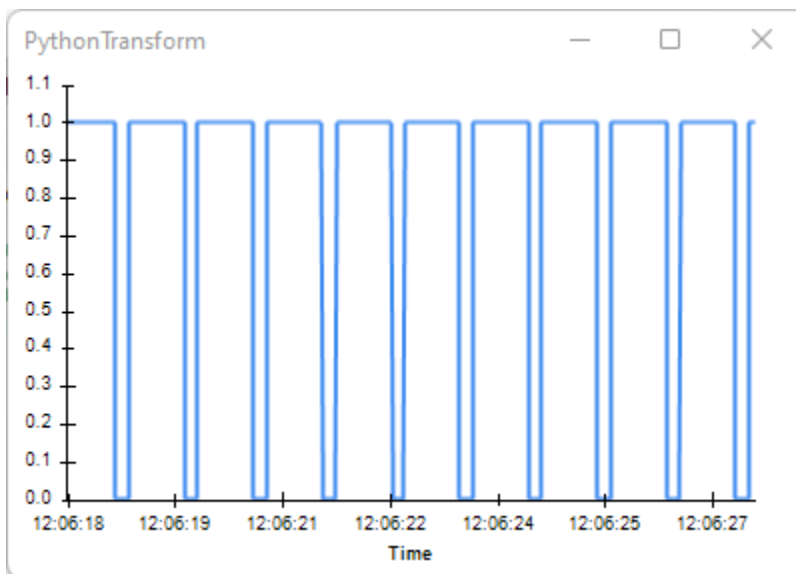
OK

Cancel

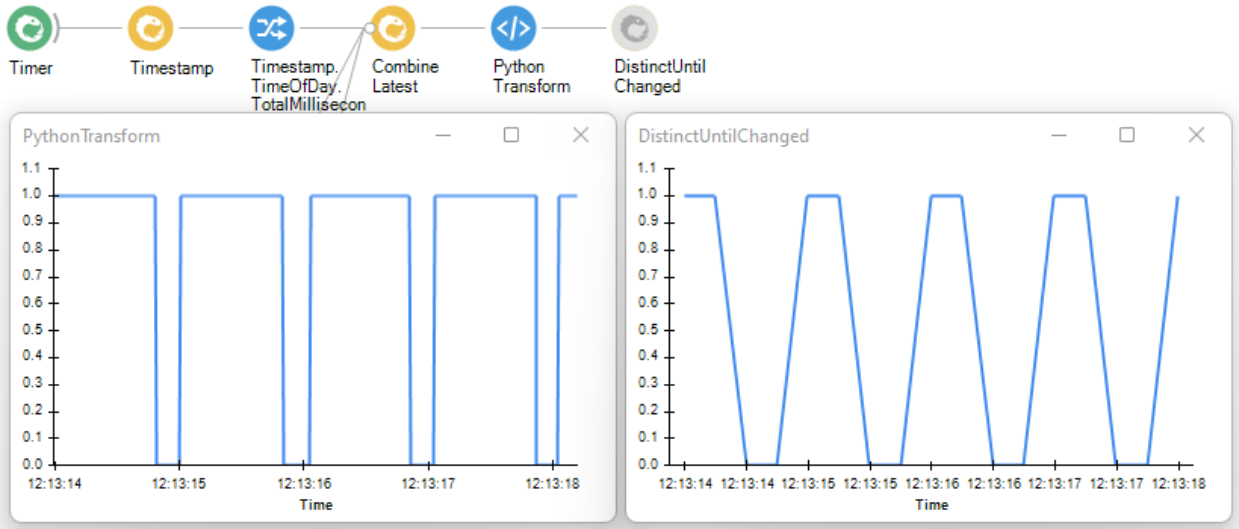
Before continuing on with constructing the workflow, this is a good place to test the logic works as intended. Set the values for the ON and OFF times by specifying the values within the “Int64” nodes. In the script above, the ON time was “Item2” of the Tuple coming from the “Combine Latest” node. This means that the top “Int64” is responsible for the ON time value, while the bottom one is responsible for the OFF time value. Our script also assumes that these durations are specified as milliseconds. In the test below, the ON time is set to 800ms and the OFF time is set to 200ms. As a note, for real-world fiber photometry experiments, periods of acquisition that are this short are not suggested.



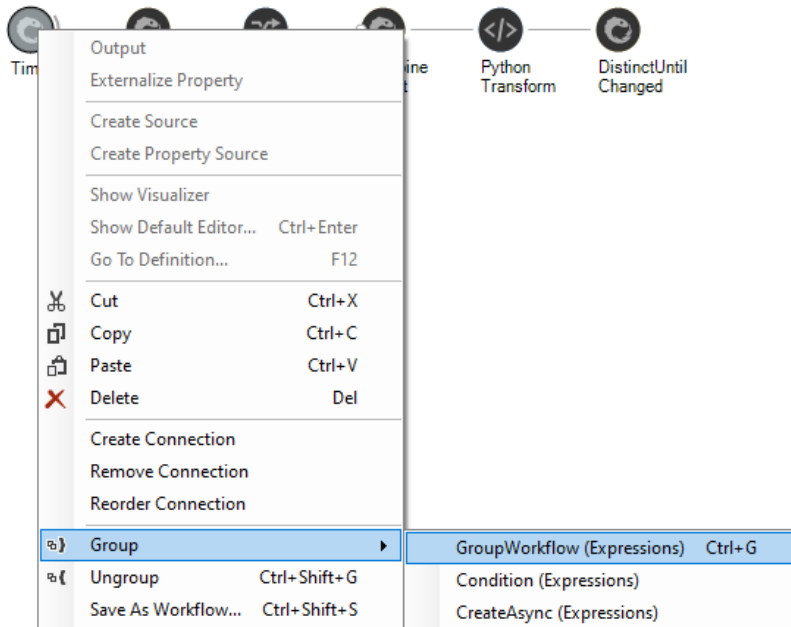
Once these values are set, run this software trigger section of the workflow by disabling all other nodes in the workflow and clicking start. Open the visualizer for the “Python Transform” node to check if our boolean signal appears correct.



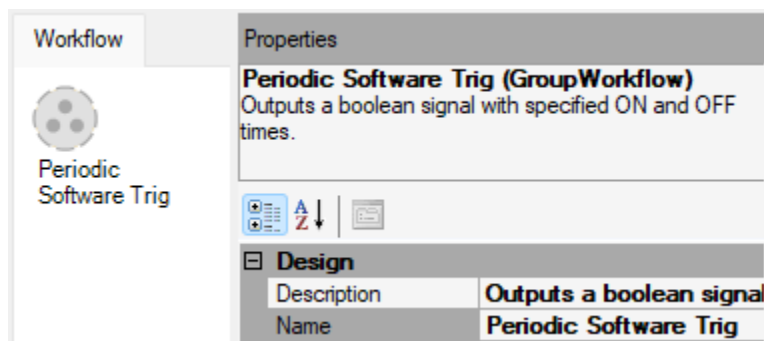
Here, the “Python Transform” periodically outputs True values for 800ms and False values for 200ms, as intended. However, this “Python Transform” outputs a value every time the timer’s data stream produces a new value. We saw above that this occurs every ~20ms. If this were connected to the “Acquisition Control” node, it would be sending identical commands every ~20ms, which is not intended. The only information we want to send to the “Acquisition Control” node is when this signal changes from LOW to HIGH or HIGH to LOW. This is a case where the “Distinct Until Changed” node is applicable. This node will only output a value when the input changes.



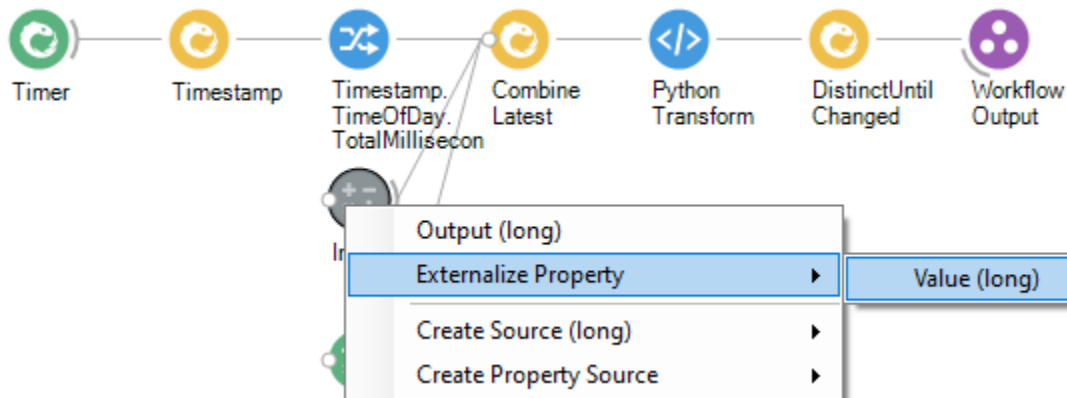
This software trigger logic is now ready, but there are some organizational changes that can be implemented to provide an easier user experience. First, we can group together all of these nodes into a single grouped workflow. This way the details of these operations will not distract the user. Do this by highlighting all of the software trigger logic, right clicking, and selecting “Group → GroupWorkflow”. This will encapsulate all of the logic into a single node.



The properties panel of the “Group Workflow” node allows you to name it and provide a description of the encapsulated process.



There are two properties within the software trigger logic that we want users to have easy access to, the ON and OFF times. Open up the encapsulated workflow by double clicking the node. You will see that a “Workflow Output” node has automatically been added to the end of the data stream. This allows the values to exit the encapsulated workflow. To make the ON and OFF times easily accessible, we want to externalize the “Value” property of both “Int64” nodes. Do this by right clicking each node and selecting “Externalize Property → Value”.



This will add “Value (ExternalizedMapping)” nodes to the inputs of the “Int64” nodes and make these properties accessible from the properties panel of the grouped workflow. Both of these “Value” nodes need to have unique “DisplayNames”. To name them, select the node, then expand the “Value” section of the properties panel and set the “Display Name”.

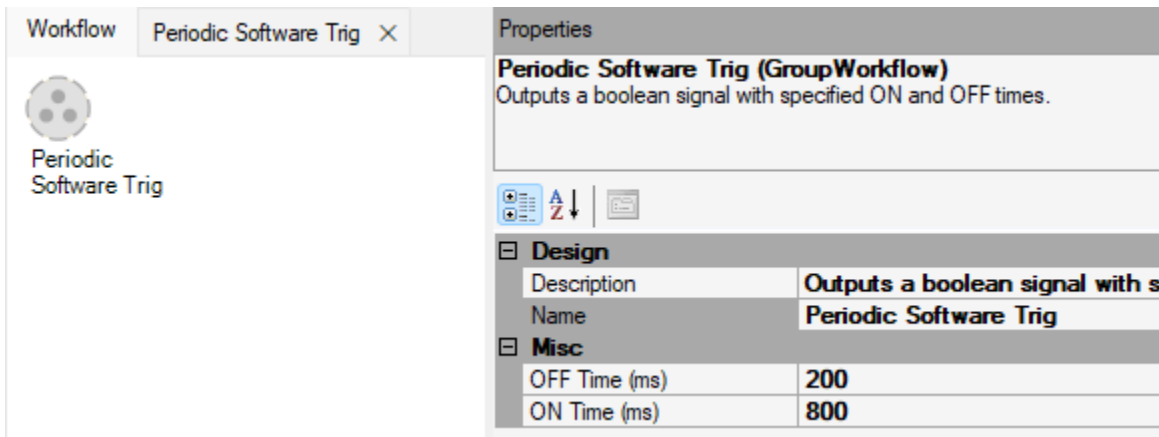
Workflow: Periodic Software Trig

Properties

**OFF Time (ms) (ExternalizedMapping)**  
Specifies a set of properties to be externalized from a workflow element.

<b>Misc</b>	
ExternalizedProperties	(Collection)
<b>Properties</b>	
<b>Value (OFF Time (ms))</b>	
Category	
Description	
DisplayName	<b>OFF Time (ms)</b>

With these externalized properties added, the ON and OFF times can be specified from the properties panel of the grouped workflow.



Now that the software trigger is complete and organized, its output can be connected to an “Acquisition Control” node configured to control the photometry stream. Then, the output of the “Acquisition Control” node can be connected to the “FP3002” node of the standard photometry workflow.

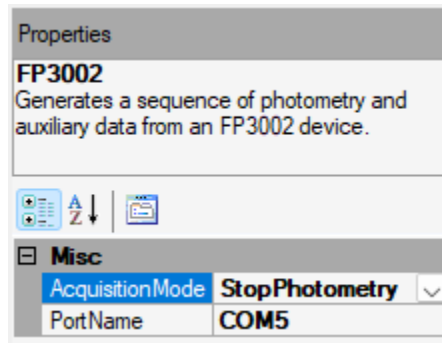


## Periodic Control, Delayed Start

This section discusses methods for adding automatic or manual delayed starts to the periodic control workflows presented in the previous two sections of this chapter. With minor changes to the “Basic Periodic Control” and “Periodic Control, Variable Duty Cycle” workflows, we can enable these delayed start features.

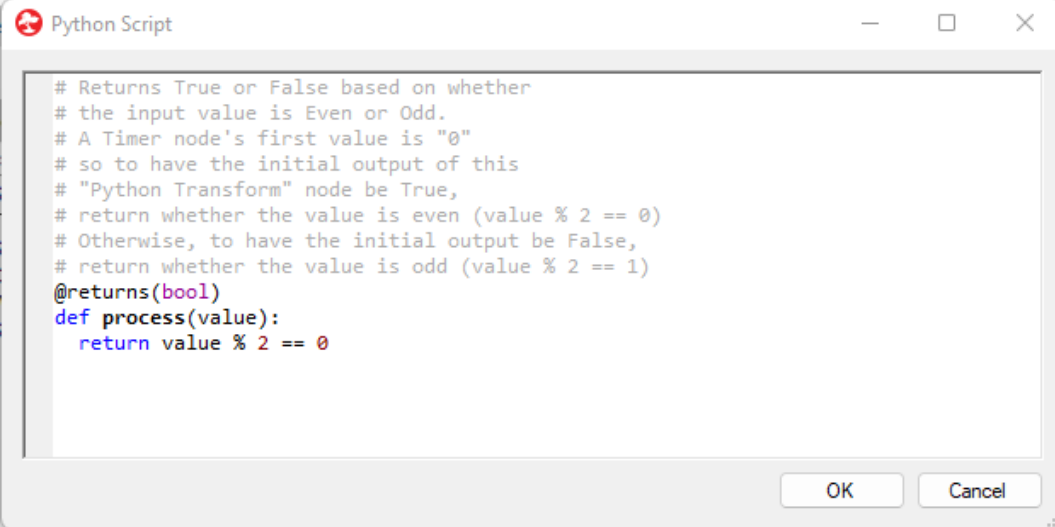
### **Basic Periodic Control, Automated Delayed Start**

In order to add an automatic delayed start to the [Basic Periodic Control](#) workflow , begin by configuring the “Basic Periodic Control” workflow to begin with data acquisition OFF. Do this by specifying the “Acquisition Mode” property of the “FP3002” node to be “Stop Photometry”.



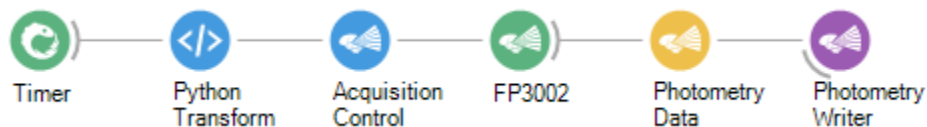


Then, configure the “Python Transform” script to output True as its first value. Since the “Timer” node will output “0” as its first value, the python script should return “ $value \% 2 == 0$ ”.

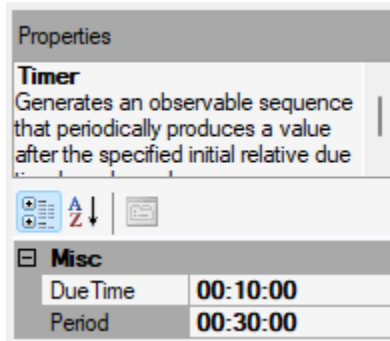


```
# Returns True or False based on whether
# the input value is Even or Odd.
# A Timer node's first value is "0"
# so to have the initial output of this
# "Python Transform" node be True,
# return whether the value is even (value % 2 == 0)
# Otherwise, to have the initial output be False,
# return whether the value is odd (value % 2 == 1)
@returns(bool)
def process(value):
    return value % 2 == 0
```

This way, when the workflow is started, the “FP3002” node will command the FP3002 system to stop data acquisition, then when the “Timer” node produces its first value, the “Start Photometry” command will be generated and sent through the “FP3002” node to the FP3002 system.

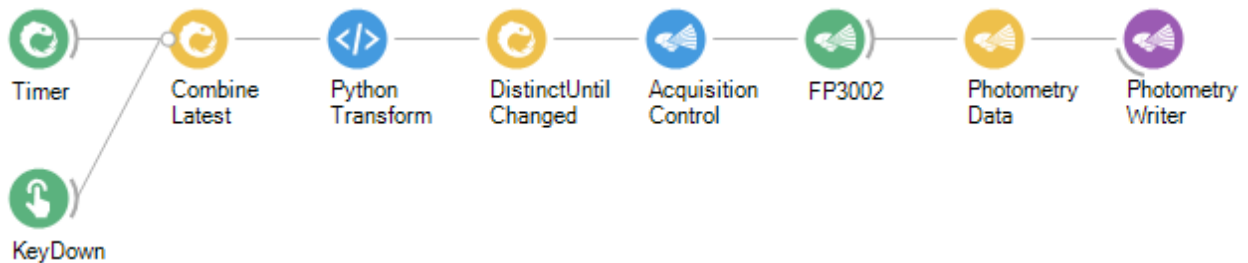


Now, the user is free to specify the duration of the automatic delayed start using the “Due Time” property of the “Timer”. In the example below, the workflow will wait 10 minutes to start basic periodic control of data acquisition. Then, during the periodic control, it will alternate between acquiring for 30 minutes and not acquiring for 30 minutes.

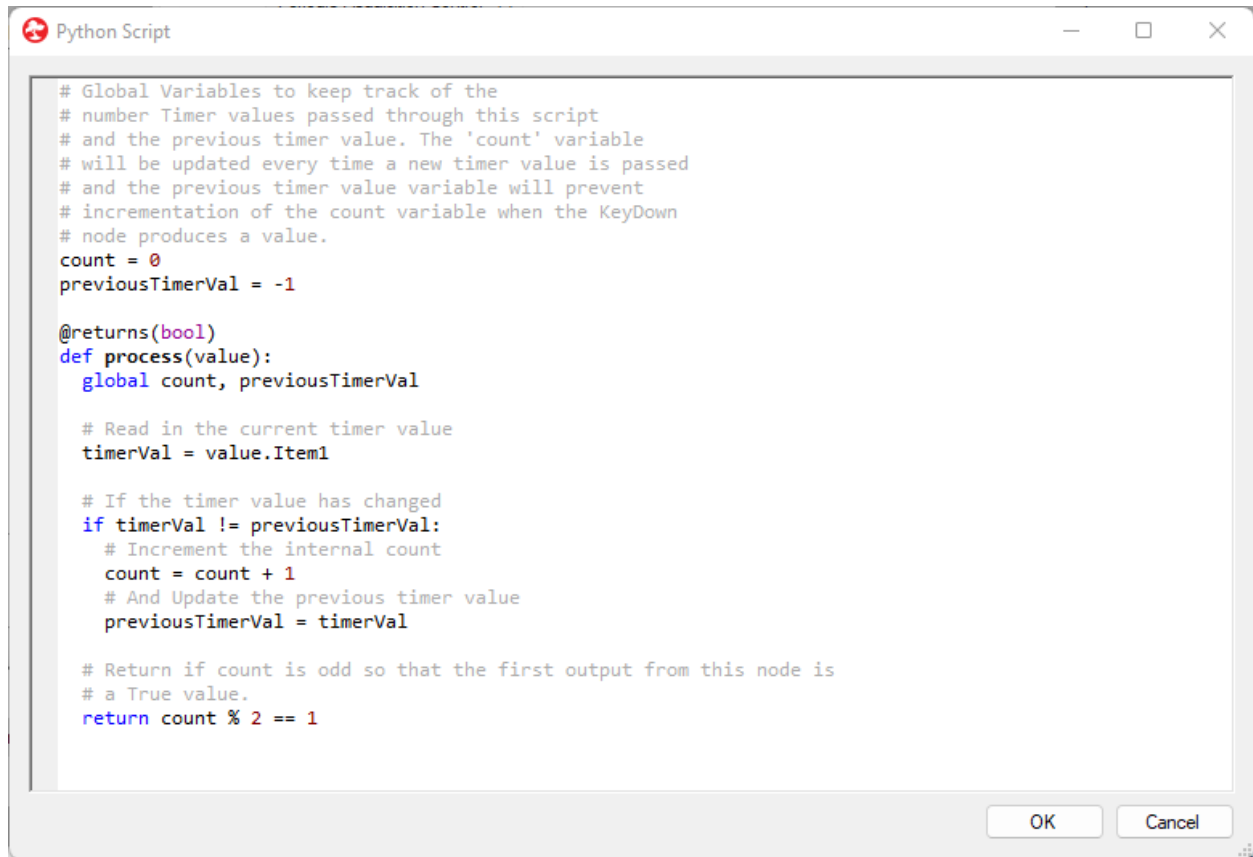


## Basic Periodic Control, Manual Delayed Start

This “Basic Periodic Control” workflow can also have a manual delay to periodic control. First, configure the “Basic Periodic Control” workflow to begin with no data acquisition. Do this by specifying the “Acquisition Mode” property of the “FP3002” node to “Stop Photometry” and configuring the python script to output True as its first value. Then verify the “Due Time” property of the “Timer” node is set to zero and use a “Combine Latest” node to combine the “Timer” node with a “Key Down” node. The “Combine Latest” node will only output its first value after both the “Timer” and “Key Down” data streams have produced a value.



Now, some minor changes to the “Python Transform” node need to be made. Currently, the python script assumes that the input is an integer value, but now its input is a Tuple. Also, the “Python Transform” outputs whether or not the value from the “Timer” node is even or odd. However, with the delayed start, the first value from the “Timer” that the python script sees could be even or odd depending on the duration of the manual delay. To make this less arbitrary and to specify the initial output of the “Python Transform” node, we need an internal counter in the python script.



```
# Global Variables to keep track of the
# number Timer values passed through this script
# and the previous timer value. The 'count' variable
# will be updated every time a new timer value is passed
# and the previous timer value variable will prevent
# incrementation of the count variable when the KeyDown
# node produces a value.
count = 0
previousTimerVal = -1

@returns(bool)
def process(value):
    global count, previousTimerVal

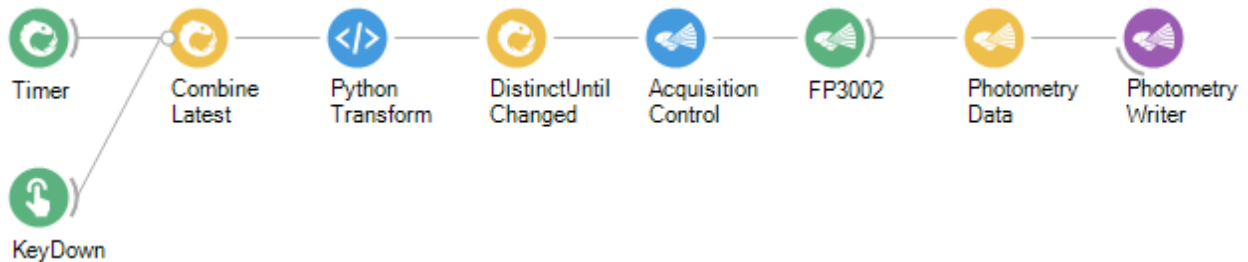
    # Read in the current timer value
    timerVal = value.Item1

    # If the timer value has changed
    if timerVal != previousTimerVal:
        # Increment the internal count
        count = count + 1
        # And Update the previous timer value
        previousTimerVal = timerVal

    # Return if count is odd so that the first output from this node is
    # a True value.
    return count % 2 == 1
```

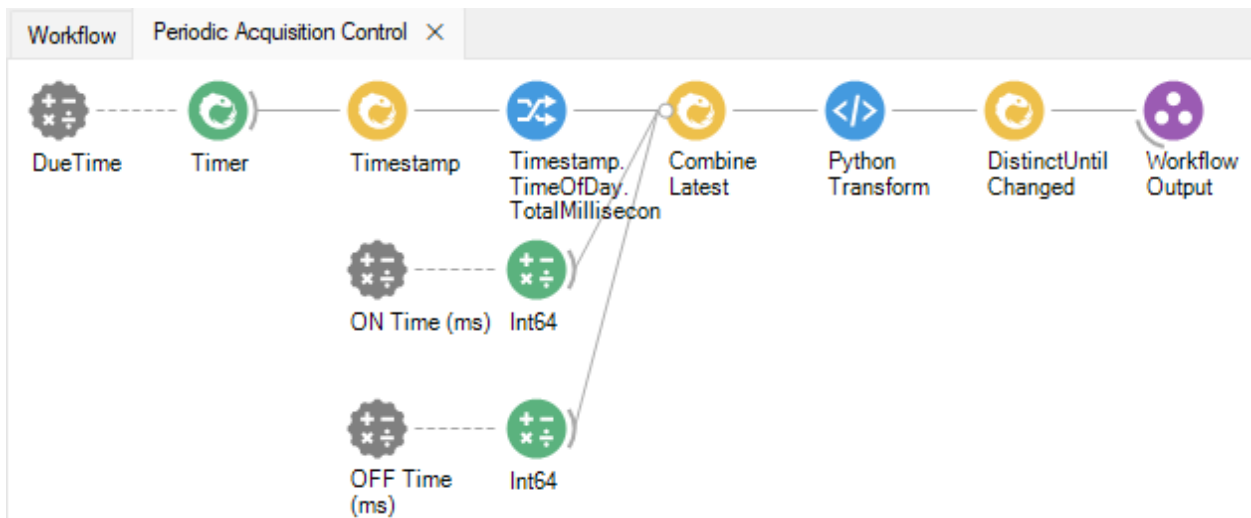
OK Cancel

This script's main two features are that it has an internal counter to dictate the output of the script. This allows the user to more easily specify the first output of the script. This script also prevents the incrementation of the internal count when the "Key Down" node produces a value. This way if the keystroke used to start the periodic control is pressed again, it will not affect data acquisition. To clean up the workflow, it is useful to specify the "Filter" property of the "KeyDown" node so that only one keystroke can be used to trigger periodic data acquisition. Also, it is useful to add a "Distinct Until Changed" node after the "Python Transform" node so that subsequent key presses will not produce repeated commands acquisition control commands.



## Periodic Control, Variable Duty Cycle, Automatic Delayed Start

To implement an automatic delay to the "Periodic Control, Variable Duty Cycle" workflow, we will specify the "Due Time" property of the "Timer" node. To keep with the organizational scheme that we developed during construction of that workflow, we can externalize the "Due Time" Property of the "Timer" node so that it appears in the grouped workflow's properties panel. You can do this by double clicking the grouped workflow to open it in a new tab, then right click the "Timer" node and select "Externalize Property → Due Time"



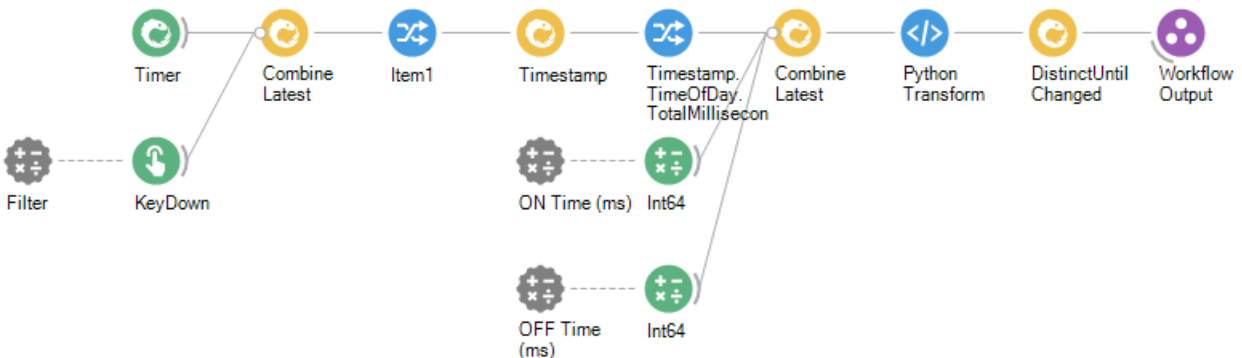
Now you can specify the duration of the delay to periodic data acquisition within the properties panel of the grouped workflow. In the example below, periodic data acquisition will start after a 10 minute delay. Once started it will alternate between 40 minutes of data acquisition and 20 minutes of no data acquisition.

The screenshot shows a workflow editor with a workflow named 'Periodic Acquisition Control'. The workflow consists of five nodes: 'Periodic Acquisition Control', 'Acquisition Control', 'FP3002', 'Photometry Data', and 'Photometry Writer'. The 'Properties' panel for the 'Periodic Acquisition Control (GroupWorkflow)' node is open, showing the following details:

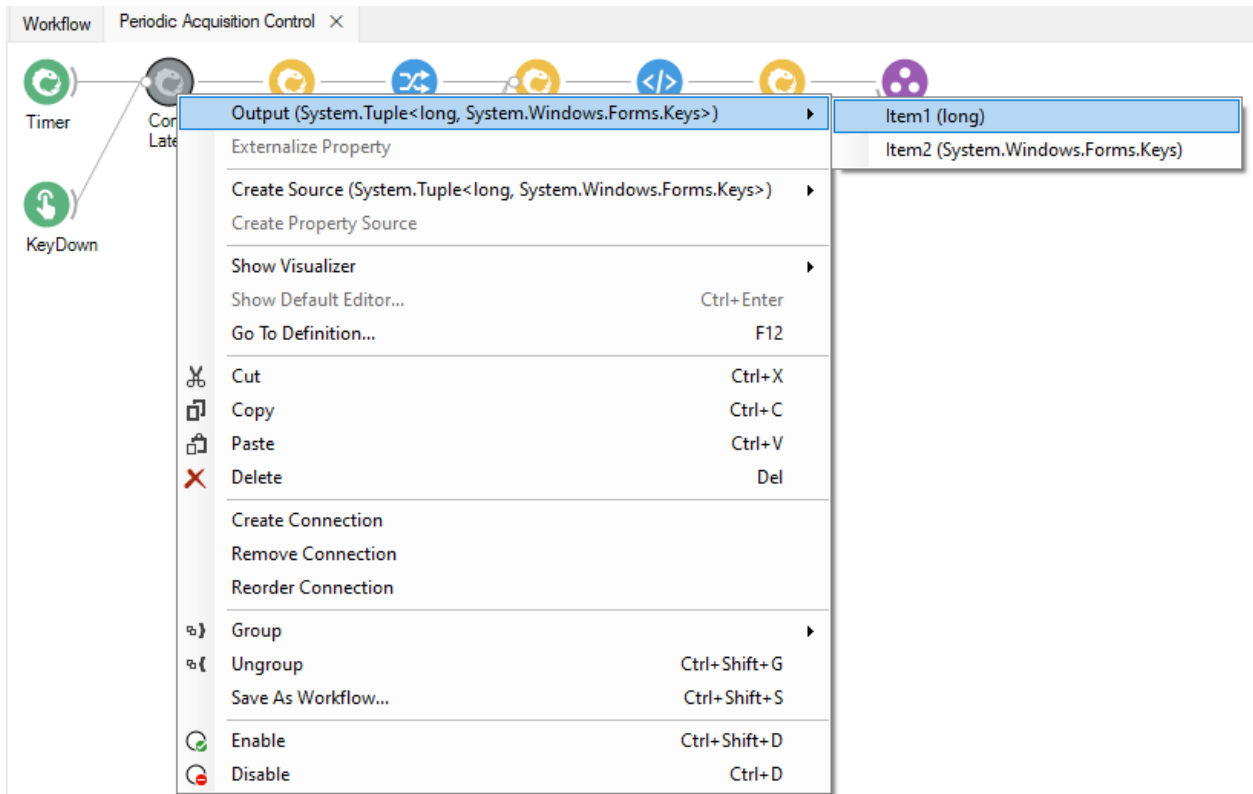
Design	
Description	Outputs a boolean signal with specified ON and OFF times.
Name	Periodic Acquisition Control
Misc	
Due Time	00:10:00
OFF Time (ms)	1200000
ON Time (ms)	2400000

## Periodic Control, Variable Duty Cycle, Manual Delayed Start

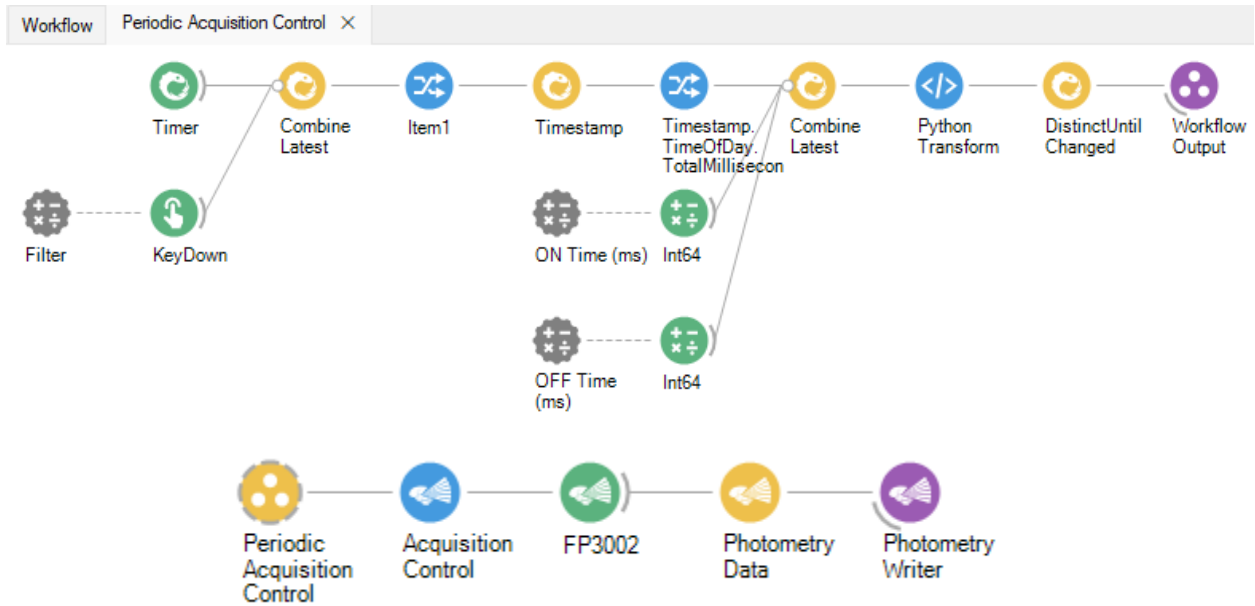
To add a manual delay to this workflow, instead of specifying the “Due Time” property, we will use a “Combine Latest” node to combine the “Timer” node and a “KeyDown” node. This way the “PythonTransform” node will not receive its first value until the specified key is pressed.



Now, we have two “Combine Latest” nodes in the same data stream leading up to the “Python Transform” node. This adds a little complexity to the data types going into python script. We can fix this data type mix-match either within the python script or within the workflow. In this case, the easiest way to fix this discrepancy is to have the “Combine Latest” node only output the first item in its Tuple (the value from the “Timer” node). Do this by right clicking the “Combine Latest” node and select “Output → Item1”.



This workflow already has a “Distinct Until Changed” node to prevent repeated outputs and already has a python script resilient to extra key presses so no other changes need to be made. It is a good idea to externalize the “Filter” property of the “KeyDown” node so that it is accessible from the grouped workflow’s property panel.



## **Second Order Periodic Control**

We have been using “periodic control” to describe cycling between states of data acquisition and no data acquisition, where the “ON time” is the duration of the data acquisition phase and the “OFF time” is the duration of the no data acquisition phase. In this second order periodic control section we discuss another level of control for data acquisition. In some experiments, implementation of periodic control is not enough, sometimes we need to implement control over when “periodic control” occurs. This concept of “Second Order Periodic Control” involves using a second software trigger to trigger when to start and stop periodic control of data acquisition. Similar to the software triggers of the periodic control workflow, this additional software trigger can be constructed to automatically or manually trigger periodic data acquisition.

Before we construct manual and automatic second order periodic control workflows, let’s detail the desired functionality of these workflows. For both workflows, we want to construct a software trigger that will alternate between two states: no data acquisition and periodic data acquisition. At the start of the no data acquisition state, the software trigger should output a “False” value. Then during the periodic data acquisition state, the software trigger should alternate between “True” and “False” values with user-specified ON and OFF times. For an automatic second order control workflow, the user should be able to specify the duration of the no data acquisition and periodic data acquisition states. For a manual second order control workflow, the user should be able to trigger a state change between no data acquisition and periodic data acquisition states with a key press.



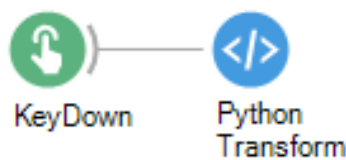
Similar to the previous data acquisition workflows, these second order periodic control workflows can be separated into three sections: the software trigger, the acquisition control node, and the standard photometry section. Our starting point in constructing this workflow will be the “Periodic Control, Variable Duty Cycle” workflow. Begin by renaming the grouped workflow containing the software trigger logic, here we will name it “Second Order Periodic Acq Control”.



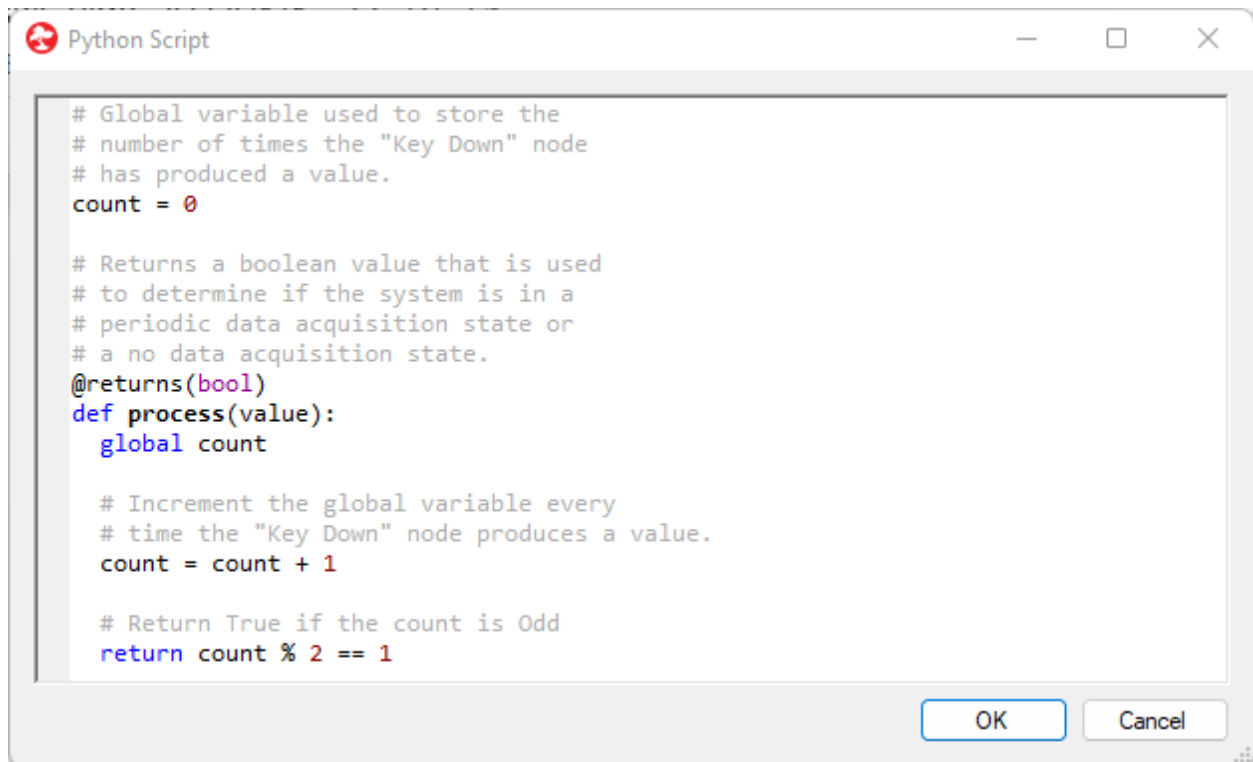
To change this first order periodic control workflow to a second order periodic control workflow, all we need to do is reconfigure the software trigger logic.

## Manual Second Order Periodic Control

In order to configure the software trigger logic for manual second order periodic control, we will use a “Key Down” node to toggle between periods of periodic data acquisition and no data acquisition. Open the grouped workflow containing the software trigger logic. Insert a “Key Down” node connected to a “Python Transform” node inside of this grouped workflow. We will use these two nodes to create a data stream that toggles between True and False when the user presses a key.



The python script in this new “Python Transform” node will contain an internal counter and output whether the key was pressed an Even or Odd number of times. We will use the output of this “Python Transform” node to dictate whether the system is in a periodic acquisition control state or a no data acquisition state. A True value will be used to trigger periodic data acquisition, while a False value will be used to trigger a stop to any data acquisition.



```
# Global variable used to store the
# number of times the "Key Down" node
# has produced a value.
count = 0

# Returns a boolean value that is used
# to determine if the system is in a
# periodic data acquisition state or
# a no data acquisition state.
@returns(bool)
def process(value):
    global count

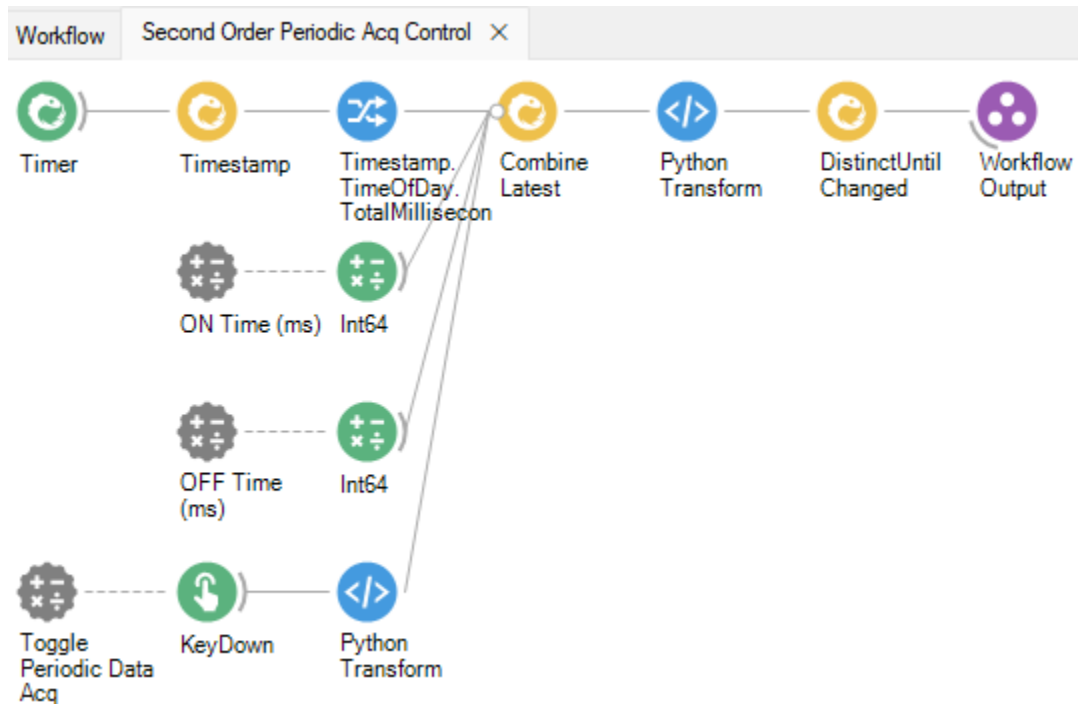
    # Increment the global variable every
    # time the "Key Down" node produces a value.
    count = count + 1

    # Return True if the count is Odd
    return count % 2 == 1
```

OK Cancel

This “Python Transform” will have its first output as True so that the FP3002 system will begin in the no data acquisition state at the start of the workflow then toggle to the periodic data acquisition state on the first keystroke.

Be sure to externalize the “Filter” property of the “Key Down” node so that it is accessible from the properties panel of the grouped workflow. Here we also edit the display name of the externalized “Filter” property to indicate its function. Specify the “Suppress Repetitions” property to “True” to prevent the “Key Down” node from producing many values for a single prolonged key press. With the “Key Down” node configured, the “Filter” property externalized, and the python script implemented, we are ready to connect this to the “Combine Latest” node.



With some additional python logic in the “Python Transform” node immediately after the “Combine Latest” node, we will finish constructing our software trigger. The boolean value dictating whether the system should be in a periodic data acquisition state or a no data acquisition state will be contained in the fourth element of the input Tuple. Begin by reading in this value as a local variable, we will name it “periodicDAQ”.

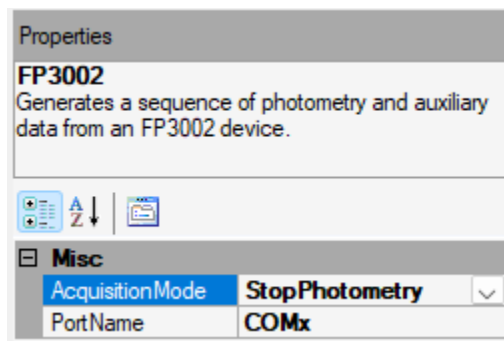
```
currentTime = value.Item1
onTime = value.Item2
offTime = value.Item3
periodicDAQ = value.Item4
```

Then, encompass the if, elif, else statement containing the periodic data acquisition logic with an if, else statement such that the periodic data acquisition logic only runs

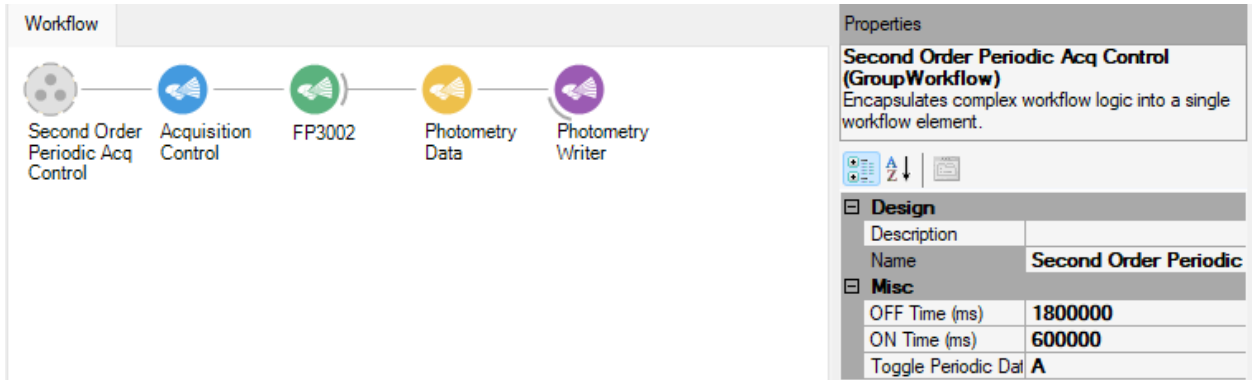
when the “periodicDAQ” variable is True. Otherwise, the script should only update the “startTime” variable and return False.

```
# If the system is in the periodic data acquisition state
# run the periodic data acquisition logic
if periodicDAQ:
    # If currently in the ON time of the cycle, return True
    if dt < onTime:
        return True
    # If currently in the OFF time of the cycle, return False
    elif dt < onTime + offTime:
        return False
    # If the full cycle has passed, update the start time and return True
    else:
        startTime = currentTime
        return True
# Otherwise, while in the no data acquisition state return False
else:
    return False
```

With this additional python logic our software trigger is complete. The “Acquisition Mode” property of the “FP3002” node should be set to “Stop Photometry”. This is because the software trigger logic dictates that the system should be in the no data acquisition state at the start of the Bonsai workflow.



In the example below, the system will begin in the no data acquisition state when the workflow is started. Then, when the “A” key is pressed, it will enter the periodic data acquisition state, with an ON time of 10 minutes and an OFF time of 30 minutes. The system will continue automatically cycling between data acquisition and no data acquisition with the user-specified duty cycle until the “A” key is pressed again to toggle OFF the periodic data acquisition.

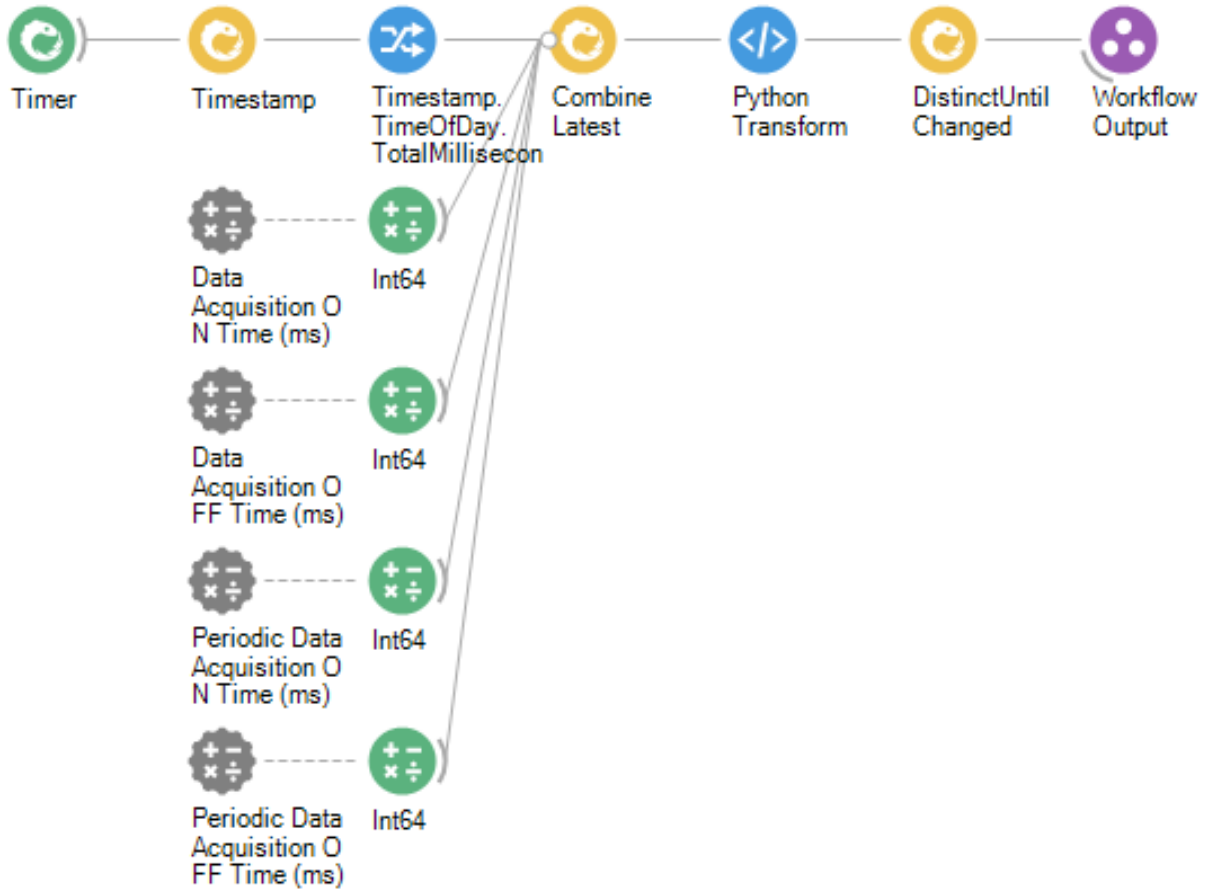


The screenshot displays a workflow editor interface. On the left, a horizontal sequence of five steps is shown: 'Second Order Periodic Acq Control' (grey circle), 'Acquisition Control' (blue circle), 'FP3002' (green circle), 'Photometry Data' (yellow circle), and 'Photometry Writer' (purple circle). On the right, the 'Properties' panel for the selected 'Second Order Periodic Acq Control' step is visible. It includes a description, a name field, and a 'Misc' section with two rows: 'OFF Time (ms)' set to 1800000 and 'ON Time (ms)' set to 600000. A 'Toggle Periodic Data' button with the letter 'A' is also present.

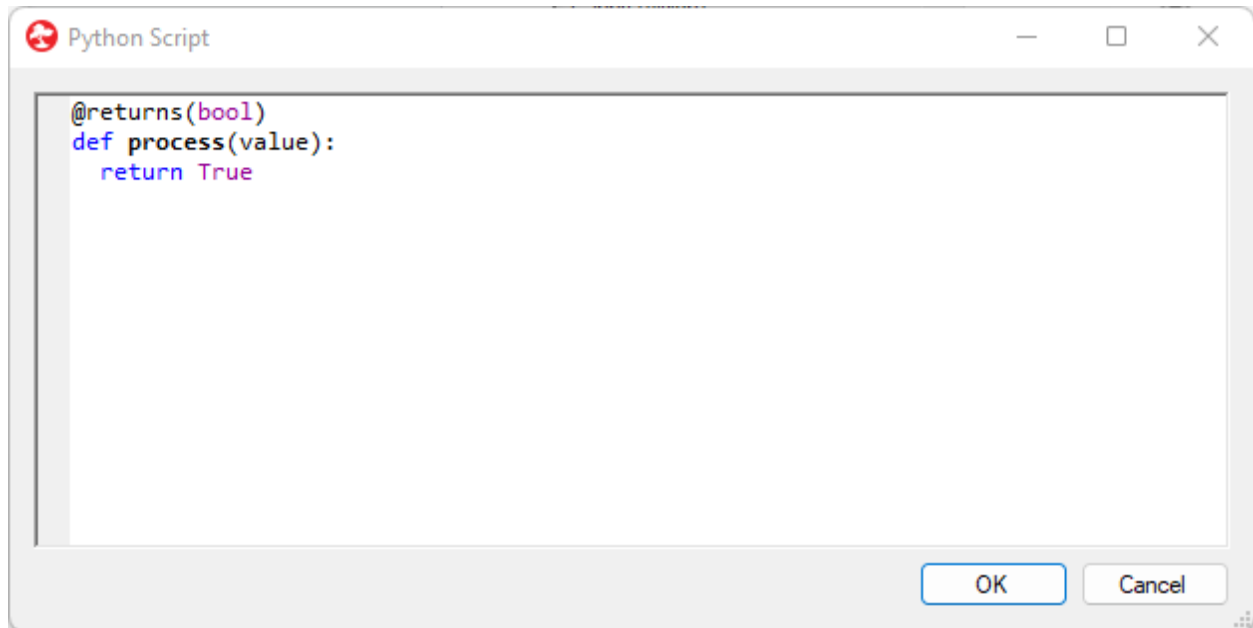
Properties	
<b>Second Order Periodic Acq Control (GroupWorkflow)</b> Encapsulates complex workflow logic into a single workflow element.	
[Icons]	
<b>Design</b>	
Description	
Name	Second Order Periodic
<b>Misc</b>	
OFF Time (ms)	1800000
ON Time (ms)	600000
Toggle Periodic Data	A

## Automated Second Order Periodic Control

In order to configure the software trigger logic for automated second order periodic control, we will include two additional “Int64” nodes to the input of the “Combine Latest” node. These will contain the duration of the periodic data acquisition and no periodic data acquisition states. Be sure to externalize the “Value” property for each of these new “Int64” nodes and give them unique display names.



Now we have all the information we need entering the “Python Transform” node through the “Combine Latest” node. Here we have the current time of day, in total milliseconds, the ON/OFF durations of data acquisition during the periodic data acquisition state, and the ON/OFF durations of the periodic data acquisition and no data acquisition states. Begin with a fresh python script by deleting the current “Python Transform” node and reinserting a new one.



Start the script by reading in each element of the incoming Tuple. Be sure to double check the order in which the “Int64” nodes are connected.

```
@returns(bool)
def process(value):

    # Read in values from the input Tuple
    currentTime = value.Item1
    daqONTime = value.Item2
    daqOFFTime = value.Item3
    periodicDaqONTime = value.Item4
    periodicDaqOFFTime = value.Item5

    return True
```

For our script we will need to keep track of two start times: the start time of data acquisition and the start time of periodic data acquisition. Declare two global variables for storing these values. Initialize these to a negative value so that the script will be able to determine if it is the first time it is being run during a workflow.

```
# Global variables for storing the time that the system entered the periodic
# data acquisition state and when the current period of data acquisition started.
daqStartTime = -1
periodicDaqStartTime = -1

@returns(bool)
def process(value):
    global daqStartTime, periodicDaqStartTime
```

Before implementing our logic for periodic data acquisition, let's handle the edge case that occurs the first time this script is run. In this case, we want to update our global start time variables to the current time. To do this, after the script reads in the input Tuple, if either of the global start time variables are negative then update them to the current time and return True.

```
# If this is the first time the script is run during a workflow, then update the
# global start time variables and output True to start data acquisition.
if daqStartTime < 0 or periodicDaqStartTime < 0:
    daqStartTime = currentTime
    periodicDaqStartTime = currentTime
    return True
```

This way, when the workflow is started, this python script will immediately update the global start time variables and output True, indicating that data acquisition should begin. That way, on the start of the workflow, the FP3002 system will be in the periodic data acquisition state. Now we can begin implementing our logic for periodic data acquisition. After our edge case test, calculate the amount of time the system has spent in the current data acquisition ON/OFF cycle and in the current periodic data acquisition ON/OFF cycle, being sure to account for a date change.

```
# Calculate the duration of time spent in the current cycle of data acquisition ON/OFF
dt = currentTime - daqStartTime
# Calculate the duration of time spent in the current cycle of periodic data acquisition ON/OFF
DT = currentTime - periodicDaqStartTime

# Account for date change
if startTime > currentTime:
    dt = dt + (1000.0 * 3600.0 * 24.0)
    DT = DT + (1000.0 * 3600.0 * 24.0)
```



Next, add an if, elif, else statement to determine if the system should be in the periodic data acquisition state, no data acquisition state, or at the end of the current periodic data acquisition ON/OFF cycle.

```
# If in the periodic data acquisition state
if DT < periodicDaqONTime:

# Else if in the no data acquisition state
elif DT < periodicDaqONTime + periodicDaqOFFTime:

# Otherwise, we have reach the end of the current periodic data acquisition ON/OFF cycle.
else:
```

Now we need to populate this if, elif, else statement. In the elif portion of the statement, we have determined that the system should be in the no data acquisition state. In this case, the script should only output a False value. Meanwhile, in the else portion of the statement, the script needs to reset to a new periodic data acquisition ON/OFF cycle. To do this, we need to update the global start time variables to the current time and output True.

```
# If in the periodic data acquisition state
if DT < periodicDaqONTime:

# Else if in the no data acquisition state
elif DT < periodicDaqONTime + periodicDaqOFFTime:
    return False
# Otherwise, we have reach the end of the current periodic data acquisition ON/OFF cycle.
else:
    daqStartTime = currentTime
    periodicDaqStartTime = currentTime
    return True
```

In the if portion of the if, elif, else statement, we need to include logic for alternating between data acquisition ON/OFF states. This logic will be quite similar to our logic for alternating between periodic data acquisition ON/OFF states. So we can add another if, elif, else statement here to determine if the system should be in the data acquisition state, no data acquisition state, or at the end of the current data acquisition ON/OFF cycle.

```
# If in the periodic data acquisition state
if DT < periodicDaqONTime:
    # If in the data acquisition state
    if dt < daqONTime:

# Else if in the no data acquisition state
elif dt < daqONTime + daqOFFTime:

# Otherwise, we have reached the end of the current data acquisition ON/OFF cycle.
else:
```

Now we need to populate this new if, elif, else statement. In the if portion of the statement we have determined that the system should be acquiring so the script

should simply output True. In the elif portion, data acquisition should stop so output False. Finally, in the else portion, we need to reset the current data acquisition ON/OFF cycle by updating only the “daqStartTime” global variable to the current time and outputting True.

```
# If in the periodic data acquisition state
if DT < periodicDaqONTime:
    # If in the data acquisition state
    if dt < daqONTime:
        return True
    # Else if in the no data acquisition state
    elif dt < daqONTime + daqOFFTime:
        return False
    # Otherwise, we have reached the end of the current data acquisition ON/OFF cycle.
else:
    daqStartTime = currentTime
    return True
```

Our python script is now complete. When the workflow starts, the script will update the global start time variables and output True to start data acquisition. The system will remain in the periodic data acquisition for the specified duration, using the logic contained within the *if DT < periodicDaqONTime:* statement to cycle between data acquisition ON/OFF states. Then after a specified amount of time, the system will enter a state of no periodic data acquisition when data acquisition will remain off until the next cycle of periodic data acquisition. The full script is shown below:

```
Python Script

# Global variables for storing the time that the system entered the periodic
# data acquisition state and when the current period of data acquisition started.
daqStartTime = -1
periodicDaqStartTime = -1

@returns(bool)
def process(value):
    global daqStartTime, periodicDaqStartTime

    # Read in values from the input Tuple
    currentTime = value.Item1
    daqONTime = value.Item2
    daqOFFTime = value.Item3
    periodicDaqONTime = value.Item4
    periodicDaqOFFTime = value.Item5

    # If this is the first time the script is run during a workflow, then update the
    # global start time variables and output True to start data acquisition.
    if daqStartTime < 0 or periodicDaqStartTime < 0:
        daqStartTime = currentTime
        periodicDaqStartTime = currentTime
        return True

    # Calculate the duration of time spent in the current cycle of data acquisition ON/OFF
    dt = currentTime - daqStartTime
    # Calculate the duration of time spent in the current cycle of periodic data acquisition ON/OFF
    DT = currentTime - periodicDaqStartTime

    # Account for date change
    if startTime > currentTime:
        dt = dt + (1000.0 * 3600.0 * 24.0)
        DT = DT + (1000.0 * 3600.0 * 24.0)

    # If in the periodic data acquisition state
    if DT < periodicDaqONTime:
        # If in the data acquisition state
        if dt < daqONTime:
            return True
        # Else if in the no data acquisition state
        elif dt < daqONTime + daqOFFTime:
            return False
        # Otherwise, we have reached the end of the current data acquisition ON/OFF cycle.
        else:
            daqStartTime = currentTime
            return True
    # Else if in the no data acquisition state
    elif DT < periodicDaqONTime + periodicDaqOFFTime:
        return False
    # Otherwise, we have reach the end of the current periodic data acquisition ON/OFF cycle.
    else:
        daqStartTime = currentTime
        periodicDaqStartTime = currentTime
        return True
```

OK Cancel

With the addition of the “Int64” nodes and the new python script, our software trigger is complete. In the example below, data acquisition will begin with the start of the workflow. Then for three hours, the system will alternate between acquiring for ten minutes and not acquiring for thirty minutes. After the three hours of periodic data acquisition, the system will stop any data acquisition for one hour. Finally, after an hour of no data acquisition, the system will begin three hours of periodic data acquisition again.

The screenshot shows a workflow editor with the following nodes in sequence: Second Order Periodic Acq Control, Acquisition Control, FP3002, Photometry Data, and Photometry Writer. The Properties panel for the 'Second Order Periodic Acq Control' node is open, displaying the following settings:

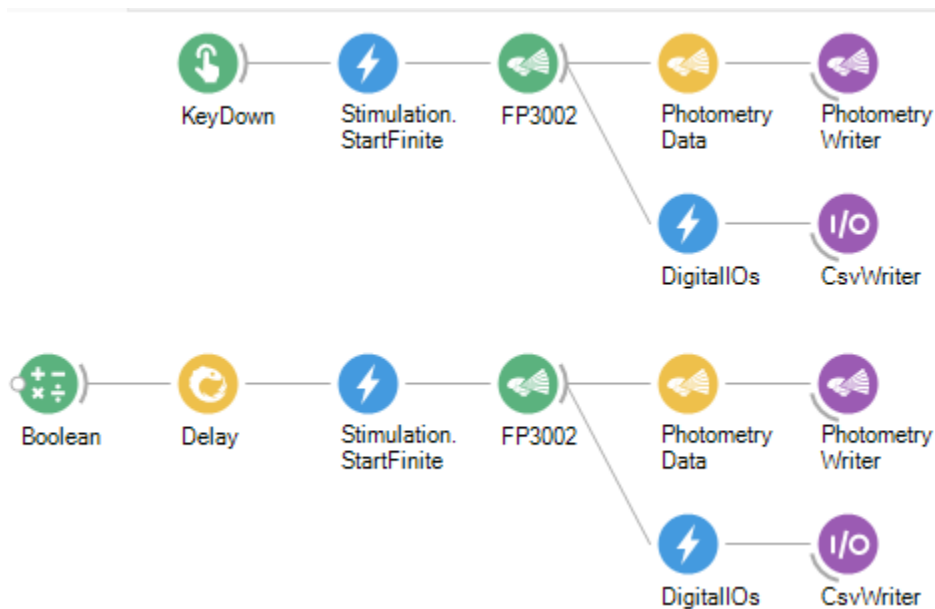
Second Order Periodic Acq Control (GroupWorkflow)	
Produces a second order periodic boolean signal with user specified ON and OFF Times (in milliseconds).	
<b>Design</b>	
Description	Produces a :
Name	Second Ord
<b>Misc</b>	
Data Acquisition OFF Time (ms)	1800000
Data Acquisition ON Time (ms)	600000
Periodic Data Acquisition OFF Time (ms)	21600000
Periodic Data Acquisition ON Time (ms)	10800000

# Chapter 5: Stimulation

This chapter explores methods for controlling stimulation of the FP3002 system during fiber photometry experiments. The workflows presented here build in complexity over the course of the chapter and all build off of the “Standard Photometry” workflow. One major theme of this chapter is the difference between automatic and manual control over stimulation. We explore how to use “Timer” nodes to add a level of automatic control and how to use “Key Down” nodes to add a level of manual control.

## Delayed Start

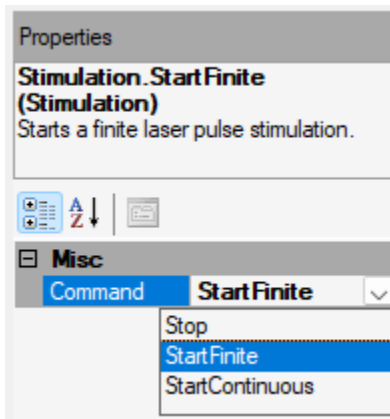
The “Standard Photometry” workflow can be expanded to allow for stimulation to be triggered after a duration of time has passed since the workflow has been started. This example workflow makes use of the “Stimulation” node and a software trigger to command the FP3002 system to start stimulation. The software triggered used for this workflow will depend on if the user wishes to automatically or manually trigger the start of stimulation. For an automated delayed start, we will use a “Boolean” node followed by a “Delay (Reactive)” node. For a manual delayed start, we will use the “KeyDown” node. This workflow also allows the user to record the ON/OFF state of the laser, timestamped using the system’s internal clock.



To create a workflow that implements a delayed start to stimulation, begin with the “Standard Photometry” workflow. Calibrate the laser within the “FP3002 Setup” window. For more information on how to calibrate and align the laser, see the “[FP3002](#)” entry in the “[Appendix I: Node Glossary](#)” .



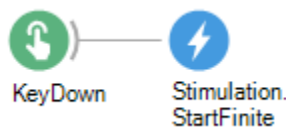
Now that the “Standard Photometry” workflow is configured, we must create the “Start Stimulation” command from a software trigger and pass it to the “FP3002” node. This is where the use of the “Stimulation” node comes into play. The “Stimulation” node has three options for the “Command” property: “Stop”, “Start Finite”, and “Start Continuous”. In this case, we want this node to command the FP3002 system to pulse the laser a finite amount of times, so the “Start Finite” option should be selected. The reason for not choosing the “Start Continuous” command is that this command will ignore the “Pulse Count” parameter specified within the “FP3002 Setup” window and command the system to pulse the laser until it the “Stop Stimulation” command is sent.



The “Stimulation” node will send its specified command whenever it receives any input value. With this in mind, a software trigger can be implemented before the “Stimulation” node such that the software trigger sends a value to the “Stimulation” node after a delay.





## Manual Trigger:

If stimulation is desired to be started manually, the “Key Down” node is a good choice for a source node. The “Key Down” node gives the option to filter by keystroke so that only a single button on the keyboard can be used to trigger data acquisition. There is also the option to “Suppress Repetitions”. This option prevents the “KeyDown” node from sending repeated messages while holding down a key for too long. In the example below, the “Key Down” node will send a value every time the user presses the “A” button, and will only send the value once if the “A” button is held down for an extended period of time. Once the “A” button is pressed, the “Stimulation” node will output the command to start a finite laser pulse train.



Properties

**KeyDown**  
Produces a sequence of events whenever a keyboard key is depressed.

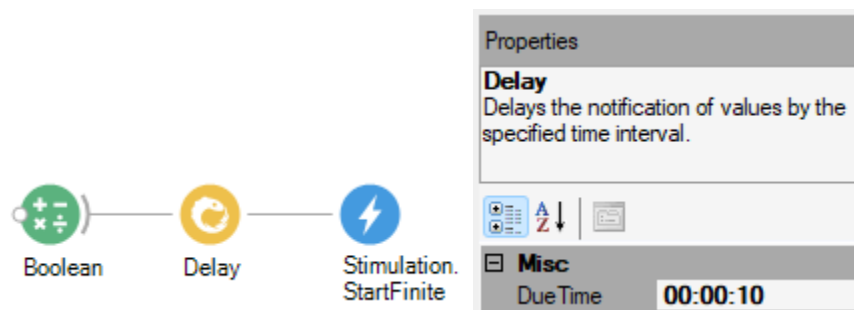
   

**Misc**

Filter	A
SuppressRepetitio	True

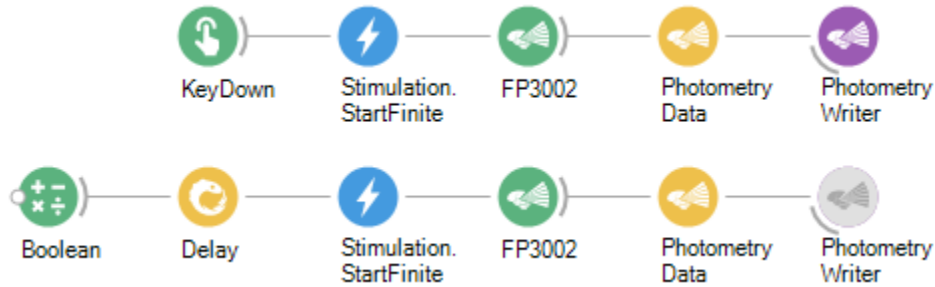
## Automated Trigger:

If instead, the desired delay to stimulation is to be a precise amount of time since the workflow has started, a “Boolean” node followed by a “Delay (Reactive)” node as the software trigger would be more appropriate. The “Delay (Reactive)” node delays the notification of values by the specified time interval. Since the “Boolean” node produces a value once at the start of the workflow, the “Stimulation” node will not produce its first and only command until the amount of time specified in the “Delay (Reactive)” node has passed since the workflow has started. In the example below, the “Stimulation” node will produce its first and only command, to start a finite laser pulse train, ten seconds after the workflow begins.

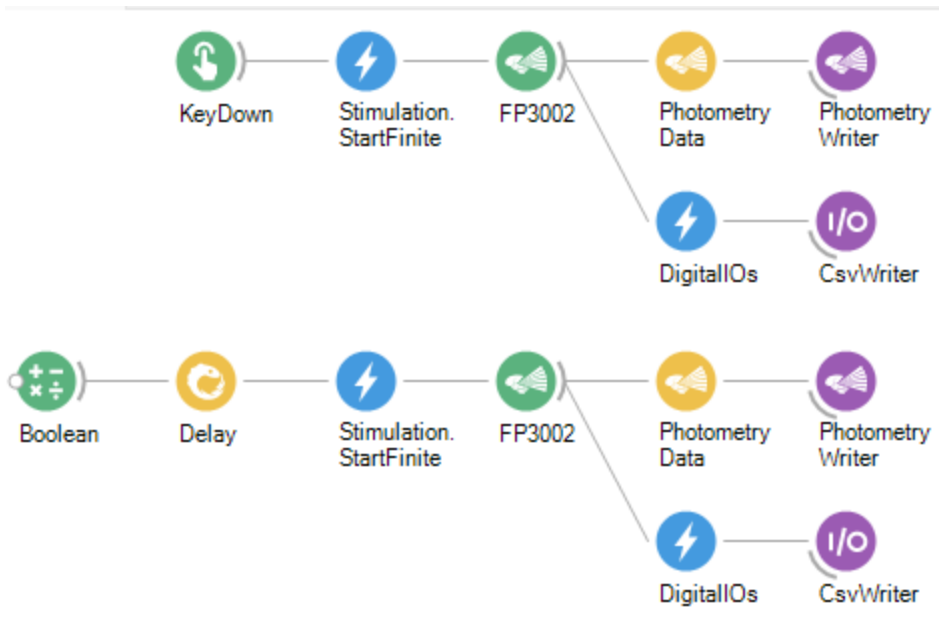




Once the desired software trigger is connected to the “Stimulation” node, the “Stimulation” node must be connected to the input of the “FP3002” node to complete the workflow. This will allow the command, created by the “Stimulation” node, to be sent through the “FP3002” node to the FP3002 system



The above workflows successfully trigger a delayed start to stimulation during a photometry experiment. However, there is an important last step for constructing any stimulation workflow. The “FP3002” node knows precisely when the laser changes state. We can gain access to that information by using a “Digital IOs” node. The “Digital IOs” node is similar to the “Photometry Data” where it processes the information coming from the “FP3002” node. However, instead of processing data related to photometry, it processes information coming from the digital input and output ports on the FP3002 system. If the “Output 1 Routing” parameter in the “FP3002 Setup” window is set to “Both”, then the internal TTL signal controlling the state of the laser will be sent to both the laser and the digital output 1 port. Due to this functionality, the state of the laser will be represented by the TTL signal on the digital output 1 port, that we can record using the “Digital IOs” node. To do this, connect the “Digital IOs” node after the “FP3002” node, in parallel with the “Photometry Data” node. Then set the “Output1” property of the “Digital IOs” node to “True”. This will cause the recorded laser state signal to be timestamped using the system’s internal clock, making it already aligned to the recorded photometry data. What is left to do is connect the “Digital IOs” node to a “Csv Writer” node.



# Hardware Control

In some experiments it is desired to control the stimulation of the FP3002 system using an external device. The FP3002 system possesses two digital input ports that accept +5V digital signals. These ports can be configured such that a TTL signal can dictate when stimulation is occurring. Using the standard photometry workflow, open the “FP3002 Setup” window by double clicking the “FP3002” node while the workflow is stopped. Then, navigate to the “Digital IOs” tabpage.

The screenshot shows the 'FP3002 Setup' window with the 'Digital IOs' tab selected. The window has a title bar with standard OS controls. Below the title bar is a 'Calibration' section with sub-tabs: 'Emission Alignment', 'Excitation Sequence', 'Excitation Power', 'Opto-Stimulation', and 'Digital IOs'. The 'Digital IOs' sub-tab is active. On the left, there are four dropdown menus: 'Digital Input 0' (set to 'Start Stimulation: Finite'), 'Digital Input 1' (set to 'None'), 'Digital Output 0' (set to 'Strobe'), and 'Digital Output 1' (set to 'Both'). Below these is a 'Miscellaneous' section with a 'Screen Brightness' slider and a 'Clock Configuration' dropdown (set to 'This Device'). On the right, there is a text area with detailed descriptions for 'Digital Inputs' and 'Digital Outputs'. At the bottom, there is a 'Digital IOs Calibration' section with explanatory text and four buttons: 'Previous', 'Write To Persistent Registers', 'Save All Settings', and 'Load All Settings'.

**Digital IOs**

Digital Input 0: Start Stimulation: Finite

Digital Input 1: None

Digital Output 0: Strobe

Digital Output 1: Both

**Miscellaneous**

Screen Brightness: [Slider]

Clock Configuration: This Device

**Digital Inputs**

**Events:** Samples a +5V TTL signal, generating timestamps on the specified type of voltage change.

**Control:** Allows a +5V TTL signal to control data acquisition of the FP3002 system and/or an external camera.

**Start Stimulation:** Allows a +5V TTL signal to start a finite or continuous pulse train.

**Digital Output 0**

**Software:** The system will output a +5V TTL signal generated within Bonsai, using the Digital Output Node.

**Strobe:** Outputs a +5V TTL signal that will be HIGH while the camera is exposing, and LOW between frames.

**Trigger State:** Outputs a +5V TTL will be HIGH while an LED is ON and LOW while an LED is OFF.

**Digital Output 1**

**Both:** Internal signal used to control the laser will be sent to both the laser and the Digital Output 1 port.

**BNC:** Internal signal will be sent to the Digital Output 1 port only. Timing can be configured by changing laser settings.

**Laser:** Internal signal will be sent to the laser only.

**Digital IOs Calibration:**

Digital IOs- Use drop down menus to configure how the system will send and receive TTL pulses. Note: Digital Output 1 is yoked to laser function and

Screen Brightness- Use slider to change LCD screen brightness.

Clock Configuration- Use drop down menu to select the master clock to use for photometry data. The system will default to the internal clock. If using an external device as the master clock, set clock configuration to 'External Device'.

Previous Write To Persistent Registers Save All Settings Load All Settings

In the “Digital IOs” section, configure either the “Digital Input 0” or “Digital Input 1” setting to be “Start Stimulation Finite” or “Start Stimulation Continuous”. The key difference between the “Start Stimulation Finite” and the “Start Stimulation Continuous” settings is that the “Start Stimulation Finite” will trigger a finite pulse train, only triggering the number of pulses specified within the “Pulse Count” property. Meanwhile, the “Start Stimulation Continuous” setting will conduct a pulse train for the whole duration of the HIGH +5V TTL signal. Whenever conducting stimulation in a photometry experiment, be sure to record the state of the laser. To do this, specify the “Output 1 Routing” property within the “FP3002 Setup” window to be “Both” so that the internal TTL signal controlling the laser will be sent to both the internal laser and the digital output 1 port.

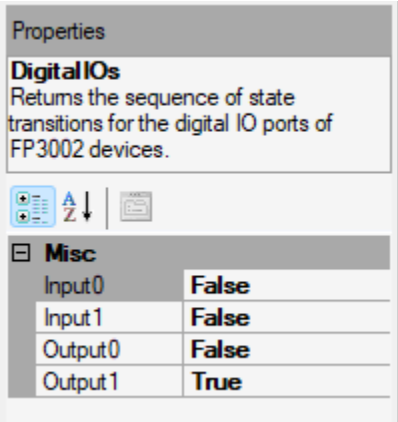
<b>Digital IOs</b>	
<b>Digital Input 0</b>	Start Stimulation: Finite
<b>Digital Input 1</b>	None
<b>Digital Output 0</b>	Software
<b>Digital Output 1</b>	Both

This will allow the +5V TTL signal on the specified port to control the stimulation of the FP3002 system. While the TTL signal is HIGH, the system will be stimulating and while the TTL signal is LOW, the system will not be stimulating. The pulse train parameters can be configured within the “Opto-Stimulation” tabpage.

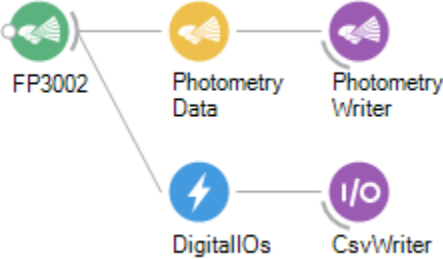
The screenshot displays the 'FP3002 Setup' software window, specifically the 'Opto-Stimulation' tab. The interface includes several control panels and a visualization area.

- Navigation:** Tabs for 'Emission Alignment', 'Excitation Sequence', 'Excitation Power', 'Opto-Stimulation', and 'Digital IOs'. Buttons for 'Measure Power' and 'Align Laser' are located at the top left.
- Parameter Settings:**
  - Wavelength:** A dropdown menu currently set to '635nm'.
  - Amplitude:** A slider control set to '50.0%'.
  - Frequency:** A slider control set to '10.0 Hz'.
  - Pulse Width:** A slider control set to '50 ms' and '50%'.
  - Pulse Train Type:** A dropdown menu set to 'Finite' with a 'Count: 10'.
- Excitation Sequence Graph:** A bar chart titled 'Excitation Sequence' showing four pulses of different colors (purple, blue, green, red) over a time axis from 0 to 562.5 ms. The pulses are labeled on the y-axis as L415, L470, L560, and 635nm.
- Opto-Stimulation Calibration:** A text area providing instructions:
  - To enable Opto Stim, insert the laser key into the stub wall of the system, and turn clockwise. Then select the desired wavelength from the drop down menu and connect the white ribbon cable inside the system to the corresponding laser.
  - For 635 nm only: Click "Align Laser", and adjust the XY translator inside the FP3002 system to align the laser to the desired patch cord branch.
  - For 450 nm only: Attach a secondary patch cord (mono branch or bifurcating) to the secondary FC connector on the stub wall of the system.
  - For both lasers: Click "Measure Power" and use an external power meter to set laser power. Then configure all other stimulation parameters
- Navigation Buttons:** 'Previous', 'Write To Persistent Registers', 'Save All Settings', 'Load All Settings', and 'Next' are located at the bottom.

Now the laser state can be recorded using the “Digital IOs” node connected after the “FP3002” node, in parallel with the “Photometry Data” node. Be sure to configure the “Digital IOs” node such that the “Output1” property is set to “True”.

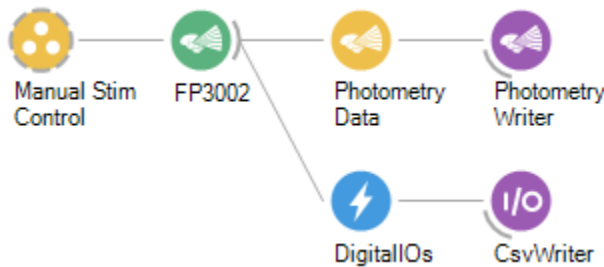


This “Digital IOs” node can be connected directly to a “Csv Writer” node to record the timestamped laser state. This data set will already be aligned to the photometry data set since it also uses the system’s internal clock to timestamp the state changes of the laser.

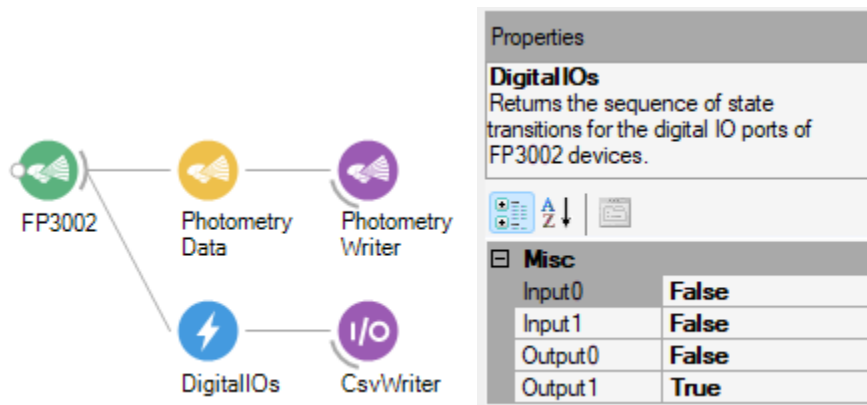


# Manual Control

This section discusses two ways to construct a Bonsai workflow that will give the user full manual control over stimulation. Both cases utilize the “Stimulation” node for creating commands for starting and stopping stimulation. These commands will be sent through the “FP3002” node to the FP3002 system. In one case, we use separate keys for the start and stop commands, and in the other case we use any key to toggle between stimulation states.



In both cases, begin with the “Standard Photometry” workflow and add the “Digital IOs” node in parallel to the “Photometry Data” node. This way the workflow will be able to record both the photometry data and the laser state. Be sure to set the “Output 1” property to “True”. The “Digital IOs” node should also be connected to a “Csv Writer” in order to save the laser state signal to a .csv file.



## Case #1: Separate Keys

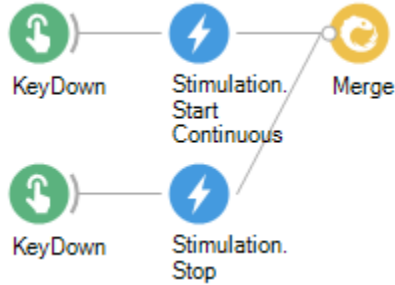
For the case in which separate keys will be used for the “Stop Stimulation” and “Start Stimulation” commands, pair two “Stimulation” nodes with two “Key Down” nodes. One “Stimulation” node should have a “Command” property of “Stop” and the other should have a “Command” property of “Start Finite” or “Start Continuous”. Be sure to set the “Filter” properties of the “Key Down” nodes so that they are distinct from each other. In the example below, the “Key Down” node connected to the “Stimulation.Stop” node has a “Filter” property set to “B”, while the other “Key Down” node has a “Filter” property of “A”. This way the “A” key will trigger the start of stimulation while the “B” key will trigger the end of stimulation.

The image displays four screenshots of software property windows, arranged in a 2x2 grid. Each window shows the configuration for a specific node.

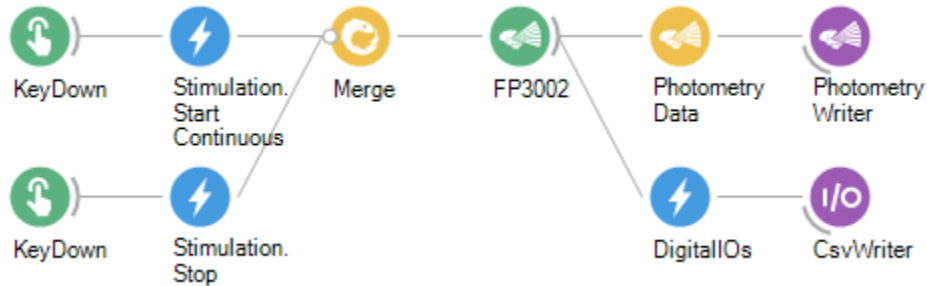
- Top-Left Window:** Properties for a **KeyDown** node. Description: "Produces a sequence of events whenever a keyboard key is depressed." The **Misc** section shows **Filter** set to **A** and **SuppressRepetition** set to **False**.
- Top-Right Window:** Properties for a **Stimulation.StartContinuous (Stimulation)** node. Description: "Starts continuous laser pulse stimulation." The **Misc** section shows **Command** set to **StartContinuous**.
- Bottom-Left Window:** Properties for a **KeyDown** node. Description: "Produces a sequence of events whenever a keyboard key is depressed." The **Misc** section shows **Filter** set to **B** and **SuppressRepetition** set to **False**.
- Bottom-Right Window:** Properties for a **Stimulation.Stop (Stimulation)** node. Description: "Stops laser pulse stimulation." The **Misc** section shows **Command** set to **Stop**.



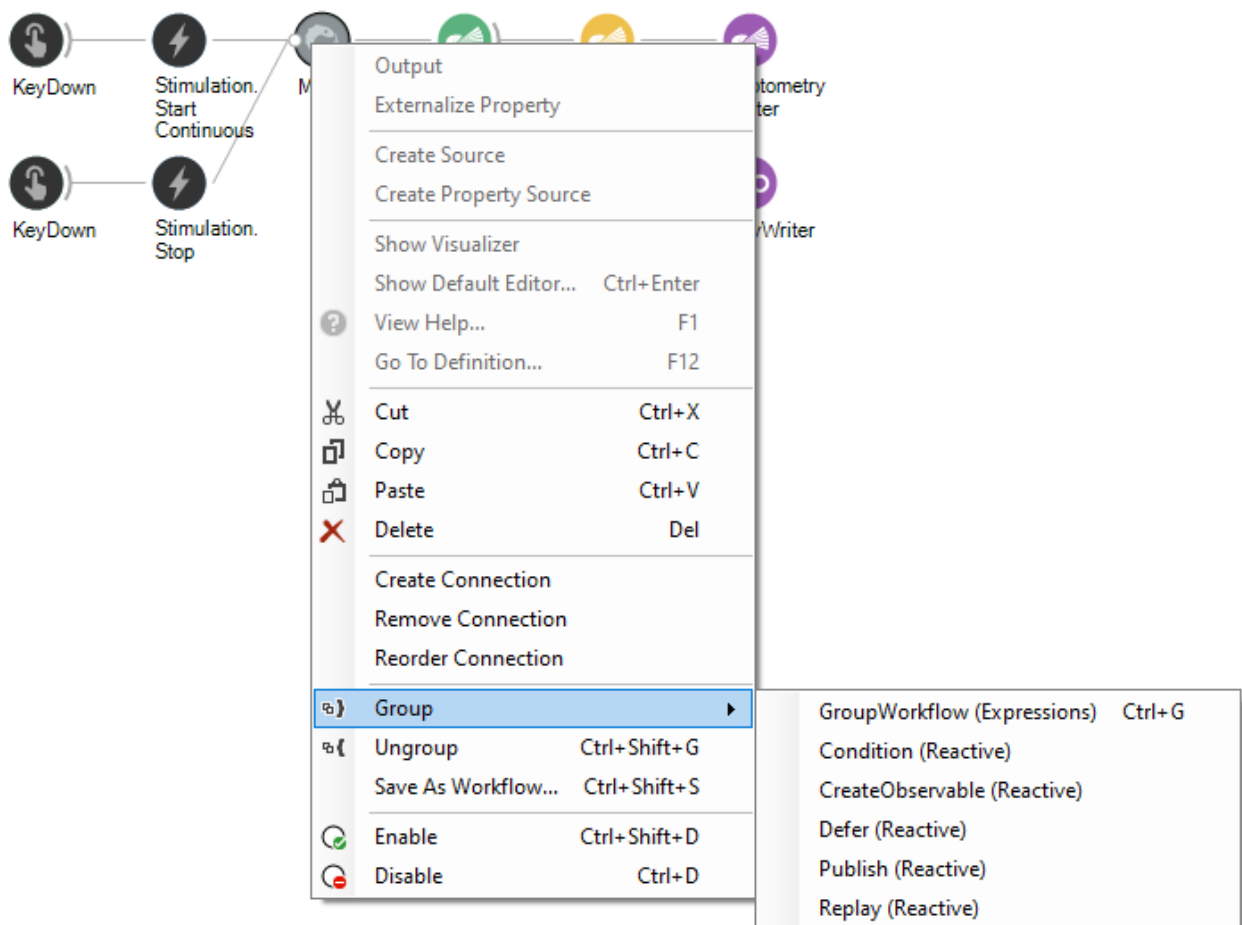
Currently, there are two data streams for producing commands to be sent to the “FP3002” node. These two data streams need to be converted to a single data stream without combining elements from both data streams into a different data type. For situations like this, the “Merge” node is ideal for converting from two data streams into one data stream. This node accepts elements from data streams that produce data of the same type and outputs the most recent element from either data stream.



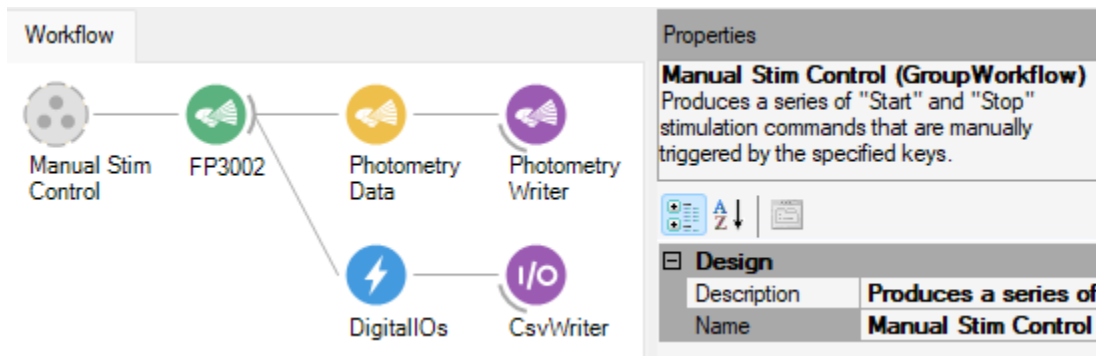
Now that the commands controlling stimulation are merged together into a single data stream, the merged data stream can be connected to the input of the “FP3002” node so that the commands can be sent to the FP3002 system after the workflow has been started.



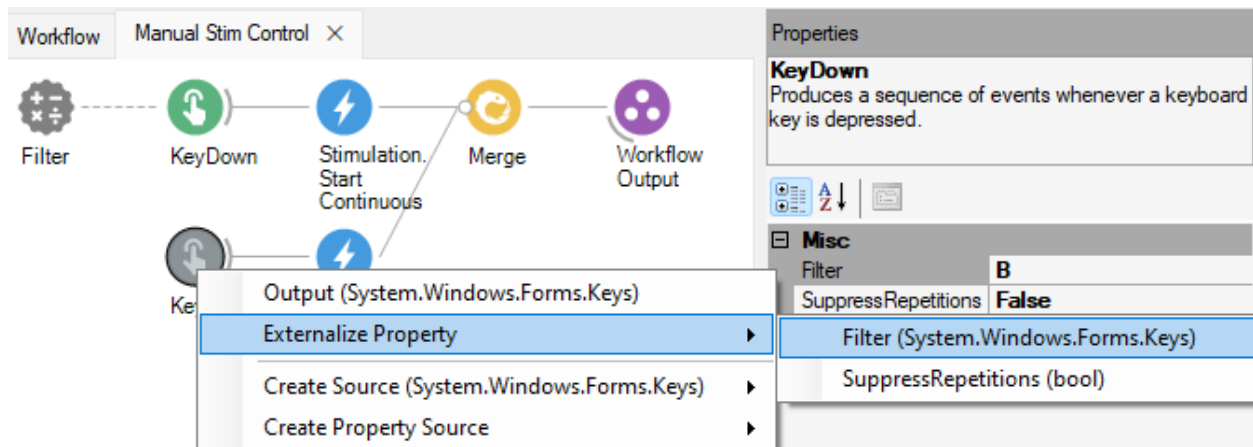
This workflow has all of the logic needed to conduct a fiber photometry experiment with manual control over laser stimulation. However, it is important to add some organization to allow for easier understanding of the workflow after some time has passed. In this workflow, we have five nodes used to control the sending of stimulation commands as well as two important parameters for the user to know before the experiment (the key filters). We can group these stimulation control nodes into a single grouped workflow, while still keeping the “Filter” properties of the “Key Down” nodes easily accessible from the top level of the workflow. To do this, select all of the nodes prior to the “FP3002” node, right click and select “Group → GroupWorkflow”.



This will encapsulate the stimulation control nodes into a single grouped workflow that the user can name and provide a description for.



In order to make the "Filter" properties easily accessible by the user, open the grouped workflow by double clicking it. Then right click each "Key Down" node and select "Externalize Property → Filter". This will create an "Externalized Mapping" node for each "Key Down" node.



These need to have unique names, so be sure to expand the “Filter” property of the “Externalized Mapping” nodes and specify unique and informative “Display Name” properties for both nodes. In the example, the filters are named “Start Stim Key” and “Stop Stim Key”.

The screenshot shows a workflow editor window titled "Manual Stim Control". The workflow consists of the following nodes: "StartStimKey" (grey), "KeyDown" (green), "Stimulation.Start Continuous" (blue), "Merge" (yellow), and "Workflow Output" (purple). A second path starts with "StopStimKey" (grey), "KeyDown" (green), and "Stimulation.Stop" (blue), which also feeds into the "Merge" node. The "Properties" panel on the right shows the "StartStimKey (ExternalizedMapping)" node selected. Under the "Filter" section, the "Filter" property is expanded to show a table with the following data:

Filter	(StartStimKey)
Category	
Description	
DisplayName	<b>StartStimKey</b>

With the “Filter” properties externalized, they are now easily accessible from the properties panel of the grouped workflow. Although this particular workflow was somewhat basic, a little time spent organizing the workflow can go a long way. This is especially true for when workflows start to become more complex like in the later portions of this “Stimulation” chapter.

The screenshot shows a workflow editor window titled "Manual Stim Control". The workflow consists of the following nodes: "Manual Stim Control" (grey), "FP3002" (green), "Photometry Data" (yellow), "Photometry Writer" (purple), "DigitalIOs" (blue), and "CsvWriter" (purple). The "Properties" panel on the right shows the "Manual Stim Control (GroupWorkflow)" node selected. Under the "Design" section, the "Description" property is expanded to show a table with the following data:

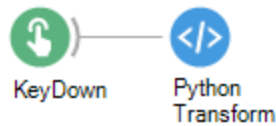
Description	Produces a series of
Name	<b>Manual Stim Control</b>

Under the "Misc" section, the "StartStimKey" and "StopStimKey" properties are expanded to show a table with the following data:

StartStimKey	<b>A</b>
StopStimKey	<b>B</b>

## Case #2: Toggable

For the case in which the stimulation state is to be toggled by any keystroke, the stimulation control logic must be altered. Begin with a “Key Down” node connected to a “Python Transform” node. We will then write a script within the “Python Transform” node that will contain an internal counter keeping track of the number of times the “Key Down” node has been triggered. Then we will output whether that internal counter is even or odd. This way the output from the python script will alternate between True and False every time a key is pressed.



```
Python Script
# Global 'count' Variable incremented every time an element
# is passed through this node. The count is initialized to -1
# so that the first output will be True (the "Start Stimulation"
# trigger).
count = -1

# This node returns a boolean value (True or False).
@returns(bool)
def process(value):
    global count    # Allow global access to the 'count' variable.

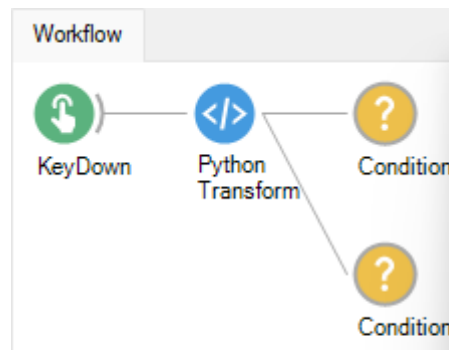
    # Increment the 'count' variable every time an element is passed through this node.
    count = count + 1

    # Return True if the count is Even and False if the count is Odd.
    return count % 2 == 0

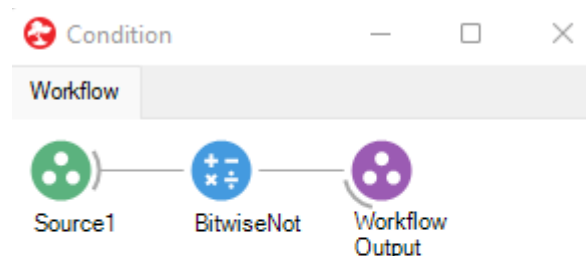
OK Cancel
```

This python script initializes the ‘count’ variable to ‘-1’ when the Bonsai workflow is started. Then every time a value is passed through the node, the count variable is incremented. The script outputs `count % 2 == 0` which is True if the count is Even and False if the count is Odd.

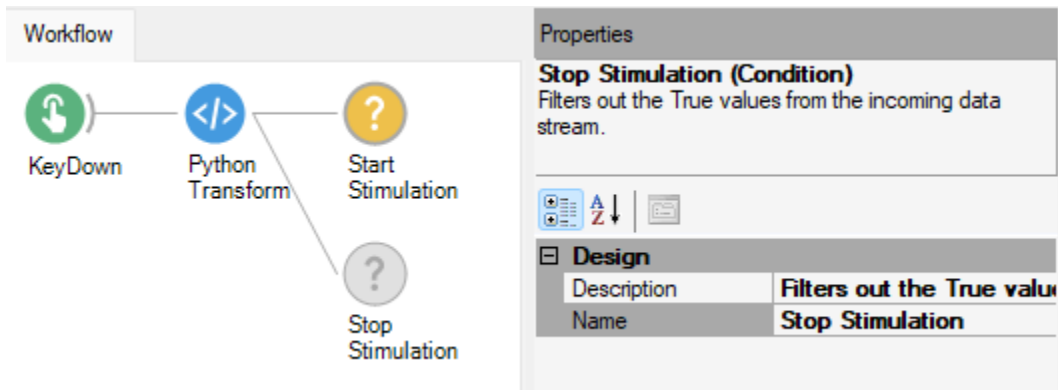
Once the software trigger is configured to alternate between True and False we need to have the True values send the “Start Stimulation” command to the “FP3002” node and we need the False values to send the “Stop Stimulation” command. In order to accomplish this, we need to separate the boolean values coming from the python script into two data streams, one containing the True values the other containing the False values. An easy way to separate into multiple data streams based on a condition is to use the “Condition” node. Connect two “Condition” nodes to the “Python Transform” node in parallel to each other.



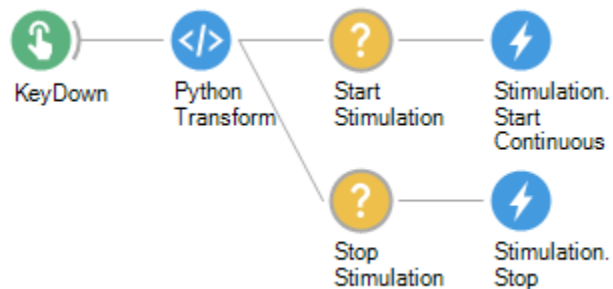
The “Condition” node filters the elements of an observable sequence according to the condition specified by the encapsulated workflow. This means that the values appearing at the input of the “Condition” node will only appear at its output when the logic inside of the node returns a True value. Since the values coming from the “Python Transform” node are boolean values, no additional logic needs to be added within the “Condition” node to filter out all of the False values from the data stream. However, in order to filter out all of the True values, a “Bitwise Not” node needs to be added to the “Condition” node. This “Bitwise Not” node will convert all of the True values to False and vice versa within the “Condition” node, causing the node to only output the values from its input when they are False values. Configure one of the “Condition” nodes by double clicking it and inserting a “Bitwise Not” node between the “Source 1” and “Workflow Output” nodes.



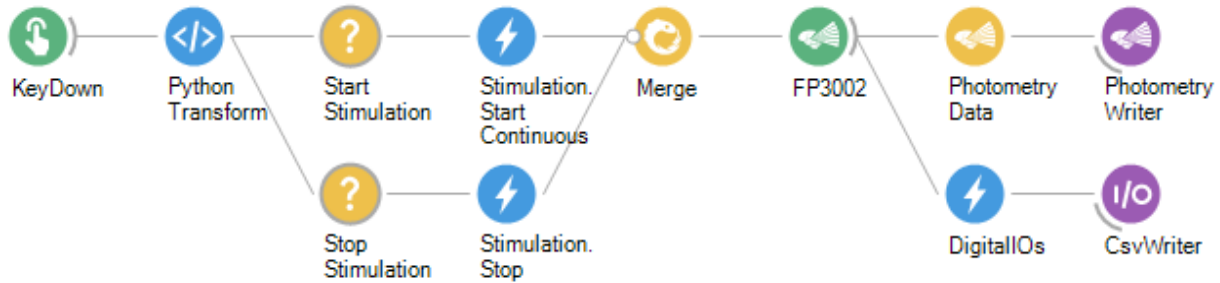
The “Condition” nodes also give the option to set the name and description of the node. It is useful to utilize these options to keep the logic organized.



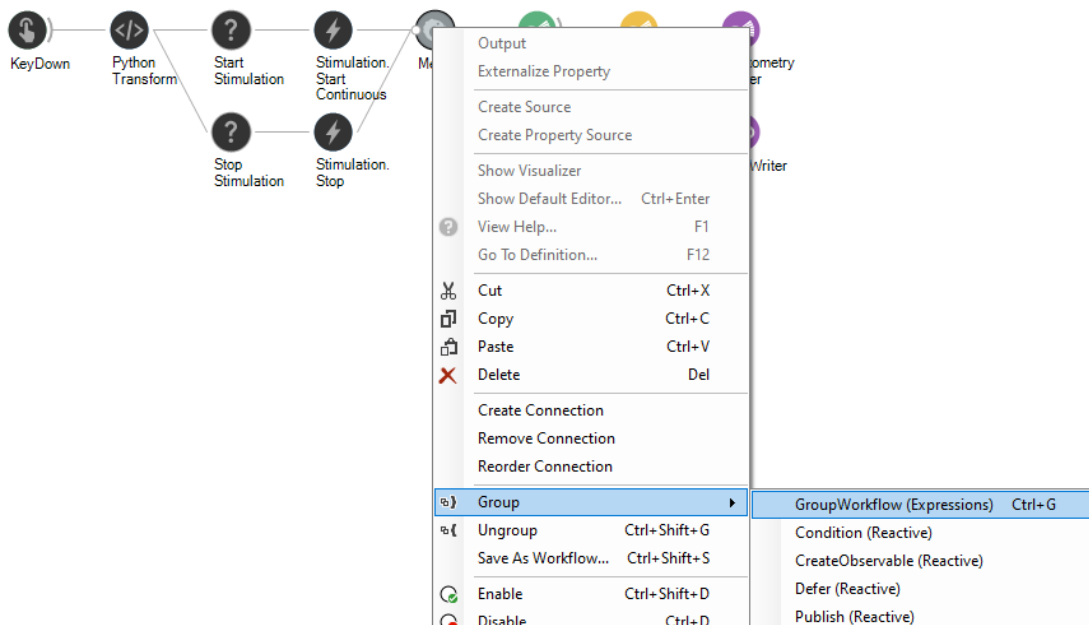
Now that we have the boolean values separated into two data streams, connect “Stimulation” nodes to each of the “Condition” nodes. Configure the “Stimulation” nodes such that the one receiving True values outputs a “Start Stimulation” command, and the other outputs the “Stop Stimulation” command.



We now have to convert these two data streams outputting stimulation commands into a single data stream that outputs the most recent stimulation command. To do this, connect both of the “Stimulation” nodes to a single “Merge” node. This “Merge” node will output the most recent command produced by the two “Stimulation” nodes. To complete the workflow logic, connect the “Merge” node to the input of the “FP3002” node.

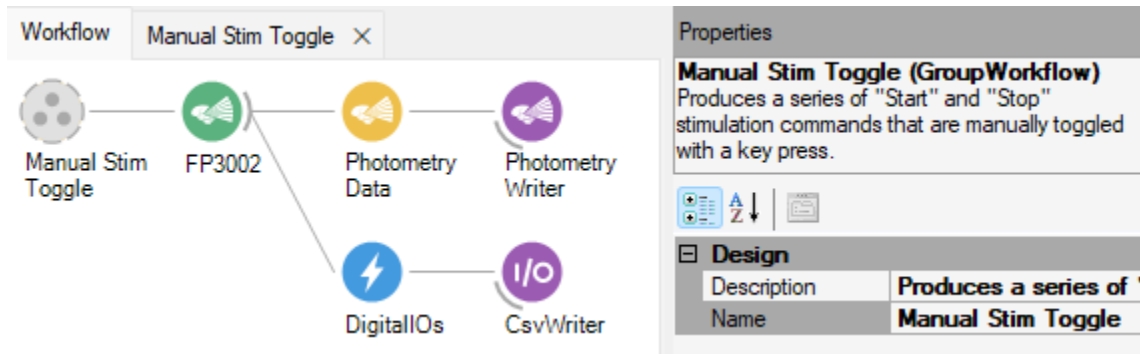


This workflow now contains all of the necessary logic for conducting a fiber photometry experiment with stimulation toggled by a key press. However, there are seven nodes used solely for the purpose of producing the stimulation commands. This can cause confusion for the user and can be easily organized to improve reusability. To organize this work, let’s group together all of the stimulation command logic into a single grouped workflow. Select all of the nodes prior to the “FP3002” node, then right click and select “Group → GroupWorkflow”.

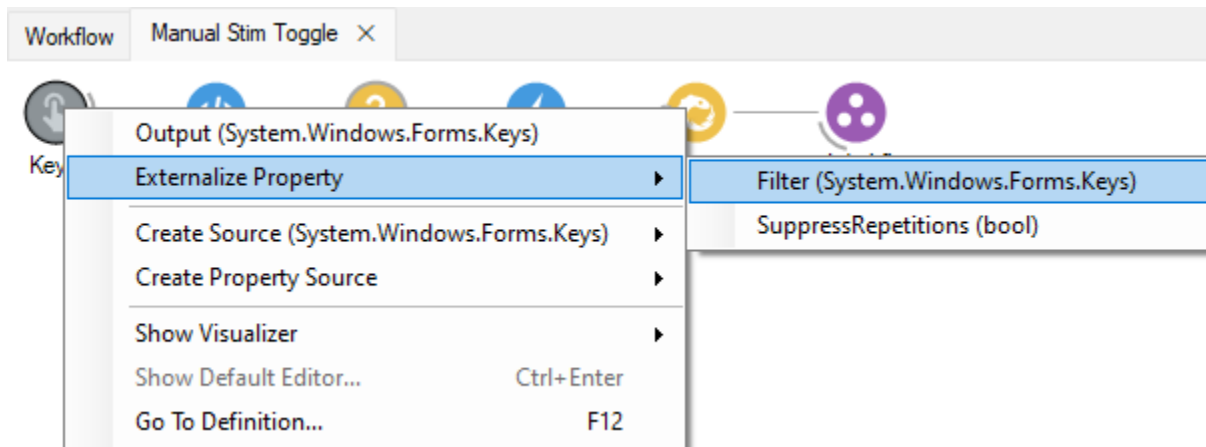




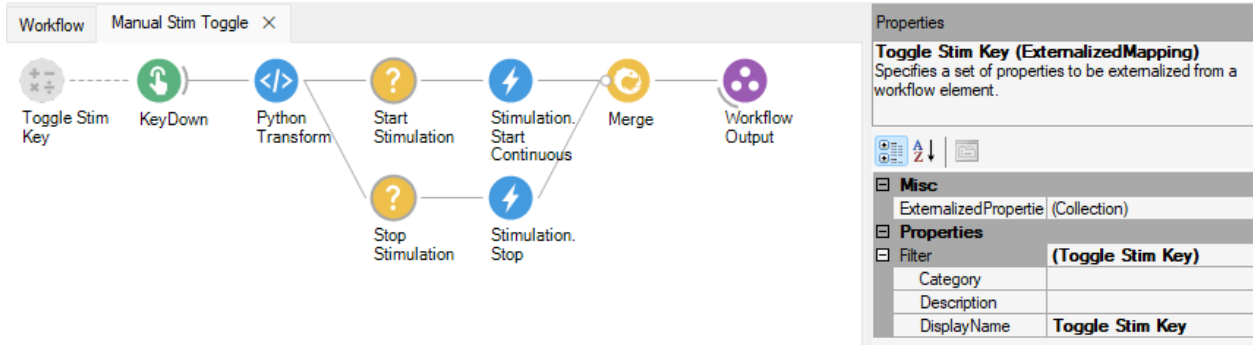
This will encapsulate the stimulation control nodes into a single grouped workflow that the user can name and provide a description for.



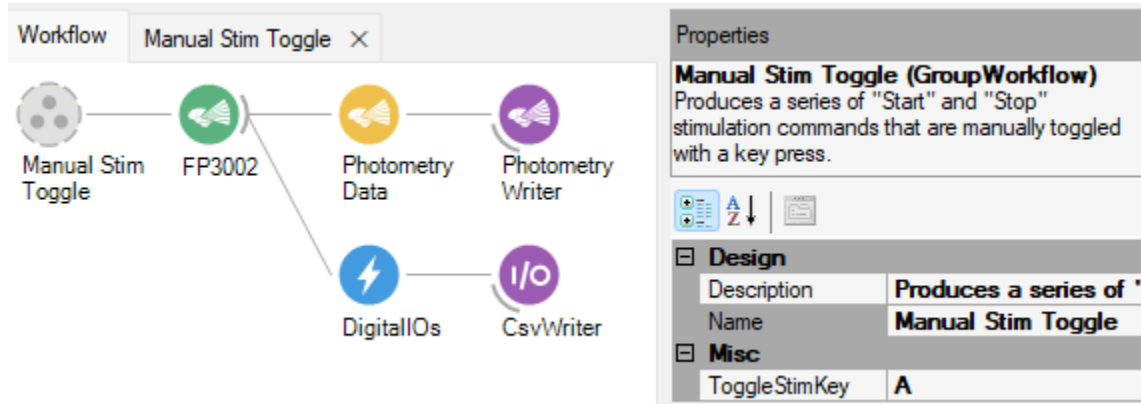
In case the user wants to use a specific key press to toggle stimulation, we can easily make the "Filter" property of the "Key Down" node accessible from the top level of the workflow. To do this, open the grouped workflow by double clicking it, right click the "Key Down" node, and select "Externalize Property → Filter". This will create an "Externalized Mapping" node for the "Key Down" node.



You can specify the display name of the "Filter" property by expanding the "Filter" property of the "Externalized Mapping" node and specifying a unique and informative "Display Name" property. In the example, the filter property is named "Toggle Stim Key".

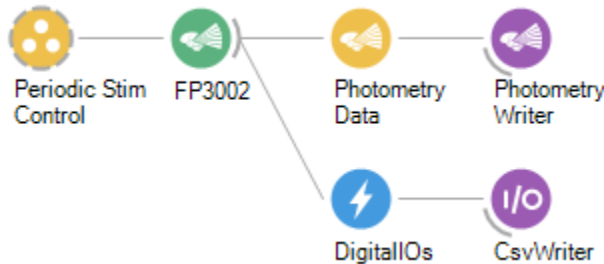


With the “Filter” property of the “KeyDown” node externalized, it is now easily accessible from the properties panel of the grouped workflow.

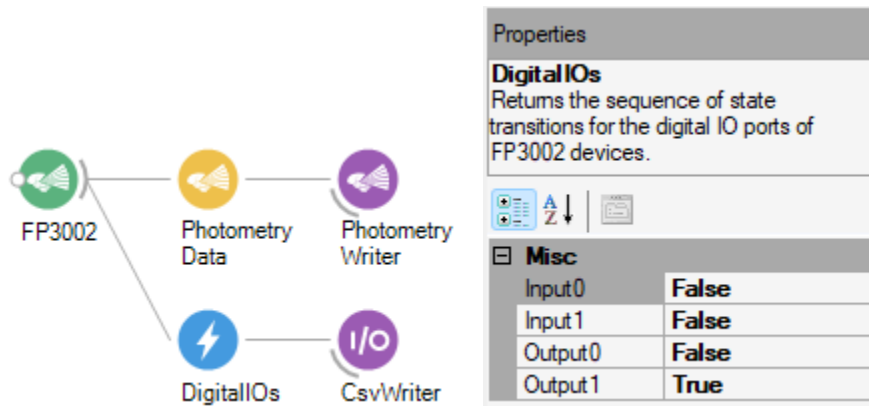


# Basic Periodic Control

In this section, we discuss a method for periodic control of stimulation with a 50% duty cycle. This means that the system will alternate the stimulation state, spending an equal amount of time in each state. For cases where periodic control is desired, but a 50% duty cycle is not desired, please see the [“Period Control, Variable Duty Cycle”](#) section of this chapter.

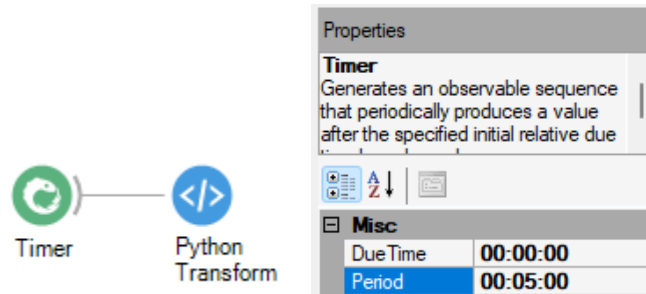


To construct this workflow, begin with the “Standard Photometry” workflow and add the “Digital I/Os” node in parallel to the “Photometry Data” node. This way the workflow will be able to record both the photometry data and the laser state. Be sure to set the “Output1” property of the “Digital I/Os” node to “True”. The “Digital I/Os” node should also be connected to a “Csv Writer” in order to actually save the laser state signal to a .csv file.



The rest of the workflow consists of logic for controlling the production of the stimulation commands. One way to construct the software trigger logic is to connect a “Timer” node to a “Python Transform” node. The “Timer” node can be configured to output an incremented value after a specified time interval has elapsed. Then the

“Python Transform” node can accept the integer from the “Timer” node and output True or False based on whether the value is Even or Odd. This setup will output a periodic stream of boolean values with a 50% duty cycle as desired.



To properly configure the “Timer” node, set the “DueTime” to zero, unless a delay to the first command is desired. Then specify the “Period” property to be the duration of time for stimulation to occur each cycle. Since the duty cycle of this particular workflow is 50%, this will also be the period of time that stimulation does not occur during each cycle.

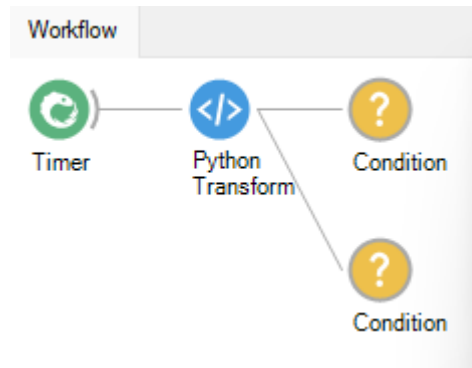
```
Python Script

# Returns True or False based on whether
# the input value is Even or Odd.
# A Timer node's first value is "0"
# so to have the initial output of this
# "Python Transform" node be True,
# return whether the value is even (value % 2 == 0)
# Otherwise, to have the initial output be False,
# return whether the value is odd (value % 2 == 1)
@returns(bool)
def process(value):
    return value % 2 == 0
```

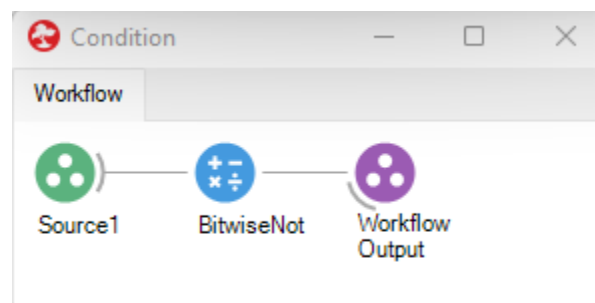
The script within the “Python Transform” node simply checks whether or not the value is Even or Odd. The script does this by outputting `count % 2 == 0`, which is True if the value is Even and False if the value is Odd. Since the first value that the “Timer” node outputs is “0”, True will be the first value output by the “Python Transform” node.

Once the software trigger is configured to alternate between True and False we need to have the True values send the “Start Stimulation” command to the “FP3002” node and

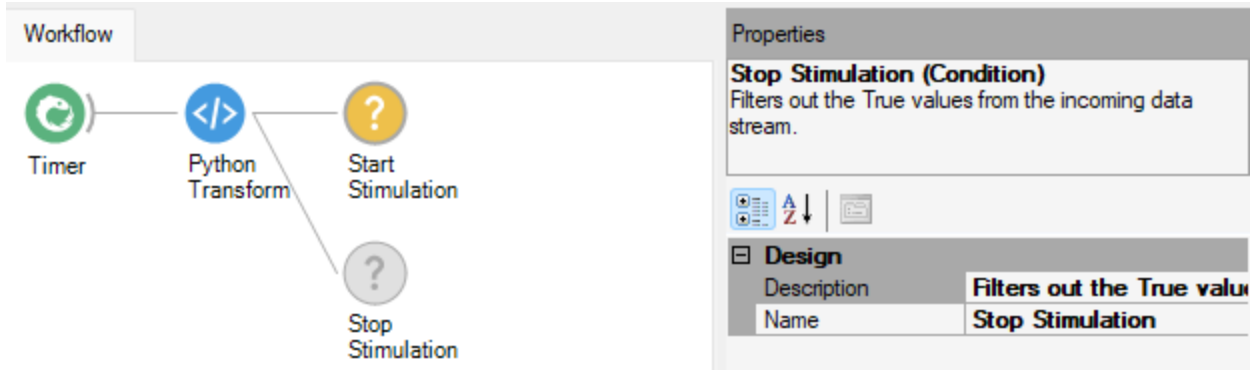
we need the False values to send the “Stop Stimulation” command. In order to accomplish this, we need to separate the boolean values coming from the python script into two data streams, one containing the True values the other containing the False values. An easy way to separate into multiple data streams based on a condition is to use the “Condition” node. Connect two “Condition” nodes to the “Python Transform” node in parallel to each



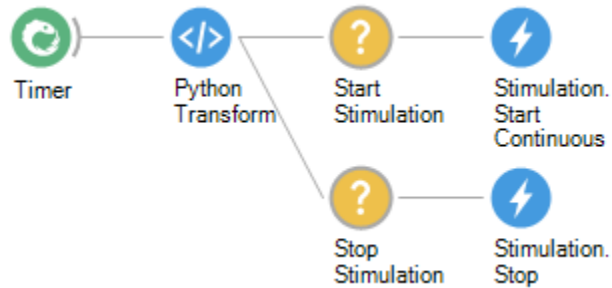
The “Condition” node filters the elements of an observable sequence according to the condition specified by the encapsulated workflow. This means that the values appearing at the input of the “Condition” node will only appear at its output when the logic inside of the node returns a True value. Since the values coming from the “Python Transform” node are boolean values, no additional logic needs to be added within the “Condition” node to filter out all of the False values from the data stream. However, in order to filter out all of the True values, a “Bitwise Not” node needs to be added to the “Condition” node. This “Bitwise Not” node will convert all of the True values to False and vice versa within the “Condition” node, causing the node to only output the values from its input when they are False values. Configure one of the “Condition” nodes by double clicking it and inserting a “Bitwise Not” node between the “Source 1” and “Workflow Output” nodes.



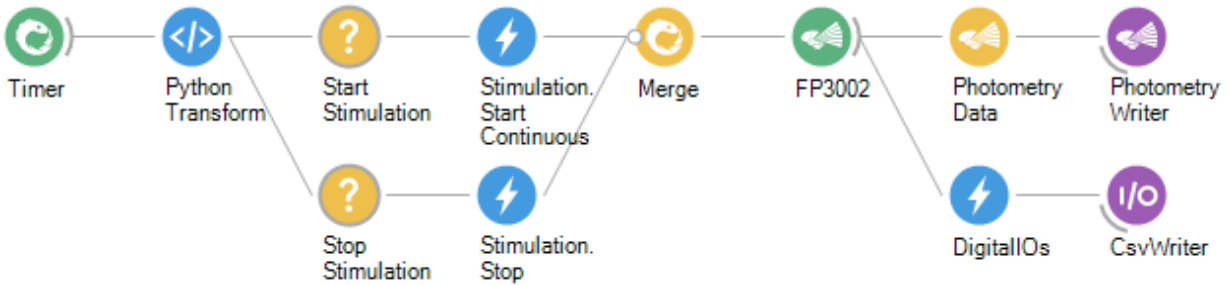
The “Condition” nodes also give the option to set the name and description of the node. It is useful to utilize these options to keep the logic organized.



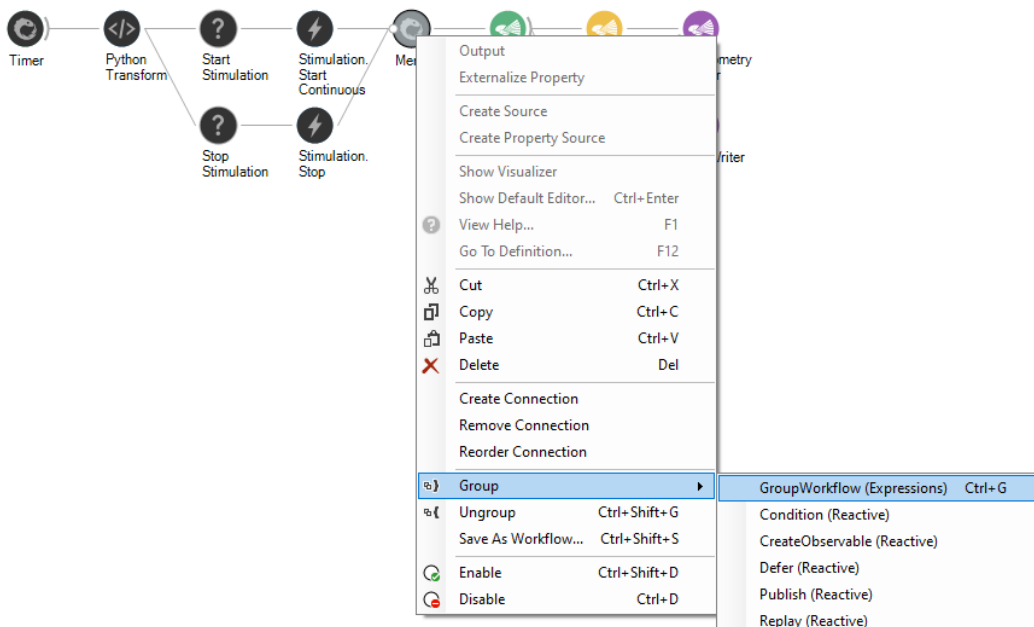
Now that we have the boolean values separated into two data streams, connect “Stimulation” nodes to each of the “Condition” nodes. Configure the “Stimulation” nodes such that the one receiving True values outputs a “Start Stimulation” command, and the other outputs the “Stop Stimulation” command.



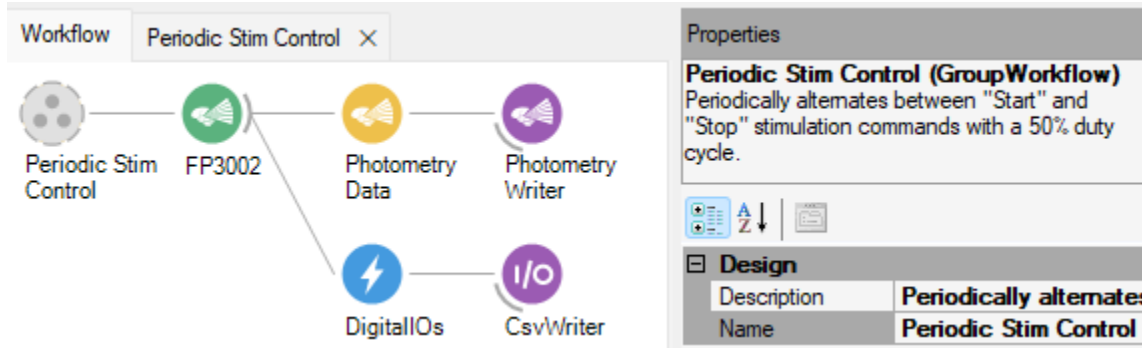
We now have to convert these two data streams outputting stimulation commands into a single data stream that outputs the most recent stimulation command. To do this, connect both of the “Stimulation” nodes to a single “Merge” node. This “Merge” node will output the most recent command produced by the two “Stimulation” nodes and can be connected directly to the “FP3002” node.



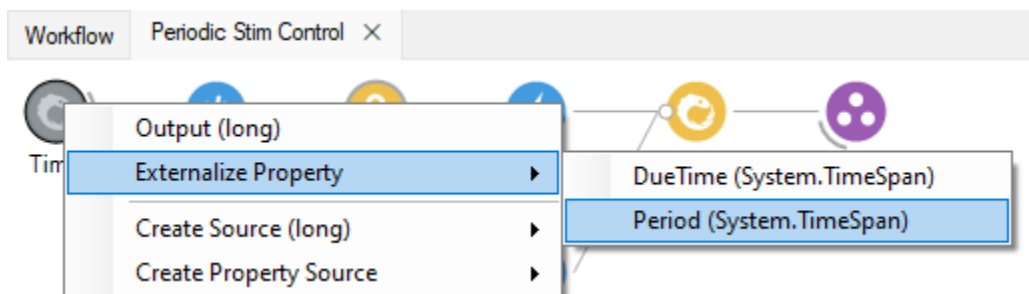
This workflow now contains all of the necessary logic for conducting a fiber photometry experiment with periodic stimulation. However, there are seven nodes used solely for the purpose of producing the stimulation commands. This can cause confusion for the user and can be easily organized to improve reusability. To organize this work, let’s group together all of the stimulation command logic into a single grouped workflow. Select all of the nodes prior to the “FP3002” node, then right click and select “Group → GroupWorkflow”.



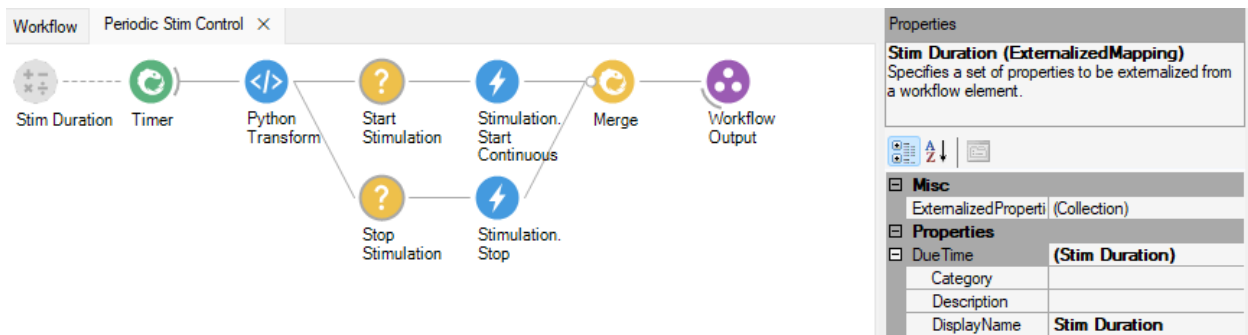
This will encapsulate the stimulation control nodes into a single grouped workflow that the user can name and provide a description for.



This now appears more organized, but the duration of stimulation needs to be easily accessible by the user. To do this, open the grouped workflow by double clicking it, right click the "Timer" node and select "Externalize Property → Period". This will create an "Externalized Mapping" node for the "Timer" node.



You can specify the display name of the "Period" property by expanding the "Period" property of the "Externalized Mapping" node and specifying a unique and informative "Display Name" property. In the example, the due time property is named "Stim Duration".





With the “Period” property of the “Timer” node externalized, it is now easily accessible from the properties panel of the grouped workflow.

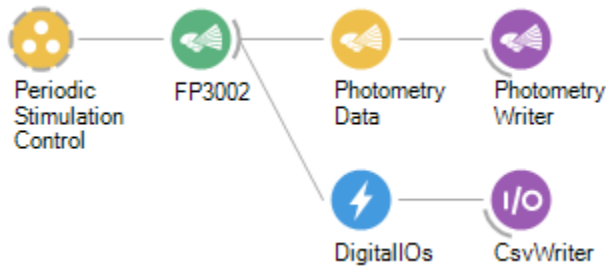
The screenshot displays a workflow editor interface. On the left, a workflow diagram shows a sequence of nodes: 'Periodic Stim Control' (grey circle with three dots), 'FP3002' (green circle with a fan), 'Photometry Data' (yellow circle with a fan), 'Photometry Writer' (purple circle with a fan), 'DigitalIOs' (blue circle with a lightning bolt), and 'CsvWriter' (purple circle with 'I/O'). A line connects the 'FP3002' node to the 'Properties' panel on the right.

The 'Properties' panel on the right is titled 'Periodic Stim Control (GroupWorkflow)' and contains the following information:

- Description: Periodically alternates between "Start" and "Stop" stimulation commands with a 50% duty cycle.
- Design section:
  - Description: Periodically alternates
  - Name: Periodic Stim Control
- Misc section:
  - Stim Duration: 00:05:00

# Periodic Control, Variable Duty Cycle

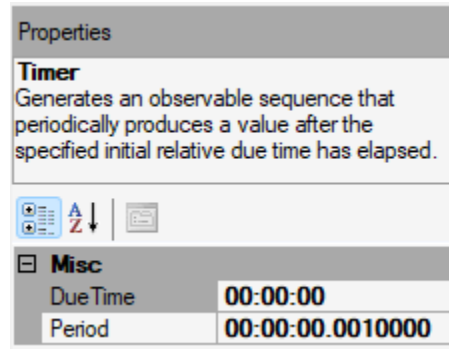
The “Stimulation: Basic Periodic Control” workflow is a particular implementation of the periodic control concept where the duty cycle is 50%. This concept can be generalized to allow for any duty cycle. In this variable duty cycle, period control workflow, the user has full control over the duration of stimulation and the duration of no stimulation.



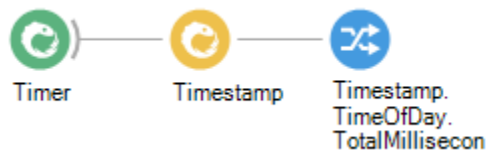
To construct this workflow, begin with the “Standard Photometry” workflow and add the “Digital IOs” node in parallel to the “Photometry Data” node. This way the workflow will be able to record both the photometry data and the laser state signal. Be sure to set the “Output 1” property to “True”. The “Digital IOs” node should also be connected to a “CsvWriter” in order to actually save the laser state signal to a .csv file.

Properties	
<b>Digital IOs</b>	
Returns the sequence of state transitions for the digital IO ports of FP3002 devices.	
[Icons]	
<b>Misc</b>	
Input0	False
Input1	False
Output0	False
Output1	True

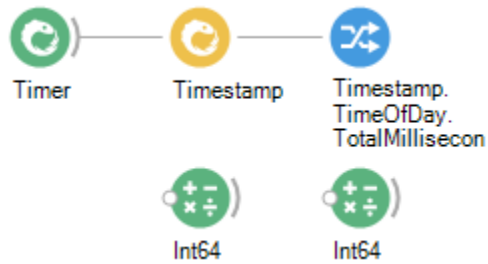
Similar to the previous Stimulation workflows, the rest of the workflow consists of logic for producing stimulation commands. Beginning with the software trigger logic, we must configure it to alternate between outputting True and False values with a specified duty cycle. Start with a “Timer” node configured with a “Period” of 0.001 seconds. This will force the “Timer” node to output a value as fast as possible.



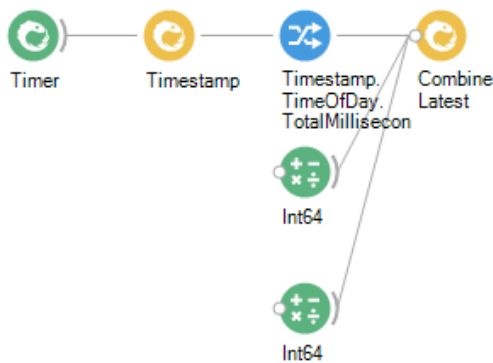
The “Timer” node does not have the precision to output a value every 1ms so it is advised to actually timestamp the values coming from this node using the computer’s timestamp. To do this, connect a “Timestamp” node after the “Timer” node. Then select the output of the “Timestamp” node to be “Time Of Day, Total Milliseconds” by right clicking the “Timestamp” node and selecting “Output → Timestamp → TimeOfDay → TotalMilliseconds”.



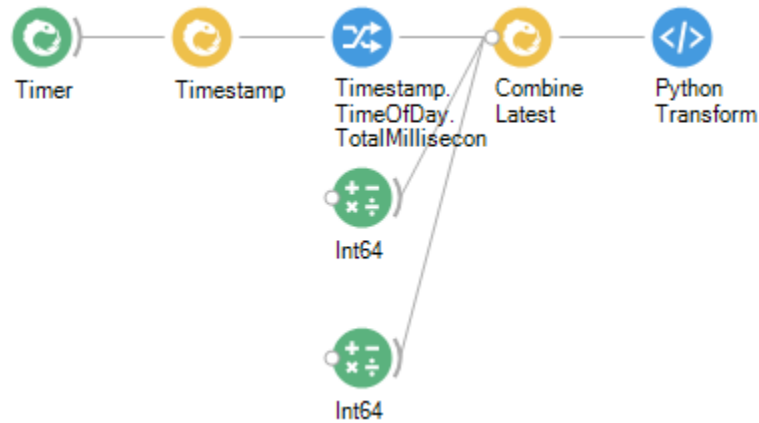
The goal for this software trigger is to alternate between outputting True and False values, where the user can specify how long to stay in each state. With this in mind, we need a way to store the duration spent stimulating versus not stimulating. This is easily done by inserting two “Int64” nodes into the workflow, parallel to the timestamped timer data stream. The “Int64” nodes will output the specified integer value once at the start of the Bonsai workflow.



With these three data streams, we have all the information we need to pass through a python script to periodically output boolean values with the desired duty cycle. However, to get this information into a “Python Transform” node, these three data streams must be combined. In this particular case, connecting the three data streams to a “Combine Latest” node is appropriate. This node will accept inputs from each data stream and output a Tuple containing the values from each stream every time any stream has produced a value. Since the “Int64” nodes only produce a value once, at the start of the Bonsai workflow, the output of the “Combine Latest” node will only produce a value when the timer’s data stream produces a value. This output will contain the current time of day in milliseconds and the ON/OFF times specified in the “Int64” nodes.



Now that we have one data stream containing all the information we need, we can connect a “Python Transform” node after the “Combine Latest” node to process the information and output the appropriate boolean value. In this python script, we will keep track of each time a new stimulation cycle starts, find the amount of time spent in the current cycle, and output a boolean value accordingly. The below script is hardcoded to have its initial output to be a True value.



```
Python Script

# Global 'startTime' variable that keeps track
# of the time that the current stimulation cycle
# started.
startTime = -1

@returns(bool)
def process(value):
    global startTime

    # Local variables for containing the information
    # coming into the script. The 'value' variable
    # is the Tuple coming from the "Combine Latest"
    # node. To access elements of a tuple you use the
    # ".itemN" notation. The order of the elements of the
    # Tuple follows from the input connections into the "Combine Latest"
    # node from Top to Bottom.
    currentTime = value.Item1
    onTime = value.Item2
    offTime = value.Item3

    # Only called for the first element passed through
    # the script. The only time this value is negative is
    # when it is initialized.
    if startTime < 0:
        # Set the start time of the cycle to the current time.
        startTime = currentTime

        # In this case the first output is True.
        return True

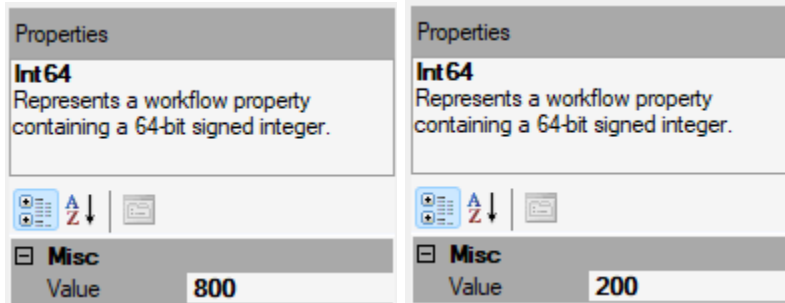
    # Calculate the amount of milliseconds spent in the current cycle
    dt = currentTime - startTime

    # Account for date change
    if startTime > currentTime:
        dt = dt + (1000.0 * 3600.0 * 24.0)

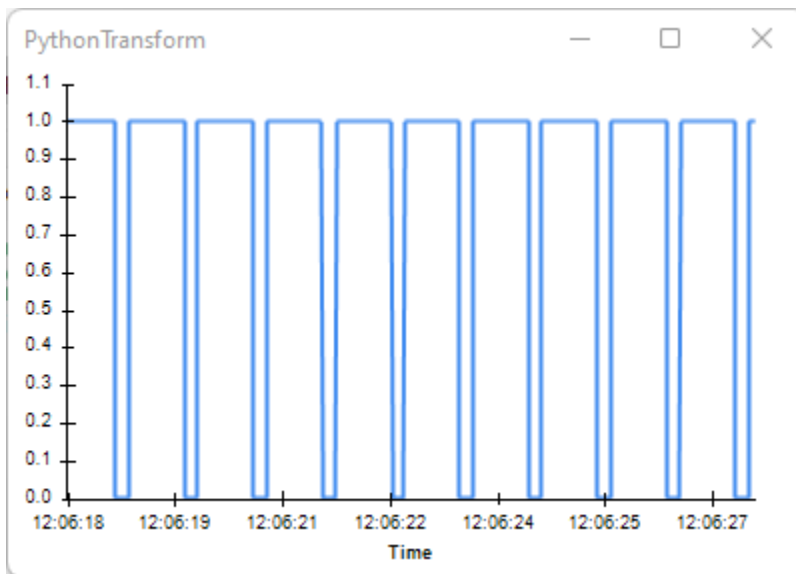
    # If currently in the ON time of the cycle, return True.
    if dt < onTime:
        return True
    # Else if currently in the OFF time of the cycle, return False.
    elif dt < onTime + offTime:
        return False
    # Otherwise, the full cycle has passed, update the start time and return True
    else:
        startTime = currentTime
        return True

OK Cancel
```

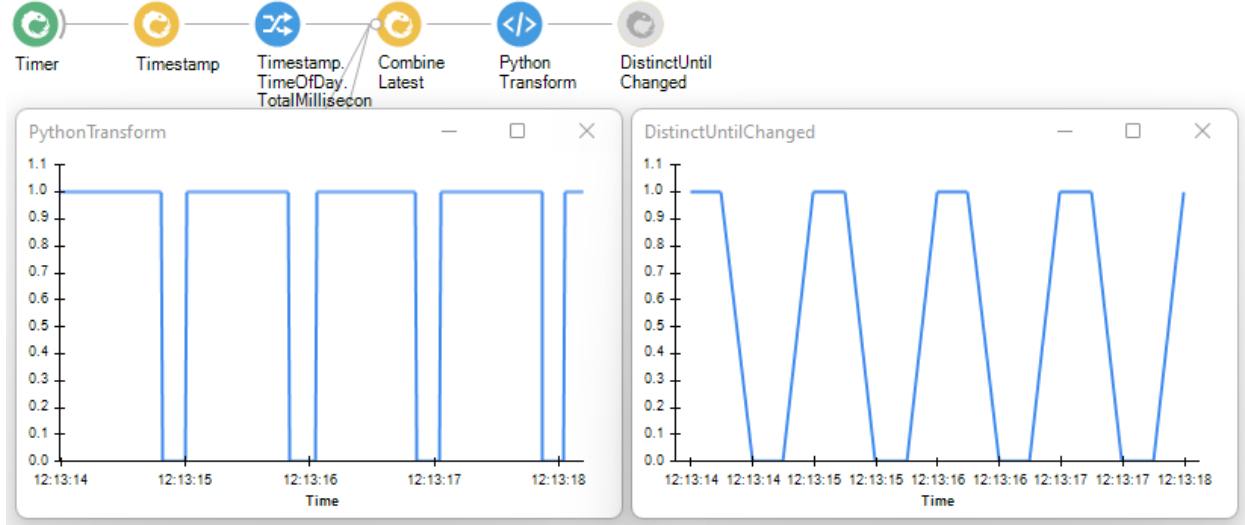
Before continuing on with constructing the workflow, this is a good place to test the logic works as intended. Set the values for the ON and OFF times by specifying the values within the “Int64” nodes. In the script above, the ON time was “Item2” of the Tuple coming from the “Combine Latest” node. This means that the top “Int64” is responsible for the ON time value, while the bottom one is responsible for the OFF time value. Our script also assumes that these durations are specified as milliseconds. In the test below, the ON time was set to 800ms and the OFF time was set to 200ms.



Once these values are set, run this software trigger section of the workflow and open the visualizer for the “Python Transform” node to check if our boolean signal appears correct.

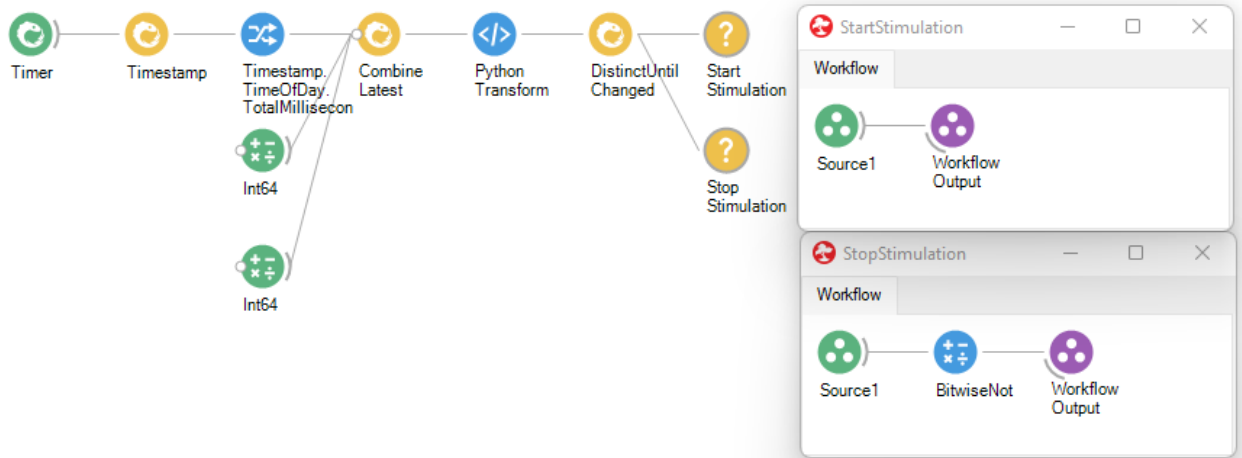


The above signal appears correct, the “Python Transform” successfully outputs True values for 800ms then False values for 200ms and repeats itself. However, this “Python Transform” outputs a value every time the timer’s data stream has a new value. If this were connected to our “Stimulation” node logic, it would be sending commands every ~20ms. Sending stimulation commands at this rate can interfere with the pulse train logic set within the “FP3002 Setup” window. The only information we want to send to the “Stimulation” nodes is when this signal changes from LOW to HIGH or HIGH to LOW. This is a case where the “Distinct Until Changed” node is applicable. This node will only output a value when the input changes.

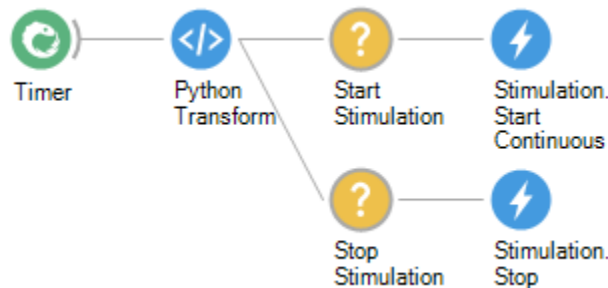




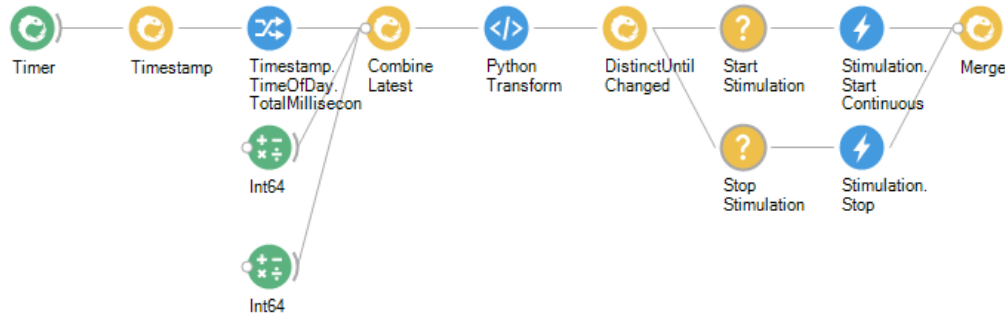
Once the software trigger is configured to alternate between True and False we need to have the True values send the “Start Stimulation” command to the “FP3002” node and we need the False values to send the “Stop Stimulation” command. Here is where we implement the “Condition” nodes to split the incoming data stream into two. Please see the previous stimulation control workflows for more information about this concept.



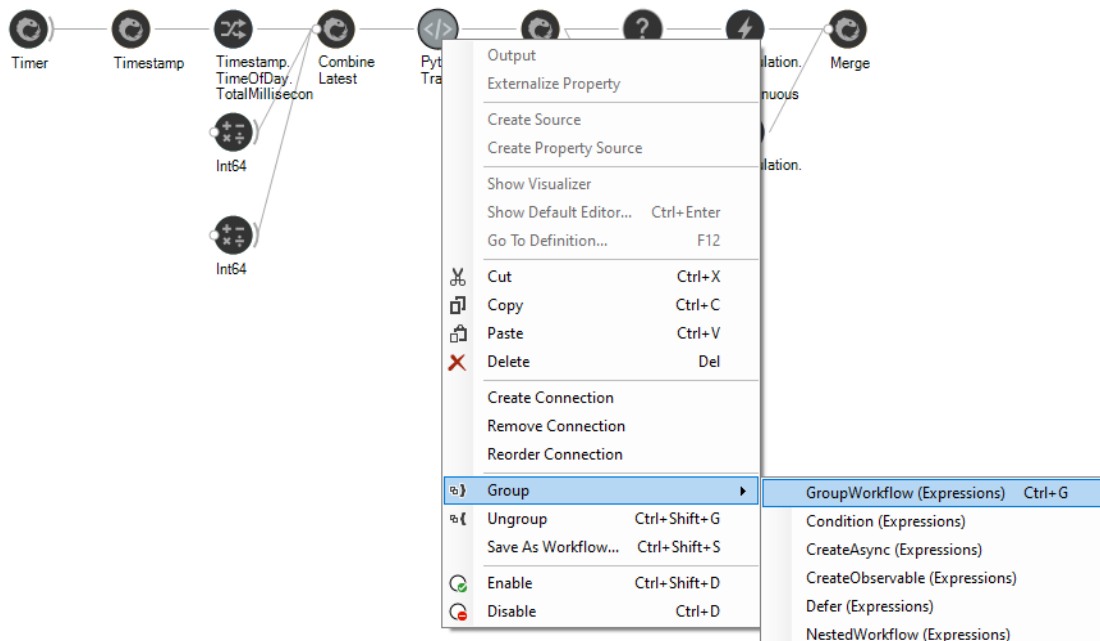
Now that we have the boolean values separated into two data streams, connect “Stimulation” nodes to each of the “Condition” nodes. Configure the “Stimulation” nodes such that the one receiving True values outputs a “Start Stimulation” command, and the other outputs the “Stop Stimulation” command.



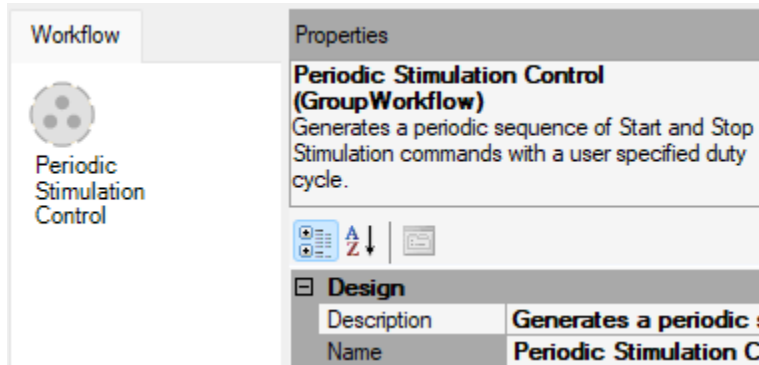
We now have to convert these two data streams outputting stimulation commands into a single data stream that outputs the most recent stimulation command. To do this, connect both of the “Stimulation” nodes to a single “Merge” node. This “Merge” node will output the most recent command produced by the two “Stimulation” nodes and can be connected directly to the “FP3002” node.



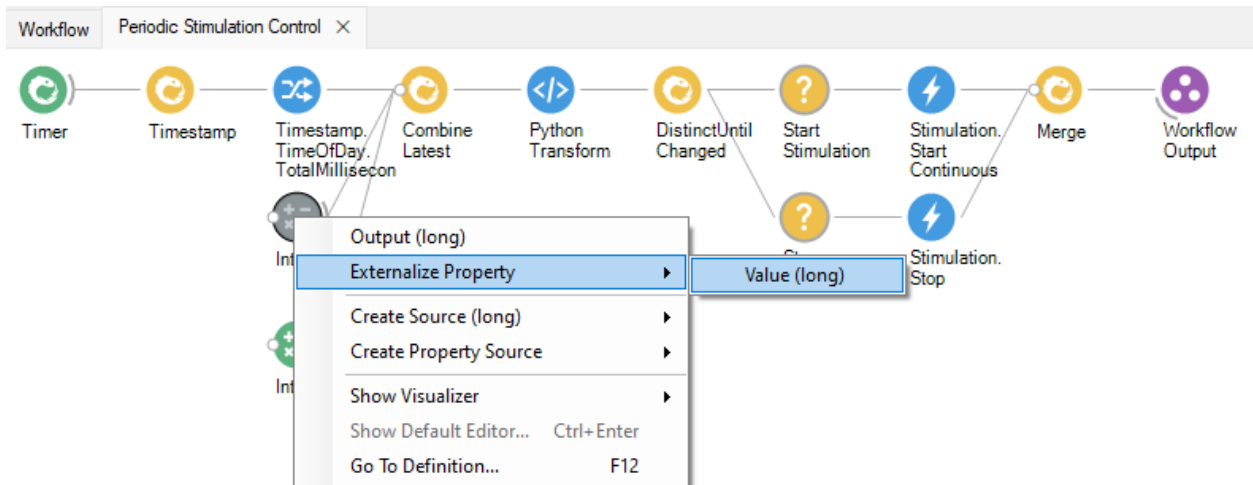
This stimulation command production logic is now ready, but there are some organizational changes that can be implemented to provide an easier user experience. First, we can group together all of these nodes into a single grouped workflow. This way the details of these operations will not distract the user. Do this by highlighting all of the software trigger logic, right clicking, and selecting “Group → GroupWorkflow”. This will encapsulate all of the logic into a single node.



The properties panel of the “Group Workflow” node allows you to name it and provide a description of the encapsulated process.



There are two properties within the software trigger logic that we want users to have easy access to, the ON and OFF times. Open up the encapsulated workflow by double clicking the node. You will see that a “Workflow Output” node has automatically been added to the end of the data stream. This allows the values to exit the encapsulated workflow. To make the ON and OFF times easily accessible, we want to externalize the “Value” property of both “Int64” nodes. Do this by right clicking each node and selecting “Externalize Property → Value”.



This will add “Value (ExternalizedMapping)” nodes to the inputs of the “Int64” nodes. Both of these “Value” nodes need to have unique “DisplayNames”. To name them, select the node, then expand the “Value” section of the properties panel and set the “Display Name”.

The screenshot shows a workflow editor with a workflow named "Periodic Stimulation Control". The workflow consists of several nodes: Timer, Timestamp, Timestamp.TotalMillisecond, Combine Latest, Python Transform, DistinctUntil Changed, Start Stimulation, Stimulation.Start Continuous, Merge, and Workflow Output. There are also two "Int64" nodes, one for "ON Time (ms)" and one for "OFF Time (ms)", which are connected to the "Combine Latest" node. The properties panel for the "OFF Time (ms)" node is expanded, showing the "Value" section with the "DisplayName" set to "OFF Time (ms)".

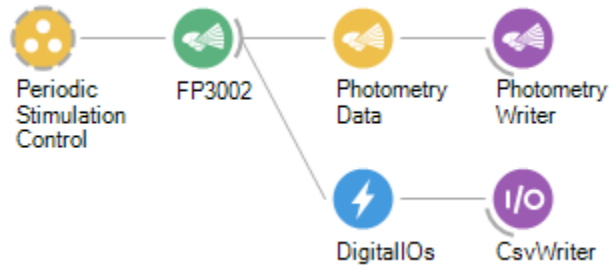
Properties	
<b>OFF Time (ms) (ExternalizedMapping)</b> Specifies a set of properties to be externalized from a workflow element.	
Misc	
ExternalizedProperty (Collection)	
Properties	
Value	(OFF Time (ms))
Category	
Description	
DisplayName	OFF Time (ms)

With these externalized properties added, the ON and OFF times can be specified from the properties panel of the grouped workflow.

The screenshot shows the properties panel for the "Periodic Stimulation Control" workflow. The panel is divided into sections: "Design" and "Misc". The "Design" section contains "Description" and "Name". The "Misc" section contains "OFF Time (ms)" and "ON Time (ms)".

Properties	
<b>Periodic Stimulation Control (GroupWorkflow)</b> Generates a periodic sequence of Start and Stop Stimulation commands with a user specified duty cycle.	
Design	
Description	Generates a periodic...
Name	Periodic Stimulation C...
Misc	
OFF Time (ms)	2000
ON Time (ms)	8000

Now that the software trigger is complete and organized, its output can be connected to the “FP3002” node of the “Standard Photometry” workflow with laser state recording.

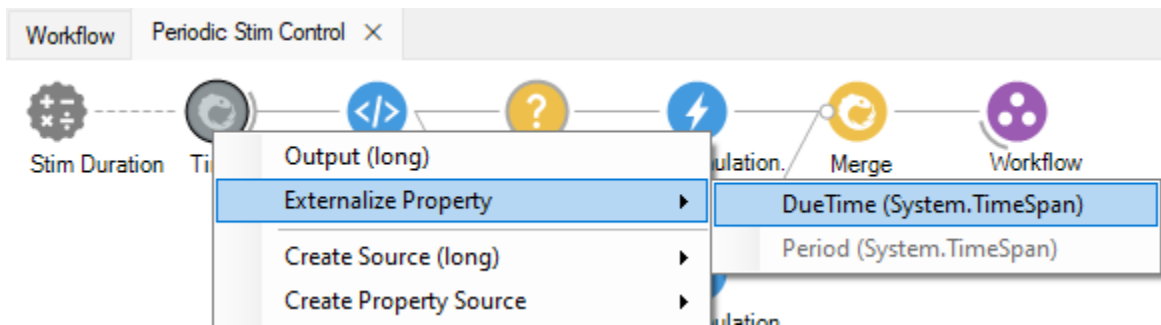


## Periodic Control, Delayed Start

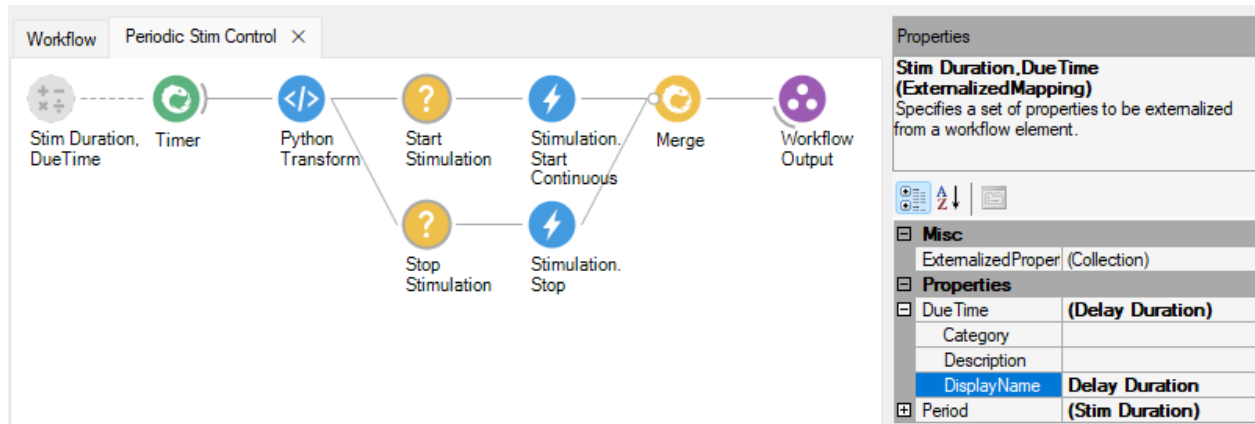
The previous workflows discussing methods for periodic control over stimulation can be expanded to include a delayed start to periodic control. This technique allows for the user to manually or automatically trigger periodic stimulation after a duration of time has passed since the Bonsai workflow has been started. With minor changes to the “Basic Periodic Control” and “Periodic Control, Variable Duty Cycle” workflows, we can enable this delayed start feature.

### Basic Periodic Control, Automated Delayed Start

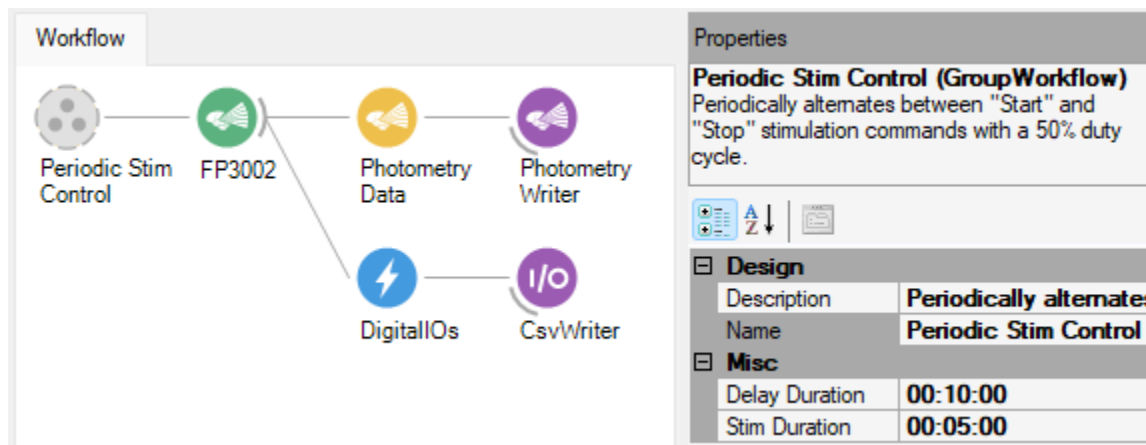
To add an automatic delayed start to the [Basic Periodic Control](#) workflow, simply specify the “Due Time” property of the “Timer” node. This will cause the “Timer” node to wait for that specified amount of time until it outputs its first value. We can make this “Due Time” easily accessible from the top level of the workflow by opening the grouped workflow and externalizing the “Due Time” property of the “Timer” node.



Set the “Display Name” of the externalized “Due Time” property to something unique and informative.



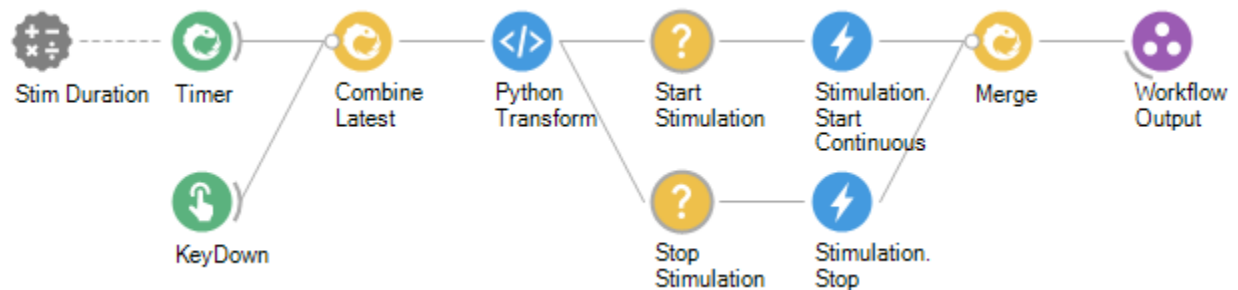
Now the delay duration can easily be set from the properties panel of the grouped workflow containing all of the stimulation command production logic. In the example below, the first “Start Stimulation” command will occur 10 minutes after the Bonsai workflow was started and then the stimulation state will alternate every 5 minutes.



## Basic Periodic Control, Manual Delayed Start

This “Basic Periodic Control” workflow can also have a manual delay to periodic control. To do this, verify that the “Due Time” is set to zero, then use a “Combine Latest” node to combine the “Timer” node with a “KeyDown” node. The “Combine

Latest” node will only output its first value after both the “Timer” and “KeyDown” data streams have produced a value.



Now, some minor changes to the “Python Transform” node need to be made. Currently, the python script assumes that the input is an integer value, but now its input is a Tuple. Also, the “Python Transform” outputs whether or not the value from the “Timer” node is even or odd. However, with the delayed start, the first value from the “Timer” that the python script sees could be even or odd depending on how long the manual delay is. To make this less arbitrary and to specify the initial output of the “Python Transform” node to be True, we need an internal counter in the python script.

```
Python Script

# Global Variables to keep track of the
# number Timer values passed through this script
# and the previous timer value. The 'count' variable
# will be updated every time a new timer value is passed
# and the previous timer value variable will prevent
# incrementation of the count variable when the KeyDown
# node produces a value.
count = 0
previousTimerVal = -1

@returns(bool)
def process(value):
    global count, previousTimerVal

    # Read in the current timer value
    timerVal = value.Item1

    # If the timer value has changed
    if timerVal != previousTimerVal:
        # Increment the internal count
        count = count + 1
        # And Update the previous timer value
        previousTimerVal = timerVal

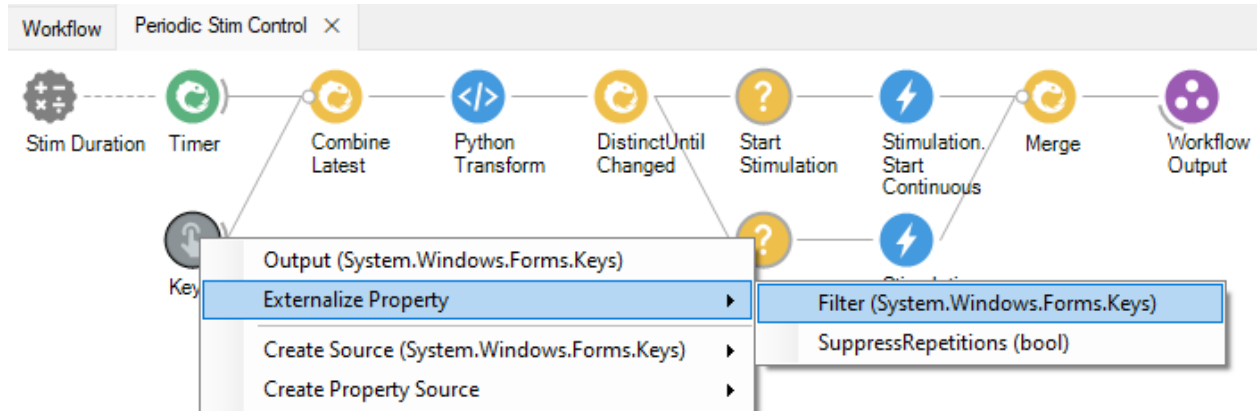
    # Return if count is odd so that the first output from this node is
    # a True value.
    return count % 2 == 1

OK Cancel
```

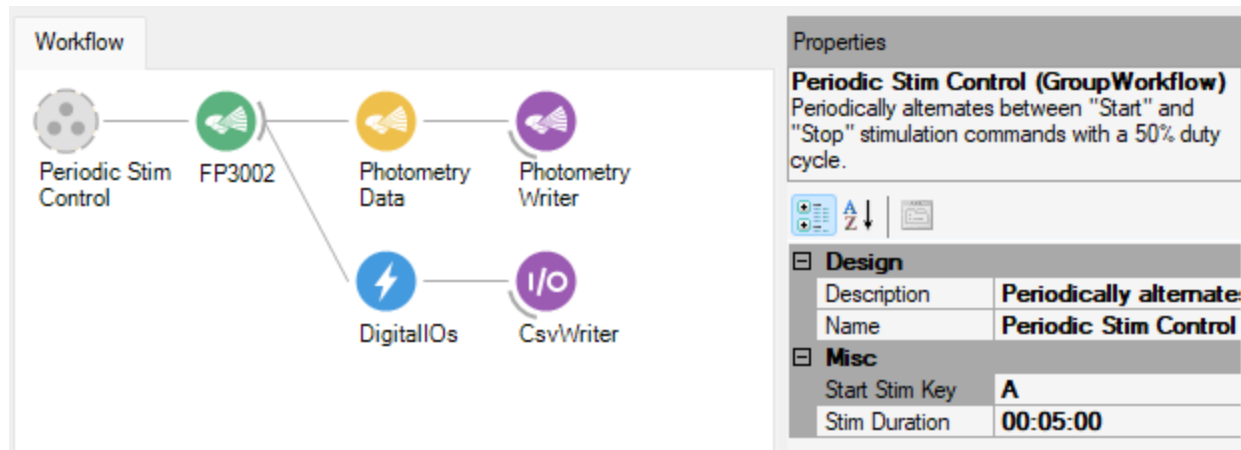


This script's main two features are that it has an internal counter to dictate the output of the script. This allows the user to more easily specify the first output of the script. This script also prevents the incrementation of the internal count when the "Key Down" node produces a value. This way if the keystroke used to start the periodic control is pressed again, it will not toggle the stimulation state. To clean up the workflow, it is useful to specify and externalize the "Filter" property of the "Key Down" node so that only one keystroke can be used to trigger stimulation.

Also, it is useful to add a “Distinct Until Changed” node after the “Python Transform” node so that subsequent key presses will not produce repeated commands acquisition control commands.

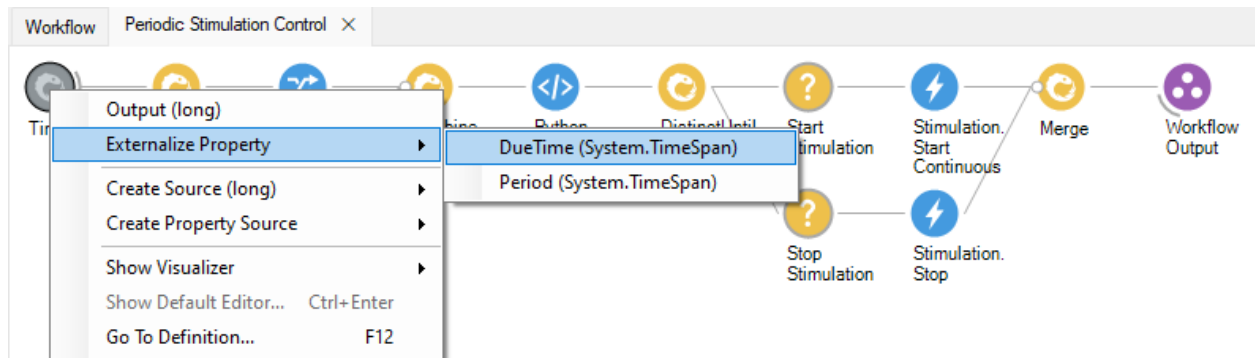


Once the externalized “Filter” property has a unique and informative “Display Name”, the filter can be set from the properties panel of the grouped workflow. In the example below, periodic stimulation will start once the user presses the “A” key and once started, it will alternate states every 5 minutes.

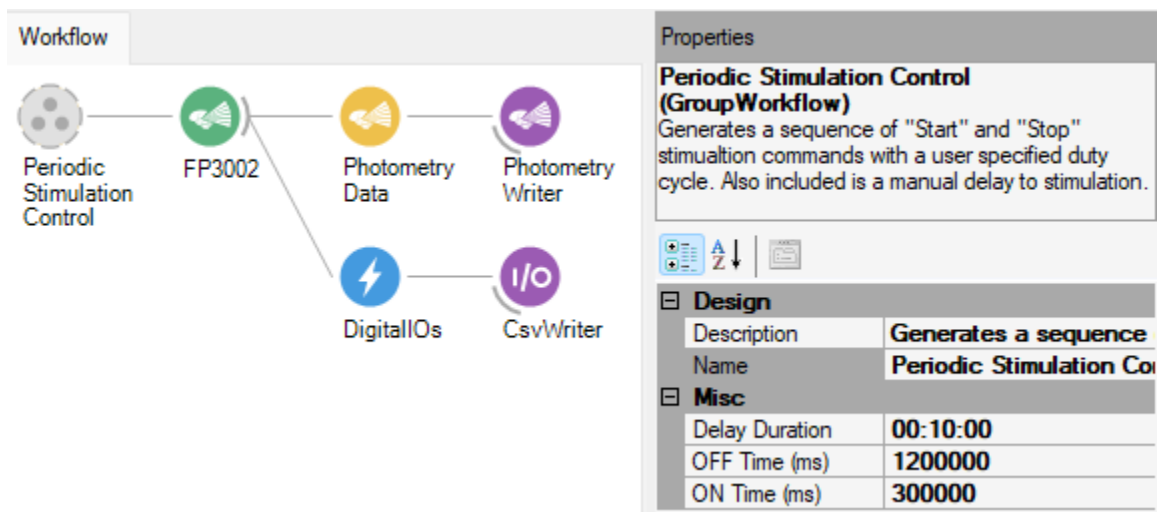


## Periodic Control, Variable Duty Cycle, Automatic Delayed Start

To implement an automatic delay to the variable duty cycle periodic control workflow, we will specify the “Due Time” property of the “Timer” node. To keep with the organizational scheme that we developed during construction of that workflow, we can externalize the “Due Time” Property of the “Timer” node so that it appears in the grouped workflow’s properties panel. You can do this by double clicking the grouped workflow to open it in a new tab, then right click the “Timer” node and select “Externalize Property → Due Time”



Once the externalized “DueTime” property has a unique and informative “DisplayName”, you can specify the duration of the delay to periodic stimulation within the properties panel of the grouped workflow. In the example below, periodic stimulation will start after a 10 minute delay. Once started it will alternate between 5 minutes of stimulation and 20 minutes of no stimulation.



**Properties**

**Periodic Stimulation Control (GroupWorkflow)**  
Generates a sequence of "Start" and "Stop" stimulation commands with a user specified duty cycle. Also included is a manual delay to stimulation.

**Design**

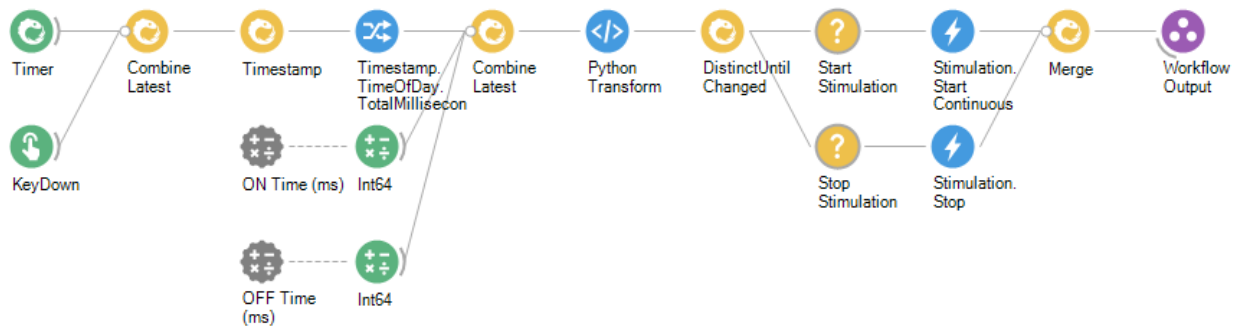
Description	Generates a sequence
Name	Periodic Stimulation Co

**Misc**

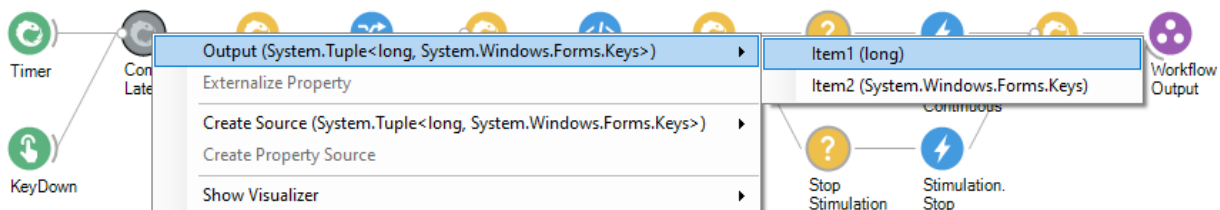
Delay Duration	00:10:00
OFF Time (ms)	1200000
ON Time (ms)	300000

## Periodic Control, Variable Duty Cycle, Manual Delayed Start

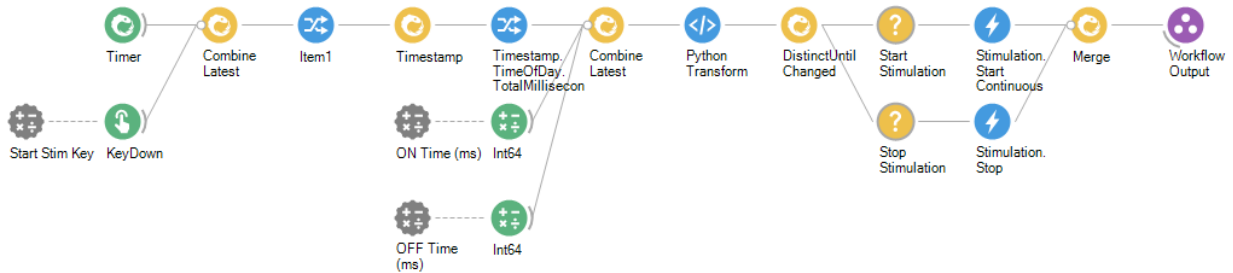
To add a manual delay to this workflow, instead of specifying the “Due Time” property, we will use a “Combine Latest” node to combine the “Timer” node and a “KeyDown” node. This way the “PythonTransform” node will not receive its first value until the specified key is pressed.



Now, we have two “Combine Latest” nodes in the same data stream leading up to the “Python Transform” node. This adds a little complexity to the data types going into python script. We can fix this data type mix-match either within the python script or within the workflow. In this case, the easiest way to fix this discrepancy is to have the “Combine Latest” node only output the first item in its Tuple (the value from the “Timer” node). Do this by right clicking the “Combine Latest” node and select “Output → Item1”.



This workflow already has a “Distinct Until Changed” node to prevent repeated outputs and already has a python script resilient to extra key presses so no other changes need to be made. It is a good idea to externalize the “Filter” property of the “Key Down” node so that it is accessible from the grouped workflow’s property panel.



Once the externalized “Filter” property has a unique and informative “Display Name”, the key used to trigger periodic stimulation can be set within the grouped workflow’s property panel. In the example below, periodic stimulation will start when the “A” key is pressed, then it will alternate between 5 minutes of stimulation and 20 minutes of no stimulation.

**Workflow**

**Properties**

**Periodic Stimulation Control (GroupWorkflow)**  
Generates a sequence of "Start" and "Stop" stimulation commands with a user specified duty cycle. Also included is a manual delay to stimulation.

**Design**

Description	Generates a sequenc
Name	Periodic Stimulation C

**Misc**

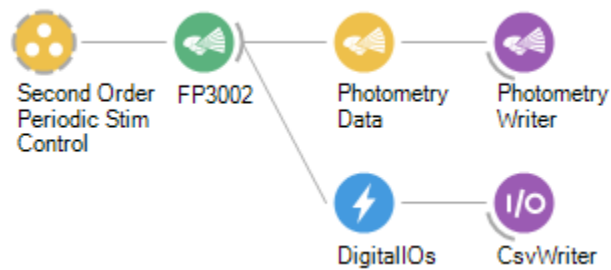
OFF Time (ms)	1200000
ON Time (ms)	300000
Start Stim Key	A

## **Second Order Periodic Control**

We have been using “periodic control” to describe cycling between starting and stopping stimulation, where the “ON time” is the duration of the laser pulse train and the “OFF time” is the duration of the no stimulation phase. In this second order periodic control section we discuss another level of control for stimulation. In some experiments, implementation of periodic control is not enough, sometimes we need to implement control over when “periodic control” occurs. This concept of “Second Order Periodic Control” involves using a second software trigger to trigger when to start and stop periodic control of stimulation. Similar to the software triggers of the periodic control workflow, this additional software trigger can be constructed to automatically or manually trigger periodic stimulation.

Before we construct manual and automatic second order periodic control workflows, let’s detail the desired functionality of these workflows. For both workflows, we want to construct a software trigger that will alternate between two states: no stimulation and periodic stimulation. At the start of the no stimulation state, the software trigger should output a “False” value. Then during the periodic stimulation state, the software trigger should alternate between “True” and “False” values with user-specified ON and OFF times. For an automatic second order control workflow, the user should be able to specify the duration of the laser pulse trains and the duration of no stimulation. For a manual second order control workflow, the user should be able to trigger a state change between no stimulation and periodic stimulation states with a key press.

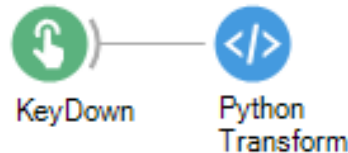
Similar to the previous data acquisition workflows, these second order periodic control workflows can be separated into three sections: the software trigger, the generation of stimulation commands, and the standard photometry section with the addition of the “Digital IOs” node to record the laser state. Our starting point in constructing this workflow will be the “Periodic Control, Variable Duty Cycle” workflow. Begin by renaming the grouped workflow containing the software trigger and the generation of stimulation commands logic, here we will name it “Second Order Periodic Stim Control”.



To change this first order periodic control workflow to a second order periodic control workflow, all we need to do is reconfigure the software trigger logic.

## Manual Second Order Periodic Control

In order to configure the software trigger logic for manual second order periodic control, we will use a “Key Down” node to toggle between periods of periodic stimulation and no stimulation. Open the “Second Order Periodic Stim Control” grouped workflow. Insert a “Key Down” node connected to a “Python Transform” node inside of this grouped workflow. We will use these two nodes to create a data stream that toggles between True and False when the user presses a key.



The python script in this new “Python Transform” node will contain an internal counter and output whether the key was pressed an Even or Odd number of times. We will use the output of this “Python Transform” node to dictate whether the system is in a periodic stimulation control state or a no stimulation state. A True value will be used to trigger periodic stimulation, while a False value will be used to trigger a stop to any stimulation.

```
Python Script

# Global variable used to store the number of times
# the "Key Down" node has produced a value.
count = 0

# Returns a boolean value that is used to determine if the system is in a
# periodic stimulation state or a no stimulation state.
@returns(bool)
def process(value):
    global count

    # Increment the global variable every
    # time the "Key Down" node produces a value.
    count = count + 1

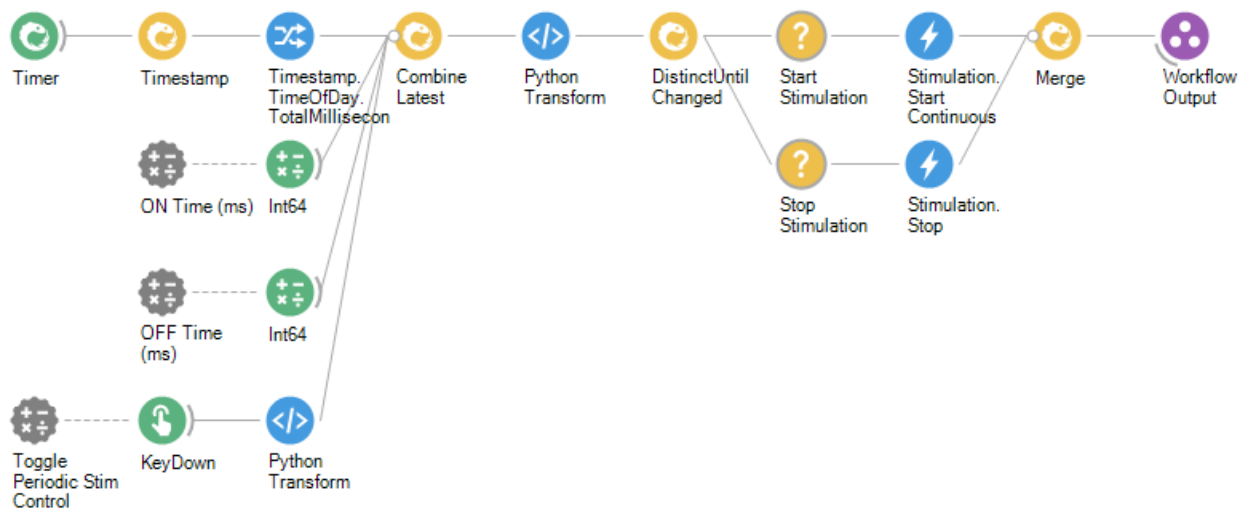
    # Return True if the count so that the first keystroke will trigger a
    # start to periodic stimulation.
    return count % 2 == 1

OK Cancel
```



This “Python Transform” will have its first output as True so that the FP3002 system will begin in the no stimulation state at the start of the workflow then toggle to the periodic stimulation state on the first keystroke.

Be sure to externalize the “Filter” property of the “Key Down” node so that it is accessible from the properties panel of the grouped workflow. Here we also edit the display name of the externalized “Filter” property to indicate its function. Specify the “Suppress Repetitions” property to “True” to prevent the “Key Down” node from producing many values for a single prolonged key press. With the “Key Down” node configured, the “Filter” property externalized, and the python script implemented, we are ready to connect this to the “Combine Latest” node.



With some additional python logic in the “Python Transform” node immediately after the “Combine Latest” node, we will finish constructing our software trigger. The boolean value dictating whether the system should be in a periodic stimulation state or a no stimulation state will be contained in the fourth element of the input Tuple. Begin by reading in this value as a local variable, we will name it “periodicStim”.

```
currentTime = value.Item1
onTime = value.Item2
offTime = value.Item3
periodicStim = value.Item4
```

Then, encompass the if, elif, else statement containing the periodic stimulation logic with an if, else statement such that the periodic stimulation logic only runs when the

“periodicStim” variable is True. Otherwise, the script should only update the “startTime” variable and return False.

```
# If the system is in the periodic stimulation state run the periodic stimulation logic
if periodicStim:
    # If currently in the ON time of the cycle, return True
    if dt < onTime:
        return True
    # If currently in the OFF time of the cycle, return False
    elif dt < onTime + offTime:
        return False
    # If the full cycle has passed, update the start time and return True
    else:
        startTime = currentTime
        return True
# Otherwise, while in the no stimulation state return False
else:
    startTime = currentTime
    return False
```

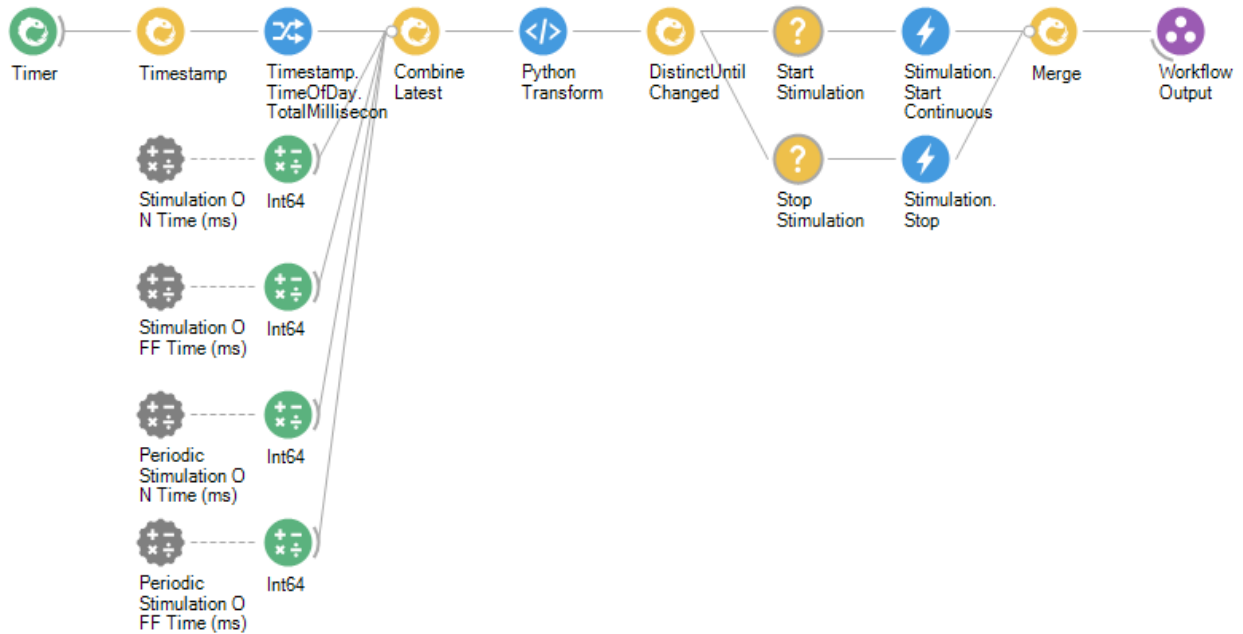
With this additional python logic our software trigger is complete. In the example below, the system will begin in the no stimulation state when the workflow is started. Then, when the “A” key is pressed, it will enter the periodic stimulation state, with a laser pulse train duration of 5 minutes followed by 20 minutes of no stimulation. The system will continue automatically cycling between stimulation and no stimulation with the user-specified duty cycle until the “A” key is pressed again to toggle OFF the periodic stimulation.

The screenshot displays a software interface with a workflow diagram on the left and a properties panel on the right. The workflow diagram shows a sequence of components: 'Second Order Periodic Stim Control' (grey circle), 'FP3002' (green circle), 'Photometry Data' (yellow circle), 'Photometry Writer' (purple circle), 'DigitalIOs' (blue circle), and 'CsvWriter' (purple circle). The 'FP3002' component is highlighted with a red box. The properties panel on the right is titled 'Second Order Periodic Stim Control (GroupWorkflow)' and contains the following information:

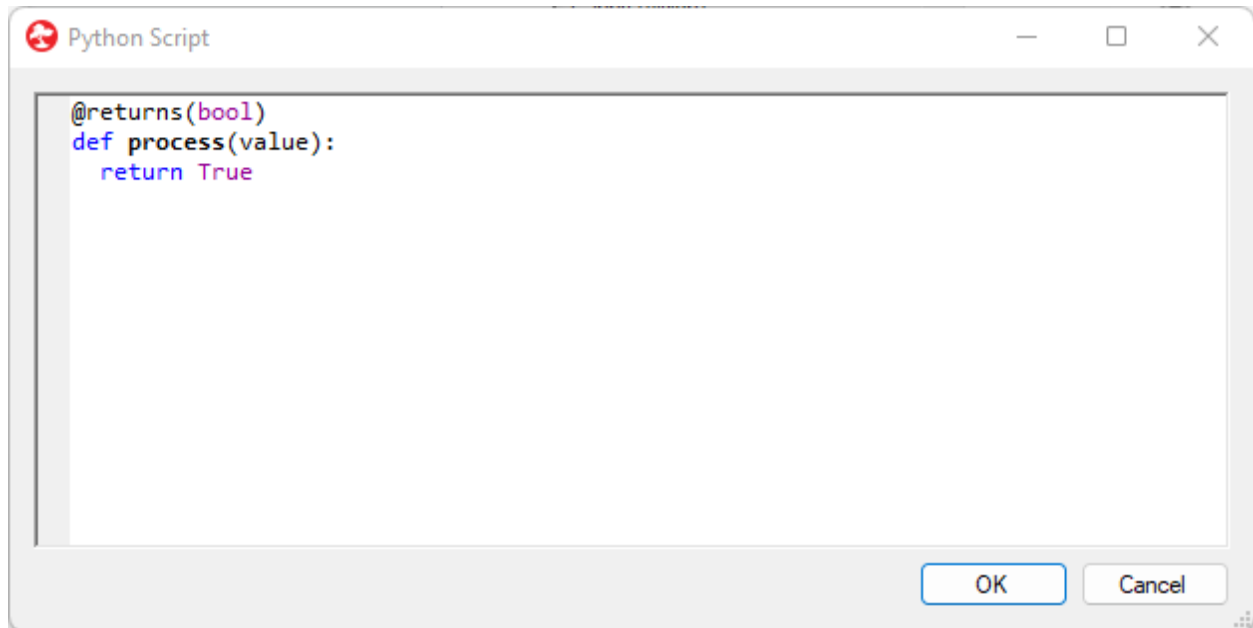
Second Order Periodic Stim Control (GroupWorkflow)	
Description	Generates a sequence of "Start" and "Stop" stimulation commands with a user specified duty cycle. Also included is a manual delay to stimulation.
Name	Second Order Periodic Stim Control
<b>Design</b>	
Description	Generates a sequence of "Start" and "Stop" stimulation commands with a user specified duty cycle. Also included is a manual delay to stimulation.
Name	Second Order Periodic Stim Control
<b>Misc</b>	
OFF Time (ms)	1200000
ON Time (ms)	300000
Toggle Periodic Stimulation	A

## Automated Second Order Periodic Control

In order to configure the software trigger logic for automated second order periodic control, we will include two additional “Int64” nodes to the input of the “Combine Latest” node. These will contain the duration of the periodic stimulation and no periodic stimulation states. Be sure to externalize the “Value” property for each of these new “Int64” nodes and give them unique display names.



Now we have all the information we need entering the “Python Transform” node through the “Combine Latest” node. Here we have the current time of day, in total milliseconds, the ON/OFF durations of stimulation during the periodic stimulation state, and the ON/OFF durations of the periodic stimulation and no periodic stimulation states. Begin with a fresh python script by deleting the current “Python Transform” node and reinserting a new one.



Start the script by reading in each element of the incoming Tuple. Be sure to double check the order in which the “Int64” nodes are connected.

```
@returns(bool)
def process(value):

    # Read in values form the input Tuple
    currentTime = value.Item1
    stimONTime = value.Item2
    stimOFFTime = value.Item3
    periodicStimONTime = value.Item4
    periodicStimOFFTime = value.Item5

    return True
```

For our script we will need to keep track of two start times: the start time of a pulse train and the start time of periodic stimulation. Declare two global variables for storing these values. Initialize these to a negative value so that the script will be able to determine if it is the first time it is being run during a workflow.

```
# Global variables for storing the time that the system entered the periodic
# stimulation state and when the current laser pulse train started.
stimStartTime = -1
periodicStimStartTime = -1

@returns(bool)
def process(value):
    global stimStartTime, periodicStimStartTime
```

Before implementing our logic for periodic stimulation, let's handle the edge case that occurs the first time this script is run. In this case, we want to update our global start time variables to the current time. To do this, after the script reads in the input Tuple, if either of the global start time variables are negative then update them to the current time and return False.

```
# If this is the first time the script is run during a workflow, then update the
# global start time variables and output False
if stimStartTime < 0 or periodicStimStartTime < 0:
    stimStartTime = currentTime
    periodicStimStartTime = currentTime
    return False
```

This way, when the workflow is started, this python script will immediately update the global start time variables and output False, indicating that stimulation should not begin. That way, on the start of the workflow, the FP3002 system will be in a no stimulation state. Now we can begin implementing our logic for periodic stimulation. After our edge case test, calculate the amount of time the system has spent in the current stimulation ON/OFF cycle and in the current periodic stimulation ON/OFF cycle, being sure to account for a date change.

```
# Calculate the duration of time spent in the current cycle of stimulation ON/OFF
dt = currentTime - stimStartTime
# Calculate the duration of time spent in the current cycle of periodic stimulation ON/OFF
DT = currentTime - periodicStimStartTime

# Account for date change
if startTime > currentTime:
    dt = dt + (1000.0 * 3600.0 * 24.0)
    DT = DT + (1000.0 * 3600.0 * 24.0)
```

Next, add an if, elif, else statement to determine if the system should be in the periodic stimulation state, the no periodic stimulation state, or at the end of the current periodic stimulation ON/OFF cycle. Here we will configure the logic such that the system begins in a periodic stimulation ON state, but it will start its stimulation ON/OFF cycles in the OFF state. We find this logic works best in most experiments to allow for periodic stimulation to begin at the start of the experiment while not stimulating immediately after the workflow has been started.

```
# If in the periodic stimulation state
if DT < periodicStimONTime:

# Else if in the no periodic stimulation state
elif DT < periodicStimONTime + periodicStimOFFTime:

# Otherwise, we have reach the end of the current periodic stimulation ON/OFF cycle
else:
```

Now we need to populate this if, elif, else statement. In the elif portion of the statement, we have determined that the system should be in the no periodic stimulation state. In this case, the script should only output a False value. Meanwhile, in the else portion of the statement, the script needs to reset to a new periodic stimulation ON/OFF cycle. To do this, we need to update the global start time variables to the current time and output False.

```
# If in the periodic stimulation state
if DT < periodicStimONTime:

# Else if in the no periodic stimulation state
elif DT < periodicStimONTime + periodicStimOFFTime:
    return False
# Otherwise, we have reach the end of the current periodic stimulation ON/OFF cycle
else:
    stimStartTime = currentTime
    periodicStimStartTime = currentTime
    return False
```

In the if portion of the if, elif, else statement, we need to include logic for alternating between stimulation ON/OFF states. This logic will be quite similar to our logic for alternating between periodic stimulation ON/OFF states. So we can add another if, elif, else statement here to determine if the system should be in the stimulation OFF state, stimulation ON state, or at the end of the current stimulation ON/OFF cycle.

```
# If in the periodic stimulation state
if DT < periodicStimONTime:
    # If in the stimulation OFF state
    if dt < stimOFFTime:

        # If in the stimulation ON state
        elif dt < stimOFFTime + stimONTime:

            # Otherwise, we have reached the end of the current stimulation ON/OFF cycle.
            else:
```

Now we need to populate this new if, elif, else statement. In the if portion of the statement we have determined that the system should not be stimulating so the script should simply output False. In the elif portion, stimulation should start, so output True. Finally, in the else portion, we need to reset the current stimulation ON/OFF cycle by updating only the “stimStartTime” global variable to the current time and outputting False.

```
# If in the periodic stimulation state
if DT < periodicStimONTime:
    # If in the stimulation OFF state
    if dt < stimOFFTime:
        return False
    # If in the stimulation ON state
    elif dt < stimOFFTime + stimONTime:
        return True
    # Otherwise, we have reached the end of the current stimulation ON/OFF cycle.
    else:
        stimStartTime = currentTime
        return False
```

Our python script is now complete. When the workflow starts, the script will update the global start time variables and output False to ensure no stimulation is occurring. The system will remain in the periodic stimulation state for the specified duration, using the logic contained within the *if DT < periodicStimONTime:* statement to cycle between stimulation ON/OFF states. Then after a specified amount of time, the system will enter a state of no periodic stimulation when stimulation will remain OFF until the next cycle of periodic stimulation. The full script is shown below:

```
Python Script
# Global variables for storing the time that the system entered the periodic
# stimulation state and when the current laser pulse train started.
stimStartTime = -1
periodicStimStartTime = -1

@returns(bool)
def process(value):
    global stimStartTime, periodicStimStartTime

    # Read in values form the input Tuple
    currentTime = value.Item1
    stimONTime = value.Item2
    stimOFFTime = value.Item3
    periodicStimONTime = value.Item4
    periodicStimOFFTime = value.Item5

    # If this is the first time the script is run during a workflow, then update the
    # global start time variables and output False
    if stimStartTime < 0 or periodicStimStartTime < 0:
        stimStartTime = currentTime
        periodicStimStartTime = currentTime
        return False

    # Calculate the duration of time spent in the current cycle of stimulation ON/OFF
    dt = currentTime - stimStartTime
    # Calculate the duration of time spent in the current cycle of periodic stimulation ON/OFF
    DT = currentTime - periodicStimStartTime

    # Account for date change
    if startTime > currentTime:
        dt = dt + (1000.0 * 3600.0 * 24.0)
        DT = DT + (1000.0 * 3600.0 * 24.0)

    # If in the periodic stimulation state
    if DT < periodicStimONTime:
        # If in the stimulation OFF state
        if dt < stimOFFTime:
            return False
        # If in the stimulation ON state
        elif dt < stimOFFTime + stimONTime:
            return True
        # Otherwise, we have reached the end of the current stimulation ON/OFF cycle.
        else:
            stimStartTime = currentTime
            return False
    # Else if in the no periodic stimulation state
    elif DT < periodicStimONTime + periodicStimOFFTime:
        return False
    # Otherwise, we have reach the end of the current periodic stimulation ON/OFF cycle
    else:
        stimStartTime = currentTime
        periodicStimStartTime = currentTime
        return False

OK Cancel
```



With the addition of the “Int64” nodes and the new python script, our software trigger is complete. In the example below, stimulation will begin 20 minutes after the start of the workflow. Then for three hours, the system will alternate between stimulating for 5 minutes and not stimulating for 20 minutes. After the three hours of periodic stimulation, the system will stop any stimulation for one hour. Finally, after an hour of no stimulation, the system will begin three hours of periodic stimulation again, beginning with the 20 minutes of no stimulation.

**Workflow**

```

graph LR
    A[Second Order Periodic Stim Control] --> B[FP3002]
    B --> C[Photometry Data]
    B --> D[DigitalIOs]
    C --> E[Photometry Writer]
    D --> F[CsvWriter]
    
```

**Properties**

**Second Order Periodic Stim Control (GroupWorkflow)**  
 Generates a second order periodic sequence of "Start" and "Stop" stimulation commands with user specified duty cycles.

**Design**

Description	Generates a se
Name	Second Order I

**Misc**

Periodic Stimulation OFF Time (ms)	21600000
Periodic Stimulation ON Time (ms)	10800000
Stimulation OFF Time (ms)	1200000
Stimulation ON Time (ms)	300000

# Chapter 6: Machine Vision

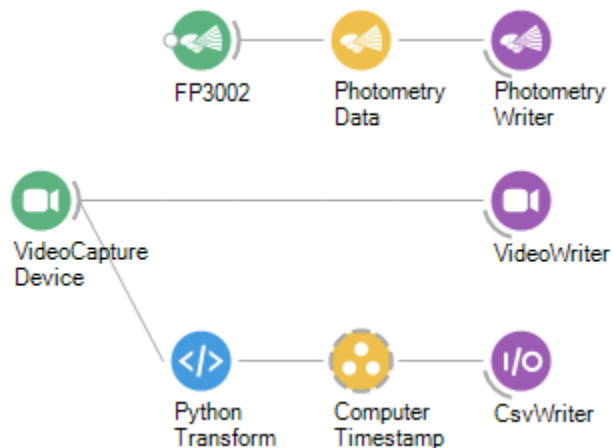
---

This chapter builds on the concept of utilizing machine vision techniques within fiber photometry experiments. We begin with an introduction on how to synchronize a behavioral camera with the standard photometry data stream. Then we discuss the implementation of image processing techniques for animal tracking in different environments. The machine vision techniques presented here consist of image segmentation to isolate an animal in an enclosure followed by binary region analysis to find the centroid of the animal. More advanced techniques can be implemented, but are outside the scope of this book. We will focus on utilizing built-in Bonsai nodes for our machine vision algorithms and showcasing how they can interface with the standard photometry workflow.

Once we establish the ability to conduct animal tracking in multi-chamber enclosures, we transition to the concept of “Closed-Loop” experiments. We use this term to describe fiber photometry experiments that are influenced by the actions of the animal(s) being observed. In this chapter, we describe how to use an animal tracking algorithm in order to control the data acquisition and/or the stimulation of the FP3002 system. We conclude this chapter with a section on common real-time analysis algorithms for processing the data generated by image processing algorithms.

## Synchronized Behavioral Camera

For experiments that require post-hoc behavioral analysis, it is possible to construct Bonsai workflows with external behavioral cameras synchronized with the photometry data. This particular workflow will save three files: the standard photometry dataset (.csv), the behavioral frame numbers timestamped using the computer's timestamp (.csv), and the video file containing the camera frames (.avi).



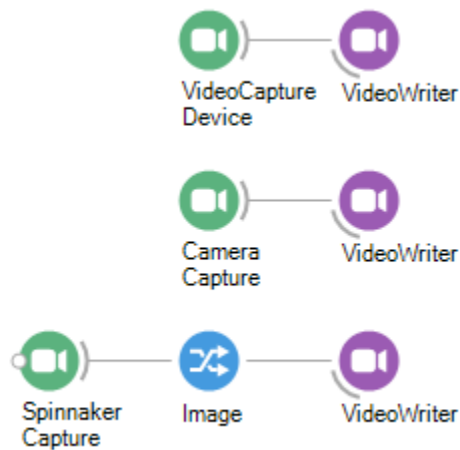
This workflow consists of two parallel data streams. One deals with processing and recording the photometry data, while the other processes and records the information from the external behavioral camera. Begin construction of this workflow with the “Standard Photometry” workflow.



This will record the photometry data, timestamped both by the system's internal clock and the computer's clock. The data set produced by the “Standard Photometry” workflow can readily alignable with the behavioral camera's data set so long as we timestamp the behavioral camera's frames with the computer's clock.

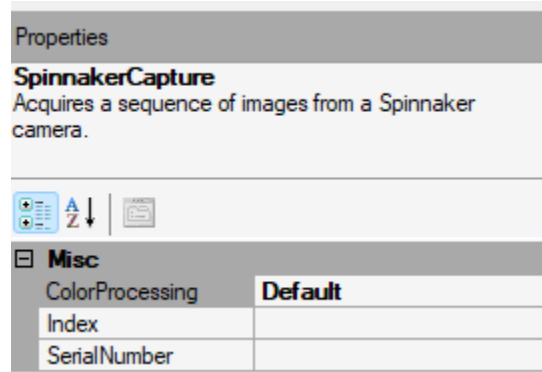
Now that the photometry data stream is constructed, we can begin constructing the behavioral camera's data stream. There are three commonly used source nodes for connecting to external cameras and producing frames from them. For Spinnaker cameras use the "Spinnaker Capture" node. For DirectShow based capture devices use the "Video Capture Device". Finally for most webcams, the "Camera Capture" node is usable. All of the workflows contained within the "Machine Vision" section will work the exact same way whether using the "Video Capture Device" or the "Camera Capture" nodes. However, the "Spinnaker Capture" node works slightly differently. The output of the "Spinnaker Capture" node is of type "SpinnakerDataFrame" while the other two nodes output elements of type "IpImage". However, the "SpinnakerDataFrame" consists of an "IpImage" and "ChunkData" so the machine vision techniques described in this and the following sections can still be used with the "Spinnaker Capture" node if the "IpImage" is selected from the "SpinnakerDataFrame".

Create an external behavioral camera data stream by connecting the desired source node to a "Video Writer" node.

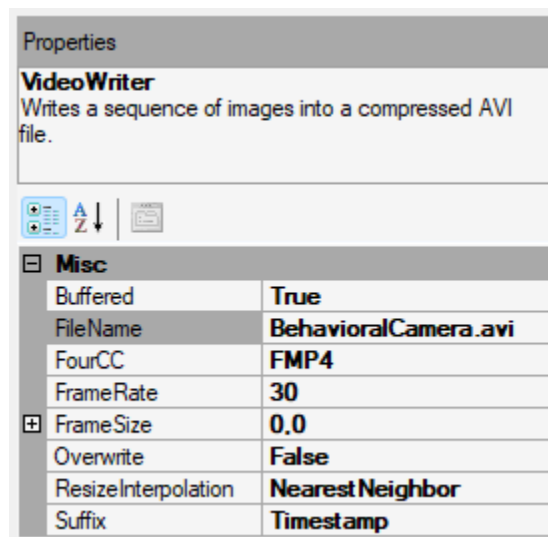


Some configuration is available for the capture nodes and the "Video Writer" node. All three capture nodes have the option to specify the camera's index. For the "Camera Capture" and "Video Capture Device" nodes, the internal camera on the "FP3002" system will not be recognized, so if only one behavioral camera is connected to the computer, it will appear on index 0. However, the "Spinnaker Capture" node will recognize the internal camera on the FP3002 system, so some care needs to be taken so that the "Spinnaker Capture" and "FP3002" nodes do not try to both access the same camera. The "FP3002" node registers the internal camera when applying a

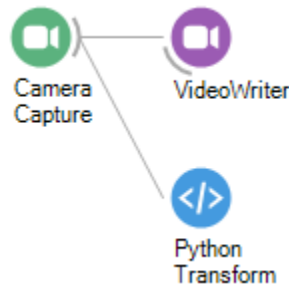
firmware update so we need to only verify that the “Spinnaker Camera” is not trying to access the internal camera. You can do this by specifying the “Index” or “SerialNumber” properties of the “Spinnaker Capture” node.



The “Video Writer” node has a variety of properties that are configurable. Similar to the “Csv Writer”, be sure to specify the “File Name”, “Overwrite”, and “Suffix” properties. Be sure to include the file extension in the “File Name” and that it matches the “FourCC”. By default “FMP4” will be used as the “FourCC” in order to save an .avi file. Next, set the “Frame Rate” property to the frame rate of the behavioral camera. This will allow the playback of the video to be at the same rate that the camera frames were acquired.



With the basic behavioral camera data stream configured, we need to synchronize this data stream with the photometry data stream by timestamping the frame count of the behavioral camera frame using the computer's clock. Most behavioral cameras do not have an internal frame counter so our first task in synchronization is to create a frame counter for the behavioral camera. Do this by connecting a "Python Transform" node to the capture node in parallel with the "Video Writer" node.



Edit the python script by double clicking the "Python Transform" node. Here we will insert a basic python counter script that will keep track of the number of behavioral camera frames that have passed through the "Python Transform" node.

```
Python Script

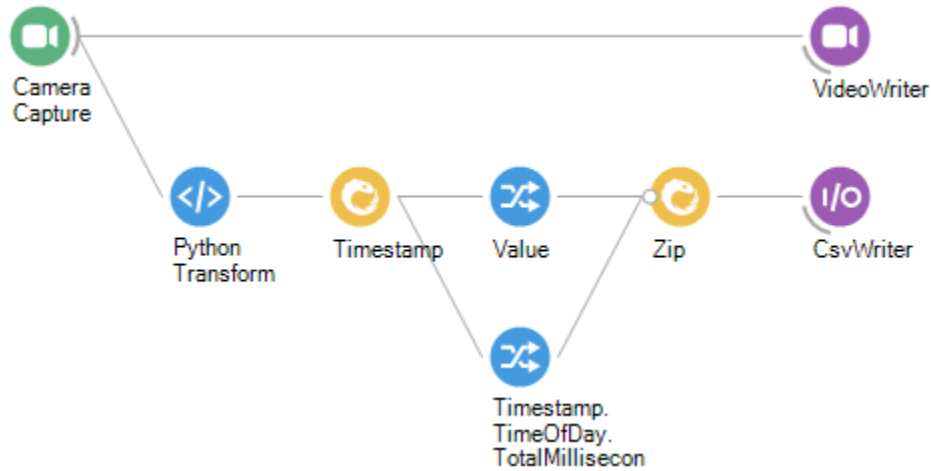
# Global variable for keeping track of the number
# of behavioral camera frames that have passed through
# this script.
count = 0

# Returns a value of type "Integer"
@returns(int)
def process(value):
    global count

    # Every time the behavioral camera passes
    # a value through this script, increment the
    # count variable and return the new count.
    count = count + 1
    return count

OK Cancel
```

Now we can timestamp the frame number of the behavioral camera using the computer's timestamp and save it to a .csv file. We will do this using the same method we used for the photometry data stream.



Again, we can group together all of the computer timestamp nodes into a single grouped workflow that can be named and described from the properties panel.

**Workflow**

Camera Capture

Python Transform

Computer Timestamp

CsvWriter

VideoWriter

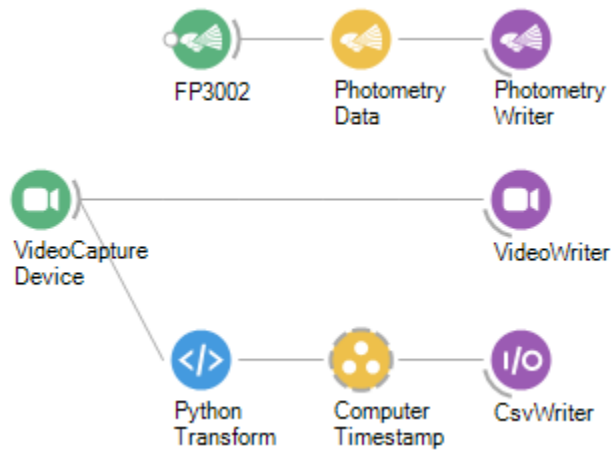
**Properties**

**Computer Timestamp (GroupWorkflow)**  
Timestamps each external camera frame using the computer's clock.

**Design**

Description	Timestamps each exte
Name	Computer Timestamp

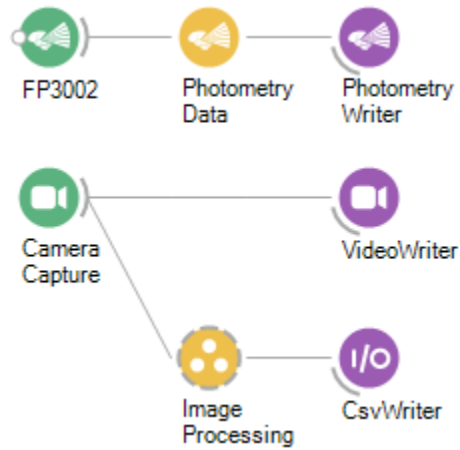
Now, the behavioral camera and photometry data streams can be implemented in the same Bonsai workflow and can be aligned using the computer's clock.



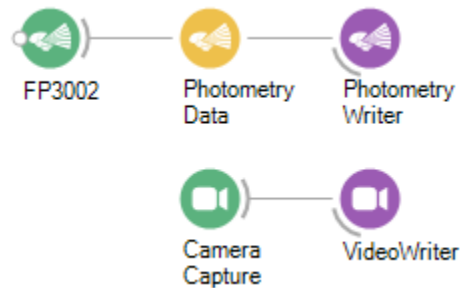


# One Chamber Animal Tracking

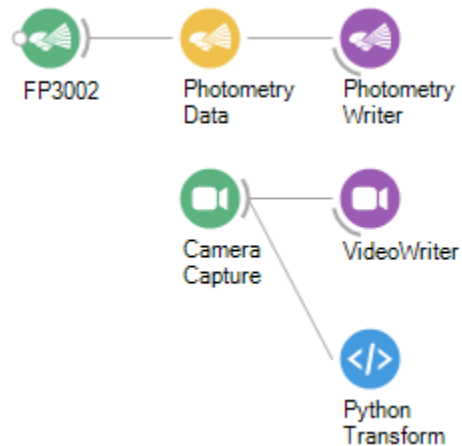
Previously we discussed how to synchronize a behavioral camera in a parallel data stream within Bonsai. This technique can be expanded to provide real-time processing of the behavioral camera's frame. In particular, this section will discuss tracking the position of an animal in a one chamber enclosure.



Begin construction of this workflow with the “Machine Vision: Synchronized Behavioral Camera” workflow discussed previously. Remove the “Python Transform” and timestamping logic from the behavioral camera’s data stream, this section of the workflow is where we will implement our image processing logic.

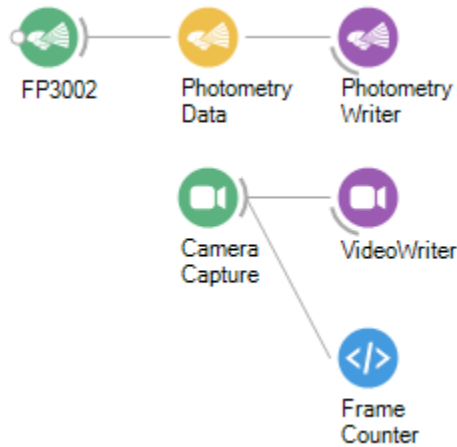


Our image processing logic needs to save four columns of data into a .csv file. We will configure our logic to save the frame number in column one, the computer’s timestamp in column two, and the XY position in columns three and four. In parallel to the “Video Writer” node, connect a “Python Transform” node to the image source node. This will contain our frame counter script from the “Machine Vision: Synchronized Behavioral Camera” workflow.

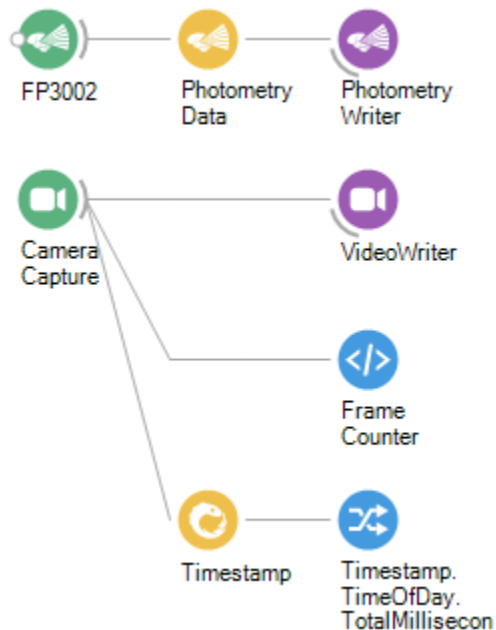


```
Python Script
count = 0
@returns(int)
def process(value):
    global count
    count = count + 1
    return count
```

Be sure to name and describe the “Python Transform” node from its properties panel. This becomes important when constructing more complex workflows.



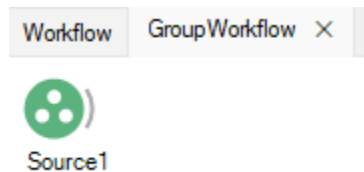
In parallel to the frame counter node, connect a “Timestamp” node to the behavioral camera’s source node, outputting the “Time Of Day, Total Milliseconds” value. Instead of timestamping the behavioral camera frames in series, like in the previous workflow, we are adding a timestamp in parallel. This will make combining our image processing data and saving to a .csv file easier.



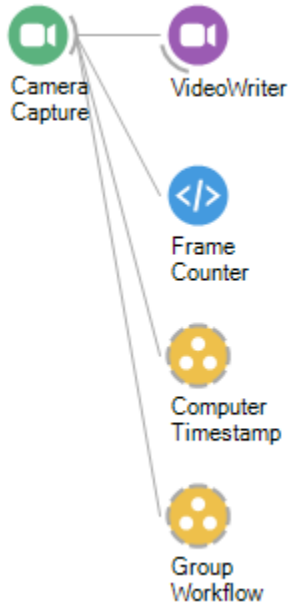
It is also useful to group the timestamp nodes into a single grouped workflow that is named and described. To do this, select both of the timestamp nodes that we just added, right click and select “Group → GroupWorkflow”.



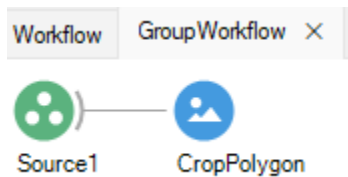
Now we need to create the logic for finding the centroid of an animal in a single chamber enclosure. We will put all of this logic inside a grouped workflow in order to keep everything organized. Begin by adding a “Group Workflow” from the toolbox into the workflow. Open the grouped workflow by double clicking it. The first node that we need to add to this grouped workflow is a “Workflow Input” node so that we can accept the behavioral camera frame into this grouped workflow.



With this source node inside of the grouped workflow, we can now navigate back to the top-level of the workflow and connect the capture node to the grouped workflow.

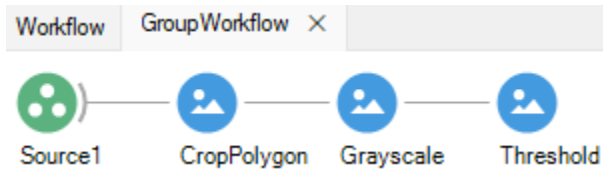


Returning to our grouped workflow, the first set of logic we want to implement in our “Find Centroid” algorithm will work to crop the image coming from the behavioral camera to the bounds of a single chamber enclosure. To do this connect a “Crop Polygon” node to the source node of the grouped workflow. This node will require reconfiguration every time the behavioral camera is moved, and should be checked before every experiment. Since there are several nodes in this workflow that require configuration, we will discuss methods for configuration after we have constructed all of the needed logic for this workflow.

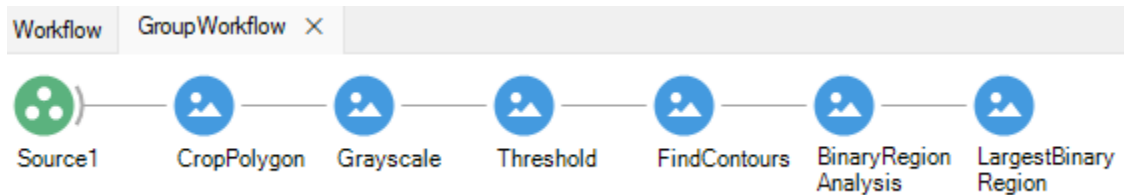


Next we need to apply an image segmentation technique to isolate the animal in the image. One way to do this is to convert the image to grayscale then threshold the grayscale image. Insert a “Grayscale” node followed by a “Threshold” node after the “Crop Polygon” node. The “Grayscale” node will convert a BGR color image to

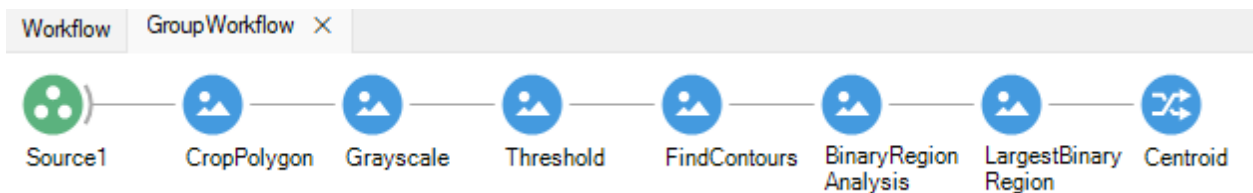
grayscale. The “Threshold” node will accept that grayscale image, isolate the animal in the image, and output a binary image.



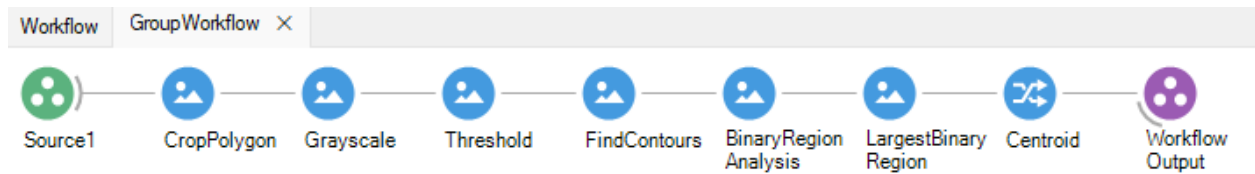
Now that we have applied image segmentation to isolate the animal, we can begin on our binary region analysis. This part of the algorithm begins with finding the contours of the animal, then we extract binary region properties from those contours and isolate the largest binary region (i.e. the animal in the enclosure). We will do this by implementing the following series of nodes after the threshold: “Find Contours”, “Binary Region Analysis”, and “Largest Binary Region”.



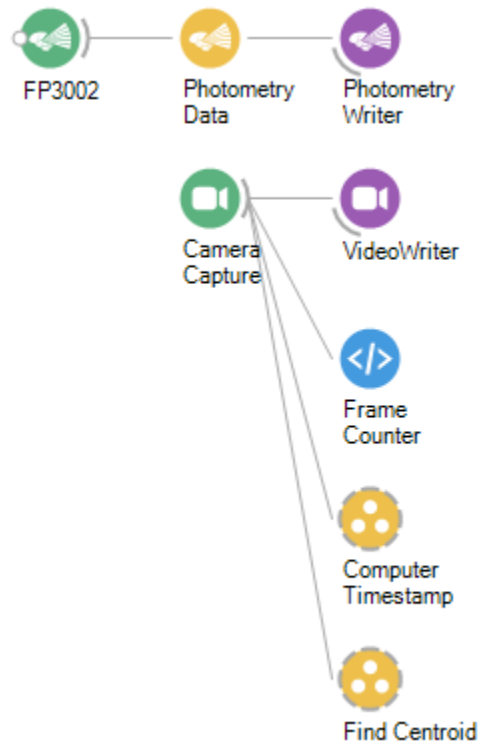
The bulk of this algorithm is complete, all we have to do is output the desired information. We can output the centroid of the largest binary region by right clicking the “Largest Binary Region” node and selecting “Output → Centroid”. Once all of our image processing nodes have been configured properly, the largest binary region will be the animal and this centroid property will be the XY position of the animal, where the X value is in pixels from the left edge of the cropped image and the Y value is in pixels from the top edge of the cropped image.



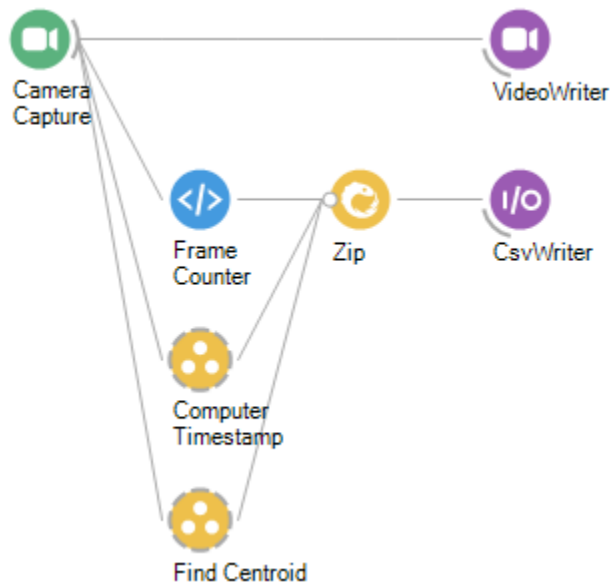
Our image processing algorithm is now complete. In order to output the centroid value from the grouped workflow, insert a “Workflow Output” node after the “Centroid” node.



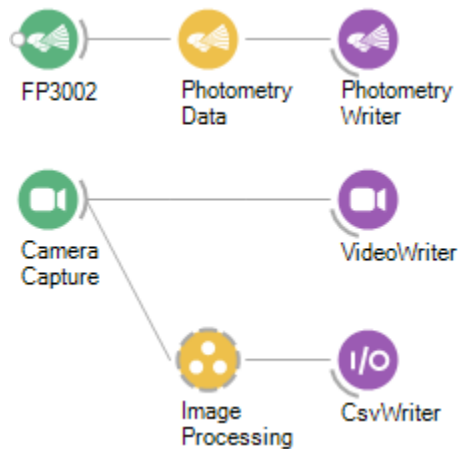
We can also add some organization by naming and describing the grouped workflow. In the example we name the grouped workflow “Find Centroid” and briefly describe the image processing algorithm we implemented.



We have three data streams for our image processing algorithm: the frame counter, computer timestamp, and the XY position of the animal. In order to save all of these data streams into a single .csv file, we need to combine them. In this case, the “Zip” node is the preferred method of combining the parallel data streams since they all produce values at the same rate. We can then connect the “Zip” node to a “Csv Writer” to actually write our data to a file.



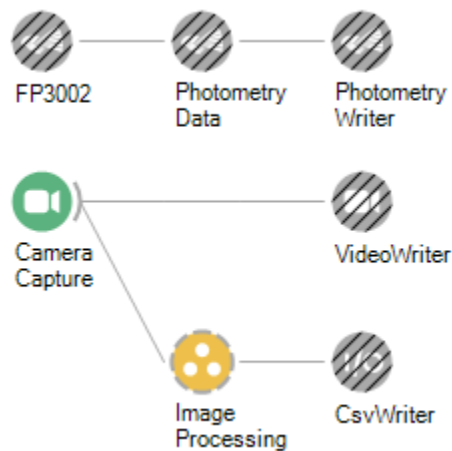
It is useful to group together all of our image processing nodes into a single grouped workflow that we can name and describe.



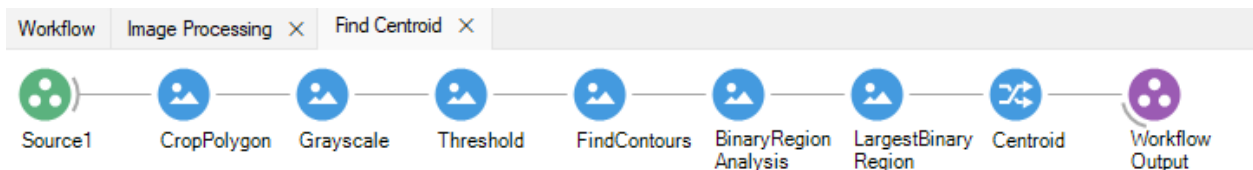


## Configuration:

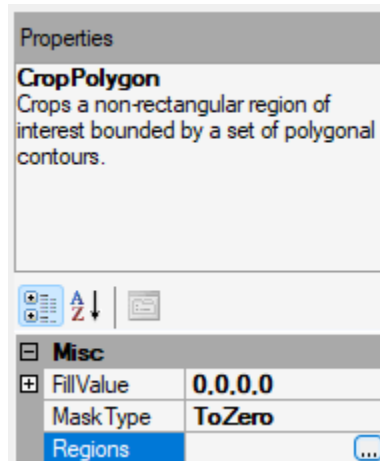
The nodes contained within our “Find Centroid” algorithm require configuration and should be reconfigured every experiment. We will configure these nodes while the workflow is running, so it is useful to temporarily disable all of the nodes contained within the photometry data stream as well as all of the writers for the behavioral camera data stream. The easiest way to do this is to select all of the nodes to be disabled and to press the “CTRL + D” hotkey to disable them. This way we can run the workflow while we configure the image processing nodes without saving any data or accessing the FP3002 system.



Open the “Find Centroid” grouped workflow contained within the “Image Processing” grouped workflow so we have access to all of the nodes that we need to configure. Then start the workflow.



First, we will configure the “Crop Polygon” node such that the image is cropped to the single chamber. While the workflow is running, click the “Crop Polygon” node, click inside of the “Regions” property text box and select the “...” button.



This will open the video feed coming into the “Crop Polygon” node and allow you to draw a region of interest encapsulating the chamber. For rectangular chambers, left click and drag a rectangle along the wall of the chamber. Once the rectangle is drawn, you can move each point individually by right clicking inside of the rectangle, close to one of the points and dragging to a new position. The rectangle needs to be selected, appearing green, to be reshaped. If it is not selected, press “Tab” to select it. You can also delete selected rectangles by pressing “Del”. Be sure that only one rectangle is drawn and if there are extra, press tab to select the undesired rectangles and delete them. For a non-rectangular chamber, you can hold “Shift” while left clicking and dragging a region of interest. This will generate an elliptical region with many points that can be reshaped to match the shape of the enclosure.



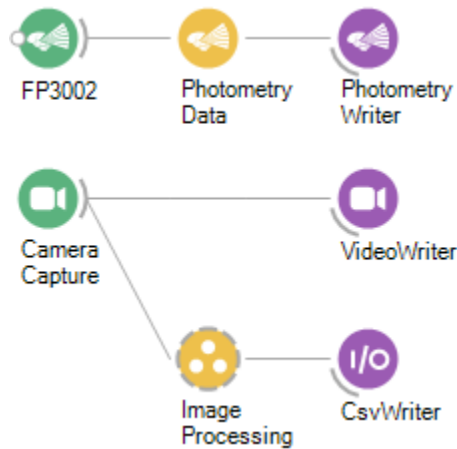
The next node that needs configuring is the “Threshold” node. First set the “Max Value” to the maximum possible pixel value. For most cases this will be 255, however when working with Mono16 images this value will be 65,535. Next, set the “Threshold Type” property to “Binary” or “BinaryInv”, depending if the animal is brighter or darker than its environment. This property will cause the image to be black and white where pixel values higher than the “Threshold Value” will be white in “Binary” mode and black in “BinaryInv” mode. Finally, we must configure the “Threshold Value” property such that the animal is the largest white region in the image. While the workflow is running, open up the visualizer for the “Threshold” node by double clicking it, and adjust the “Threshold Value” until the animal is white and the rest of the enclosure is black.



Dr Polter's lab - Institute for Neuroscience - The George Washington University

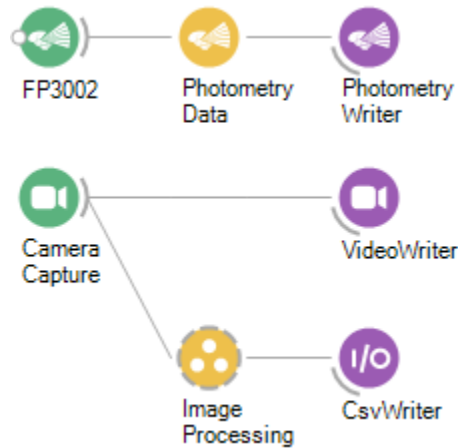
A useful feature of the visualizers for nodes in the “Bonsai.Vision” package is that when you right click the image in the visualizer, it will display the cursor position and the pixel values. You can use this feature to help configure the “Threshold” node by opening the “Grayscale” node’s visualizer and finding the approximate pixel values of the animal in the enclosure.

With the image processing logic constructed, organized, and configured, this workflow is ready to use within a fiber photometry experiment. Be sure to enable all of the nodes by highlighting them and pressing “CTRL + Shift + D”. The photometry data stream will save two data sets: the standard photometry data set and the frame numbers with computer timestamps for alignment purposes. The behavior camera data stream will also save two data sets: the raw image data as an .avi file and a .csv file containing the frame number, computer timestamp, and XY position of the animal for every behavioral camera frame.

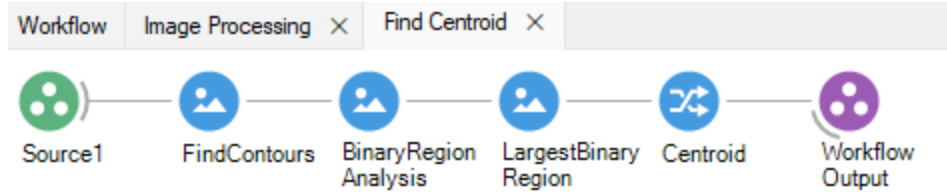


## Three Chamber Animal Tracking

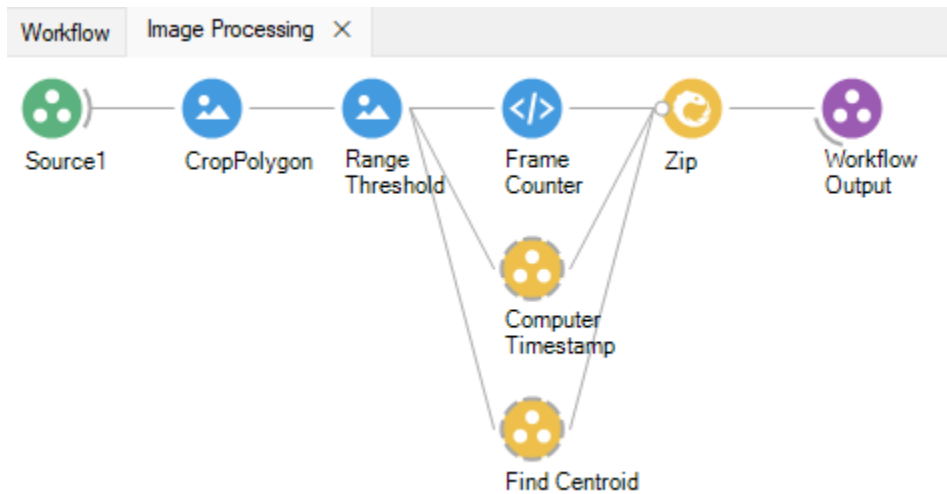
This section will expand the one chamber animal tracking workflow to allow for tracking an animal in a three chamber enclosure. Here we will save all of the same data as the previous workflow, except with an additional three columns in the behavior camera's .csv file to contain which chamber the animal is in.



In order to construct this workflow, we will begin with the one chamber animal tracking workflow and edit the image processing logic to account for three chambers. Inside of the “Image Processing” grouped workflow, we will keep the “Frame Counter” python script and the “Computer Timestamp” logic the exact same. However, we will change the “Find Centroid” grouped workflow such that the image segmentation part of the algorithm will be conducted before the grouped workflow. This means we will crop the image to the enclosure and threshold it before the image enters the “Find Centroid” grouped work. This is done so that this step in the image processing does not have to be repeated for each chamber analysis, but instead can be processed once and sent to the “Find Centroid” and chamber analyses. We can insert the image segmentation logic immediately after the “Source1” node such that it feeds the “Frame Counter”, “Computer Timestamp”, and “Find Centroid” nodes. Open the “Find Centroid” grouped workflow and remove the image segmentation logic. This includes the “Crop Polygon”, “Grayscale”, and “Threshold” nodes.

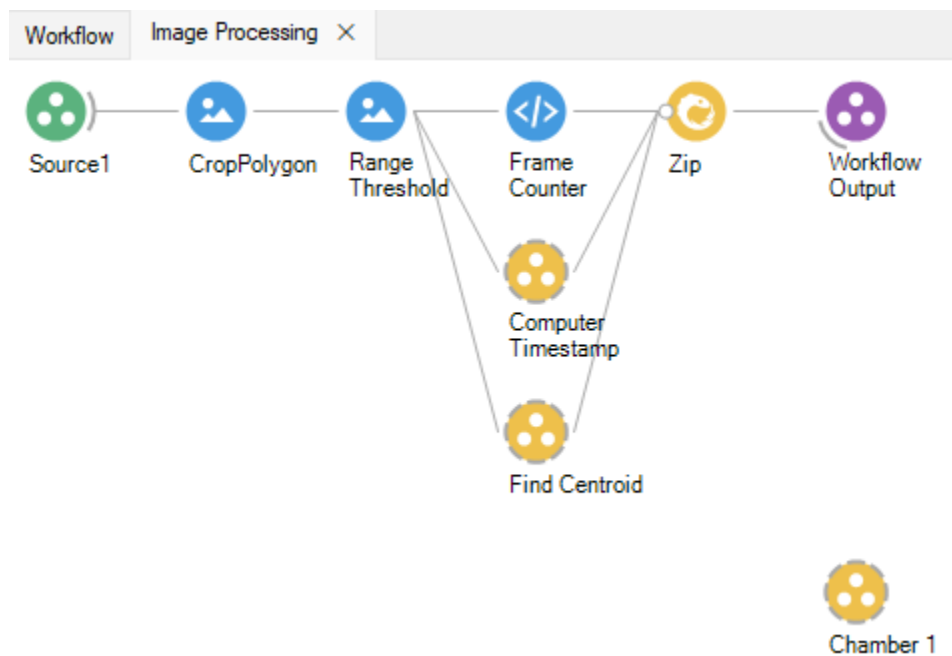


We will reimplement similar logic immediately after the “Source” node of the “Image Processing” grouped workflow. Instead of thresholding a grayscale image, let’s threshold a BGR image. This will allow you to have three dimensions to threshold the image instead of just one in the grayscale case. To do this, insert the “Crop Polygon” node followed by the “Range Threshold” node immediately after the “Source” node of the “Image Processing” grouped workflow.

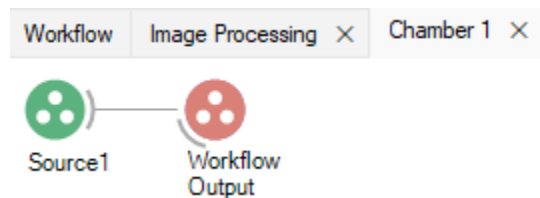


These image segmentation nodes will require configuration. We will cover some useful tricks for configuring these two nodes once all of our logic is set up. In particular, this section will cover cropping the image to non-rectangular enclosures and a way to visualize the BGR pixel values during thresholding.

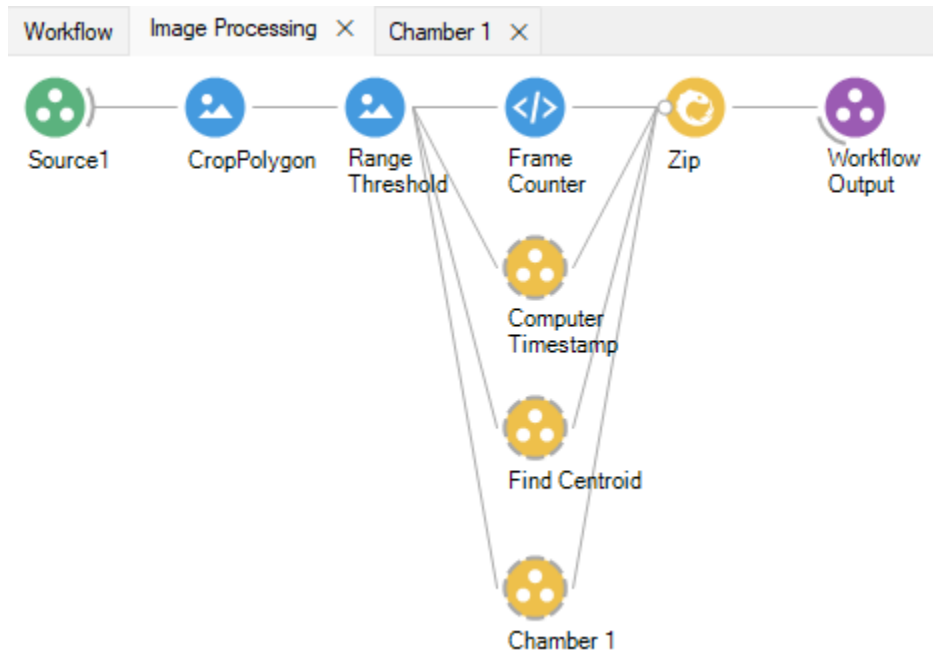
Next we will need to construct logic for determining whether or not the animal is located within a particular chamber of a multi-chamber enclosure. We will contain the logic for each chamber in separate grouped workflows. These chamber grouped workflows will accept the cropped, thresholded image and output to the “Zip” node. Insert a “Group Workflow” node and name it “Chamber 1”.



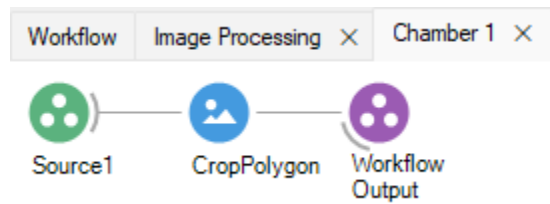
Inside of the “Chamber 1” grouped workflow, start by inserting a “Workflow Input” node followed by a “Workflow Output” node.



Now we can connect the “Range Threshold” node to the “Chamber 1” grouped workflow’s input and connect the output of the “Chamber 1” grouped workflow to the “Zip” node.

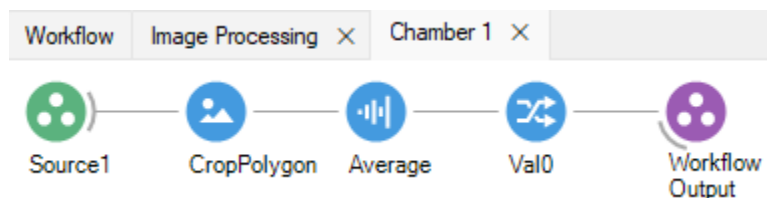


The image coming into the “Chamber 1” grouped workflow is already cropped to the full enclosure and thresholded. Within the “Chamber 1” node we need to crop the image again, this time to a particular chamber of the enclosure. Insert the “Crop Polygon” node immediately after the “Source” node of the “Chamber 1” grouped workflow.

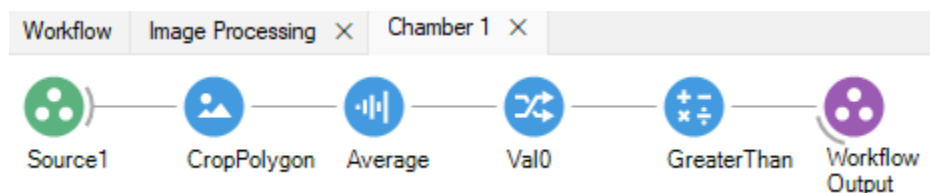




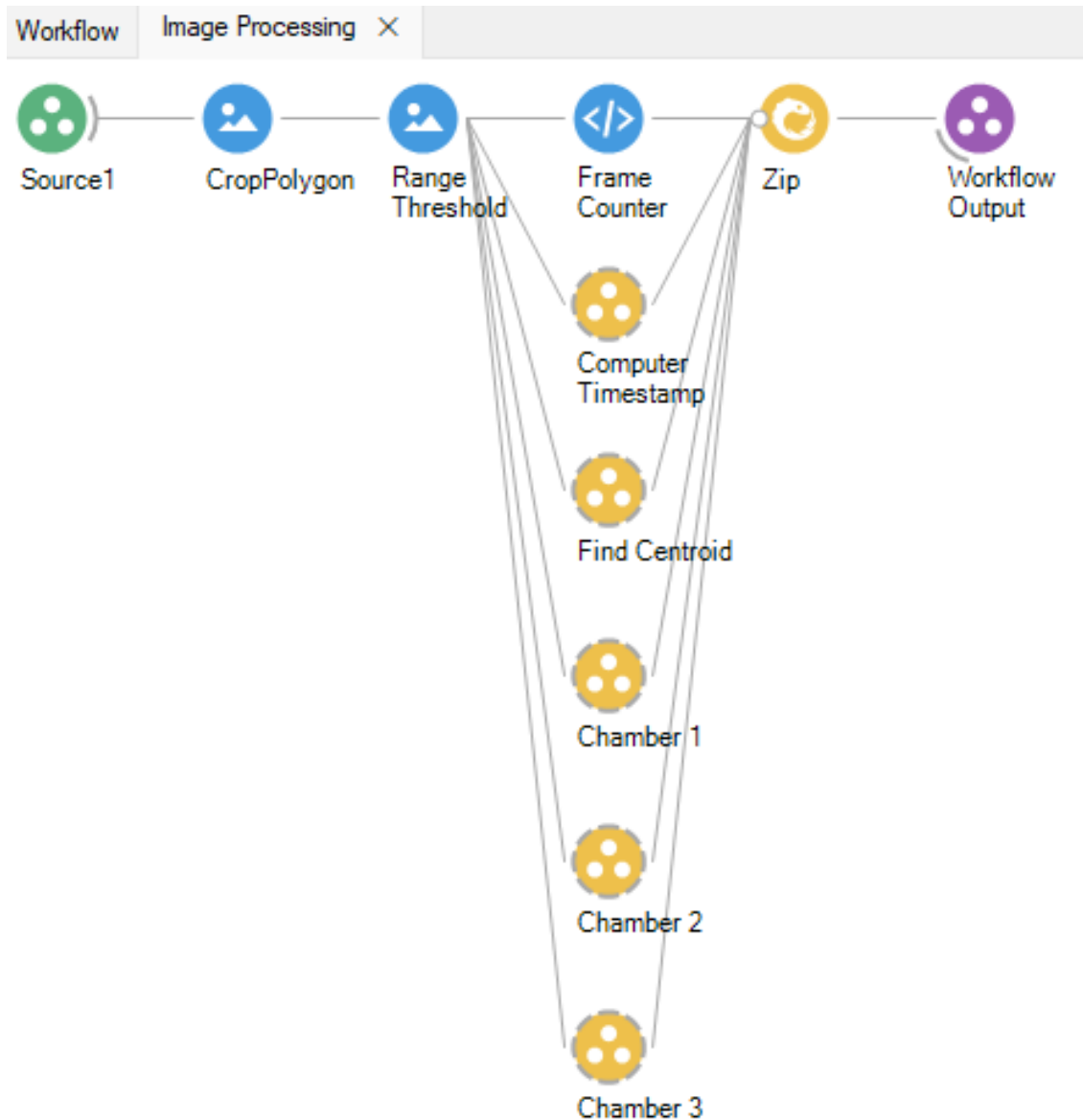
Since the image is already thresholded at this point of the workflow, after configuration, the image coming out of the “Crop Polygon” node should be mostly black. When the animal enters the chamber, the animal should appear white. This means that the average pixel value of the chamber when the animal is not in the chamber is approximately zero, and this average will greatly increase when the animal enters the chamber. This means we can use pixel averaging to distinguish whether or not the animal is located within this particular chamber. To implement pixel averaging logic, insert the “Average” node (from the Bonsai.Dsp package) and output the “Val0” element.



Currently, the above logic will output the average pixel value of the thresholded image, cropped to a particular chamber. We need this to output a “1” when the average pixel value is above a certain threshold and “0” when the average pixel value is below a certain threshold. The “Greater Than” node can be used for this purpose; insert it immediately after the “Val0” node.

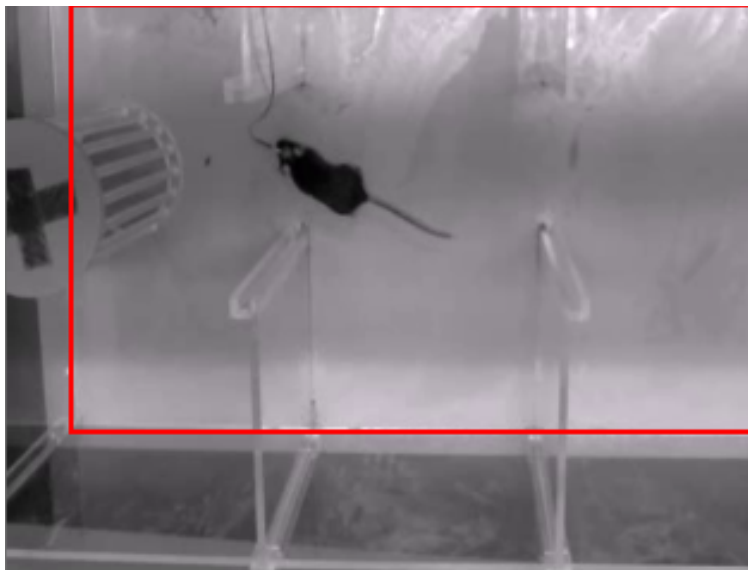


Our chamber analysis logic is now complete for “Chamber 1”. This exact same logic will be used for every other chamber in the enclosure. The only difference will be contained within the “Crop Polygon” node, configuring it to a different chamber. Duplicate the “Chamber 1” grouped workflow for each chamber of the enclosure and rename the grouped workflows to have unique names. Be sure to connect them in parallel to the “Chamber 1” grouped workflow.



## Configuration:

This workflow possesses four “Crop Polygon” nodes, one “Range Threshold” node, and three “Greater Than” nodes that must be configured before every experiment. Begin this process by selecting all of the nodes in the photometry data stream as well as all of the writer nodes in the behavioral camera’s data stream. Then click “CTRL + D” to disable them. This way we can run the workflow during configuration, without trying to access the FP3002 system or writing to storage. Then open the “Image Processing” grouped workflow and the “Chamber 1” through “Chamber 3” grouped workflows. This way we will have access to all of the nodes that need to be configured while the workflow is running. Start the workflow and begin configuration with the “Crop Polygon” and “Range Threshold” nodes in the “Image Processing” grouped workflow. Click the “Crop Polygon” node and select the “...” found in the “Regions” property. This will open a window for you to draw a region of interest. Left click and drag to draw a region of interest of the whole enclosure. You can move each point of the drawn rectangle by right clicking and dragging near an existing point inside of the shape. You can also add more points to the shape by double left clicking inside of the shape while it is selected. If the shape is not selected, press “Tab” to select it. With this window you can draw an outline of any 2D enclosure.



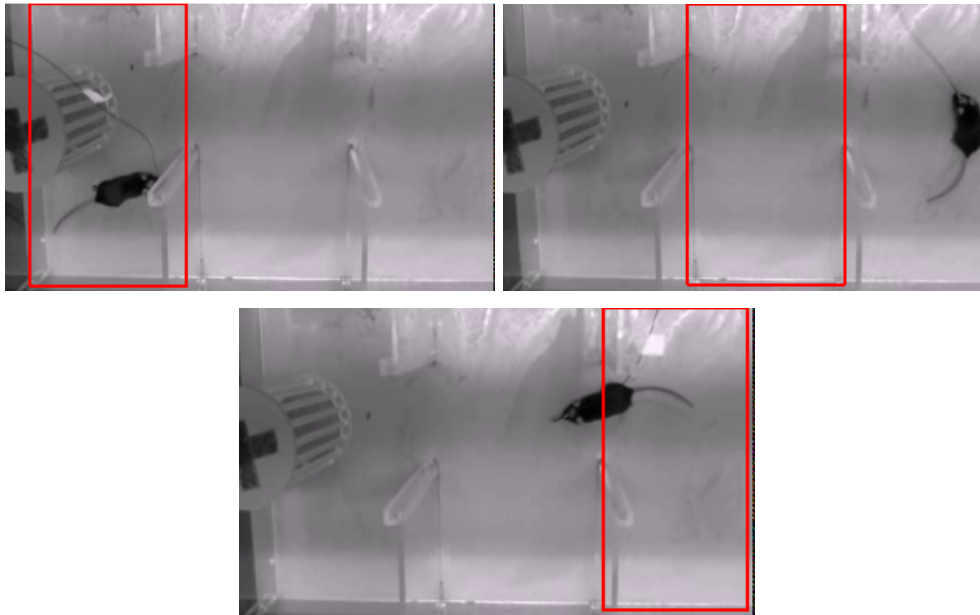
Dr Polter's lab - Institute for Neuroscience - The George Washington University

Once the “Crop Polygon” node is used to crop the image to the full enclosure, double click the “Range Threshold” node to open its visualizer. Then adjust the upper and lower limits of the threshold until the animal in the enclosure appears white and the enclosure appears black.



Dr Polter's lab - Institute for Neuroscience - The George Washington University

Then stop the workflow and disable the “Range Threshold” node and the “Find Centroid” grouped workflow. Restart the workflow and configure the “Crop Polygon” nodes found within the “Chamber 1” through “Chamber 3” grouped workflows. This time crop the image to a different chamber for each “Crop Polygon” node.

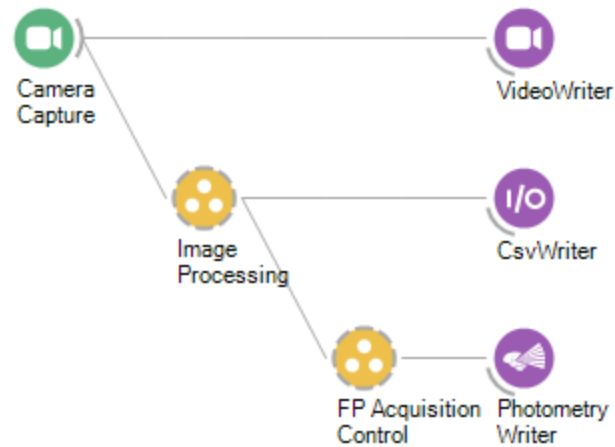


Dr Polter's lab - Institute for Neuroscience - The George Washington University

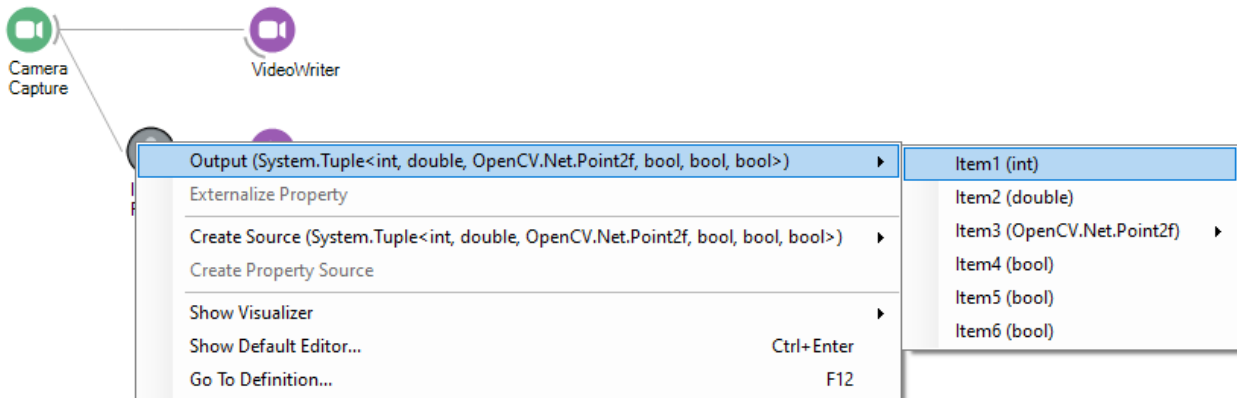
Stop the workflow again and enable the “Range Threshold” node and the “Find Centroid” grouped workflow. Restart the workflow and double click the “Greater Than” nodes to open their visualizers. Allow the animal to enter each chamber and adjust the “Value” property of each “Greater Than” node such that the node outputs False when the animal is not present and outputs True when the animal is present.

## Chamber Dependent Data Acquisition

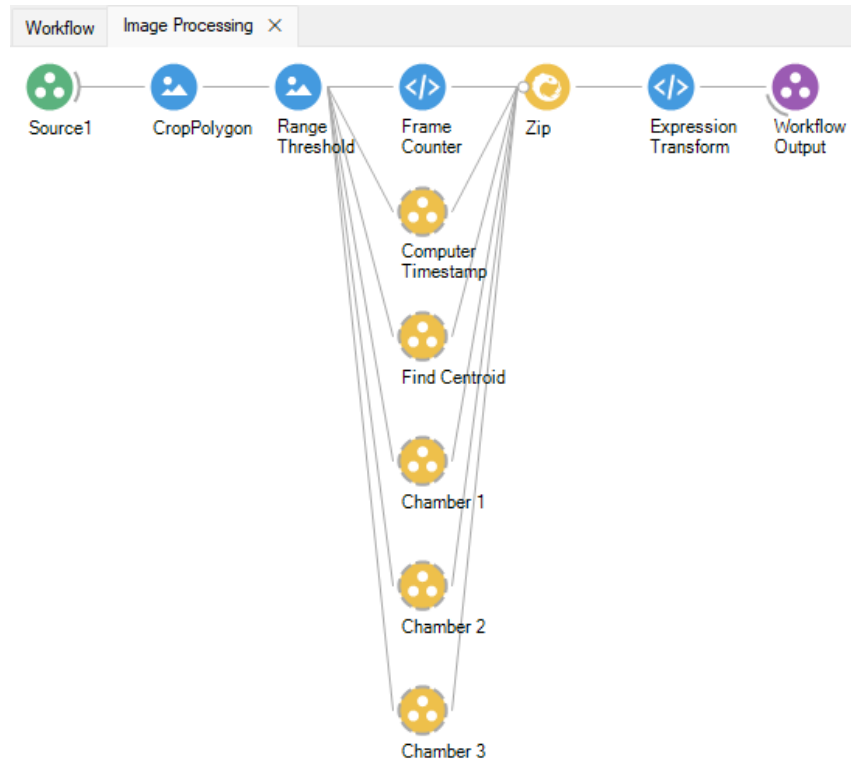
In the preceding sections, we have kept the photometry data and machine vision data streams parallel to each other so that they do not affect each other. However, in these last machine vision sections we will use the information that we acquire from our image processing algorithm to affect our photometry data stream. In particular, in this section we will cover controlling photometry data acquisition based on where the animal is located in a multi-chamber enclosure. This type of workflow logic is what we consider to be “Closed-Loop”. Over this section and the next, we will explore this concept of a fiber photometry experiment being influenced by the animal’s behavior. Here we will combine the concepts developed in the previous machine vision sections with the logic we developed in the data acquisition sections.



To begin constructing this workflow for chamber dependent data acquisition, begin with the “Three Chamber Animal Tracking” workflow. We will use the data coming from the chamber analysis grouped workflows to configure a software trigger that will control when data acquisition occurs. The first step is to add some organization to more easily access the data coming out of the “Image Processing” grouped workflow. If you right click that grouped workflow and look at its outputs, you will see that it outputs a Tuple containing “Item1” through “Item6”.



Without specifying the names of these items, it can be difficult to work with particular items of this Tuple and can lead to costly errors in the workflow. In order to specify the names of each of these items, we can use an “Expression Transform” inserted immediately after the “Zip” node inside of the “Image Processing” grouped workflow.



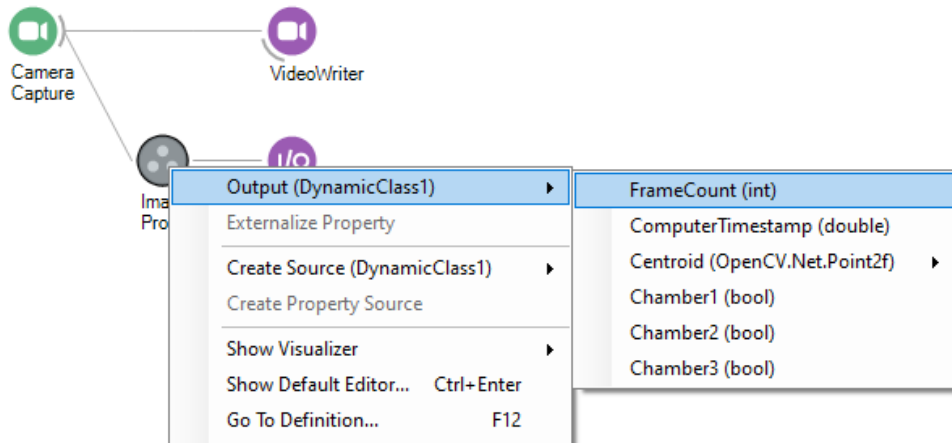
Open the editor for the “Expression Transform” by double clicking the node. Here we can create a dynamic class similar to a Tuple, where each element of the dynamic class has a user-specified name. In other words, you will be able to access each item using an informative name. Below is the syntax for renaming items one through six, creating a dynamic class similar to a tuple.

```
Expression Script
new(
  Item1 as FrameCount,
  Item2 as ComputerTimestamp,
  Item3 as Centroid,
  Item4 as Chamber1,
  Item5 as Chamber2,
  Item6 as Chamber3)
```

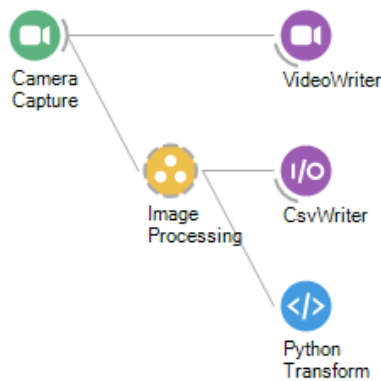
After implementing the above script within the “Expression Transform”, you can see the effect of this script by rechecking the output of the “Image Processing” node. Instead



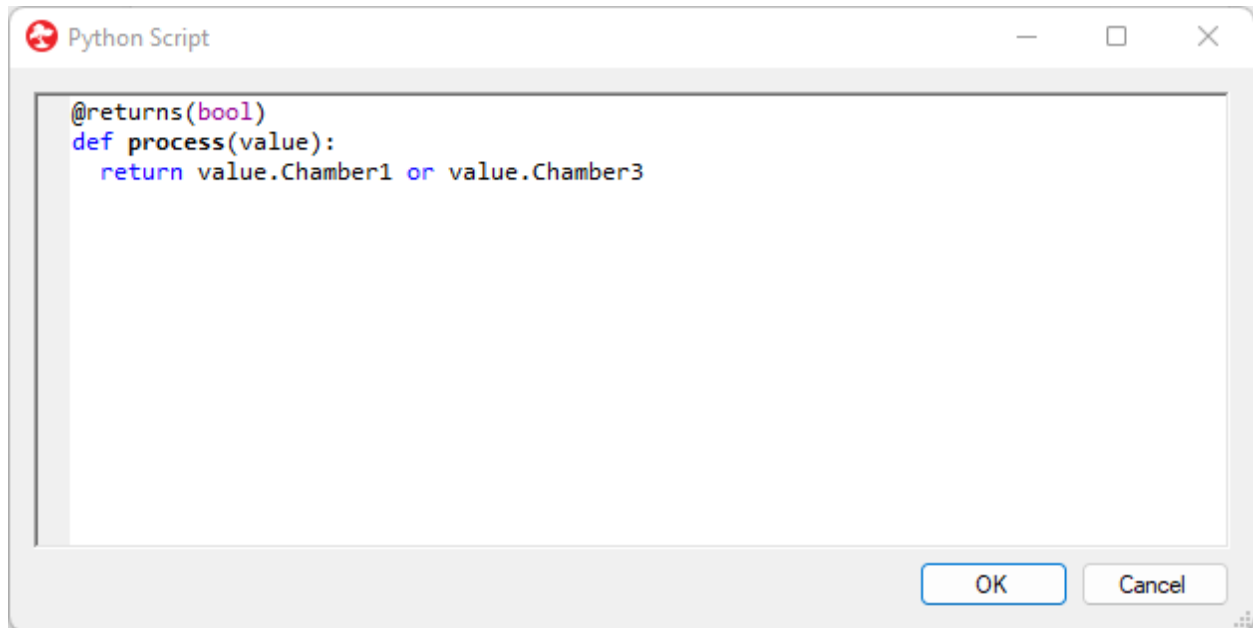
of each element appearing as “Item1” through “Item6”, we can actually see a descriptive name for each element being output.



Now, we want to configure the acquisition control logic based on which chamber the animal is located. In this example, we will be using a three chamber enclosure where the chambers go from left to right, labeled one through three. We will place the animal in the center chamber, label “Chamber 2”, and we only want to acquire photometry data while the animal is in the outer chambers, labeled “Chamber 1” and “Chamber 3”. With this desired functionality, we will want our software trigger to change to “True” when the animal enters chambers one or three and to change to “False” when the animal enters chamber two. This can be accomplished by connecting the output of the “Image Processing” node to a “Python Transform” node in parallel with the “Csv Writer” node.



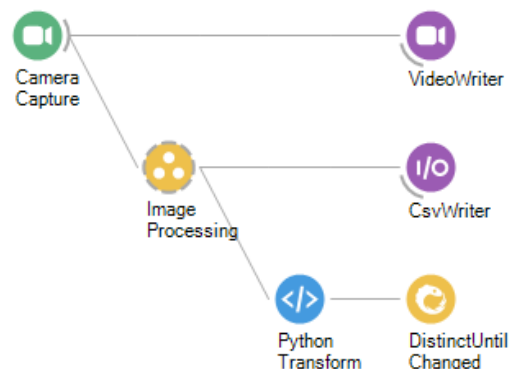
Inside the python script, we want to output the logical OR value of the “Chamber 1” and “Chamber 3” elements. This way, the “Python Transform” node will output True while the animal is in chambers one or three and False while the animal is in chamber two. Since we specified the names of items one through six, we can more easily access the values of the incoming data within this script using “value.Chamber1” and “value.Chamber3”.



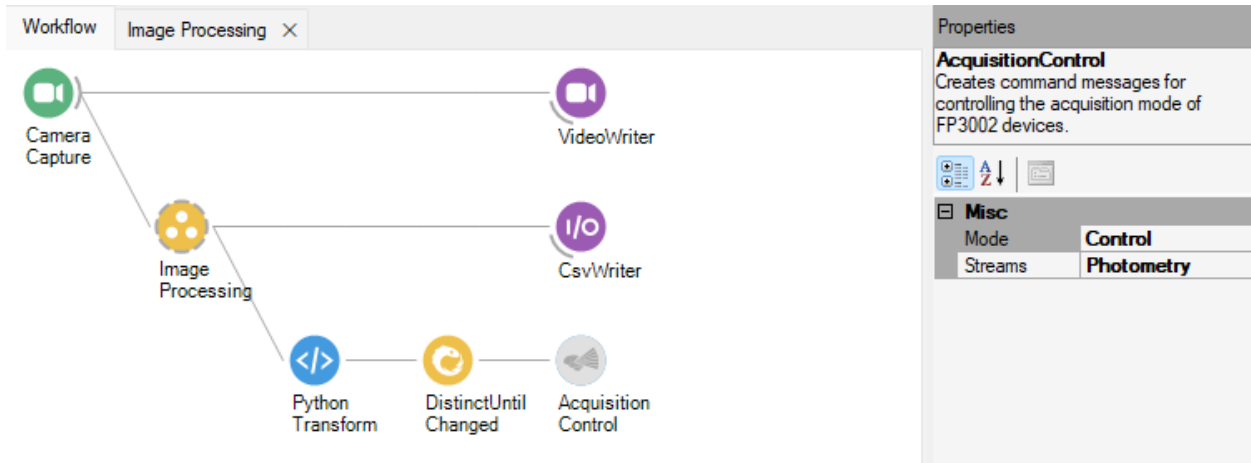
```
@returns(bool)
def process(value):
    return value.Chamber1 or value.Chamber3
```

OK Cancel

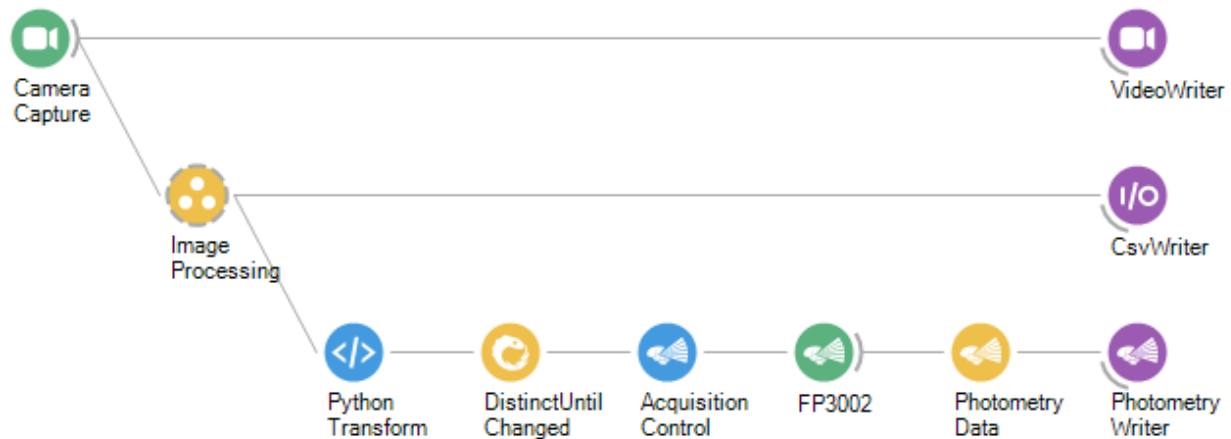
This script will output a value every behavioral camera frame, however, we only want to send acquisition control commands to the “FP3002” node when this value changes. Be sure to include a “Distinct Until Changed” node after this “Python Transform” node.



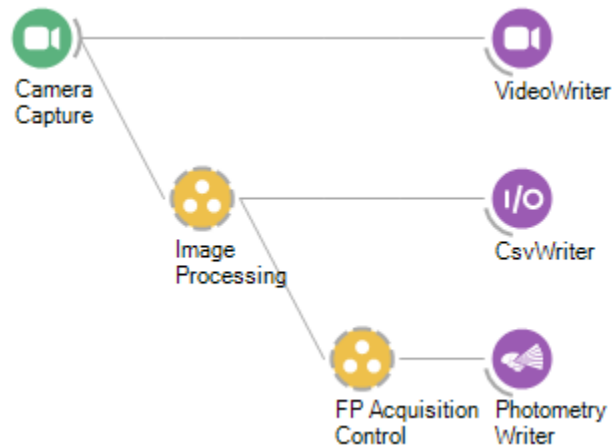
Now, we can create acquisition control commands using the “Acquisition Control” node after the “Distinct Until Changed” node. The “Acquisition Control” node should be configured to have the “Mode” and “Streams” properties set to “Control” and “Photometry”.



Our acquisition control logic is now ready to connect to our photometry data stream.



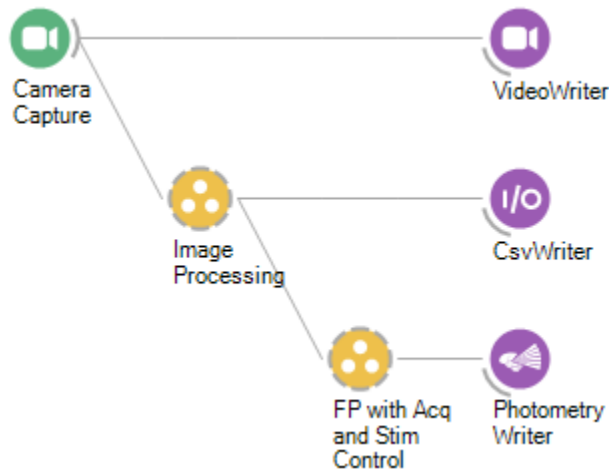
This workflow could use some more organization by grouping together the nodes used to create and control the flow of photometry data. In this example, we group all of the nodes from the “Python Transform” node to the “Photometry Data” node and name the grouped workflow “FP Acquisition Control”.



Before running an experiment with this workflow, be sure to configure the image processing grouped workflow as outlined in the [“Three Chamber Animal Tracking”](#) section.

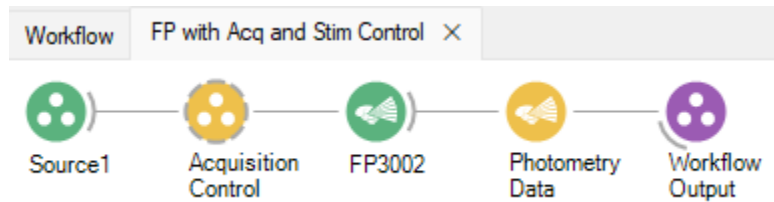
## Chamber Dependent Stimulation

This section will expand upon the previous section such that the animal will trigger stimulation when entering a particular chamber. We will keep our acquisition control logic such that data acquisition will only occur while the animal is in chambers one and two. We will add stimulation logic such that stimulation will be triggered when the animal enters chamber one.

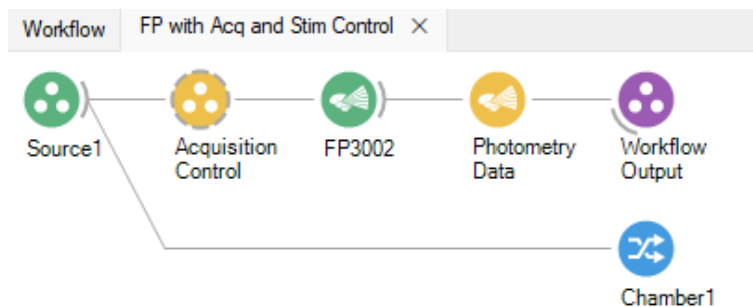
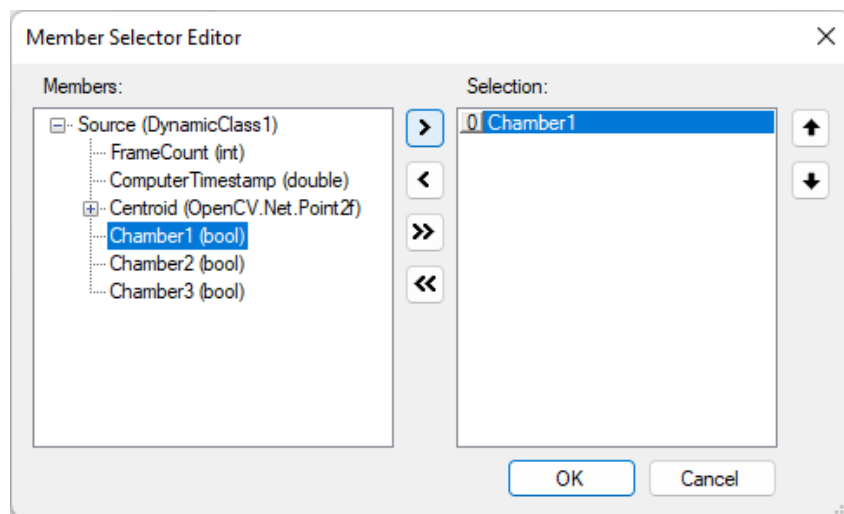


To construct this workflow, begin with the workflow created in the “Chamber Dependent Data Acquisition” section. Begin by renaming the grouped workflow containing the nodes used for creating and controlling the photometry data stream. In the previous example, this was named “FP Acquisition Control”. Since this will now also contain the stimulation control logic, we will rename this to “FP with Acq and Stim Control”.

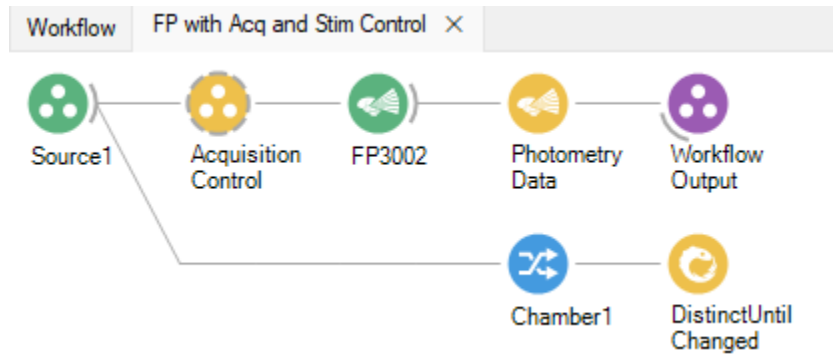
Next, open the “FP with Acq and Stim Control” grouped workflow and group together the acquisition control logic, i.e. the “Python Transform” node through the “Acquisition Control” node. This will work to keep our acquisition and stim control logic separated.



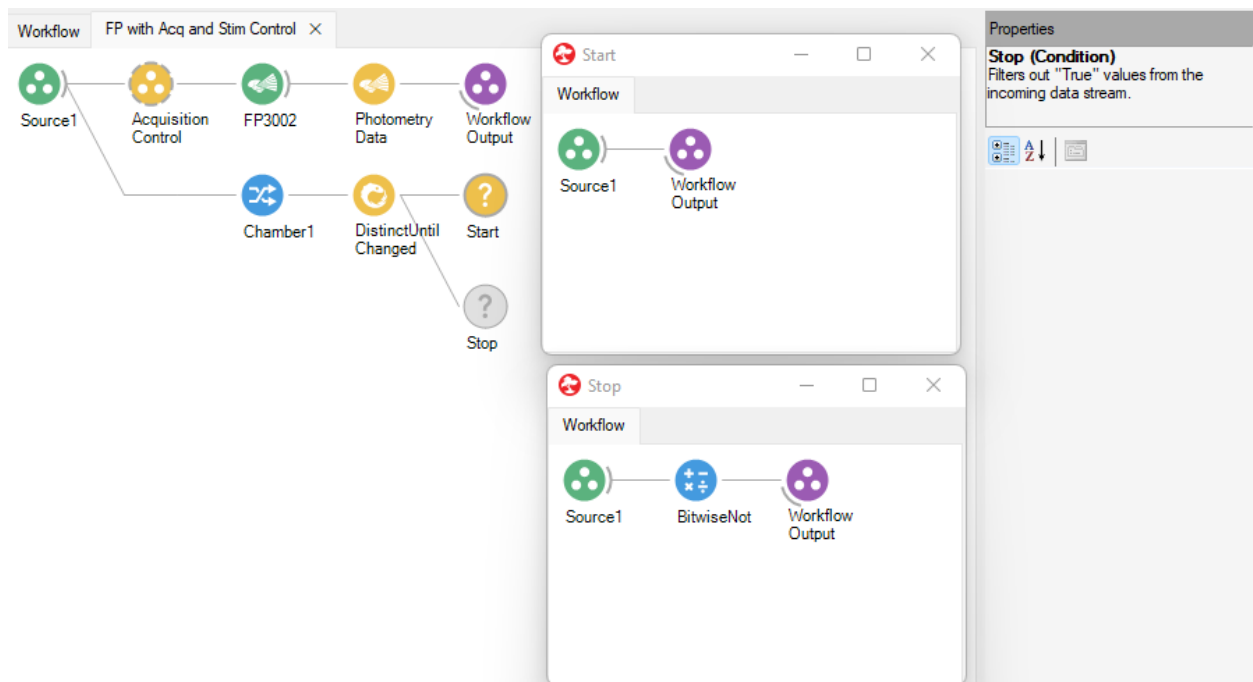
We will now connect our stimulation control logic to the output of the “Source1” node in parallel with the “Acquisition Control” grouped workflow. Begin by connecting a “Member Selector” node to the source node. Open the editor for this node, click the “Chamber1” member and select it by clicking the greater than symbol. Press “OK” when complete and you have successfully selected the “Chamber1” item coming from the source node.



For every behavioral camera frame, this “Chamber1” node will output a “True” value while the animal is in chamber one and “False” otherwise. We only want to send a stimulation command when this value changes so be sure to include a “Distinct Until Changed” node immediately after.

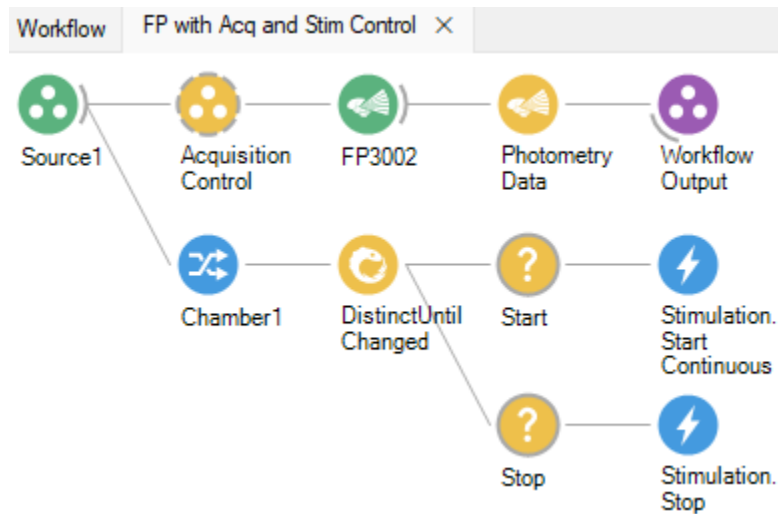


From the “Distinct Until Changed” node, connect two “Condition” nodes in parallel to deinterleave the boolean values into two data streams. One “Condition” node requires no changes to filter out the “False” values, we will name this one “Start”. The other “Condition” node requires a “Bitwise Not” node inserted between the “Source1” node and the “Workflow Output” node. We will name this “Condition” node “Stop” as it will filter out the “True” values.

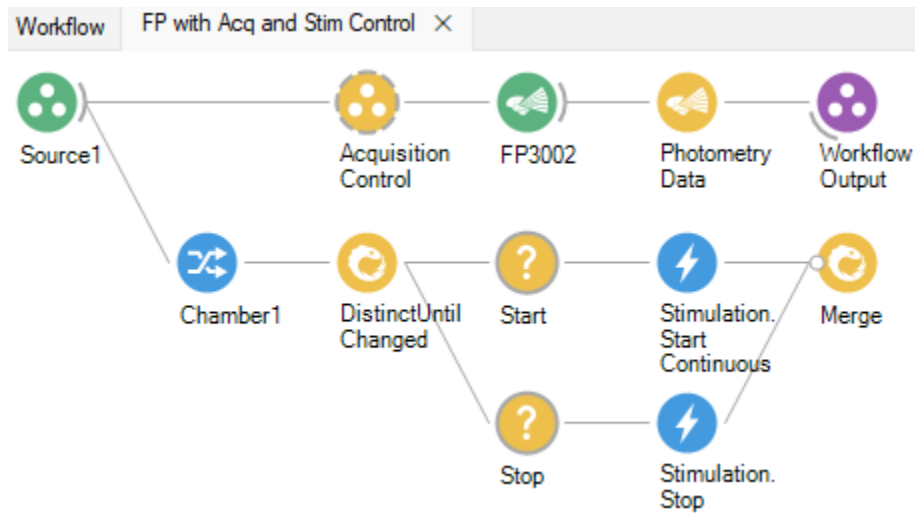


A “Stimulation” node needs to be connected to both of the “Condition” nodes. Configure the “Stimulation” node connected to the “Start” node to have a start

stimulation command. Meanwhile, configure the “Stimulation” node connected to the “Stop” node to have a stop stimulation command.

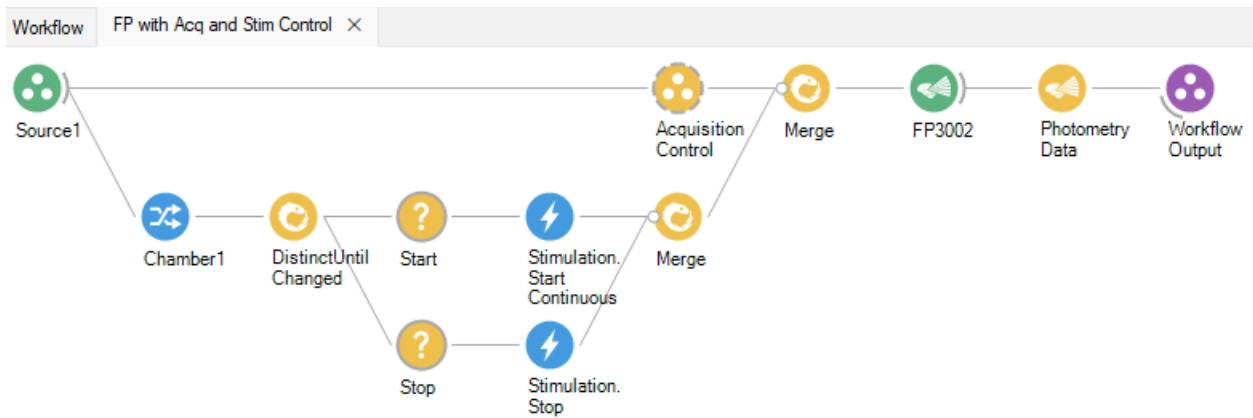


Next, merge together the two data streams containing stimulation commands.

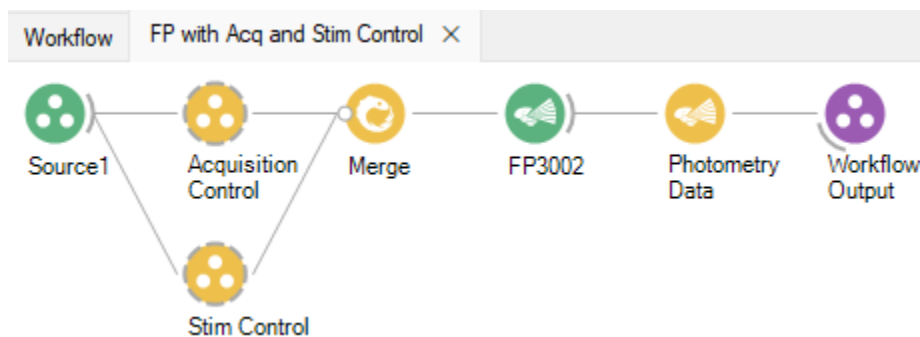




Then, merge together the acquisition control commands and the stimulation commands using a “Merge” node immediately before the “FP3002” node.



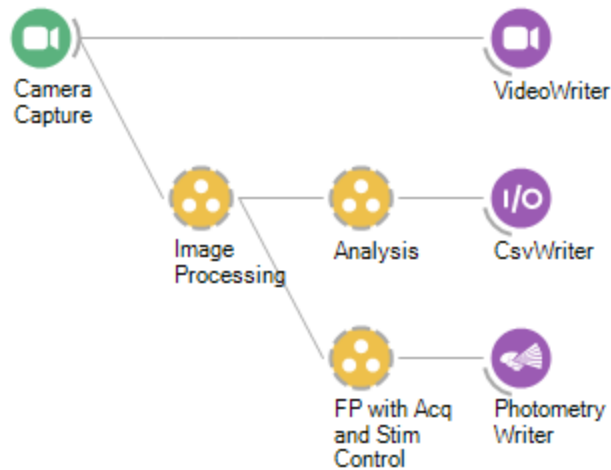
Finally, add some organization by grouping together all of the stimulation control logic. In this example, we group from the “Chamber 1” node to the first “Merge” node and name it “Stim Control”.



We now have all of the required logic for controlling data acquisition and triggering stimulation such that photometry data will be acquired while the animal is located in chambers one and three and stimulation will be triggered when the animal enters chamber one. Before running an experiment with this workflow, be sure to configure the image processing grouped workflow as outlined in the “[Three Chamber Animal Tracking](#)” section.

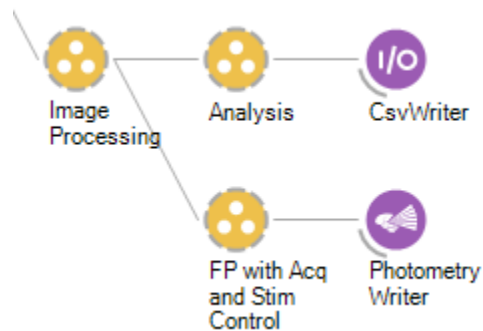
## Real Time Analysis

In this section we will discuss the addition of real time analysis algorithms that can generate more useful data from the image processing algorithms. The image processing algorithm developed and used in the previous sections produces the following data for every behavioral camera frame: Frame Count, Computer Timestamp, Centroid, and the chamber the animal is located in. With this information, we can conduct a variety of analyses in real time. This section will cover where in the workflow to insert these real time analyses and will cover three common types of real time analysis algorithms. In particular, we will implement algorithms for tracking the duration of time the animal spends in each chamber, the total distance traveled (in pixels), and the velocity (in pixels per millisecond).

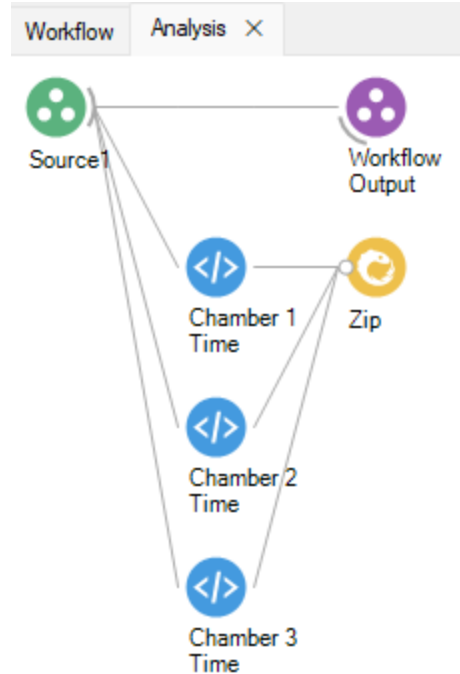


## Organization:

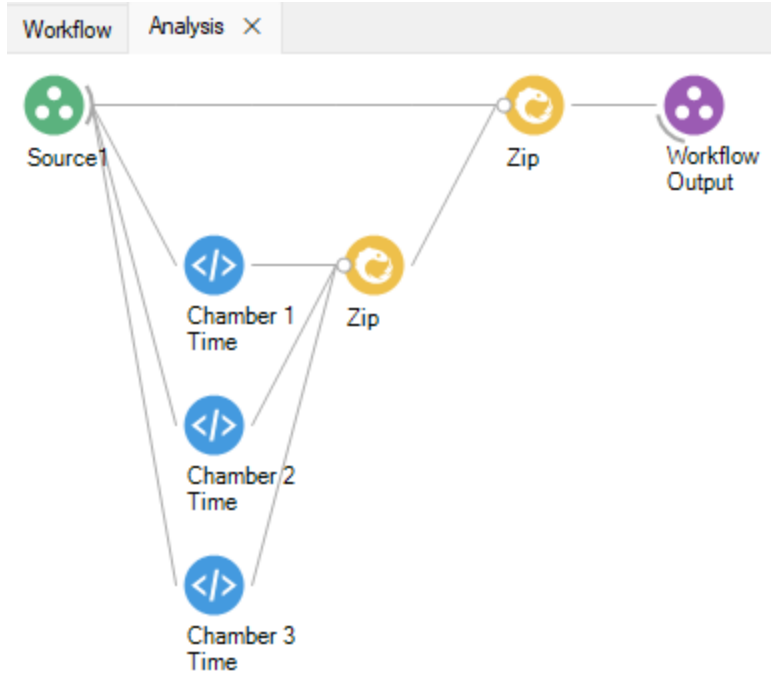
In this example, we will add these real time analysis algorithms to the workflow developed in the previous section. However, similar methods can be used to add analysis algorithms to other machine vision workflows. Our first task is to find a location within our workflow that we can insert these algorithms, without affecting the functionality of the workflow. Our analyses require the information produced by the “Image Processing” grouped workflow so we will insert them immediately after that node. However, we need to be sure that we are not affecting our stimulation and acquisition control logic for the photometry data stream. With this in mind, a good place for our real time analysis logic is to be located immediately after the “Image Processing” grouped workflow, parallel to the “FP with Acq and Stim Control” grouped workflow. Insert a “Group Workflow” node here and name it “Analysis”.



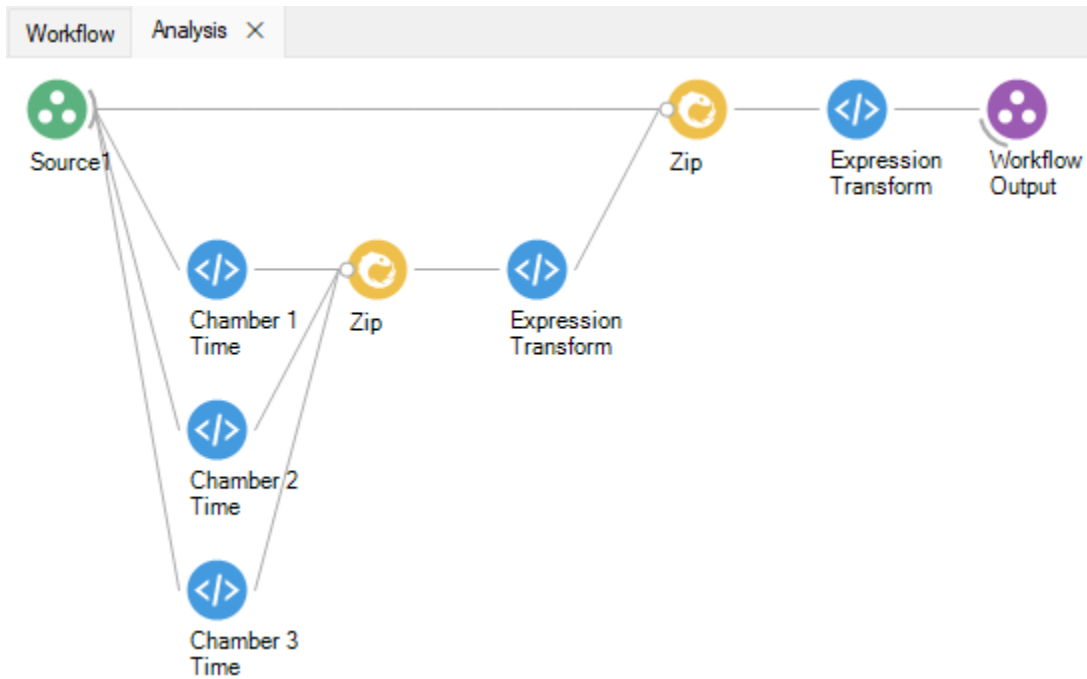
Inside of the “Analysis” grouped workflow, we will want to be sure to preserve the existing data produced by the “Image Processing” grouped workflow as well as analyze this incoming data. Here we will conduct all of our analyses in parallel data streams and combine them with our preserved incoming data. For our first analysis algorithm, we will track the duration of time the animal has spent in each chamber with units of milliseconds. Begin implementation by adding three “Python Transform” nodes, parallel to each other and the existing data stream. Combine them using a “Zip” node. Each of these three “Python Transform” nodes will be responsible for tracking the duration of time the animal spends in a particular chamber. Be sure to name these nodes to indicate which chamber they are analyzing.



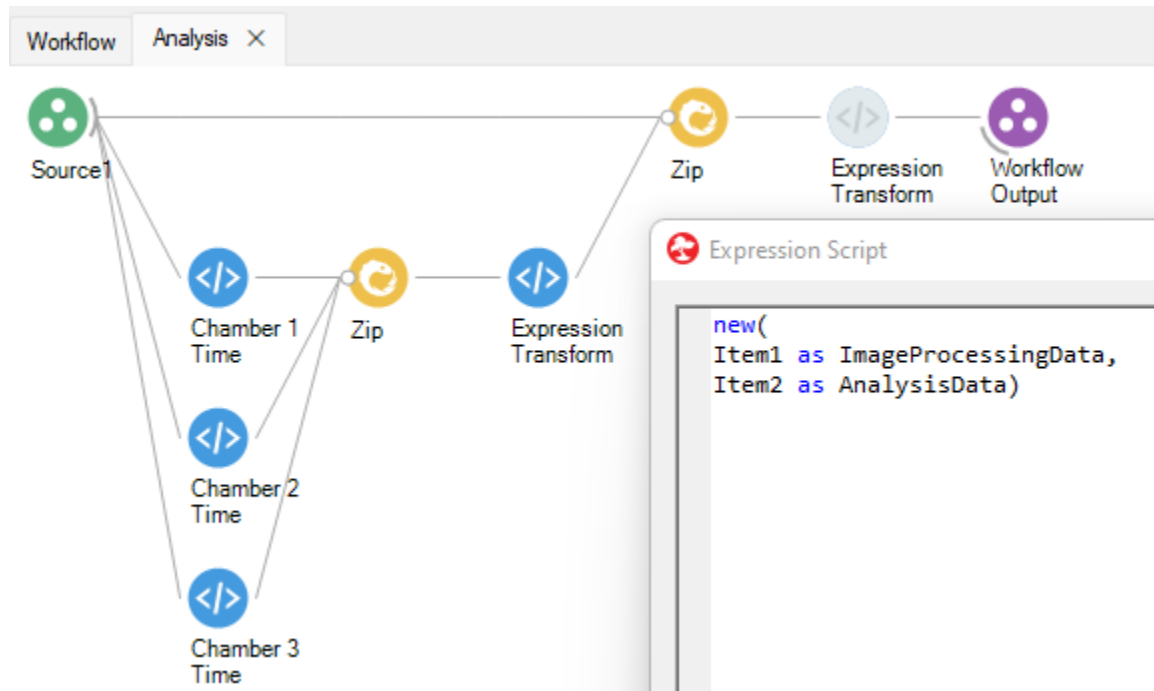
Before implementing our python scripts, let's finish the structure of the "Analysis" grouped workflow. This grouped workflow needs to output all of the image processing data as well as all of the analysis data. Since the analysis algorithms will produce data at the same rate as the image processing algorithms, we can combine these data streams with a "Zip" node connected to the "Source1" node and the already existing "Zip" node. With this format, we will be able to output our image processing data and analysis data in a format that a "Csv Writer" can handle, and we can easily add more analyses by adding them in parallel to the three "Python Transform" nodes.



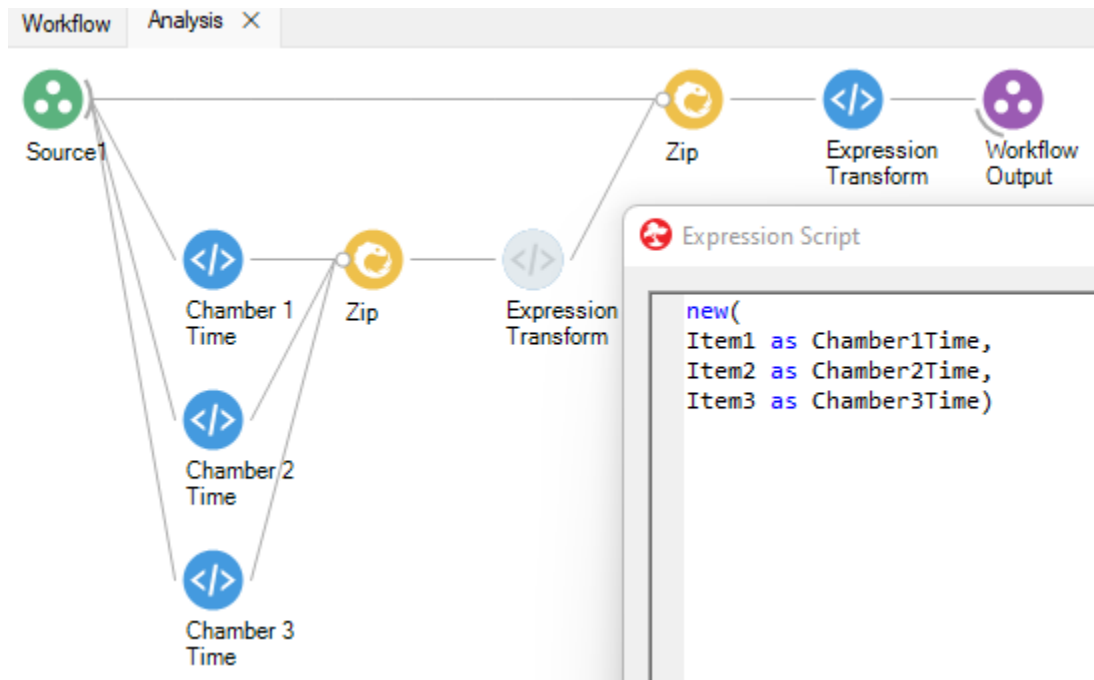
We can also add a level of organization by naming the data coming out of each “Zip” node. Do this by adding an “Expression Transform” after each “Zip” node.



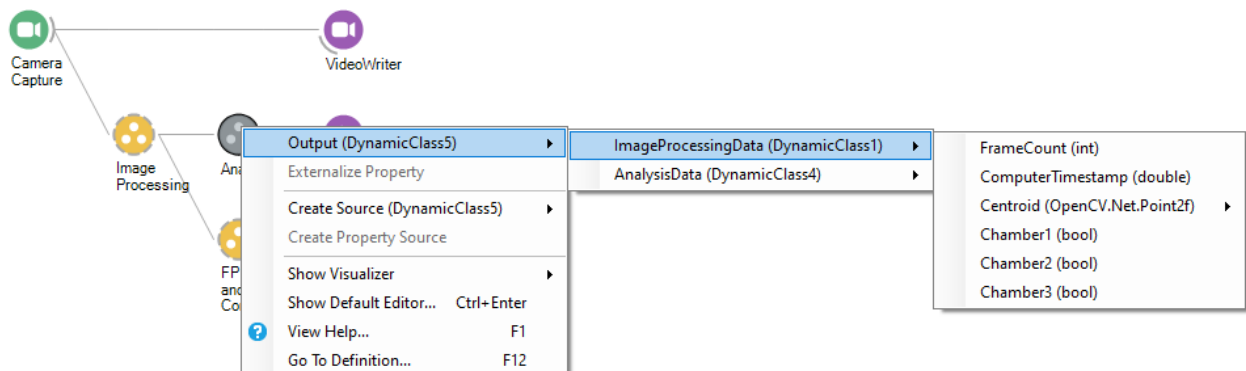
The “Expression Transform” connected to the “Workflow Output” will be used to specify between image processing data and analysis data. The below script will rename Item1 (the data coming from the “Source1” node) to “ImageProcessingData” and will rename Item2 (the data coming from the zipped analysis algorithms) to “AnalysisData”.

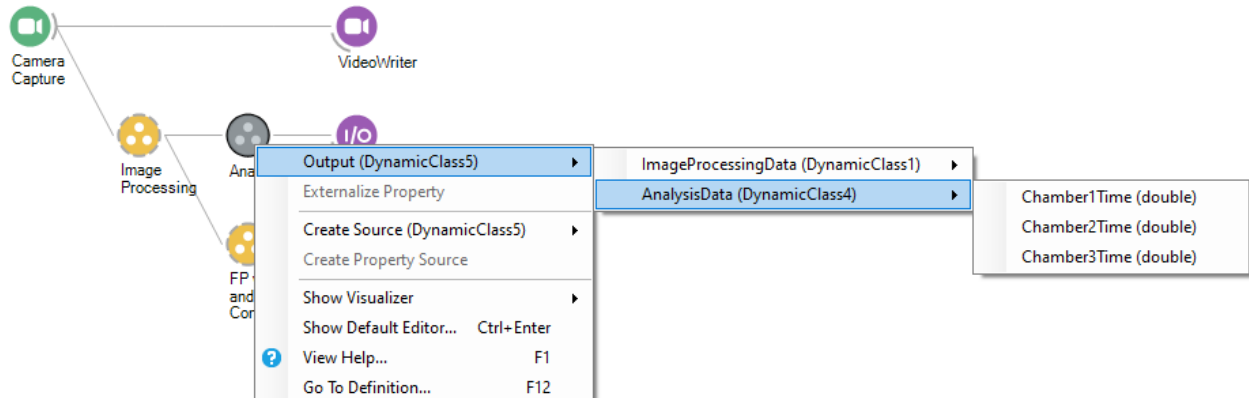


The other “Expression Transform” will rename the data coming from the zipped analyses. In this case, we will rename items one through three to “Chamber1Time” through “Chamber3Time”. We will update this “Expression Transform” node every time we add a new real time analysis algorithm.



The immediate effect of this level of organization is that the element names are displayed in the output tree of the “Analysis” grouped workflow. Right click it and navigate the “Output” tree to see all of the elements contained in the output data of this grouped workflow. This organization is also useful because we can have the “Csv Writer” node write these element names as column names in the output .csv file.





## Chamber Duration:

With our “Analysis” grouped workflow structured and organized, let’s delve into the real time analysis scripts we will be using. Open the “Analysis” grouped workflow and open the “Chamber1Time” python script editor. Here will be where we implement the logic for keeping track of the amount of time the animal spends in chamber one. When writing these scripts, it is often easiest to start with bringing in the required input information for the algorithm. In this case, in order to keep track of the amount of time spent in chamber one, we will need the timestamp and the boolean value indicating if the animal is in chamber one.

```

Python Script
@returns(bool)
def process(value):

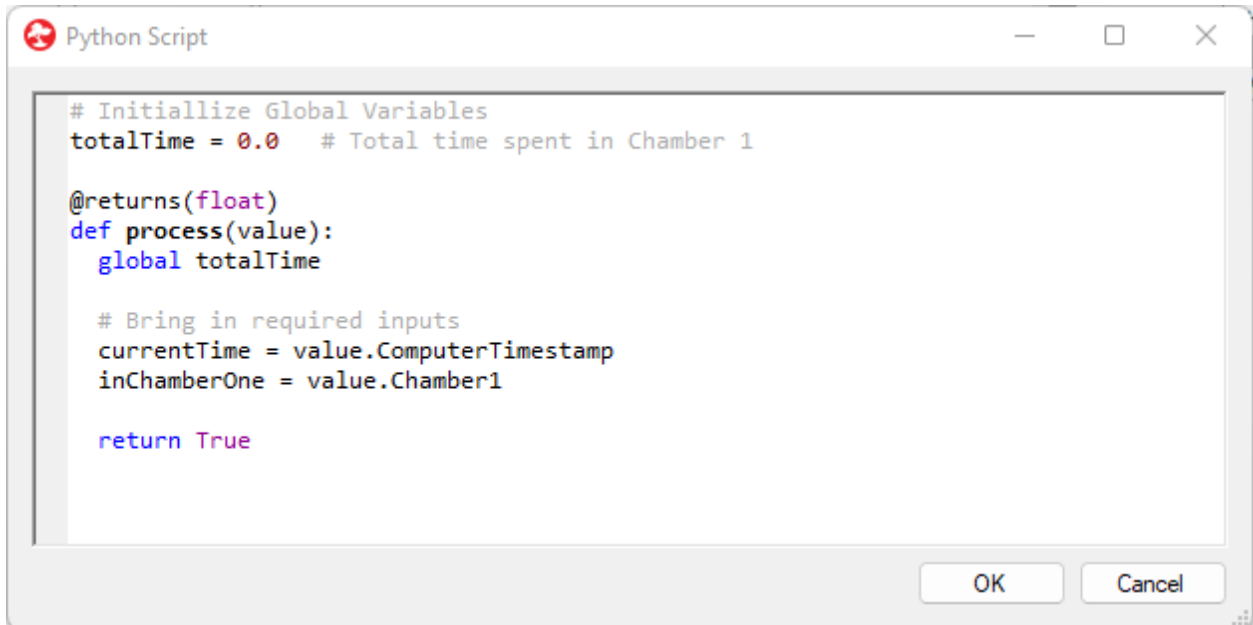
    # Bring in required inputs
    currentTime = value.ComputerTimestamp
    inChamberOne = value.Chamber1

    return True
  
```

OK Cancel



Next let's specify our output. In this case, we will be outputting a float value indicating the amount of milliseconds that the animal has spent in chamber one. Knowing this, we can specify that the return data type is a float and we can set up a global variable that will keep track of the total time spent in the chamber.

A screenshot of a 'Python Script' dialog box. The window has a title bar with a red icon, the text 'Python Script', and standard window controls (minimize, maximize, close). The main area contains Python code with syntax highlighting. The code initializes a global variable 'totalTime' to 0.0, defines a function 'process' with a float return type, and uses 'global' to access 'totalTime'. It also shows how to access attributes from a 'value' object. At the bottom right, there are 'OK' and 'Cancel' buttons.

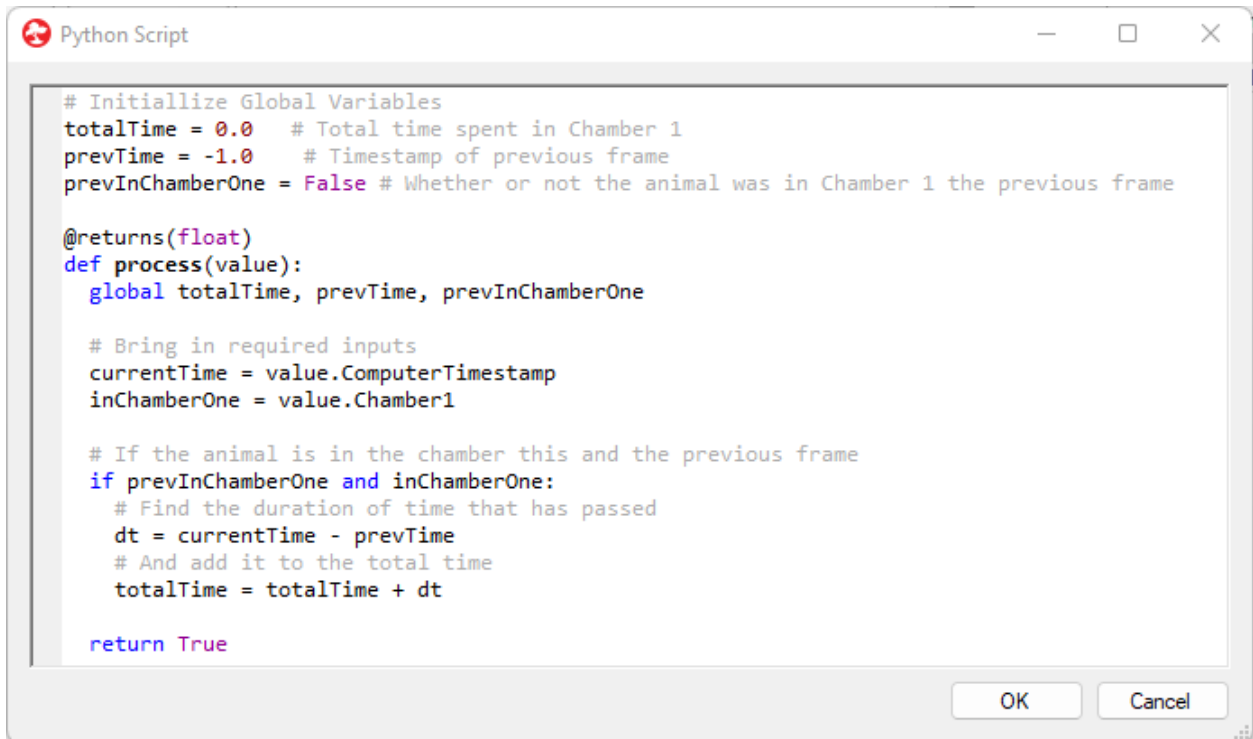
```
# Initiallize Global Variables
totalTime = 0.0 # Total time spent in Chamber 1

@returns(float)
def process(value):
    global totalTime

    # Bring in required inputs
    currentTime = value.ComputerTimestamp
    inChamberOne = value.Chamber1

    return True
```

Now we need to solidify our algorithm that will accept the timestamp and boolean indicator information and update the total time global variable. One way to do this is to determine if the animal is in chamber one during the current frame and during the previous frame. If so, take the time difference between those two frames and add it to the total time variable. Implementing this will require two more global variables to keep track of the previous frame's timestamp and boolean indicator.

A screenshot of a 'Python Script' dialog box. The dialog has a title bar with a Python logo, the text 'Python Script', and standard window controls (minimize, maximize, close). The main area contains Python code for a function named 'process'. The code initializes global variables 'totalTime', 'prevTime', and 'prevInChamberOne'. It then defines the 'process' function which takes a 'value' argument and updates the global variables based on the current frame's data. The function returns 'True'. At the bottom right, there are 'OK' and 'Cancel' buttons.

```
# Initiallize Global Variables
totalTime = 0.0 # Total time spent in Chamber 1
prevTime = -1.0 # Timestamp of previous frame
prevInChamberOne = False # Whether or not the animal was in Chamber 1 the previous frame

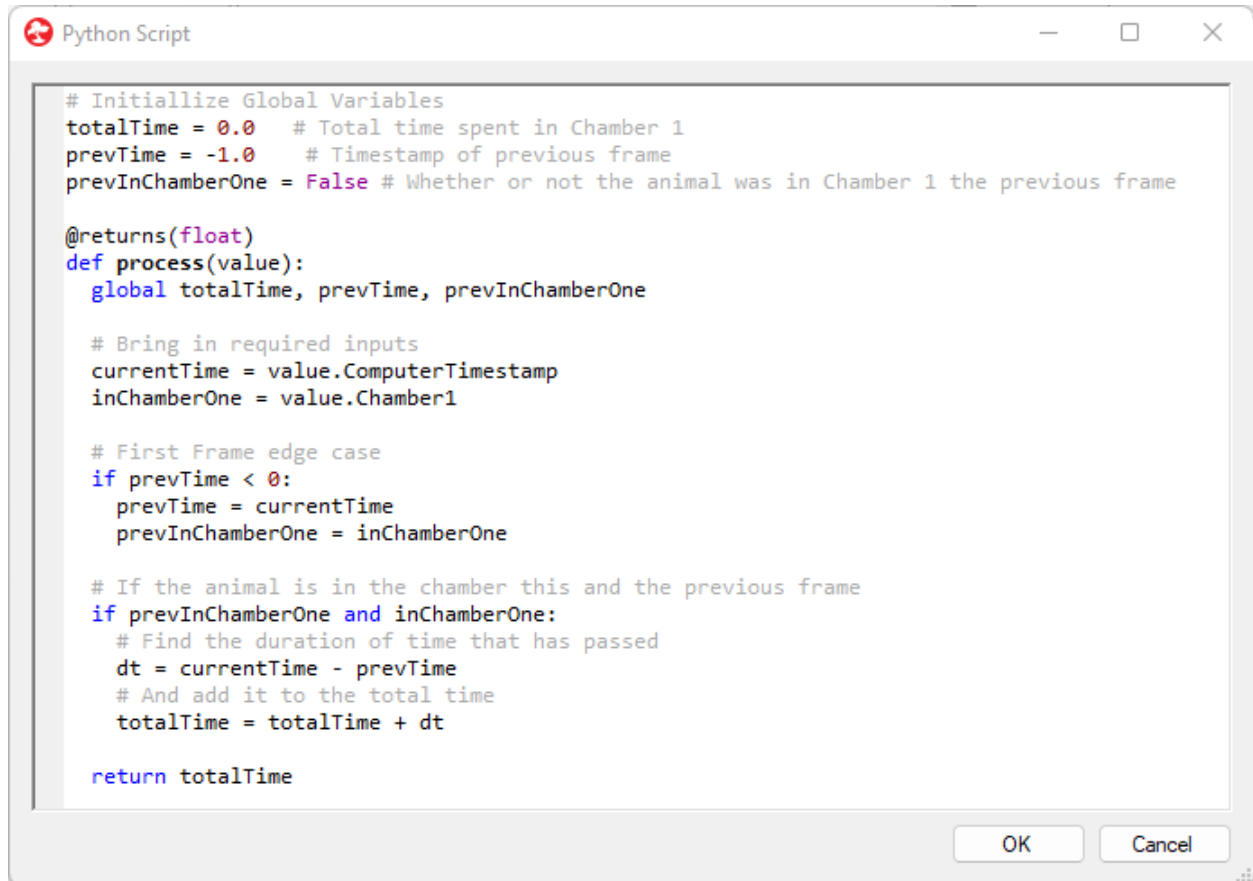
@returns(float)
def process(value):
    global totalTime, prevTime, prevInChamberOne

    # Bring in required inputs
    currentTime = value.ComputerTimestamp
    inChamberOne = value.Chamber1

    # If the animal is in the chamber this and the previous frame
    if prevInChamberOne and inChamberOne:
        # Find the duration of time that has passed
        dt = currentTime - prevTime
        # And add it to the total time
        totalTime = totalTime + dt

    return True
```

The bulk of the algorithm is complete, however, we have yet to implement logic for handling the first frame. During the first frame, we only want to update the previous frame's timestamp and boolean indicator. We also want to be sure we return the total time variable.



```
# Python Script

# Initialize Global Variables
totalTime = 0.0 # Total time spent in Chamber 1
prevTime = -1.0 # Timestamp of previous frame
prevInChamberOne = False # Whether or not the animal was in Chamber 1 the previous frame

@returns(float)
def process(value):
    global totalTime, prevTime, prevInChamberOne

    # Bring in required inputs
    currentTime = value.ComputerTimestamp
    inChamberOne = value.Chamber1

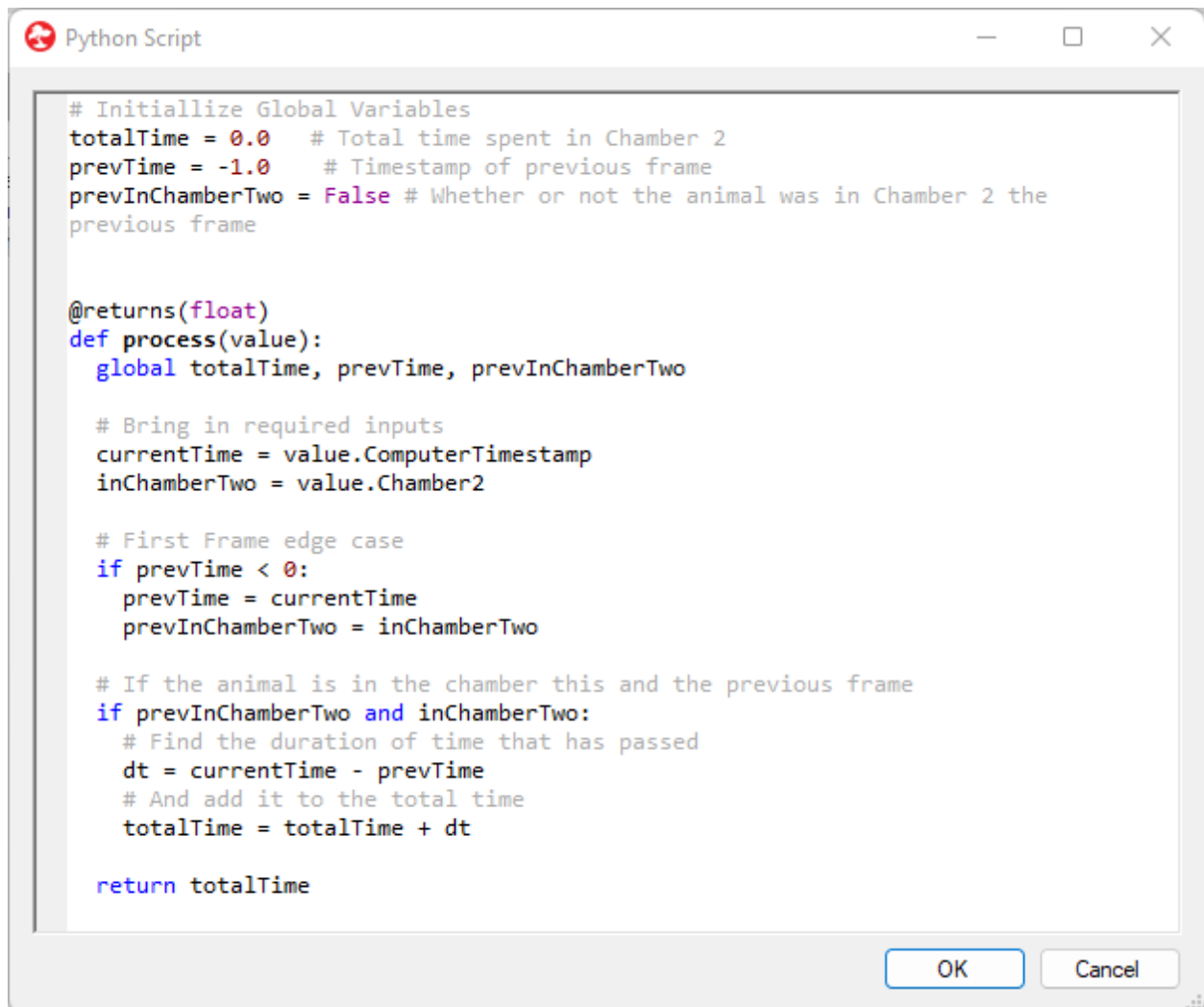
    # First Frame edge case
    if prevTime < 0:
        prevTime = currentTime
        prevInChamberOne = inChamberOne

    # If the animal is in the chamber this and the previous frame
    if prevInChamberOne and inChamberOne:
        # Find the duration of time that has passed
        dt = currentTime - prevTime
        # And add it to the total time
        totalTime = totalTime + dt

    return totalTime

OK Cancel
```

We have now completed the algorithm for tracking the duration of time the animal spends in chamber one. We can copy this script into the chamber two and three python scripts and make minor changes in order to complete those algorithms. Technically, all that has to be changed is reading in “value.Chamber1” to “value.Chamber2” or “value.Chamber3”. However, we will also change the variable names and comments to show that those scripts are for chambers two and three.



```
# Python Script

# Initiallize Global Variables
totalTime = 0.0 # Total time spent in Chamber 2
prevTime = -1.0 # Timestamp of previous frame
prevInChamberTwo = False # Whether or not the animal was in Chamber 2 the
previous frame

@returns(float)
def process(value):
    global totalTime, prevTime, prevInChamberTwo

    # Bring in required inputs
    currentTime = value.ComputerTimestamp
    inChamberTwo = value.Chamber2

    # First Frame edge case
    if prevTime < 0:
        prevTime = currentTime
        prevInChamberTwo = inChamberTwo

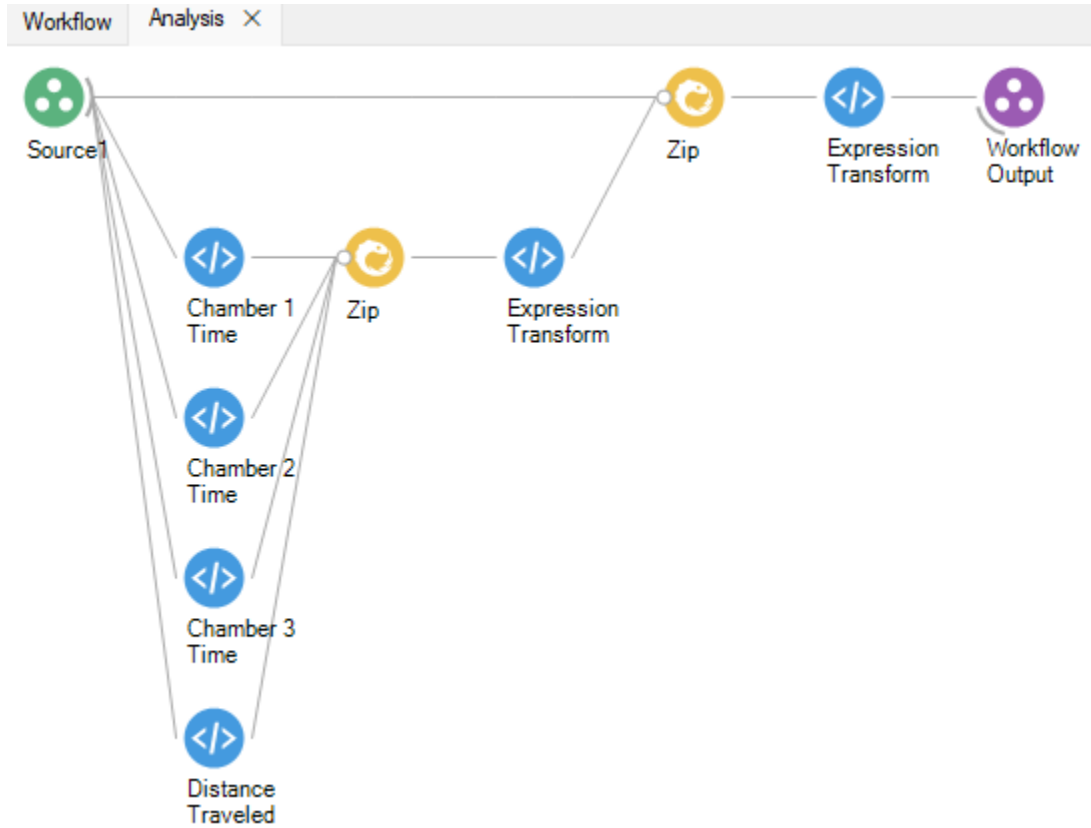
    # If the animal is in the chamber this and the previous frame
    if prevInChamberTwo and inChamberTwo:
        # Find the duration of time that has passed
        dt = currentTime - prevTime
        # And add it to the total time
        totalTime = totalTime + dt

    return totalTime


OK Cancel
```

## Distance Traveled:

The next algorithm we will go over is the distance traveled algorithm. This will accept the centroid information as an input and track the total distance, in pixels, that the animal has traveled over the course of the experiment. Begin by adding another “Python Transform” node, parallel to the chamber time scripts. Name this node “Distance Traveled”.



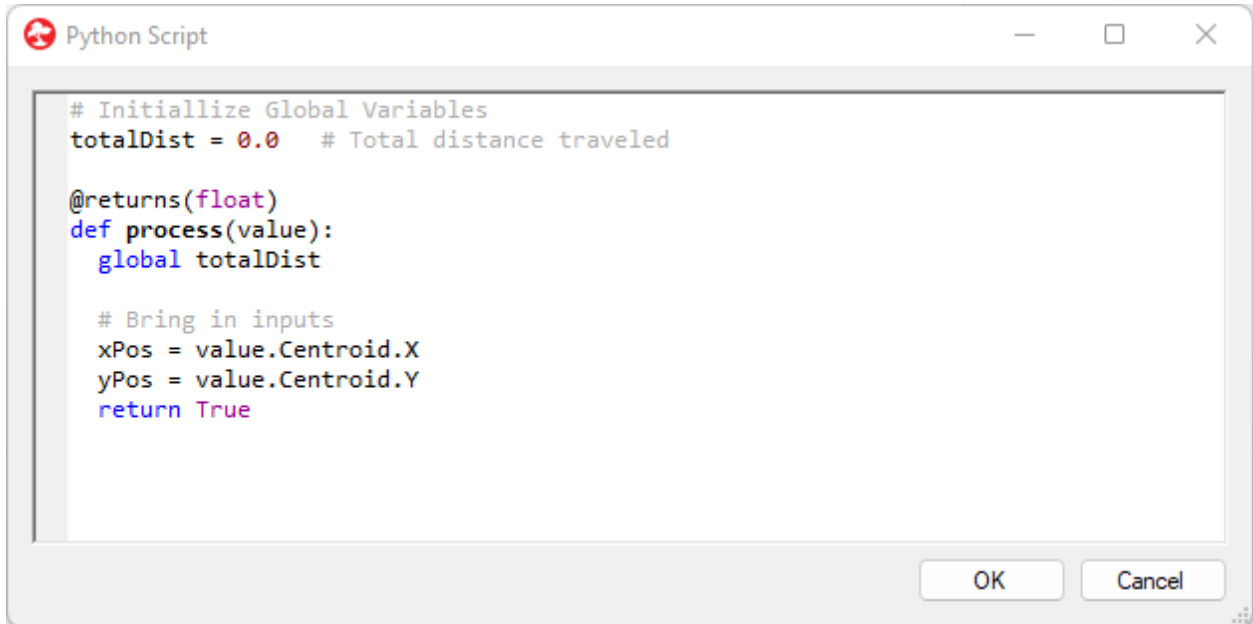
Begin the “Distance Traveled” script by bringing in the X and Y coordinate information from the Centroid input.

A screenshot of a 'Python Script' dialog box. The window title is 'Python Script' and it has standard minimize, maximize, and close buttons. The main area contains Python code for a function named 'process'. The code is as follows:

```
@returns(bool)
def process(value):
    # Bring in inputs
    xPos = value.Centroid.X
    yPos = value.Centroid.Y
    return True
```

At the bottom right of the dialog box, there are two buttons: 'OK' and 'Cancel'.

The output of this script will be a float value containing the total distance traveled. We will need a global variable to contain the distance traveled value and we need to specify that this script returns a float value.

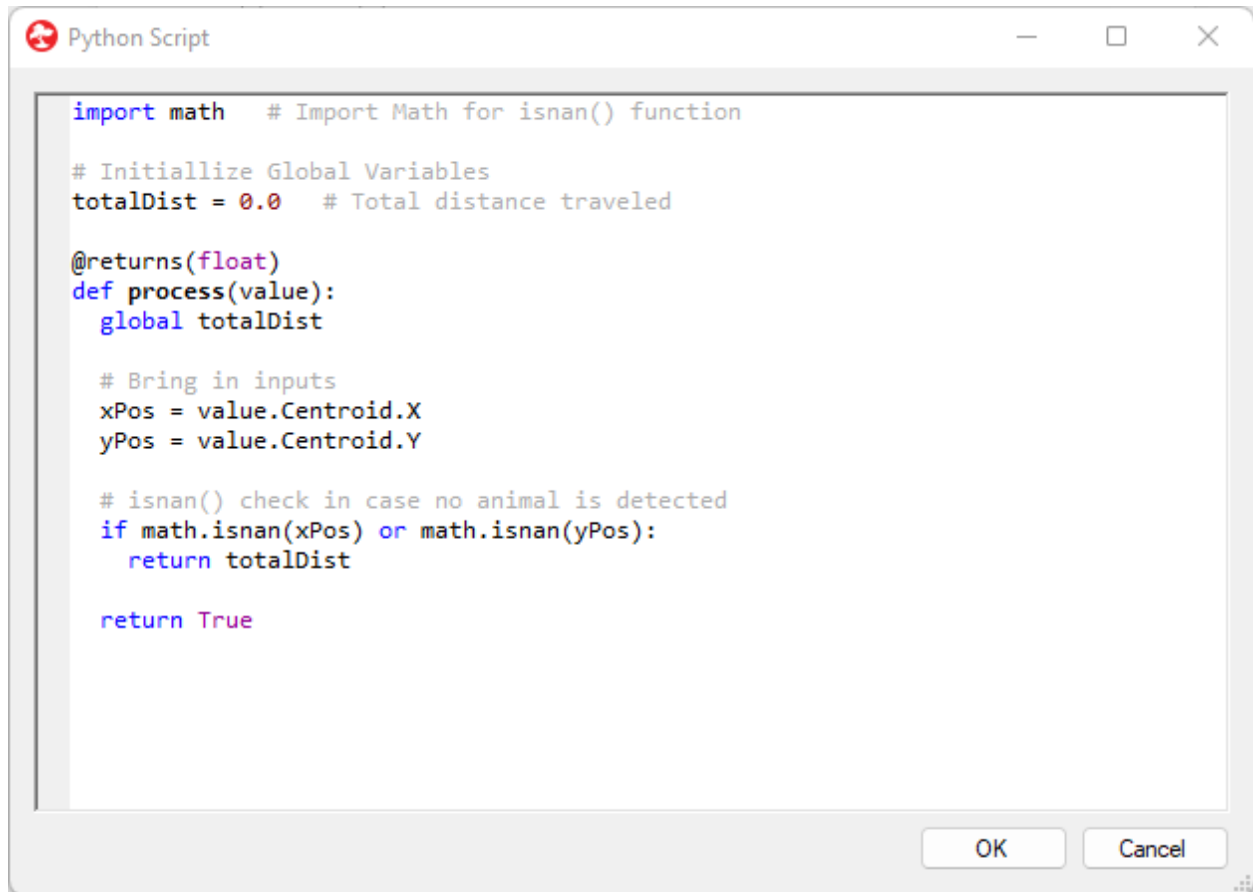
A dialog box titled "Python Script" with a red icon on the left and standard window controls (minimize, maximize, close) on the right. The main area contains Python code. At the bottom right, there are "OK" and "Cancel" buttons.

```
# Initiallize Global Variables
totalDist = 0.0 # Total distance traveled

@returns(float)
def process(value):
    global totalDist

    # Bring in inputs
    xPos = value.Centroid.X
    yPos = value.Centroid.Y
    return True
```

There is a major edge case that needs to be handled. If the animal is not detected in the enclosure, the image processing algorithm will output NaN (Not a Number) values. These values require special treatment so that we do not experience runtime errors. A simple way to handle these potential NaN values is to import the math library and to return the total distance when a NaN value is detected.



```
import math # Import Math for isnan() function

# Initialize Global Variables
totalDist = 0.0 # Total distance traveled

@returns(float)
def process(value):
    global totalDist

    # Bring in inputs
    xPos = value.Centroid.X
    yPos = value.Centroid.Y

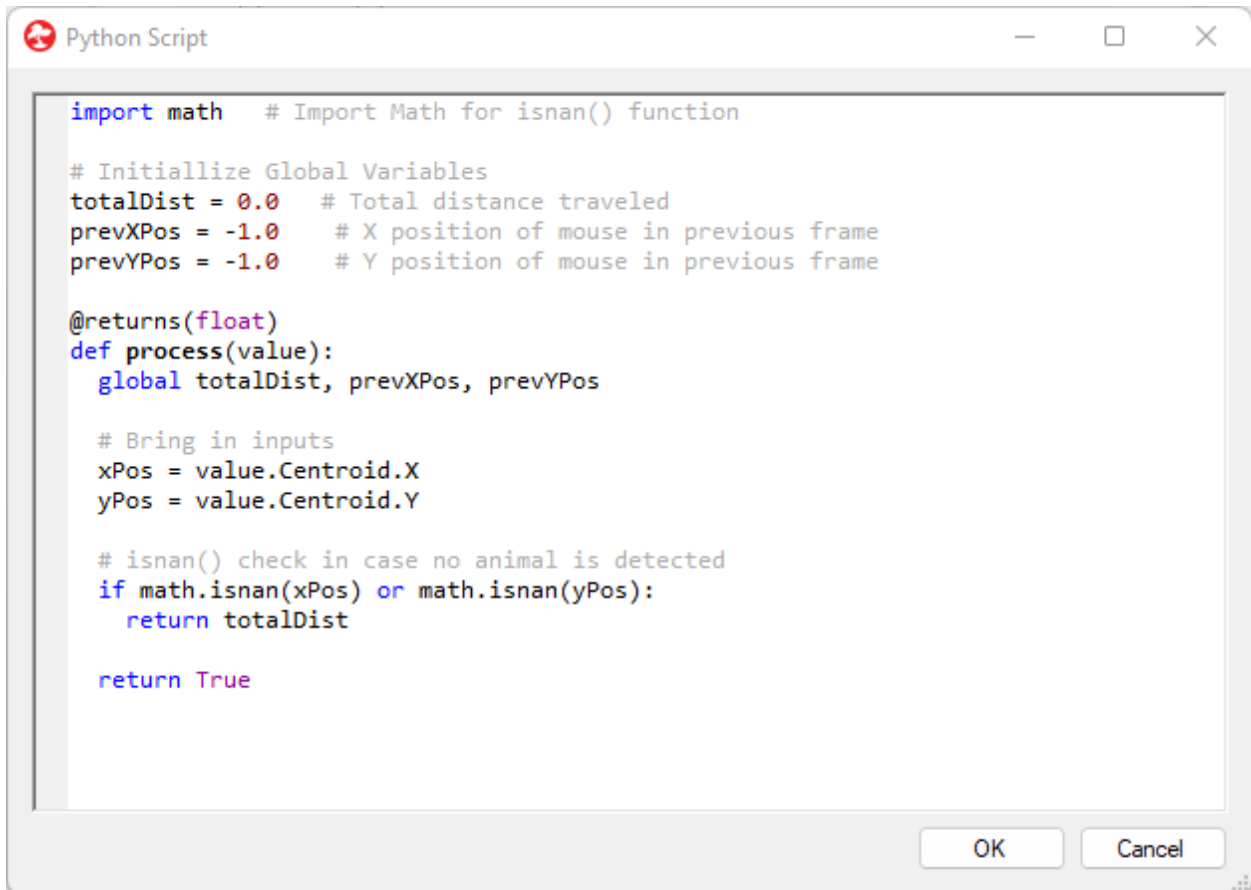
    # isnan() check in case no animal is detected
    if math.isnan(xPos) or math.isnan(yPos):
        return totalDist

    return True
```

OK Cancel



Our algorithm will find the distance traveled per frame and add that value to the total distance. With this goal in mind, we need to declare variables to store the X and Y position of the animal in the previous frame. We will initialize these to negative values, values that the imaging processing algorithm will never output, so that the algorithm can easily determine whether or not it is the first frame in which an animal is detected in the enclosure.

A screenshot of a Python Script dialog box. The window title is "Python Script" and it has standard minimize, maximize, and close buttons. The main area contains Python code for calculating distance traveled. The code includes imports, variable initialization, a function definition with a decorator, and logic for handling missing data points.

```
import math # Import Math for isnan() function

# Initiallize Global Variables
totalDist = 0.0 # Total distance traveled
prevXPos = -1.0 # X position of mouse in previous frame
prevYPos = -1.0 # Y position of mouse in previous frame

@returns(float)
def process(value):
    global totalDist, prevXPos, prevYPos

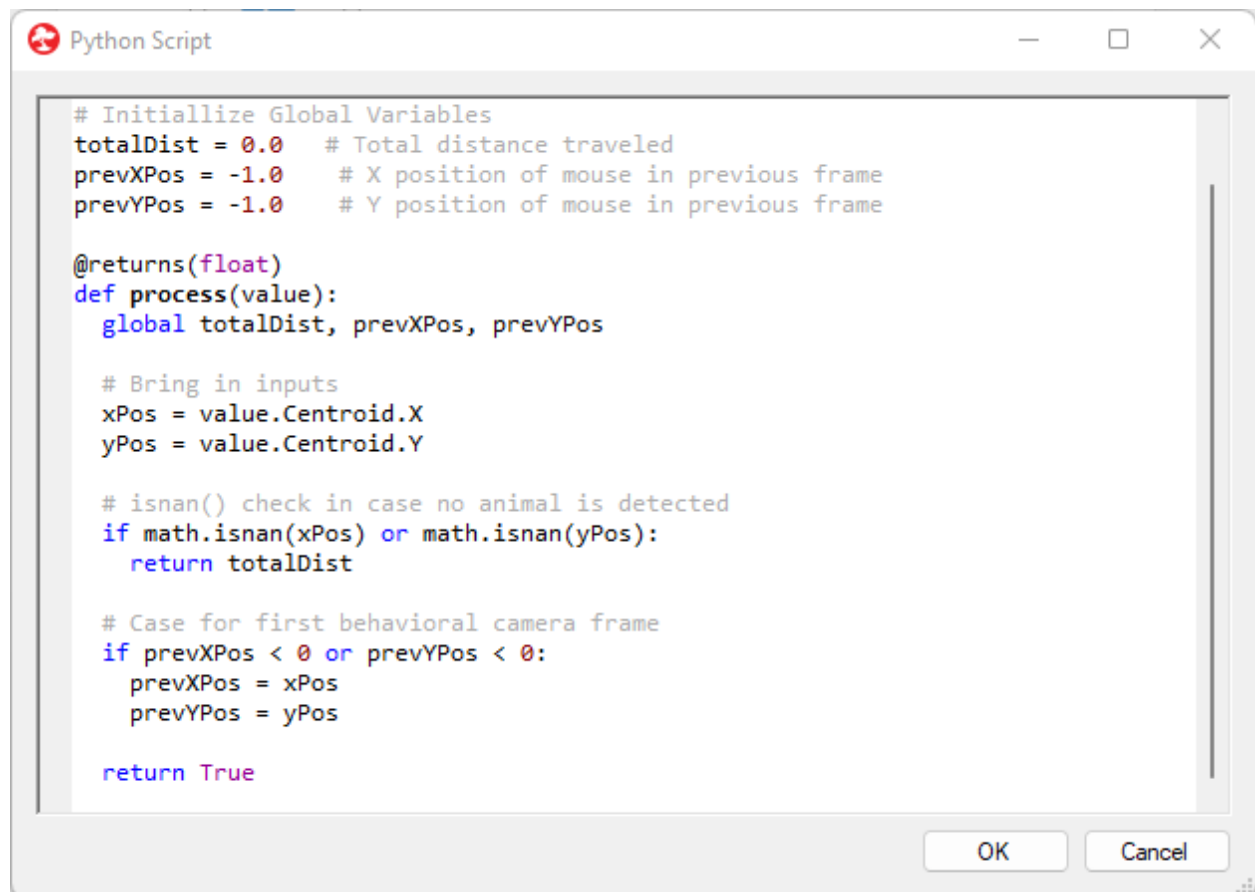
    # Bring in inputs
    xPos = value.Centroid.X
    yPos = value.Centroid.Y

    # isnan() check in case no animal is detected
    if math.isnan(xPos) or math.isnan(yPos):
        return totalDist

    return True
```

At the bottom right of the dialog box, there are two buttons: "OK" and "Cancel".

For the first frame in which an animal is detected in the enclosure, we will update the variables containing the animal's position in the previous frame.

A dialog box titled "Python Script" with a red icon on the left and standard window controls (minimize, maximize, close) on the right. The main area contains Python code for initializing global variables and defining a process function. The code includes comments for each step, such as "Initiallize Global Variables", "Bring in inputs", and "Case for first behavioral camera frame". The function "process" takes a "value" argument and returns a float. It uses "math.isnan" to check for missing values and updates "prevXPos" and "prevYPos" if an animal is detected in the first frame.

```
# Initiallize Global Variables
totalDist = 0.0 # Total distance traveled
prevXPos = -1.0 # X position of mouse in previous frame
prevYPos = -1.0 # Y position of mouse in previous frame

@returns(float)
def process(value):
    global totalDist, prevXPos, prevYPos

    # Bring in inputs
    xPos = value.Centroid.X
    yPos = value.Centroid.Y

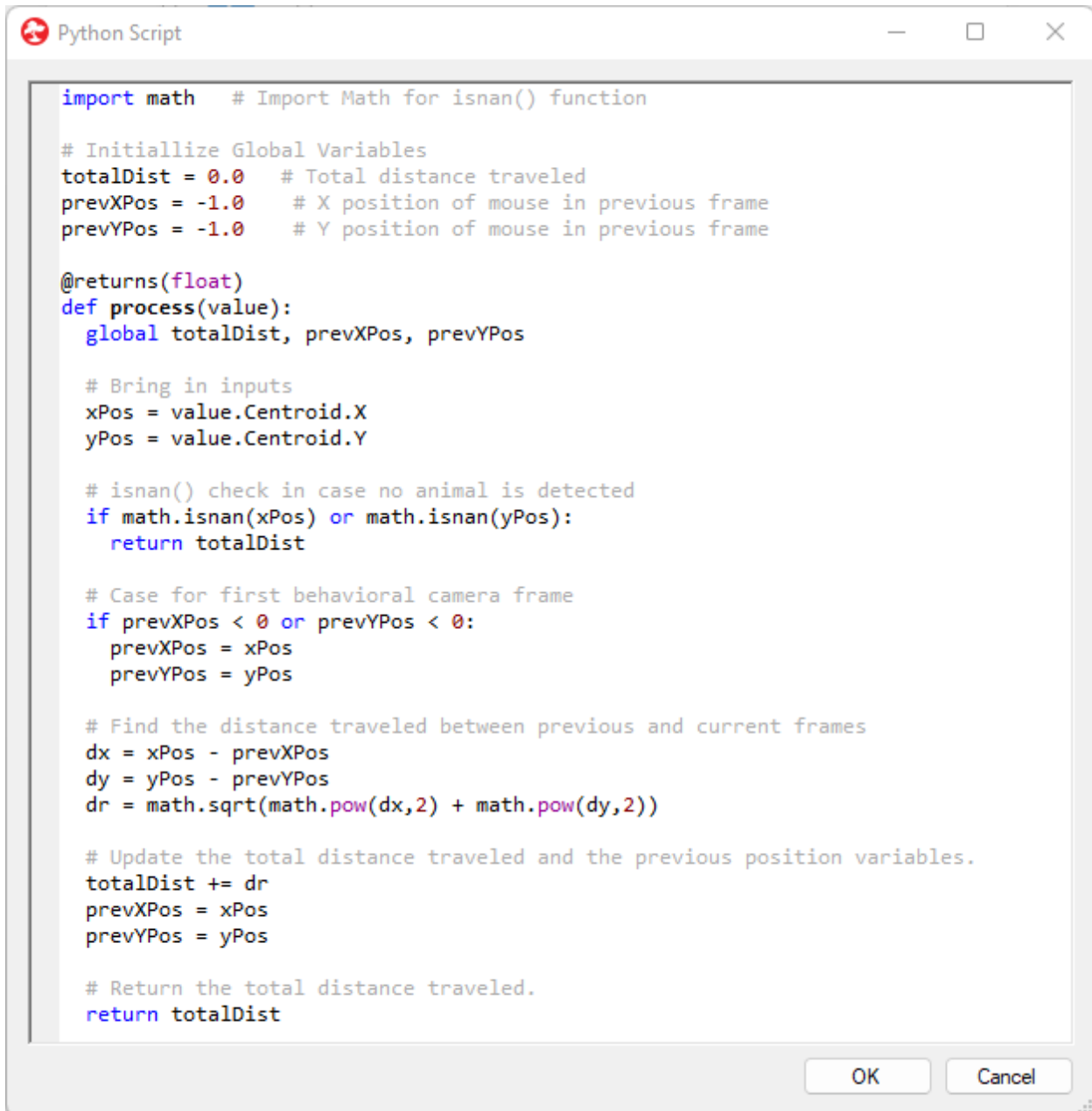
    # isnan() check in case no animal is detected
    if math.isnan(xPos) or math.isnan(yPos):
        return totalDist

    # Case for first behavioral camera frame
    if prevXPos < 0 or prevYPos < 0:
        prevXPos = xPos
        prevYPos = yPos

    return True
```

OK Cancel

Now that the major edge cases are handled, we can implement the algorithm. Here we will find the distance traveled between the previous and current frames and add it to the total distance traveled variable. Then we will update the variables containing the previous position of the animal. Finally we will return the updated total distance variable.

A screenshot of a Python Script window titled "Python Script". The window contains Python code for calculating the distance traveled by an animal between frames. The code includes comments and uses variables like totalDist, prevXPos, prevYPos, xPos, and yPos. It uses the math module for isnan() and sqrt() functions. The code is as follows:

```
import math # Import Math for isnan() function

# Initiallize Global Variables
totalDist = 0.0 # Total distance traveled
prevXPos = -1.0 # X position of mouse in previous frame
prevYPos = -1.0 # Y position of mouse in previous frame

@returns(float)
def process(value):
    global totalDist, prevXPos, prevYPos

    # Bring in inputs
    xPos = value.Centroid.X
    yPos = value.Centroid.Y

    # isnan() check in case no animal is detected
    if math.isnan(xPos) or math.isnan(yPos):
        return totalDist

    # Case for first behavioral camera frame
    if prevXPos < 0 or prevYPos < 0:
        prevXPos = xPos
        prevYPos = yPos

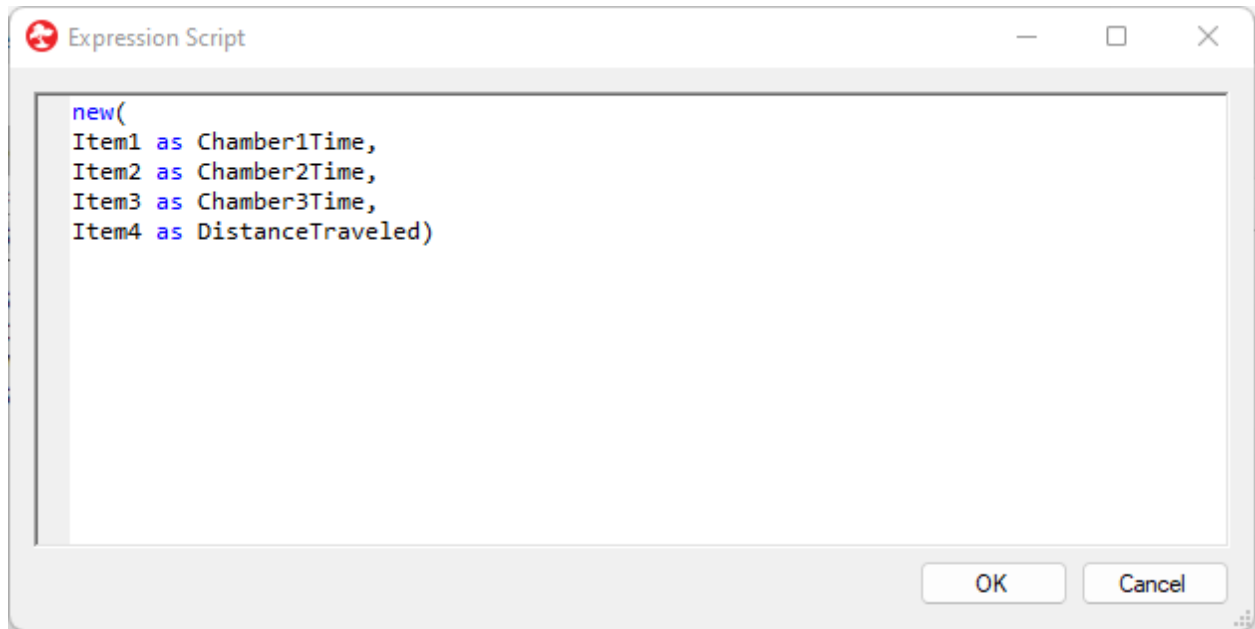
    # Find the distance traveled between previous and current frames
    dx = xPos - prevXPos
    dy = yPos - prevYPos
    dr = math.sqrt(math.pow(dx,2) + math.pow(dy,2))

    # Update the total distance traveled and the previous position variables.
    totalDist += dr
    prevXPos = xPos
    prevYPos = yPos

    # Return the total distance traveled.
    return totalDist
```

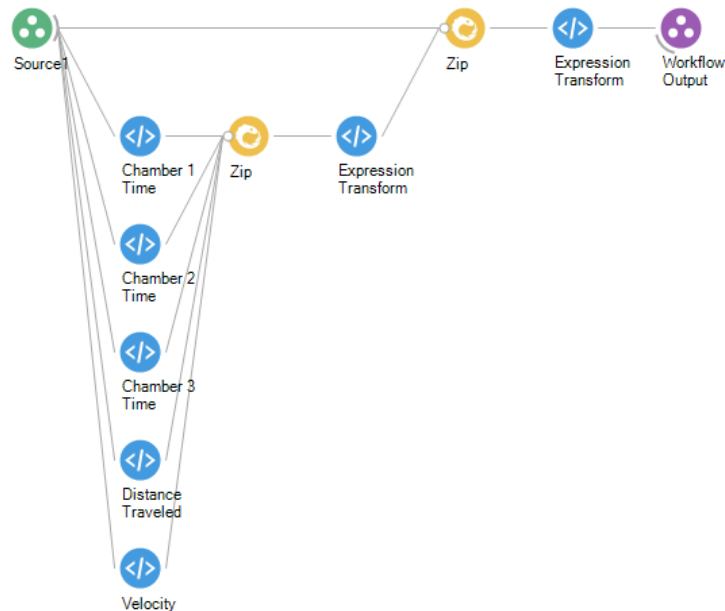
The window has standard window controls (minimize, maximize, close) and buttons for "OK" and "Cancel".

With that, the distance traveled algorithm is complete, be sure to update the “Expression Transform” that names the elements coming from the analysis algorithms.



## Velocity:

Now we will transition to the last algorithm of this section: Velocity. Here we will implement an algorithm that will output the X and Y velocity of the animal as well as the speed of the animal every frame. Begin by inserting a “Python Transform” node parallel to the previous analysis algorithms. Name it something unique such as “Velocity”.



For the velocity algorithm, we will need to bring in the input values containing the X and Y position of the animal as well as the computer timestamp of the current frame. We will also use global variables to contain all of these values for the previous frame.



Python Script



```
# Initialize Global Variables
prevXPos = -1.0 # X position of mouse in previous frame
prevYPos = -1.0 # Y position of mouse in previous frame
prevTime = -1.0 # Timestamp of previous frame

@returns(bool)
def process(value):
    global prevXPos, prevYPos, prevTime

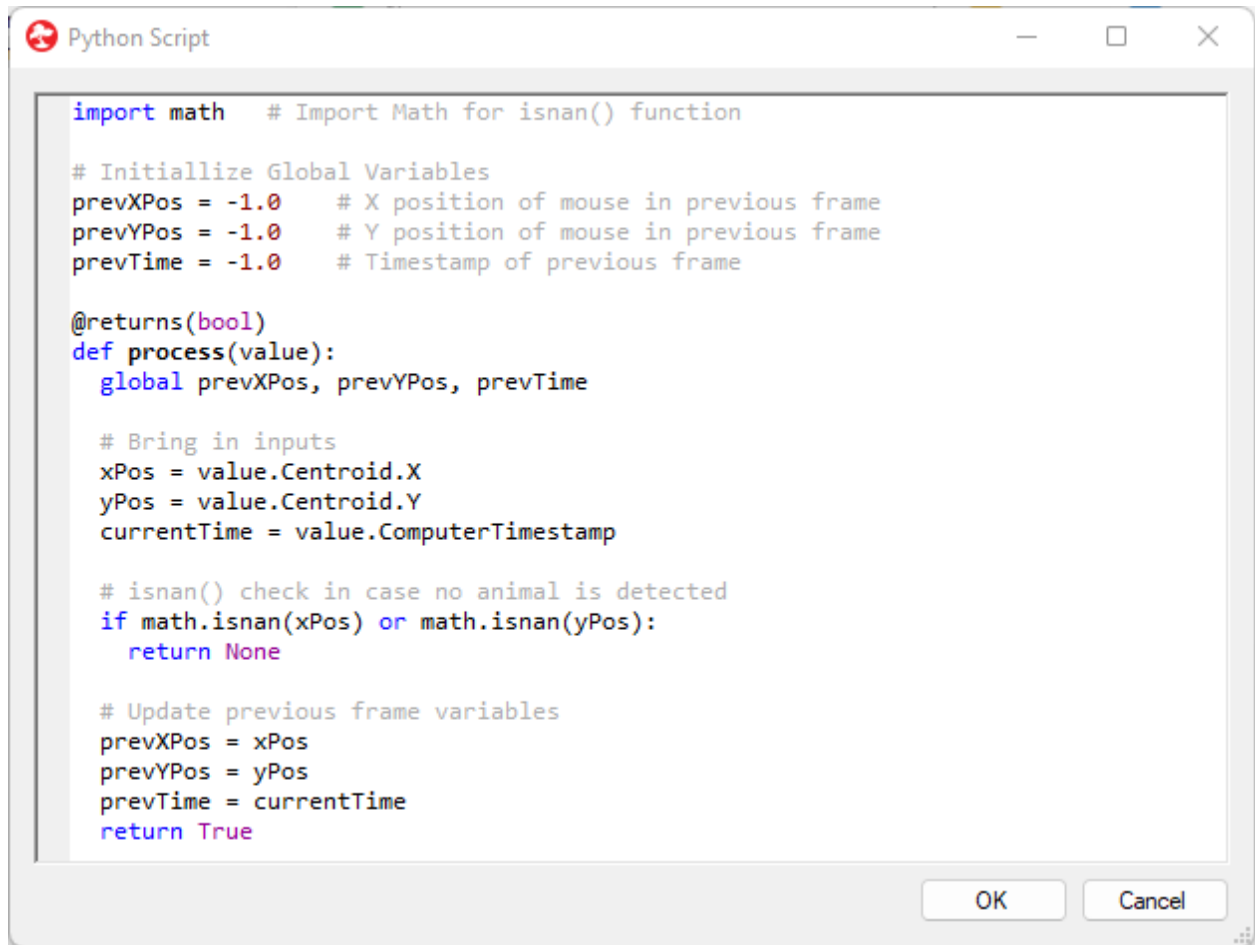
    # Bring in inputs
    xPos = value.Centroid.X
    yPos = value.Centroid.Y
    currentTime = value.ComputerTimestamp

    # Update previous frame variables
    prevXPos = xPos
    prevYPos = yPos
    prevTime = currentTime
    return True
```

OK

Cancel

Next we must handle the edge case where no animal is detected in the enclosure. This can be handled the same way as in the distance traveled algorithm, except it will return a “None” value. Be sure to import the math library again.



```
import math # Import Math for isnan() function

# Initialize Global Variables
prevXPos = -1.0 # X position of mouse in previous frame
prevYPos = -1.0 # Y position of mouse in previous frame
prevTime = -1.0 # Timestamp of previous frame

@returns(bool)
def process(value):
    global prevXPos, prevYPos, prevTime

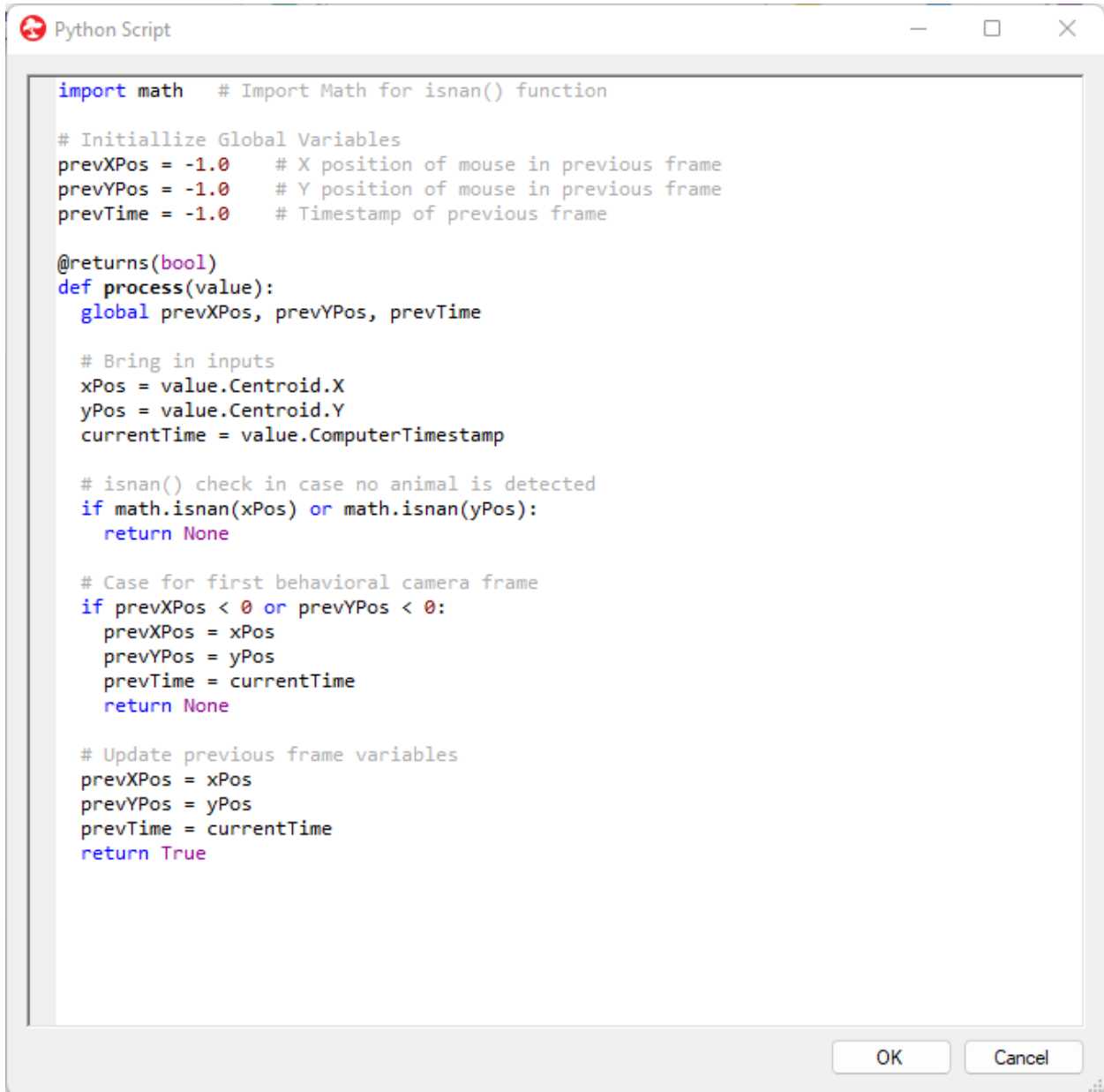
    # Bring in inputs
    xPos = value.Centroid.X
    yPos = value.Centroid.Y
    currentTime = value.ComputerTimestamp

    # isnan() check in case no animal is detected
    if math.isnan(xPos) or math.isnan(yPos):
        return None

    # Update previous frame variables
    prevXPos = xPos
    prevYPos = yPos
    prevTime = currentTime
    return True
```

OK Cancel

Then, handle the edge case for the first behavioral camera frame that the animal is detected. Do this the same way as before, with the addition of updating the previous frame's timestamp. Also, have this return a "None" value.

A screenshot of a 'Python Script' dialog box. The window has a title bar with a Python logo, the text 'Python Script', and standard window controls (minimize, maximize, close). The main area contains Python code for a function named 'process'. The code initializes global variables for mouse position and timestamp, then defines a function that checks for NaN values and handles the first frame case. The dialog has 'OK' and 'Cancel' buttons at the bottom right.

```
import math # Import Math for isnan() function

# Initiallize Global Variables
prevXPos = -1.0 # X position of mouse in previous frame
prevYPos = -1.0 # Y position of mouse in previous frame
prevTime = -1.0 # Timestamp of previous frame

@returns(bool)
def process(value):
    global prevXPos, prevYPos, prevTime

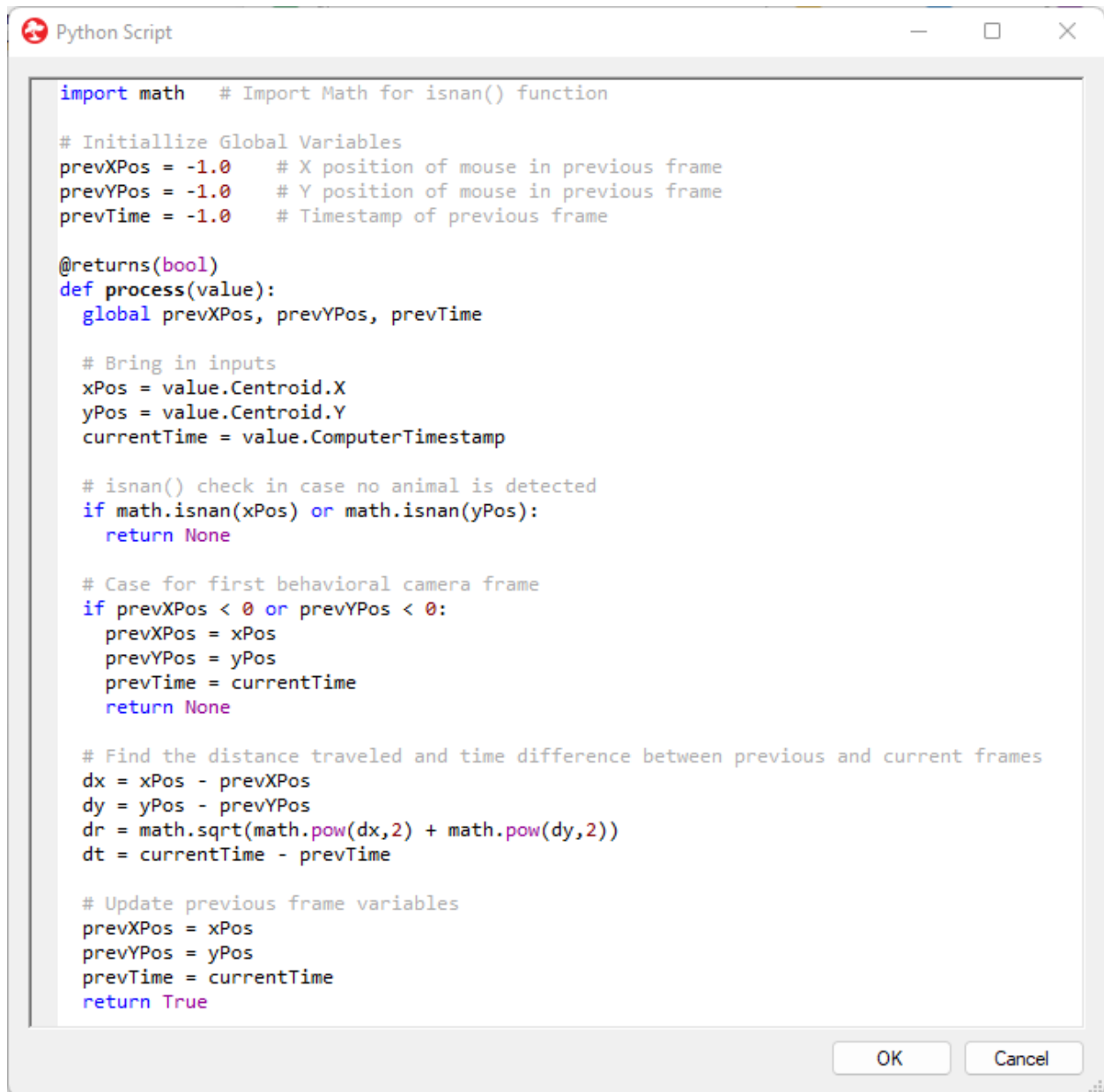
    # Bring in inputs
    xPos = value.Centroid.X
    yPos = value.Centroid.Y
    currentTime = value.ComputerTimestamp

    # isnan() check in case no animal is detected
    if math.isnan(xPos) or math.isnan(yPos):
        return None

    # Case for first behavioral camera frame
    if prevXPos < 0 or prevYPos < 0:
        prevXPos = xPos
        prevYPos = yPos
        prevTime = currentTime
        return None

    # Update previous frame variables
    prevXPos = xPos
    prevYPos = yPos
    prevTime = currentTime
    return True
```

Next, find the distance traveled between the previous frame and the current frame. Also, find the time difference.

A screenshot of a 'Python Script' dialog box. The dialog box has a title bar with a red icon, the text 'Python Script', and standard window controls (minimize, maximize, close). The main area contains Python code for processing mouse position and time data. The code includes imports, variable initialization, a function definition with a decorator, and logic for handling the first frame and calculating distance and time difference. At the bottom right, there are 'OK' and 'Cancel' buttons.

```
import math # Import Math for isnan() function

# Initiallize Global Variables
prevXPos = -1.0 # X position of mouse in previous frame
prevYPos = -1.0 # Y position of mouse in previous frame
prevTime = -1.0 # Timestamp of previous frame

@returns(bool)
def process(value):
    global prevXPos, prevYPos, prevTime

    # Bring in inputs
    xPos = value.Centroid.X
    yPos = value.Centroid.Y
    currentTime = value.ComputerTimestamp

    # isnan() check in case no animal is detected
    if math.isnan(xPos) or math.isnan(yPos):
        return None

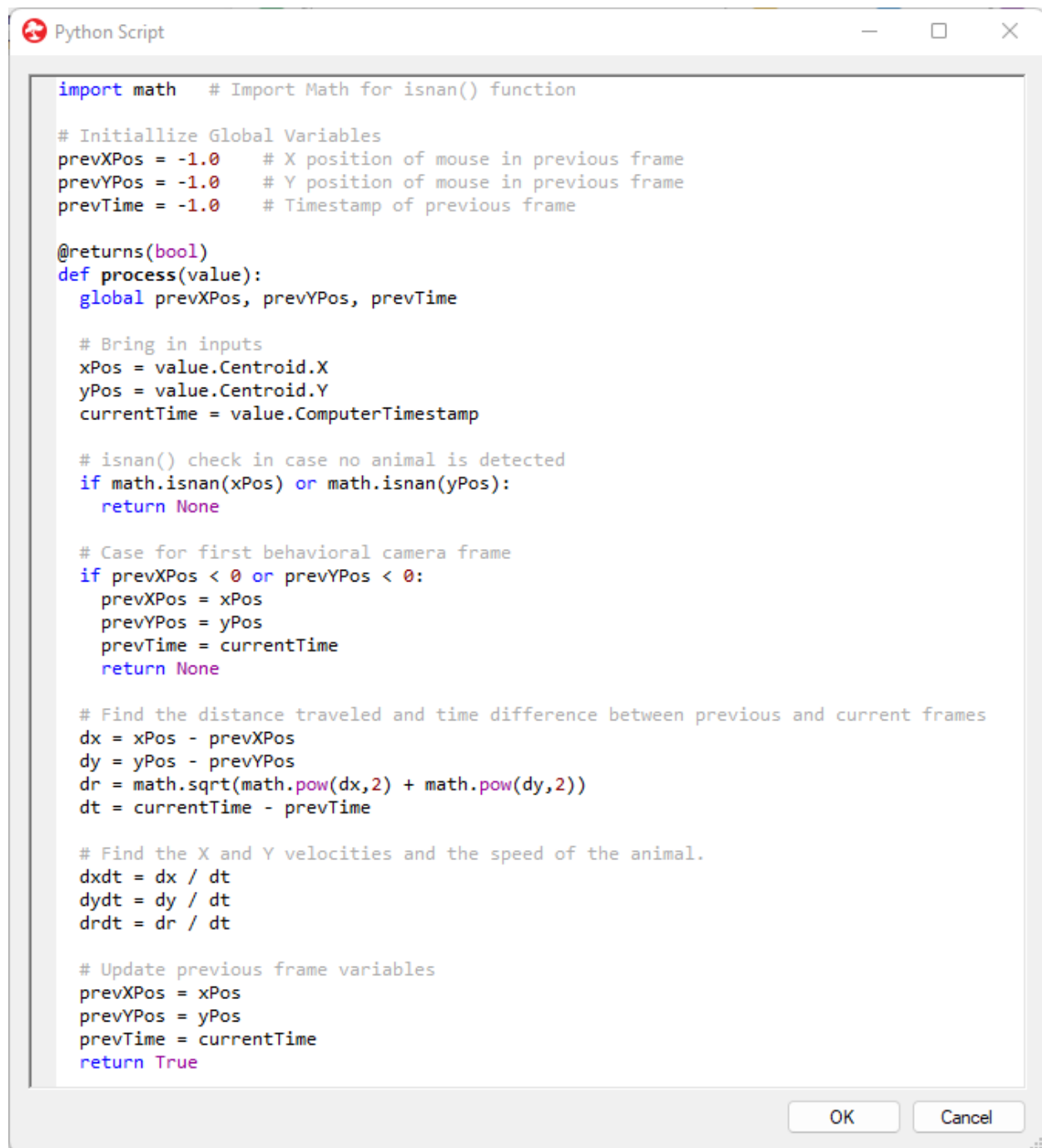
    # Case for first behavioral camera frame
    if prevXPos < 0 or prevYPos < 0:
        prevXPos = xPos
        prevYPos = yPos
        prevTime = currentTime
        return None

    # Find the distance traveled and time difference between previous and current frames
    dx = xPos - prevXPos
    dy = yPos - prevYPos
    dr = math.sqrt(math.pow(dx,2) + math.pow(dy,2))
    dt = currentTime - prevTime

    # Update previous frame variables
    prevXPos = xPos
    prevYPos = yPos
    prevTime = currentTime
    return True
```



To find the X and Y velocity as well as the speed of the animal we need to divide the “dx”, “dy”, and “dr” variables by “dt”.



```
import math # Import Math for isnan() function

# Initialize Global Variables
prevXPos = -1.0 # X position of mouse in previous frame
prevYPos = -1.0 # Y position of mouse in previous frame
prevTime = -1.0 # Timestamp of previous frame

@returns(bool)
def process(value):
    global prevXPos, prevYPos, prevTime

    # Bring in inputs
    xPos = value.Centroid.X
    yPos = value.Centroid.Y
    currentTime = value.ComputerTimestamp

    # isnan() check in case no animal is detected
    if math.isnan(xPos) or math.isnan(yPos):
        return None

    # Case for first behavioral camera frame
    if prevXPos < 0 or prevYPos < 0:
        prevXPos = xPos
        prevYPos = yPos
        prevTime = currentTime
        return None

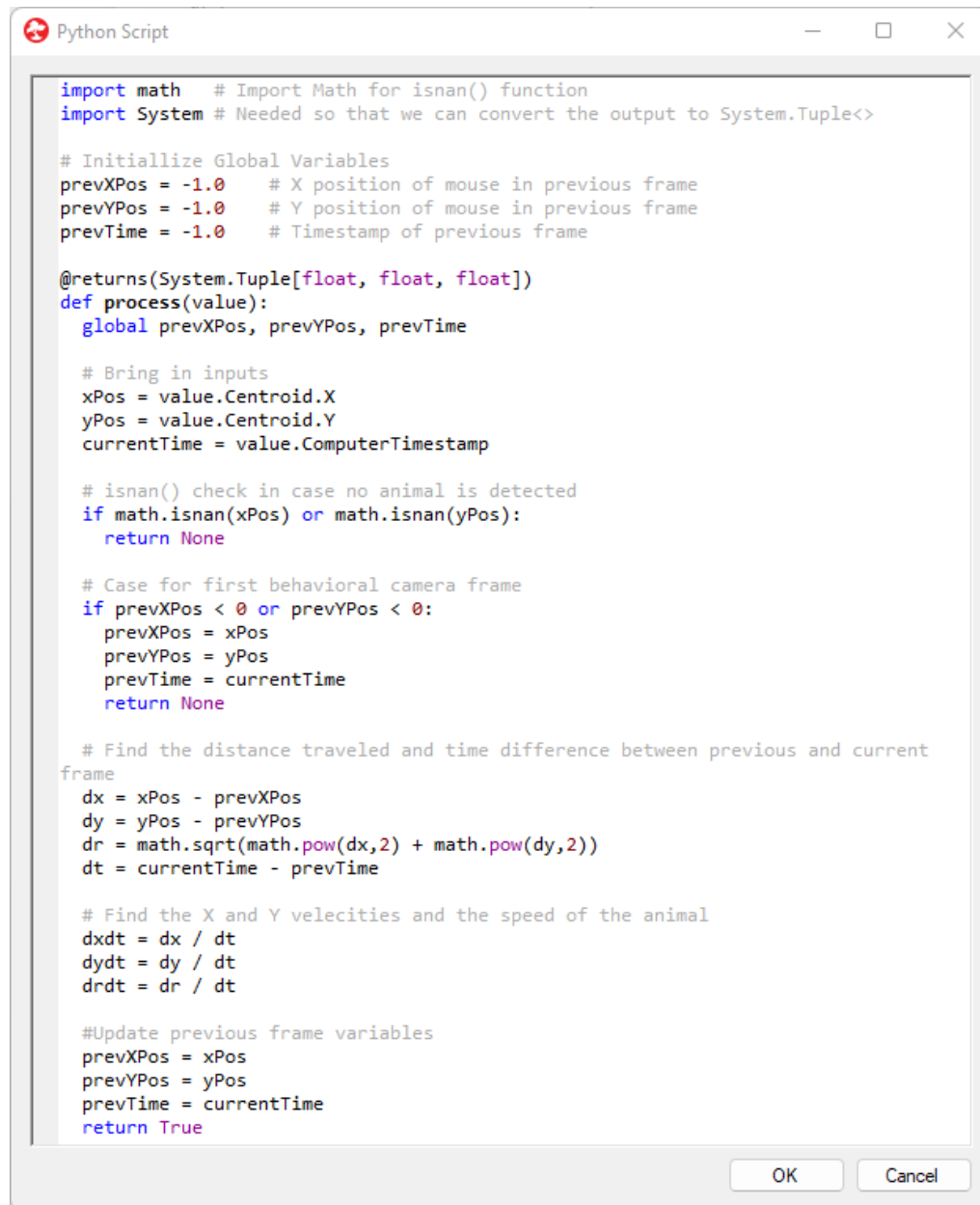
    # Find the distance traveled and time difference between previous and current frames
    dx = xPos - prevXPos
    dy = yPos - prevYPos
    dr = math.sqrt(math.pow(dx,2) + math.pow(dy,2))
    dt = currentTime - prevTime

    # Find the X and Y velocities and the speed of the animal.
    dxdt = dx / dt
    dydt = dy / dt
    drdt = dr / dt

    # Update previous frame variables
    prevXPos = xPos
    prevYPos = yPos
    prevTime = currentTime
    return True
```

The last step is to configure the output of this script. We have three values we want to output: “dxdt”, “dydt”, and “drdt”. These should be outputted as elements of a System.Tuple so that they are easily accessible within Bonsai and easily writable using

a “Csv Writer” node. Begin by importing the system library into the script and specifying the return data type.



```
Python Script

import math # Import Math for isnan() function
import System # Needed so that we can convert the output to System.Tuple<

# Initialize Global Variables
prevXPos = -1.0 # X position of mouse in previous frame
prevYPos = -1.0 # Y position of mouse in previous frame
prevTime = -1.0 # Timestamp of previous frame

@returns(System.Tuple[float, float, float])
def process(value):
    global prevXPos, prevYPos, prevTime

    # Bring in inputs
    xPos = value.Centroid.X
    yPos = value.Centroid.Y
    currentTime = value.ComputerTimestamp

    # isnan() check in case no animal is detected
    if math.isnan(xPos) or math.isnan(yPos):
        return None

    # Case for first behavioral camera frame
    if prevXPos < 0 or prevYPos < 0:
        prevXPos = xPos
        prevYPos = yPos
        prevTime = currentTime
        return None

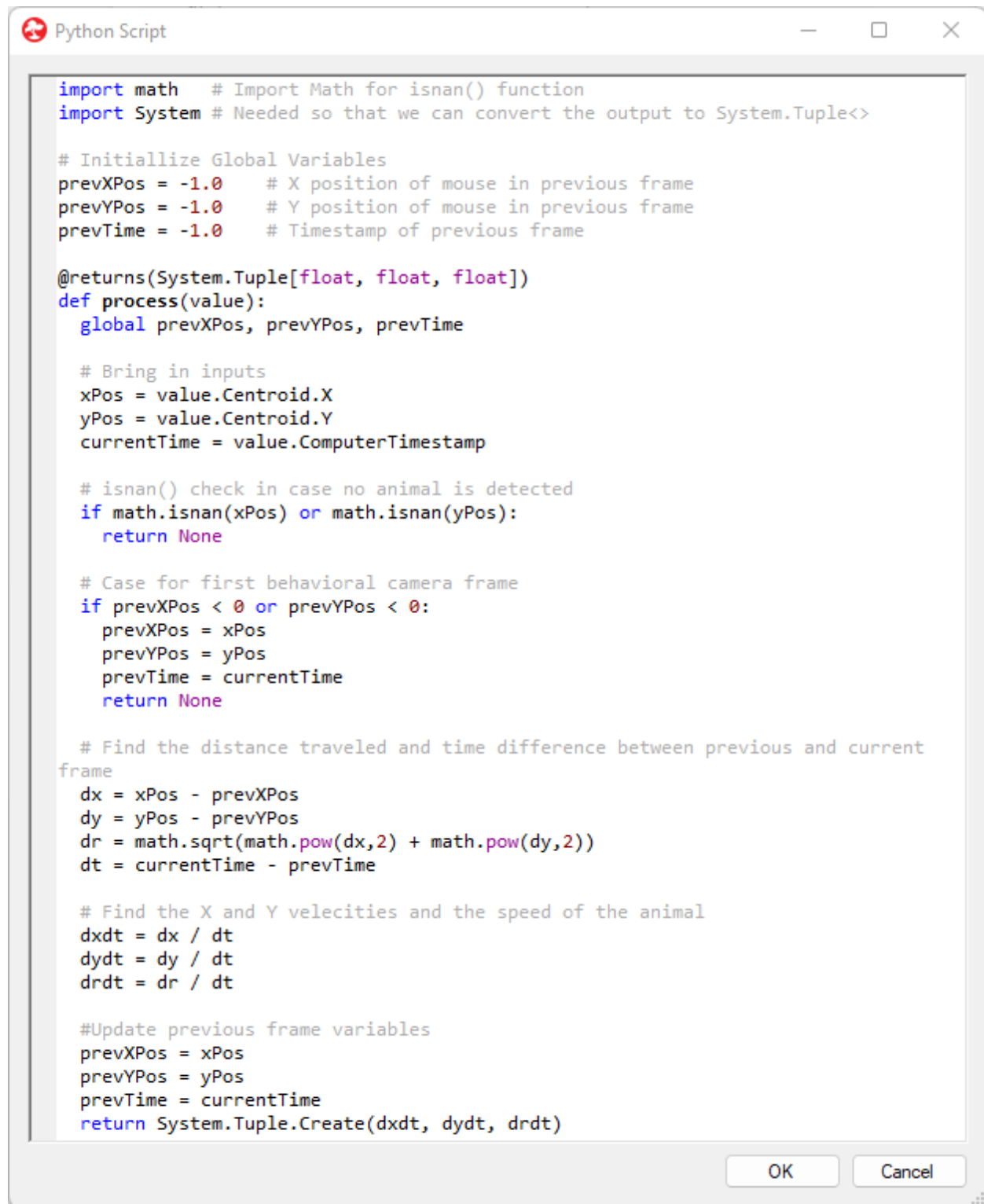
    # Find the distance traveled and time difference between previous and current
    # frame
    dx = xPos - prevXPos
    dy = yPos - prevYPos
    dr = math.sqrt(math.pow(dx,2) + math.pow(dy,2))
    dt = currentTime - prevTime

    # Find the X and Y velocities and the speed of the animal
    dxdt = dx / dt
    dydt = dy / dt
    drdt = dr / dt

    #Update previous frame variables
    prevXPos = xPos
    prevYPos = yPos
    prevTime = currentTime
    return True

OK Cancel
```

Now, specify the return value as “System.Tuple.Create(dxdt, dydt, drdt)”.



```
import math # Import Math for isnan() function
import System # Needed so that we can convert the output to System.Tuple<>

# Initialize Global Variables
prevXPos = -1.0 # X position of mouse in previous frame
prevYPos = -1.0 # Y position of mouse in previous frame
prevTime = -1.0 # Timestamp of previous frame

@returns(System.Tuple[float, float, float])
def process(value):
    global prevXPos, prevYPos, prevTime

    # Bring in inputs
    xPos = value.Centroid.X
    yPos = value.Centroid.Y
    currentTime = value.ComputerTimestamp

    # isnan() check in case no animal is detected
    if math.isnan(xPos) or math.isnan(yPos):
        return None

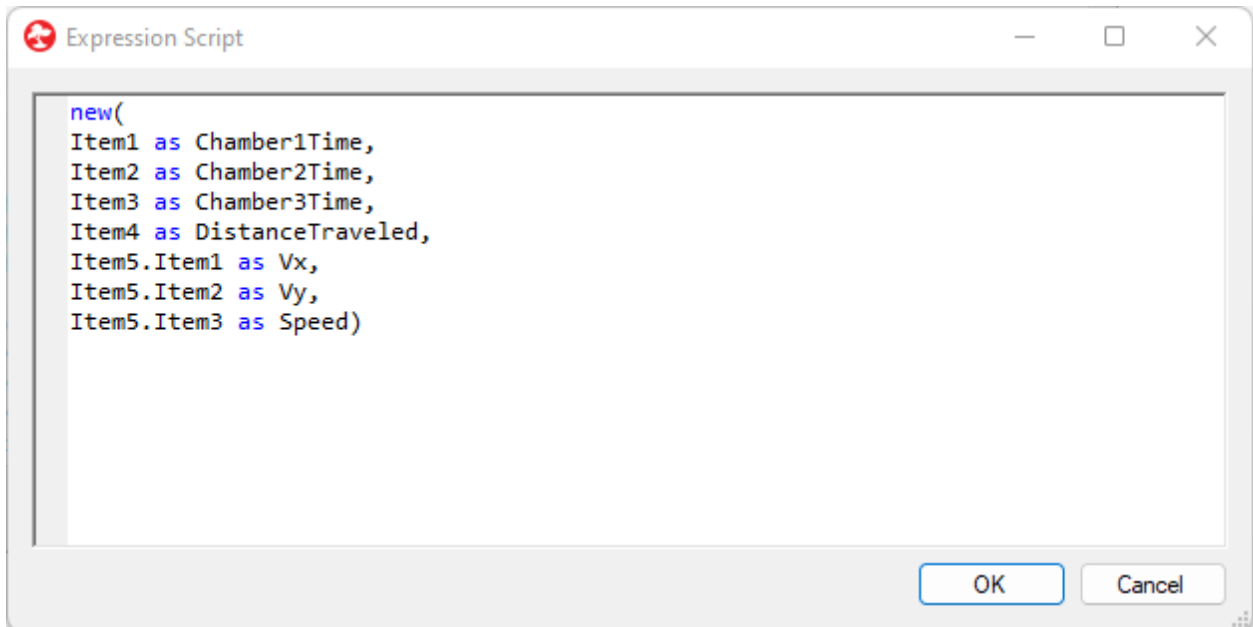
    # Case for first behavioral camera frame
    if prevXPos < 0 or prevYPos < 0:
        prevXPos = xPos
        prevYPos = yPos
        prevTime = currentTime
        return None

    # Find the distance traveled and time difference between previous and current
    # frame
    dx = xPos - prevXPos
    dy = yPos - prevYPos
    dr = math.sqrt(math.pow(dx,2) + math.pow(dy,2))
    dt = currentTime - prevTime

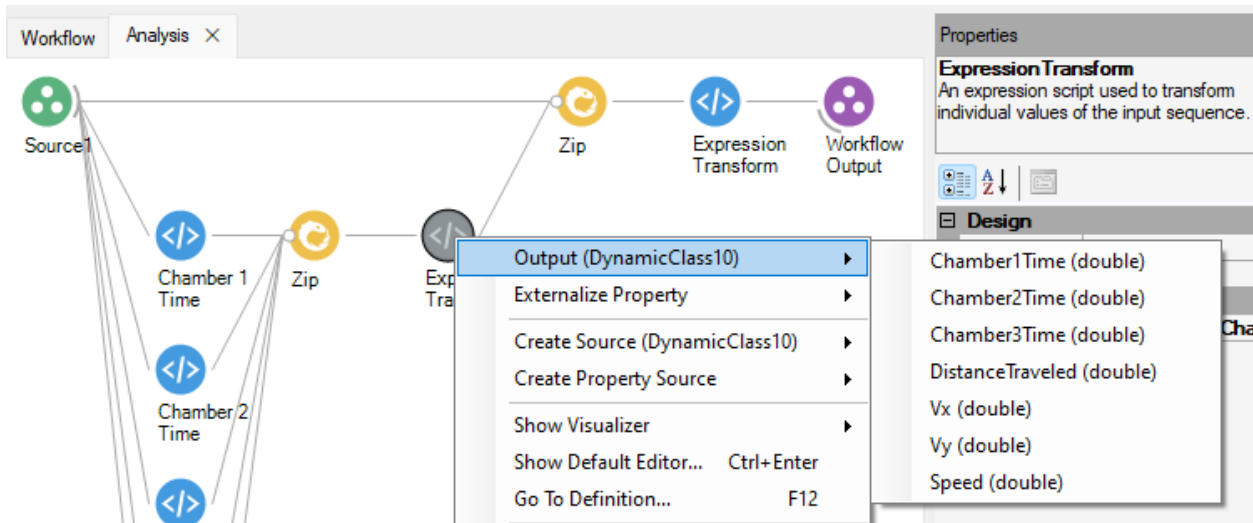
    # Find the X and Y velocities and the speed of the animal
    dxdt = dx / dt
    dydt = dy / dt
    drdt = dr / dt

    #Update previous frame variables
    prevXPos = xPos
    prevYPos = yPos
    prevTime = currentTime
    return System.Tuple.Create(dxdt, dydt, drdt)
```

With that, the velocity algorithm is complete. Be sure to update the “Expression Transform” such that each element of the velocity algorithm has an informative name.



With the “Expression Transform” written in this way, the output will be flattened so that there is not a Tuple contained within the output Tuple.



# Appendix I: Node Glossary

---

## Acquisition Control

The “Acquisition Control” node generates Harp messages that can be sent to the FP3002 system through the “FP3002” node. These messages work to command the system to start and/or stop data acquisition. This node has two configurable properties: “Mode” and “Streams”. The “Mode” property dictates whether the node will generate a “Start” command, “Stop” command, or a “Control” command. Meanwhile, the “Streams” property dictates which acquisition stream this command is going to, either the “Photometry” or the “External Camera” stream. Currently, the only acquisition stream that can be commanded is the “Photometry” stream.



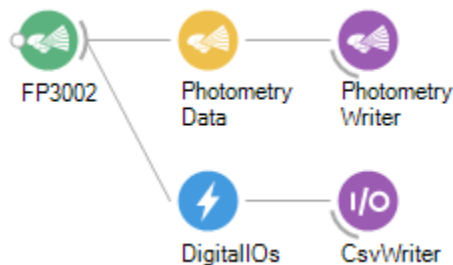
The “Mode” property can affect the type of input that this node can accept. When the “Mode” is set to “Start” or “Stop”, any data type can be sent to the input of the “Acquisition Control” node. When this occurs, the node will output the “Start” or “Stop” command every time the node accepts an input. Meanwhile, if the “Mode” property is set to “Control”, the node can only accept boolean values (True/False). When the node accepts a True value, it will output the “Start” command, and when the node accepts a False value, it will output the “Stop” command.

This node can be used in a variety of experimental designs and examples of its implementation can be found throughout the [“Data Acquisition”](#) chapter.

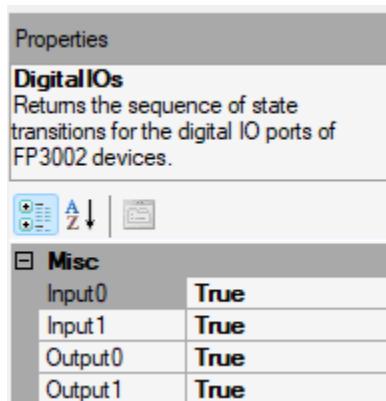
# Digital IOs

The “Digital IOs” node is used to record and timestamp the signals on the digital input and output ports on the FP3002 system. The timestamp generated with this node uses the internal clock of the FP3002 system so the data recorded by this node will already be aligned to the photometry data saved within the “Photometry Writer” node.

This node accepts the output of the “FP3002” node as its input. It processes the harp messages coming from the “FP3002” node, filtering out all messages not related to digital port states. Then it outputs data that is readily writable to a .csv file using the “Csv Writer” node.

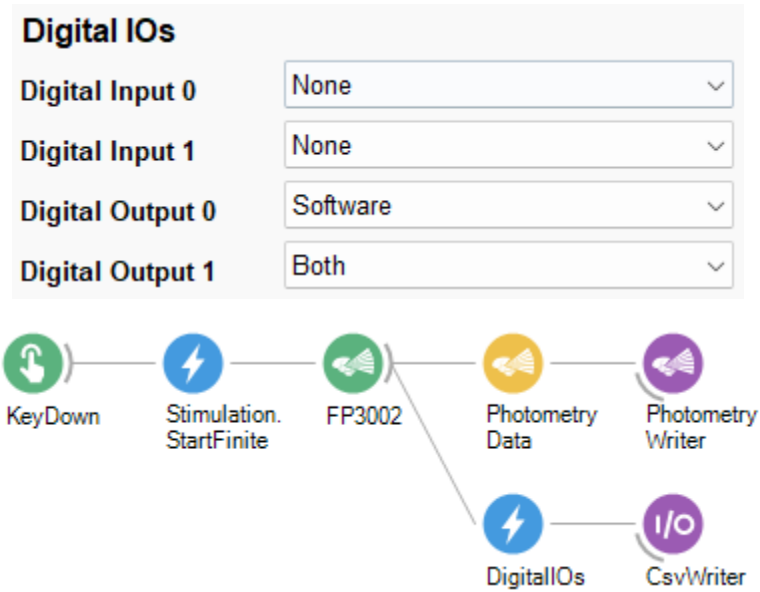


The “Digital IOs” node has four properties for specifying which Digital IO ports are being recorded. This node outputs data of type “DigitalIODataFrame”. This data type is readily writable to storage so a “Csv Writer” can be connected directly after the node, saving a .csv file with five columns containing the port name, flag, state, system timestamp, and computer timestamp.



One common use for this node is to provide precise timestamps of the laser ON/OFF state. First, be sure that the “Digital Output 1” property within the “FP3002 Setup” window is set to “Both”. This will ensure that the internal signal used to control the laser state will be sent to both the laser and the Digital Output 1 port. Then configure the

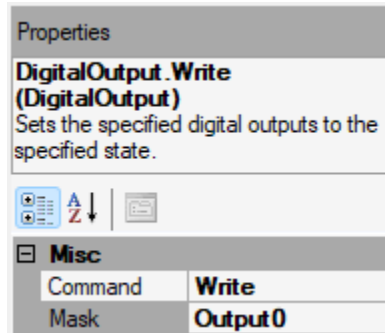
“Digital IOs” node to have the “Output1” property set to “True”. When the output of the “Digital IOs” node is connected to a “Csv Writer” it will save the laser state with timestamps using the system’s internal clock.



# Digital Output

The “Digital Output” node is used to generate Harp Messages to be sent to the FP3002 system using the “FP3002” node. These Harp Messages are mostly used to command the FP3002 system to set the Digital Output ports to a particular state. However, this can also be used to manually specify the state of the LEDs, Internal Camera Trigger, and Camera GPIO Lines.

This node has two properties to specify: “Command” and “Mask”. The “Command” property will specify the type of command contained within the Harp Message. The options are to “Set”, “Clear”, “Toggle”, or “Write” commands. The “Mask” property will dictate which signal within the system we are controlling. Most of the time, we will specify this property to “Output 0” to control the digital output 0 port.



Below are descriptions of each “Command” and “Mask” property:

## **Commands:**

Set - Sets the internal signal to HIGH

Clears - Clears the internal signal to LOW

Toggle - Toggles the internal signal to its opposite state.

## **Commands:**

Output 0/1 - Specifies the internal signal to be one of the digital output ports.

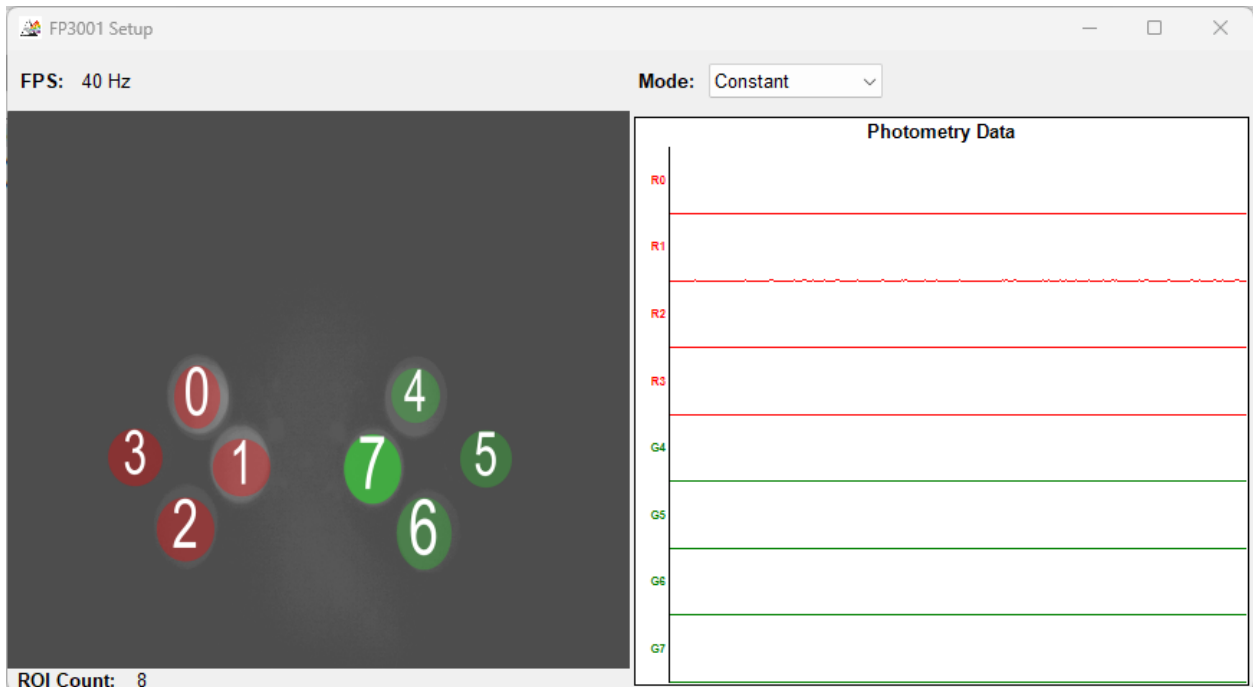


# **FP3001**

The “FP3001” node is a source node used to communicate with the FP3001 system. This node processes the information coming from the FP3001 system and generates photometry data frames to represent the data. Each photometry data frame contains an image, frame counter, system timestamp, computer timestamp, frame flags, and activity data. The “FP3001” node possesses the following properties:

**SerialNumber:** The serial number of the internal camera on the FP3001 system. The drop-down menu is automatically populated with the serial numbers of every FLIR camera found connected to the computer. If other FLIR cameras are connected to the computer (i.e. FLIR behavioral cameras, FP3001 or FP3002 systems), ensure that you are selecting the correct serial number.

Once the internal camera’s serial number is specified, double click the FP3001 node to open a calibration window. Begin data acquisition on the driver box and the calibration window should populate with images from the camera and a running plot of photometry data. There will be a signal for each ROI specified, and if none are specified the signal will represent the pixel average of the whole image.



Begin by ensuring that the FP3001 system is in focus, such that the ends of the patch cord are visible. Then, there are three properties to specify:

**FPS**: Ensure that the “FPS” property in the “FP3001 Setup” window matches the “FPS” set on the driver box. This removes the need to specify the camera’s exposure time in previous versions of the software by automatically calculating the maximum exposure time based on the “FPS”.

**Mode**: Ensure that the “Mode” property in the “FP3001 Setup” window matches the mode specified on the driver box. This property handles how frame flags are assigned to new photometry data frames coming from the FP3001 system.

**Regions**: Draw regions of interest in the desired channels and over the desired fibers of the patch cord. Below are the controls for interacting with the regions of interest in this window:

**Left-Click + Move**: If no region of interest is selected, this will draw a new region of interest. If a region of interest is selected, then this will move the region to the desired location.

**Right-Click + Move**: If a region of interest is selected, this will re-size the region to the desired dimensions.

**Left-Click**: Selects a region of interest.

**Tab**: Iteratively selects a region of interest.

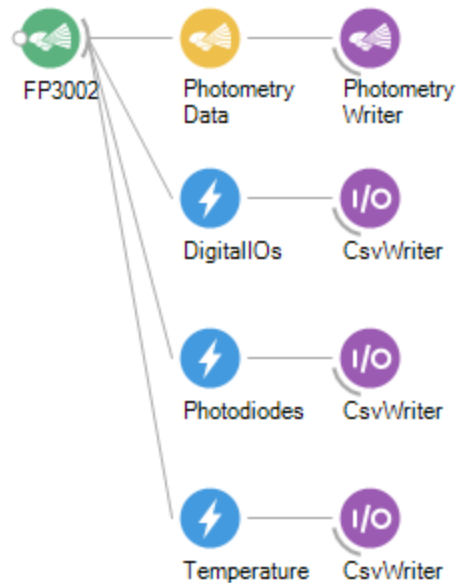
**Del**: Deletes the selected region of interest.

**CTRL**: When drawing or resizing a region, this will ensure that the region encompasses a circular region of pixels on the camera.

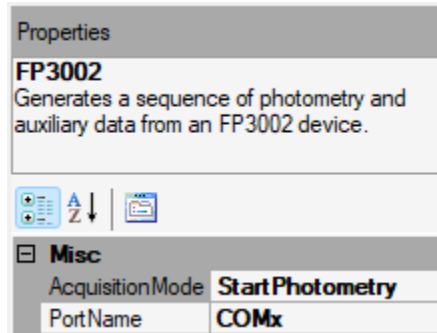
Whenever drawing ROIs, double check that the “ROI Count” in the bottom left of the “FP3001 Setup” window matches the desired number of drawn ROIs. If there are more signals than visible ROIs, then a small ROI has been accidentally drawn. You can correct this by using the “Tab” key within the “Regions” window to cycle through ROIs to select the unintended ROI. Then press the “Del” key to delete the extra ROI.

## **FP3002**

The “FP3002” node is a source node used to communicate with the FP3002 system. This node sends commands to the system to control its functionality and receives the data generated by the system. This node generates data of type “Bonsai.Harp.HarpMessage”. This data type utilizes the Harp protocol for communicating with embedded systems and requires particular nodes for processing the data contained by these Harp Messages. For example, at a user defined frequency, the FP3002 system will output photometry data in the form of Harp Messages. These messages are then processed by the “Photometry Data” node to produce data that is easier to visualize and write to storage. This same protocol is used to read in system temperature, photodiode measurements, and digital IO states from the system.



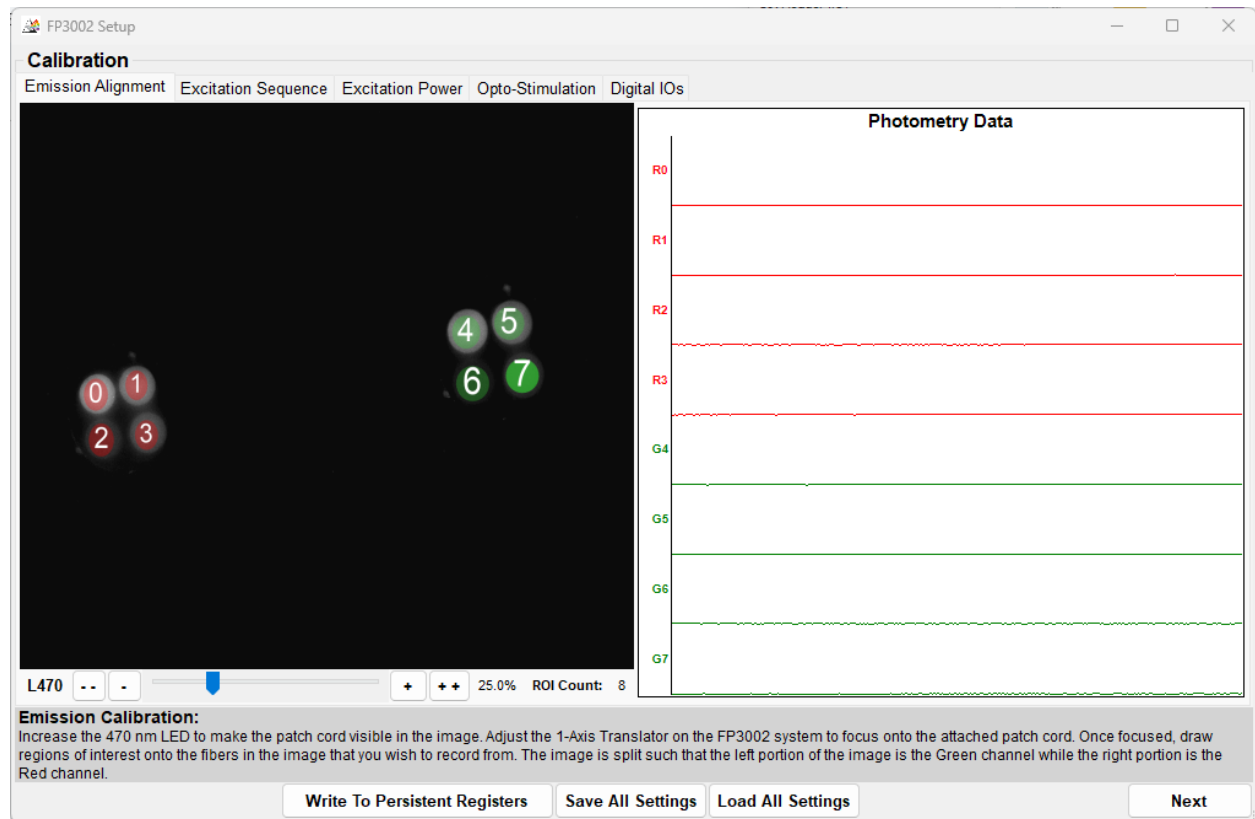
The “FP3002” node possesses the following properties within its property panel:



**Acquisition Mode:** A dropdown menu used to specify the initial state of data acquisition when the Bonsai workflow starts. When the workflow is started this is the first command that the “FP3002” node sends to the system. In most cases this command will be to either start or stop the photometry data acquisition. However, you can also use this to start or stop an external camera connected to the FP3002 system.

**Port Name:** A dropdown menu used to select the port that the FP3002 system is connected to. When selecting the correct port, the system information will populate the Bonsai command window.

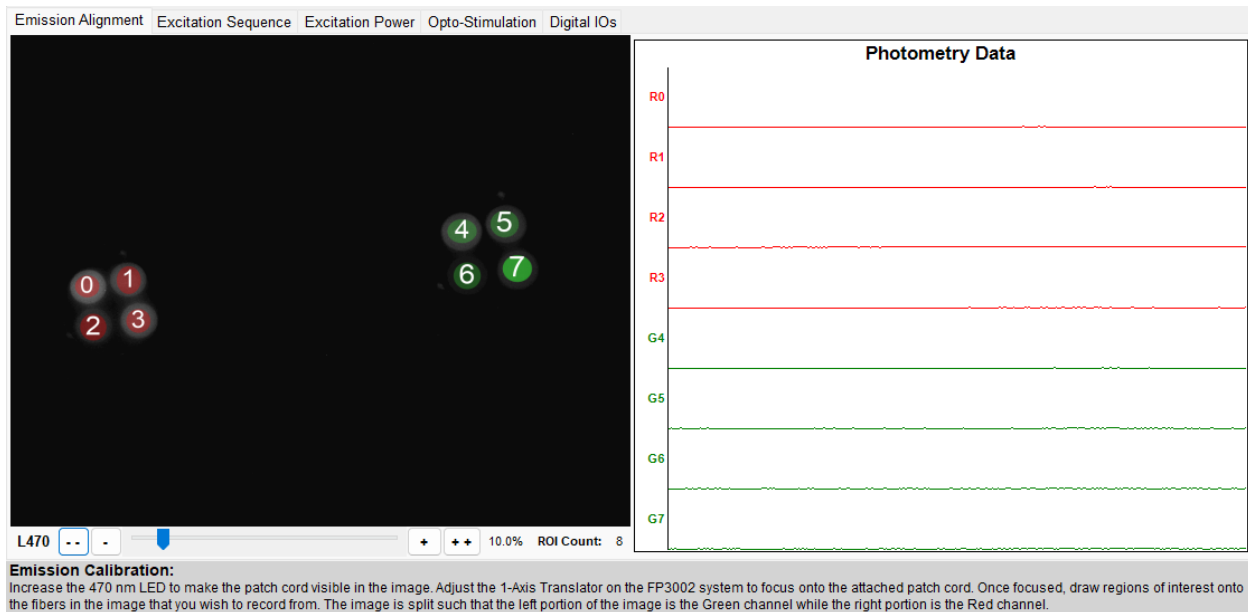
Once the “Port Name” property is specified, double click the FP3002 node to open the “FP3002 Setup” window.



There are five tabpages within the “FP3002 Setup” window, helping with different aspects of configuring the FP3002 system: “Emission Alignment”, “Excitation Sequence”, “Excitation Power”, “Opto-Stimulation”, and “Digital IOs”. At the bottom of the window are buttons for navigating the setup window, saving/loading settings, and writing settings to persistent registers on the system.

## **Emission Alignment:**

The “Emission Alignment” tabpage is broken up into two parts. The left side consists of a video feed from the internal camera as well as a slider for controlling the power of the 470nm LED. The right side consists of the photometry data coming from each user-defined region of interest. This tabpage is used to align the patch cord and define regions of interest. These regions of interest will be used to indicate to the “FP3002” node which pixels are grouped together for averaging.



To begin the emission alignment, adjust the 470nm LED using the “L470” slider to increase visibility. Then adjust the 1-Axis translator on the FP3002 system, until the fibers of the patch cord are visible. Then uses the controls below to draw and adjust the regions of interest:

Left-Click + Move: If no region of interest is selected, this will draw a new region of interest. If a region of interest is selected, then this will move the region to the desired location.

Right-Click + Move: If a region of interest is selected, this will re-size the region to the desired dimensions.

Left-Click: Selects a region of interest.

Tab: Iteratively selects a region of interest.

Del: Deletes the selected region of interest.

CTRL: When drawing or resizing a region, this will ensure that the region encompasses a circular region of pixels on the camera.

Whenever drawing regions of interest, double check that the “ROI Count” below the image matches the desired number of drawn regions of interest. If there are more signals than visible ROIs, then a small ROI has been accidentally drawn. You can correct this by using the “Tab” key within the “Regions” window to cycle through ROIs to select the unintended ROI. Then press the “Del” key to delete the extra ROI.

**Excitation Sequence:**

The “Excitation Sequence” tabpage is used specify the frame rate of data acquisition as well as the cycle of excitation LEDs used during data acquisition.

Emission Alignment   Excitation Sequence   Excitation Power   Opto-Stimulation   Digital IOs

Index	LED	Flag
0	470 nm	2
1	560 nm	4
2	415 nm	1

Frame Rate: -- - + ++ 16.0 Hz

**Excitation Sequence Calibration:**  
Set the order of the LEDs in a given trigger sequence. Their associated “Frame Flag” values will appear next to the LED names.

In order to calibrate the excitation sequence, use the table on the left to specify which LEDs will be used and the order of the LEDs in the sequence.

Index	LED	Flag
0	470 nm	2
1	560 nm	4
2	415 nm	1
	None	

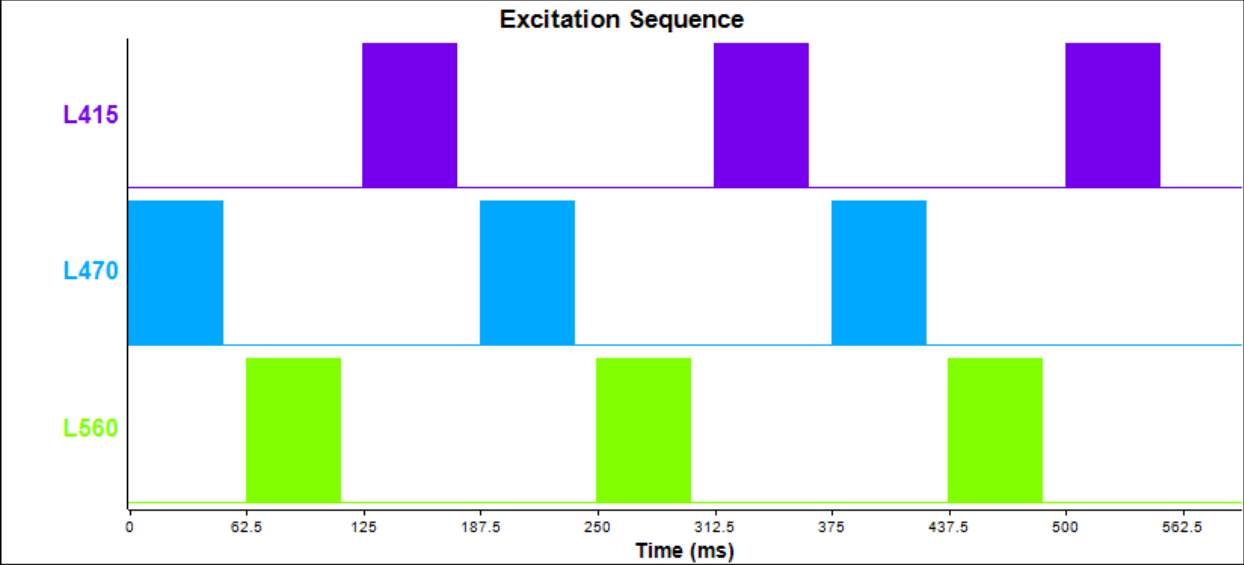
None  
415 nm  
470 nm  
560 nm

Remove Selected

To change an LED, select it, click the drop-down arrow, and select the new desired LED. To add a new LED, select the bottom empty LED cell, click the drop-down arrow, and select the desired LED. To remove an LED, select it, and click the “Remove Selected” button.



After configuring the table on the left, specify the frame rate at the top of the tabpage. Whenever the frame rate is changed or an LED is changed, added, or removed, the "Excitation Sequence" plot on the right will be updated.



## **Excitation Power:**

The “Excitation Power” tabpage is used in conjunction with a power meter to select the appropriate LED power percentage for an experiment. The power coming out of a single fiber of a patch cord should be high enough to record activity and low enough to limit photobleaching of the region of the brain that is being observed. Generally, it is recommended that the power coming out of the ferrule is to be approximately 50 $\mu$ W for 200 $\mu$ m fibers and approximately 120 $\mu$ W for 400 $\mu$ m fibers, to start. Whenever possible, use the lowest light powers possible. This will damage the tissue less and increase longevity of the experiment. These recommendations are valid for experiments under one hour. For longer experiments, consider lowering the duty cycle of the LEDs and/or lowering the LED powers.

The screenshot displays the 'Excitation Power' tab within a software application. At the top, there are five tabs: 'Emission Alignment', 'Excitation Sequence', 'Excitation Power', 'Opto-Stimulation', and 'Digital IOs'. The main area contains three rows of controls for different LED wavelengths: 415 nm, 470 nm, and 560 nm. Each row features a horizontal slider bar, a power percentage display (currently at 0.0%), and an 'Edit' button. To the left of each slider are three small input fields containing dashes. Below the interface, a section titled 'Excitation Power Calibration:' provides instructions: 'Adjust the LED Power percentage while measuring the power output from the end of the fiber. Once the desired power is found for a given LED, click "Accept" to store the value into settings.'

Calibrate each LED by clicking the “Edit” button, adjusting the amplitude until the desired output power is reached, then clicking the “Accept” button to turn off the LED.

**Excitation Power Calibration:**  
Adjust the LED Power percentage while measuring the power output from the end of the fiber. Once the desired power is found for a given LED, click "Accept" to store the value into settings.

## Opto-Stimulation

The “Opto-Stimulation” tabpage allows for the configuration of a finite or continuous length laser pulse train. It also assists in the measuring of the output power and the alignment of the 635nm laser.

**Opto-Stimulation Calibration:**  
To enable Opto Stim, insert the laser key into the stub wall of the system, and turn clockwise. Then select the desired wavelength from the drop down menu and connect the white ribbon cable inside the system to the corresponding laser.

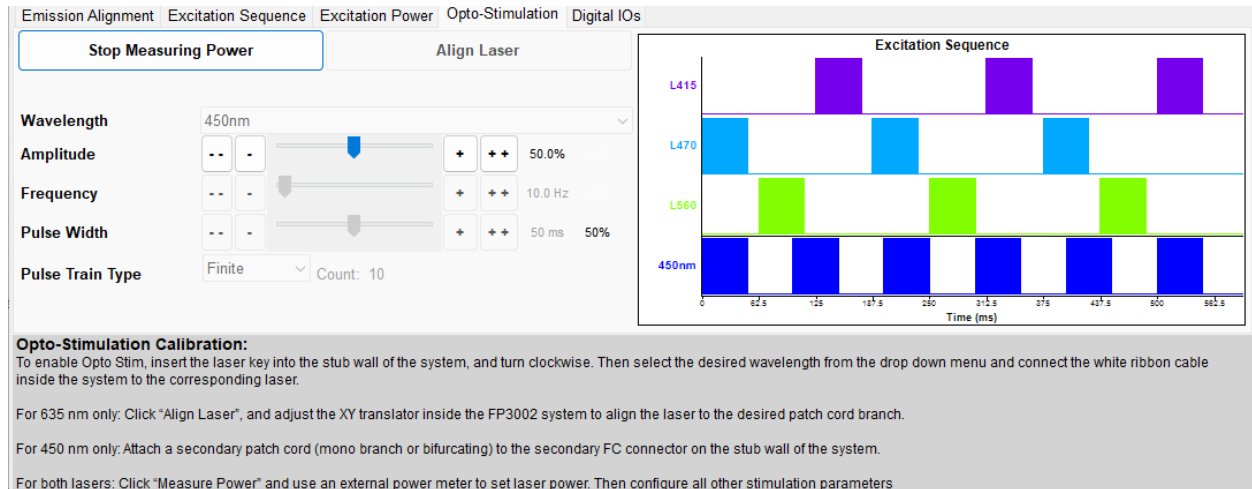
For 635 nm only: Click “Align Laser”, and adjust the XY translator inside the FP3002 system to align the laser to the desired patch cord branch.

For 450 nm only: Attach a secondary patch cord (mono branch or bifurcating) to the secondary FC connector on the stub wall of the system.

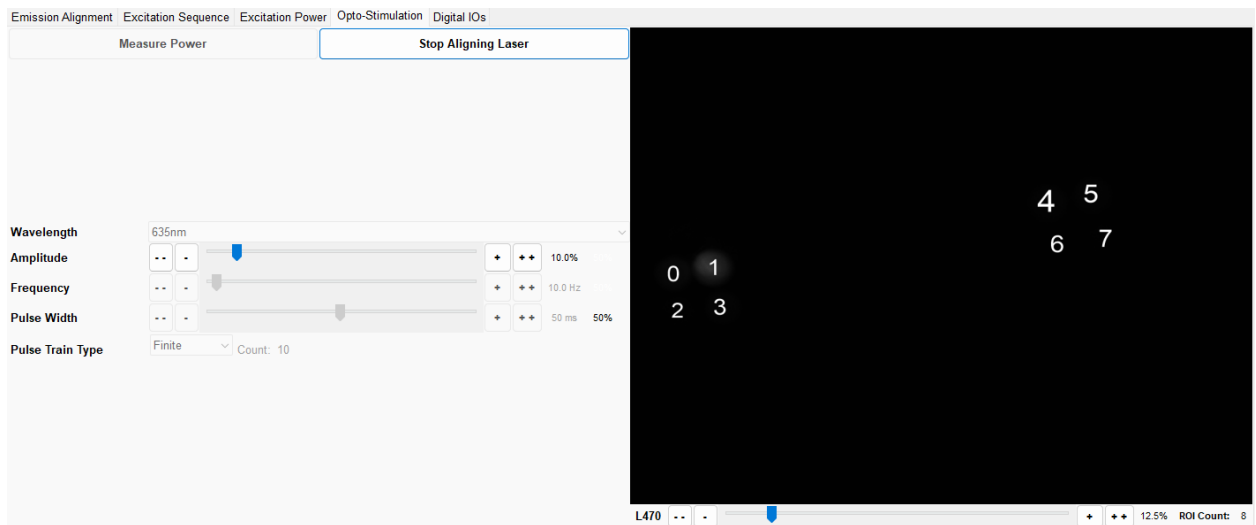
For both lasers: Click “Measure Power” and use an external power meter to set laser power. Then configure all other stimulation parameters

Begin by specifying the “Wavelength” property. If the wavelength is 450nm, then the “Measure Power” button will enable. If the wavelength is 635nm, then both the “Measure Power” and “Align Laser” buttons will be enabled. Additionally, once the “Wavelength” property is specified, then the laser pulse train will appear in the “Excitation Sequence”

**Measure Power:** Upon clicking the “Measure Power” button, the laser will run in constant mode at the specified amplitude. Adjust the “Amplitude” property until the desired output power is reached. Once complete, click the “Stop Measuring Power” button.



**Align Laser:** Upon clicking the “Align Laser” button, a low power, low duty cycle laser pulse train will begin and the image from the camera will appear instead of the “Excitation Sequence” plot. Adjust the 2-Axis translator for the 635nm laser until it is aligned with the desired fiber of the patch cord. You can also adjust the laser amplitude from 0%-30% during this phase, as well as the 470nm LED to make the fibers and laser pulse train more visible.



## Digital IOs

The “Digital IO” tabpage allows users to configure the functionality of the Digital IO ports on the FP3002 system.

**Digital IOs**

Digital Input 0: Event: Rising

Digital Input 1: Event: Rising

Digital Output 0: Software

Digital Output 1: Both

**Miscellaneous**

Screen Brightness: [Slider]

Clock Configuration: This Device

**Digital Inputs**  
**Events:** Samples a +5V TTL signal, generating timestamps on the specified type of voltage change.  
**Control:** Allows a +5V TTL signal to control data acquisition of the FP3002 system and/or an external camera.  
**Start Stimulation:** Allows a +5V TTL signal to start a finite or continuous pulse train.

**Digital Output 0**  
**Software:** The system will output a +5V TTL signal generated within Bonsai, using the Digital Output Node.  
**Strobe:** Outputs a +5V TTL signal that will be HIGH while the camera is exposing, and LOW between frames.  
**Trigger State:** Outputs a +5V TTL will be HIGH while an LED is ON and LOW while an LED is OFF.

**Digital Output 1**  
**Both:** Internal signal used to control the laser will be sent to both the laser and the Digital Output 1 port.  
**BNC:** Internal signal will be sent to the Digital Output 1 port only. Timing can be configured by changing laser settings.  
**Laser:** Internal signal will be sent to the laser only.

**Digital IOs Calibration:**  
Digital IOs- Use drop down menus to configure how the system will send and receive TTL pulses. Note: Digital Output 1 is yoked to laser function and  
Screen Brightness- Use slider to change LCD screen brightness.  
Clock Configuration- Use drop down menu to select the master clock to use for photometry data. The system will default to the internal clock. If using an external device as the master clock, set clock configuration to 'External Device'.

“Digital Input 0/1”: Specifies how the FP3002 system handles +5V TTL signals on the Digital Input ports. There are a total of 12 options for configuring these input ports.

**Digital IOs**

Digital Input 0: Event: Rising

Digital Input 1: Event: Rising

Digital Output 0: Software

Digital Output 1: Both

**Miscellaneous**

Screen Brightness: [Slider]

Clock Configuration: This Device

Event: Rising

None

Event: Rising

Event: Falling

Event: Change

Control: Trigger

Control: External Camera

Control: External Camera with Events

Control: Trigger and External Camera

Control: Trigger and External Camera with Events

Start Stimulation: Finite

Start Stimulation: Continuous

**None:** Nothing will occur when there is a TTL signal on the Digital Input port.

**Event Rising:** Sends a Harp Message to Bonsai indicating an event occurred whenever the +5V TTL signal changes from LOW to HIGH.

**Event Falling:** Sends a Harp Message to Bonsai indicating an event occurred whenever the +5V TTL signal changes from HIGH to LOW.

**Event Change:** Sends a Harp Message to Bonsai indicating an event occurred whenever the +5V TTL signal changes from LOW to HIGH or HIGH to LOW.

**Control Trigger:** Allows the +5V TTL signal to control data acquisition of the FP3002 system. While the TTL signal is HIGH, the system will be acquiring data frames and while the TTL signal is LOW, the system will stop acquiring data frames.

**Control External Camera:** Allows the +5V TTL signal to control data acquisition of an external camera. While the TTL signal is HIGH, the external camera will be acquiring data frames and while the TTL signal is LOW, the external camera will stop acquiring data frames.

**Control External Camera Events:** Allows the +5V TTL signal to control data acquisition of an external camera. While the TTL signal is HIGH, the external camera will be acquiring data frames and while the TTL signal is LOW, the external camera will stop acquiring data frames. In addition, causes the FP3002 system to send Harp events to Bonsai with every external camera exposure.

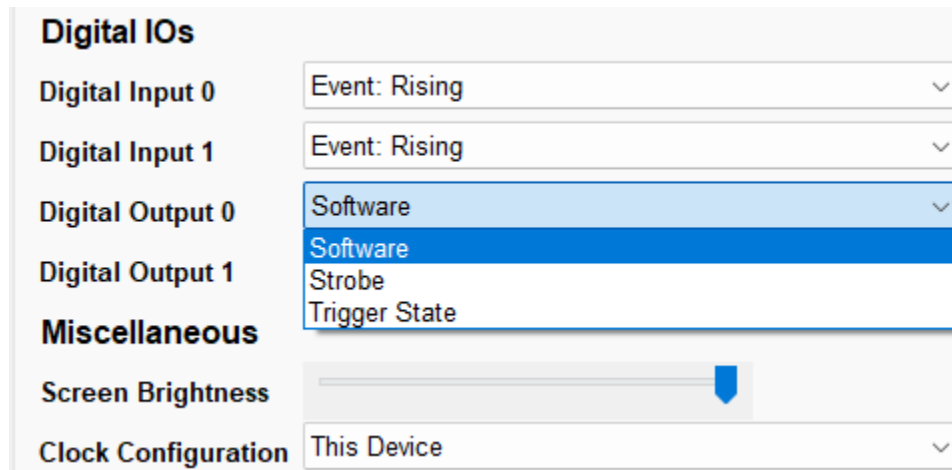
**Control Trigger and External Camera:** Allows the +5V TTL signal to control data acquisition of both the FP3002 system and an external camera. While the TTL signal is HIGH, the FP3002 system and the external camera will be acquiring data frames and while the TTL signal is LOW, the FP3002 system and the external camera will stop acquiring data frames.

**Control Trigger and External Camera Events:** Allows the +5V TTL signal to control data acquisition of both the FP3002 system and an external camera. While the TTL signal is HIGH, the FP3002 system and the external camera will be acquiring data frames and while the TTL signal is LOW, the FP3002 system and the external camera will stop acquiring data frames. In addition, it causes the FP3002 system to send Harp events to Bonsai with every change in camera exposure state.

**Start Stimulation Finite:** While the +5V TTL signal is HIGH, the FP3002 system will begin stimulation with a finite duration pulse train. The laser will pulse the number of times specified with the “Pulse Count” property.

**Start Stimulation Continuous:** While the +5V TTL signal is HIGH, the FP3002 system will begin stimulation with a continuous pulse train. The pulse train will possess all of the parameters specified in the “Stimulation Pulse”, except the pulse count will be infinite. This pulse train will continue until the TTL signal returns LOW.

“Digital Output 0” - Specifies how the FP3002 system outputs +5V TTL signals from the Digital Output 0 port. There are three options for the digital output 0 port.



The image shows a configuration interface for digital I/Os. It includes sections for Digital Input 0, Digital Input 1, Digital Output 0, Digital Output 1, Miscellaneous, Screen Brightness, and Clock Configuration. The 'Digital Output 0' dropdown menu is expanded, showing three options: 'Software', 'Strobe', and 'Trigger State'. The 'Software' option is highlighted in blue, indicating it is the selected configuration.

**Software:** Specifies that the digital output signal will be generated within Bonsai using a software trigger and the “Digital Output” node, then sent to the system through the “FP3002” node.

**Strobe:** Specifies that the camera's strobe will be sent out of the digital output 0 port such that the port will have a HIGH value while the internal camera is exposing, and a LOW value during the internal camera's dead time.

**Trigger State:** Specifies that the digital output signal will be HIGH while an LED is ON and LOW while an LED is OFF.

“Digital Output 1” - Specifies whether the digital output pin 1 state is routed to the BNC port, internal laser, or both. In experiments involving opto-stimulation, this property should be set to “Both” so that the laser can be triggered and so that the “Digital IOs” node can be used to record and timestamp the laser state. Otherwise, this property should be set to “BNC” so that the Digital Output 1 port can be used to output a TTL without triggering the internal laser.

<b>Digital IOs</b>	
Digital Input 0	Event: Rising
Digital Input 1	Event: Rising
Digital Output 0	Software
Digital Output 1	Both
<b>Miscellaneous</b>	BNC Port
	Internal Laser
Screen Brightness	Both
Clock Configuration	This Device

“Clock Synchronizer”: Specifies whether the FP3002 system outputs its own clock line, or synchronizes to an external clock. In most experiments this property is set to “ThisDevice”.



## Load / Save:

The “Load All Settings” and “Save All Settings” buttons are located at the bottom of the “FP3002 Setup” window. These buttons will load/save the FP3002 settings from/to an XML file. The XML file itself can be referenced during post-hoc analysis to verify the system settings used during a particular experiment.

```
<?xml version="1.0" encoding="utf-8"?>
<Settings>
  <DeviceInformation>
    <Id>2064-00cd</Id>
    <HardwareVersion>2.0</HardwareVersion>
    <FirmwareVersion>2.6</FirmwareVersion>
    <SerialNumber>205</SerialNumber>
  </DeviceInformation>
  <Configuration>
    <ClockSynchronizer>ThisDevice</ClockSynchronizer>
    <ScreenBrightness>0</ScreenBrightness>
  </Configuration>
  <Emission>
    <Region X="128" Y="503" Width="52" Height="64">R0</Region>
    <Region X="216" Y="491" Width="42" Height="54">R1</Region>
    <Region X="867" Y="395" Width="46" Height="42">G2</Region>
    <Region X="952" Y="373" Width="56" Height="66">G3</Region>
  </Emission>
  <Excitation>
    <TriggerPeriod Reg="9900">101.010 Hz</TriggerPeriod>
    <Sequence>
      <LED Index="0" Flag="2">L470</LED>
      <LED Index="1" Flag="4">L560</LED>
      <LED Index="2" Flag="1">L415</LED>
      <LED Index="3" Flag="0">None</LED>
      <LED Index="4" Flag="0">None</LED>
    </Sequence>
    <Power>
      <L415 Reg="10112">2.000%</L415>
      <L470 Reg="10368">3.000%</L470>
      <L560 Reg="10624">4.000%</L560>
    </Power>
  </Excitation>
  <OptoStimulation>
    <LaserWavelength>450</LaserWavelength>
    <LaserAmplitude Reg="11140">16.999%</LaserAmplitude>
    <StimPeriod Reg="32">31.250 Hz</StimPeriod>
    <StimOn Reg="4">4 ms</StimOn>
    <StimReps>34</StimReps>
  </OptoStimulation>
  <DigitalIOs>
    <DigitalOutput0>Strobe</DigitalOutput0>
    <Output1Routing>InternalLaser</Output1Routing>
    <DigitalInput0>EventFalling</DigitalInput0>
    <DigitalInput1>EventFalling</DigitalInput1>
  </DigitalIOs>
</Settings>
```

**Write to Persistent Registers:**

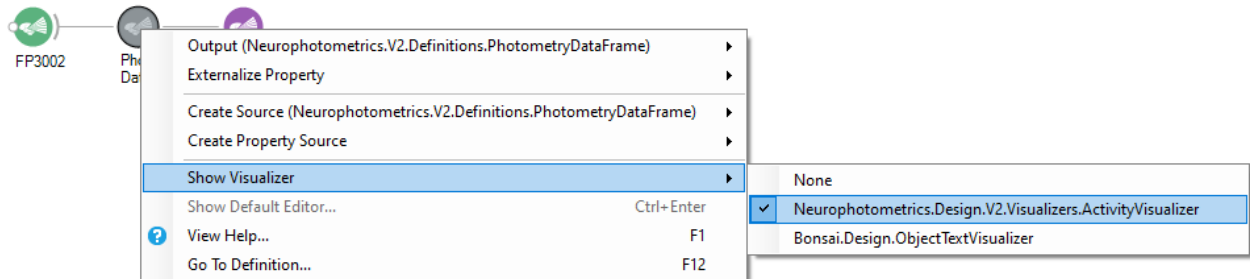
The “Write To Persistent Registers” button is also located at the bottom of the “FP3002 Setup” window. This button is used to store the configured FP3002 settings to the system’s persistent registers. This means that upon powering on the system, these settings will be loaded into the system.

# Photometry Data

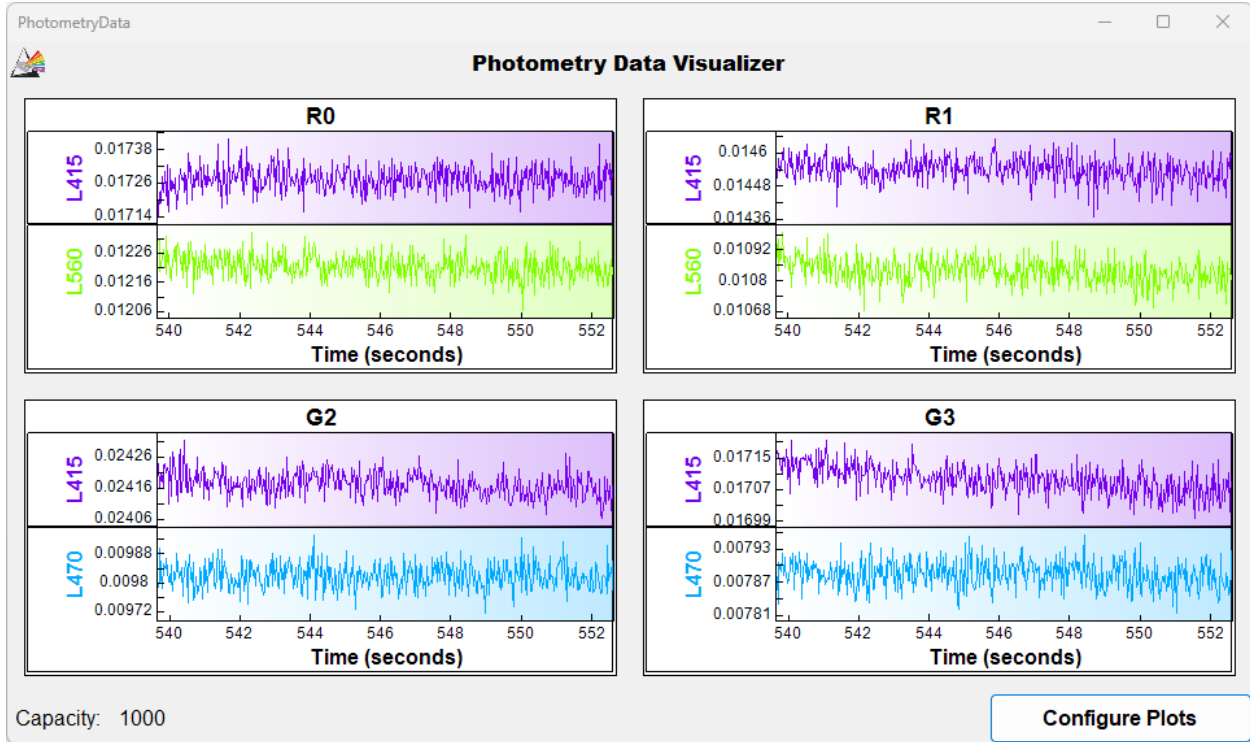
The “Photometry Data” node is used to process Harp messages from the “FP3002” node in order to extract photometry data from the incoming image data. This node filters out Harp messages unrelated to photometry and converts the remaining messages to usable data. The “Photometry Data” node accepts data of type “Bonsai.Harp.HarpMessage” and outputs data of type “PhotometryDataFrame”. This node can be connected immediately after the “FP3002” node and its output can be connected to the “Photometry Writer” node.



The “Photometry Data” node has an associated activity visualizer that will display the deinterleaved activity data during the experiment. To access this visualizer, right click the node and select the “Activity Visualizer” from the “Show Visualizer” list.



With the “Activity Visualizer” selected as the node’s visualizer, it will appear in a new window once the workflow begins.



By default, the L415 and L560 signals for the red channel regions of interest and the L415 and L470 signals for the green channel regions of interest will be displayed with a capacity of 1000 data points. The capacity of all of the signals can be configured by clicking the current capacity value. Meanwhile, the region visibility as well as the signal visibility and scaling can be configured with the “Configure Plots” button. The plot visibility and the signal visibilities are saved to the workflow.

Region	Plot Visible	L415				L470				L560			
		Visible	Scaling	Min	Max	Visible	Scaling	Min	Max	Visible	Scaling	Min	Max
R0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Auto	0.0171	0.0175	<input type="checkbox"/>	Auto	0.0000	1.0000	<input checked="" type="checkbox"/>	Auto	0.0120	0.0123
R1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Auto	0.0143	0.0146	<input type="checkbox"/>	Auto	0.0000	1.0000	<input checked="" type="checkbox"/>	Auto	0.0105	0.0109
G2	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Auto	0.0239	0.0241	<input checked="" type="checkbox"/>	Auto	0.0096	0.0099	<input type="checkbox"/>	Auto	0.0000	1.0000
G3	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Auto	0.0168	0.0171	<input checked="" type="checkbox"/>	Auto	0.0078	0.0080	<input type="checkbox"/>	Auto	0.0000	1.0000

## Photometry Writer

The “Photometry Writer” node will write the photometry data into storage in the form of a .csv file. The output file will have at least nine columns, below is a description for each column:

### Column 1, Frame Counter:

Provides a frame number for each photometry data frame. This frame number is zero based where the zeroth frame is a null frame

### Column 2, SystemTimestamp:

The timestamp generated by the system for each frame. This timestamp has units of seconds since the system turned on.

### Column 3, LED State:

Indicates which LED, if any, were on for any particular frame. Here “0” indicates None frames, “1” indicates the L415 frames, “2” indicates the L470 frames, “4” indicates the L560 frames, “6” indicates L470 + L560 frames (for FP3001 only), and “7” indicates all of the LEDs were on.

### Column 4, ComputerTimestamp:

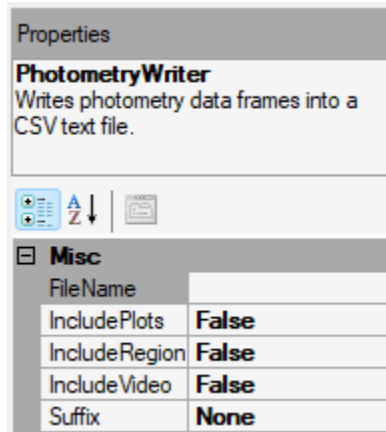
The timestamp generated by the computer for each frame. This timestamp has units of milliseconds since midnight.

### Column 5+, Region Data:

These are the columns where the relative fluorescence data will appear. Each pre-defined ROI will have its own column.

FrameCounter	SystemTimestamp	LedState	ComputerTimestamp	R0	R1	R2	R3
0	585.265568	7	36431337.2461	0.00392156...	0.00392156...	0.00392156...	0.00392156...
1	585.270528	4	36431341.7639	0.00392156...	0.00392156...	0.00392156...	0.00392156...
2	585.275552	1	36431347.271	0.00392367...	0.00406659...	0.00392282...	0.00400063...
3	585.280544	2	36431352.2779	0.00392156...	0.00398828...	0.00392282...	0.00396448...
4	585.285536	4	36431356.3459	0.00392156...	0.00401003...	0.00392282...	0.00397239...
5	585.290528	1	36431361.5949	0.00392262...	0.00404629...	0.00392657...	0.00397804...
6	585.295552	2	36431366.8484	0.00392156...	0.00398828...	0.00392156...	0.00394302...

The normal use case of the “Photometry Writer” node is to generate an output .csv file with the specified suffix contains all of the photometry data. However, this node provides additional functionality through the “Include...” properties. If any of the “Include...” properties are set to “True” then the “Photometry Writer” node will generate a folder containing the output .csv as well as the additional files from the “Include...” properties. Below is a description for each property:



FileName: The name of the output .csv file. If any of the “Include...” properties are set to “True” this filename will also help generate the name of the output folder.

Folder icon	FPData	9/12/2023 9:28 AM	File folder	
File icon	FPData.csv	9/5/2023 10:07 AM	CSV File	168 KB

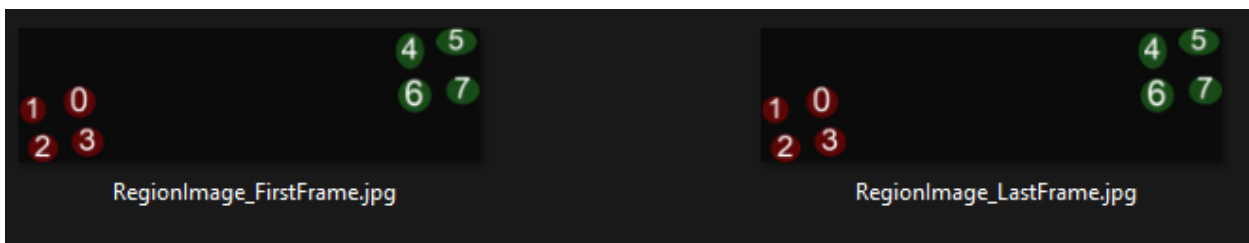
IncludePlots: When enabled, a “Plots” folder will be created that contains the activity plot for each region of interest. This property is not recommended for experiments longer than 1 hour.

Plots	9/12/2023 9:28 AM	File folder	
RegionImages	9/12/2023 9:28 AM	File folder	
VideoFiles	9/12/2023 9:28 AM	File folder	
FPData.csv	9/5/2023 10:08 AM	CSV File	211 KB

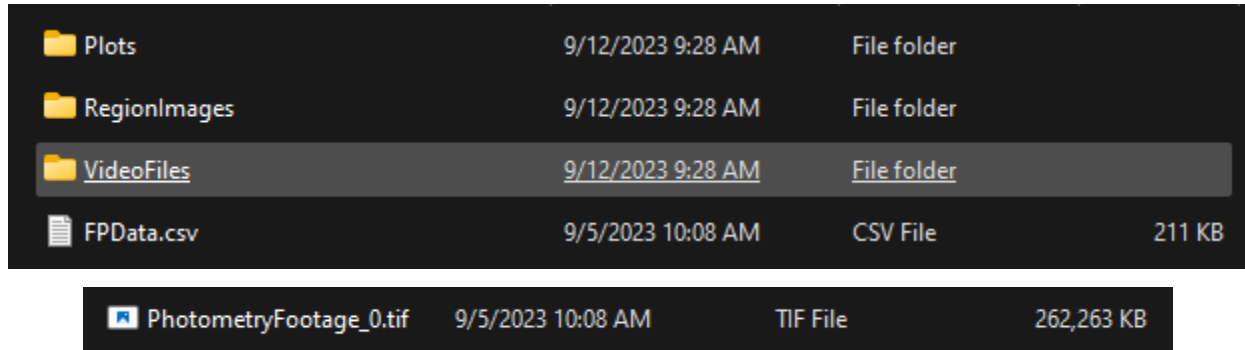
G4_Plot.jpg	9/5/2023 10:08 AM	JPG File	475 KB
G5_Plot.jpg	9/5/2023 10:08 AM	JPG File	478 KB
G6_Plot.jpg	9/5/2023 10:08 AM	JPG File	478 KB
G7_Plot.jpg	9/5/2023 10:08 AM	JPG File	478 KB
R0_Plot.jpg	9/5/2023 10:08 AM	JPG File	497 KB
R1_Plot.jpg	9/5/2023 10:08 AM	JPG File	501 KB
R2_Plot.jpg	9/5/2023 10:08 AM	JPG File	485 KB
R3_Plot.jpg	9/5/2023 10:08 AM	JPG File	502 KB

Include Regions: When enabled, a “RegionImages” folder will be created that contains the first and last image taken by the FP3002 system with the user-defined regions of interest overlaid.

Plots	9/12/2023 9:28 AM	File folder	
RegionImages	9/12/2023 9:28 AM	File folder	
VideoFiles	9/12/2023 9:28 AM	File folder	
FPData.csv	9/5/2023 10:08 AM	CSV File	211 KB



Include Video: When enabled, a “VideoFiles” folder will be created that contains multi-page *.tif* files. These *.tif* files are lossless to allow for post-hoc analysis of the raw image data coming from the FP3002 system. This feature saves a lot of data so be sure to have sufficient write speed and storage capacity before enabling.



The image shows a screenshot of a file explorer interface with a dark background. It displays a list of folders and files. The 'VideoFiles' folder is highlighted. Below the main list, a separate entry for 'PhotometryFootage\_0.tif' is shown in a lighter box.

Plots	9/12/2023 9:28 AM	File folder	
RegionImages	9/12/2023 9:28 AM	File folder	
VideoFiles	9/12/2023 9:28 AM	File folder	
FPData.csv	9/5/2023 10:08 AM	CSV File	211 KB

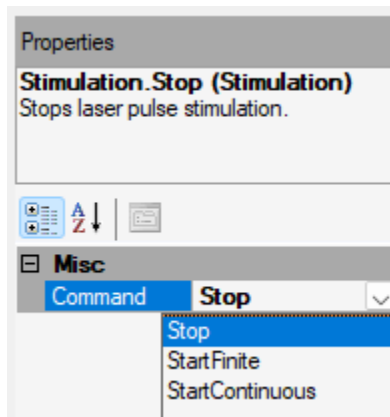
PhotometryFootage_0.tif	9/5/2023 10:08 AM	TIF File	262,263 KB
-------------------------	-------------------	----------	------------

Suffix: Allows the repetition of the same file name for multiple experiments by appending a unique suffix to the filename. The options for unique suffixes are an integer value (count) or a date-time value (timestamp).

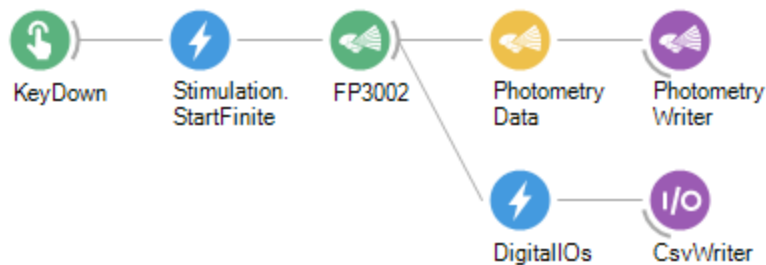


# Stimulation

The “Stimulation” node generates Harp messages that can be sent to the FP3002 system through the “FP3002” node. These messages work to command the system to start and/or stop stimulation. This node has a configurable “Command” property used to specify the type of stimulation command it will generate. The “Command” property will dictate whether the node generates a “Stop” stimulation command, “Start Finite” stimulation command, or a “Start Continuous” stimulation command.



This node accepts any data type as an input and outputs data of type “Bonsai.Harp.HarpMessage”. This way it can use any set of nodes as a software trigger and can be connected directly to the “FP3002” node to send the generated stimulation command to the FP3002 system.

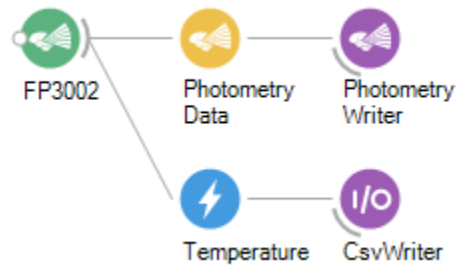


The “Stimulation” node differentiates between a “Finite” and “Continuous” stimulation. With both stimulation modes, a laser pulse train will be started using the “Amplitude”, “Wavelength”, “Pulse Frequency”, and “Pulse Width” parameters set in the “FP3002 Setup” window. However, the “Continuous” pulse train will ignore the “Pulse Count” property, continuing the pulse train until the “Stop” command is sent to the system. Meanwhile, the “Finite” pulse train will pulse the laser the amount of times specified with the “Pulse Count” property.

For detailed discussions on implementing different stimulation techniques, please visit the [“Stimulation”](#) chapter.

## Temperature

The “Temperature” node is used to record the internal temperature, in Celsius, measured in the FP3002 system. This node accepts data of type “Bonsai.Harp.HarpMessage” from the “FP3002” node every 10 seconds. It possesses an “Include Timestamp” property that allows the user to timestamp every temperature measurement using the FP3002 system’s internal clock. Depending on whether the “Include Timestamp” property is set to True or False, this node will output data of type “double” or “Bonsai.Harp.Timestamps<double>”. In either case, this node can be connected directly to a “Csv Writer” node to record temperature data.



# Appendix II: Hotkeys

---

## Bonsai:

- Ctrl + A:** Select all nodes in the workflow
- Ctrl + C:** Copies all selected nodes
- Ctrl + D:** Disable all selected nodes
- Ctrl + Enter:** Show default editor
- Ctrl + G:** Group selected nodes into a grouped workflow
- Ctrl + N:** Create a new workflow
- Ctrl + O:** Open an existing workflow
- Ctrl + S:** Save workflow
- Ctrl + V:** Paste
- Ctrl + X:** Cut
- Ctrl + Y:** Redo
- Ctrl + Z:** Undo
- Ctrl + Shift + C:** Copy as image
- Ctrl + Shift + D:** Enables all selected nodes
- Ctrl + Shift + E:** Export workflow as a .svg image
- Ctrl + Shift + G:** Ungroups the selected grouped workflow
- Ctrl + Shift + S:** Save selected nodes as a workflow
- Del:** Delete selected nodes
- F5:** Start workflow
- Ctrl + F5:** Start workflow without debugging
- F12:** Go to definition (requires Visual Studio Code installed)

## Opening FP3002 Node:

- Shift + Control + R:** Reset device settings
- Shift + Control + Alt:** Update firmware

## Regions of Interest:

- Control + Left Click + Drag:** Create uniformly scaled ROI
- Control + Right Click + Drag:** Uniformly scale existing ROI
- Right Click + Drag:** Scale existing ROI
- Left Click + Drag:** Move existing ROI
- Page Up:** Increment Image Scale (Brightness)

**Page Down:** Decrement Image Scale (Brightness)

**Delete:** Delete ROI

**Tab:** Select next ROI

**Photometry Data Plot:**

**Control + P:** Print Plot

**Control + S:** Save Plot