Fundamentals of Fiber Photometry Workflows

NEUROPHOTOMETRICS
The fiber photometry company

Christopher S. Johnson

January 3rd, 2023

NEUROPHOTOMETRICS
The fiber photometry company

Table of Contents

NEUROPHOTOMETRICS
The fiber photometry company

Preface

This document discusses the fundamentals of developing workflows for conducting experiments using the Neurophotometric's fiber photometry systems. The material is presented in a project based learning environment that builds in complexity over the course of each chapter.

The presentation begins with a chapter dedicated to setting up an environment for workflow development. Throughout this document, we utilize an open-source visual programming language called Bonsai (https://bonsai-rx.org/) to create workflows for communicating with and recording data from external sources, including our fiber photometry systems.

From there, we delve into fundamental concepts found within fiber photometry experiments. Chapter 2 presents our standard workflows for acquiring data from the 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. For 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.

Chapter 1: Hardware/Software Environment

This chapter focuses on setting up an environment for fiber photometry workflow development. We begin our discussion with the minimum hardware requirements for conducting fiber photometry experiments within Bonsai. Then, we cover all of the required downloads and installations for constructing the workflows shown in this document. We conclude with a guide on how to manage and update the 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.

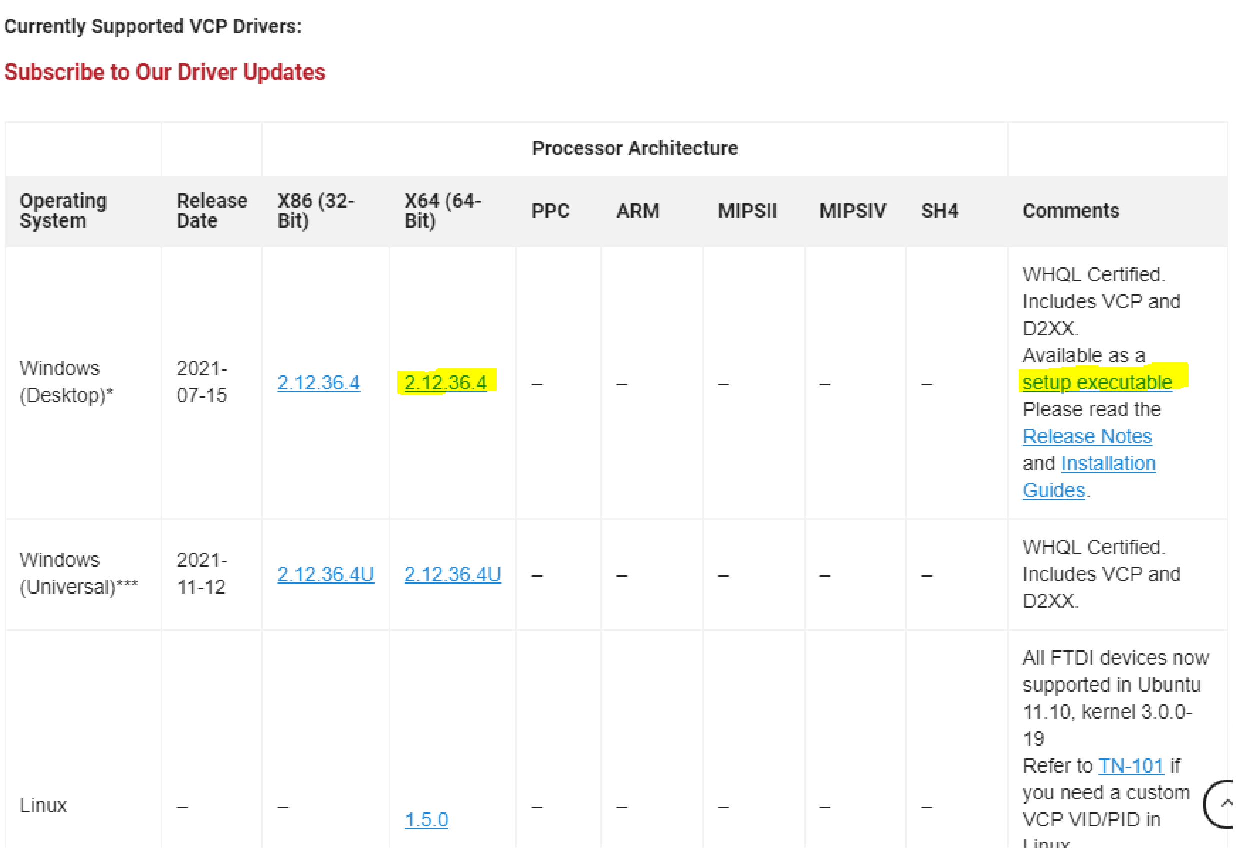
Notes
- The Neurophotometry Fiber Photometry systems must be connected directly to Type A USB 3.0 ports.
- A one meter Type A USB 3.0 to Micro-B cable, provided by Neurophotometrics, should be used to connect the FP3002 system to the computer.
- Neurophotometrics Fiber Photometry systems cannot be connected 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. First, FTDI's Virtual COM Port (VCP) driver will be installed in order for the Neurophotometrics Fiber Photometry Systems to appear as additional COM ports available to the PC. Second, the Spinnaker Software Development Kit (SDK) will allow for communication between the PC and the internal camera on the system. Finally, Bonsai will be used as a visual programming language that can easily access open source software published by Neurophotometrics.

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/), 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.

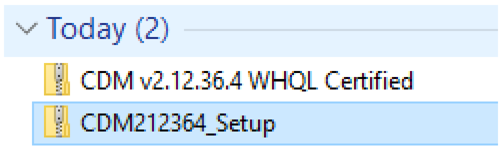


Currently Supported VCP Drivers:

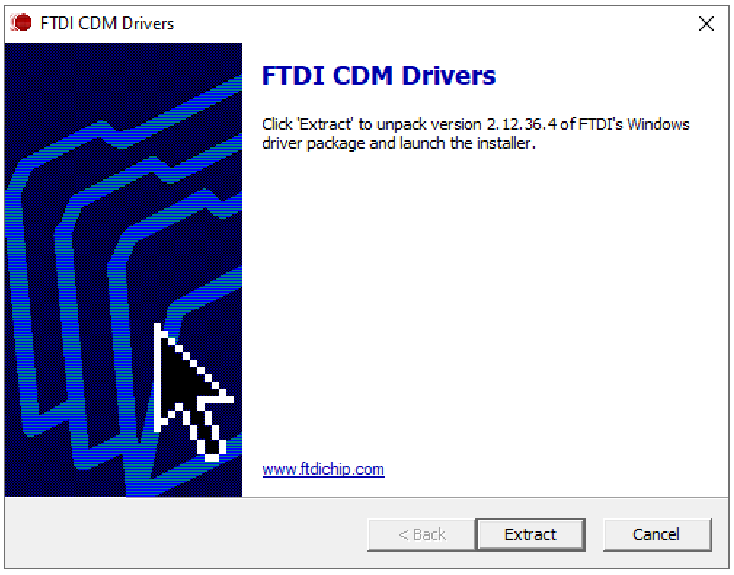
Subscribe to Our Driver Updates

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

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.


FTDI CDM Drivers

FTDI CDM Drivers

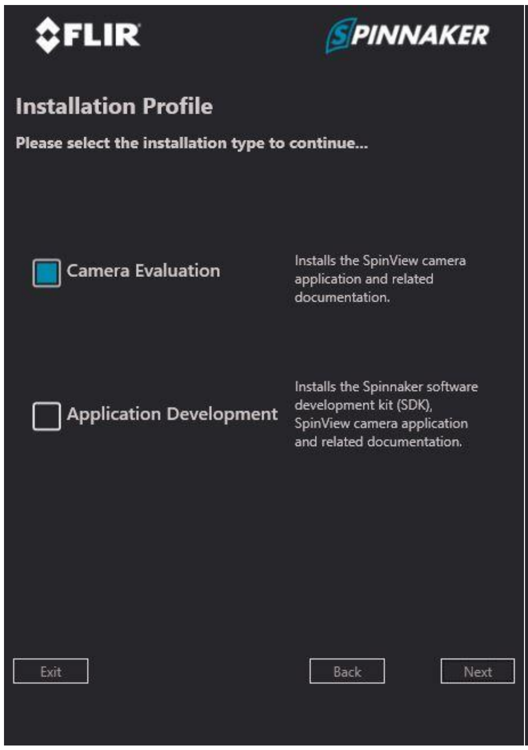
Click 'Extract' to unpack version 2.12.36.4 of FTDI's Windows driver package and launch the installer.

www.ftdichip.com

NEUROPHOTOMETRICS
The fiber photometry company

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

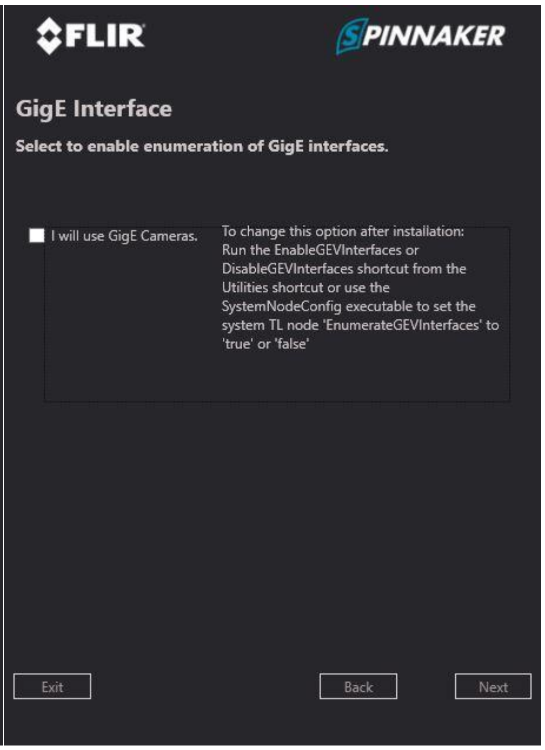
Download the [SpinnakerSDK_FULL_1.29.0.5_x64.exe](https://flir.app.boxcn.net/v/SpinnakerSDK/file/622481657674) (https://flir.app.boxcn.net/v/SpinnakerSDK/file/622481657674). Currently only the 1.29.05 version of the SDK is compatible. Run the .exe file and follow the installation wizard. When choosing 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.”


Installation Profile

Please select the installation type to continue...

[✓ Camera Evaluation] Installs the SpinView camera application and related documentation.

Application Development: Installs the Spinnaker software development kit (SDK), SpinView camera application and related documentation.


GigE Interface

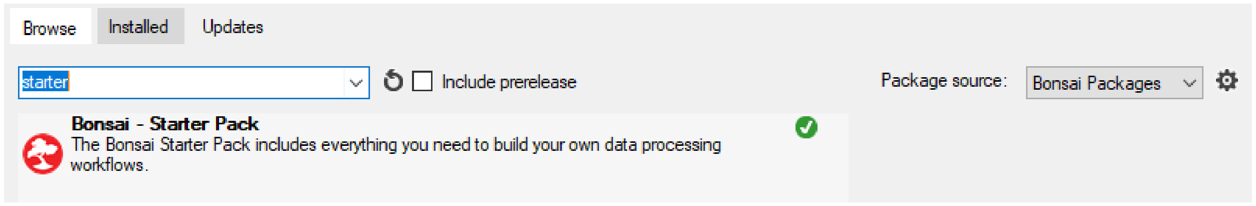
Select to enable enumeration of GigE interfaces.

☐ I will use GigE Cameras.

To change this option after installation: Run the EnableGEVInterfaces or DisableGEVInterfaces shortcut from the Utilities shortcut or use the SystemNodeConfig executable to set the system TL node 'EnumerateGEVInterfaces' to 'true' or 'false'

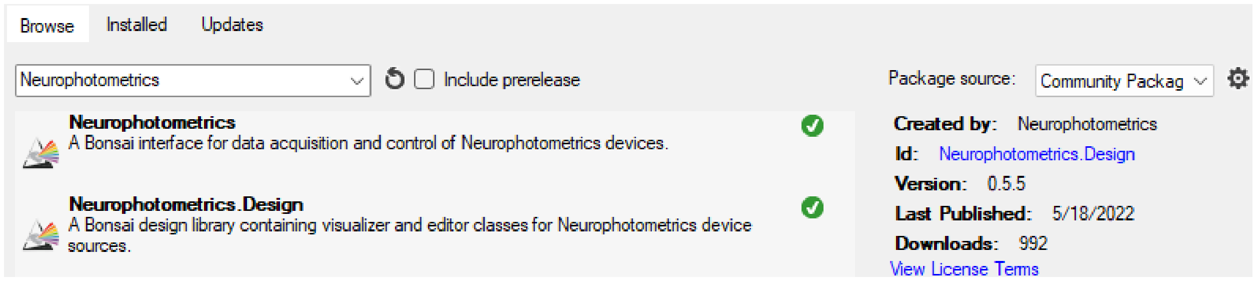
Bonsai Software:
Bonsai is an open source software that utilizes a visual programming language for creating and synchronizing heterogeneous data streams. This software interfaces with NuGet to allow for easy installation of open source software pages, including our packages for communicating with Neurophotometrics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

Download the .exe installer for the latest stable version of Bonsai (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.



Bonsai - Starter Pack
The Bonsai Starter Pack includes everything you need to build your own data processing workflows.

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.



Neurophotometrics
A Bonsai interface for data acquisition and control of Neurophotometrics devices.

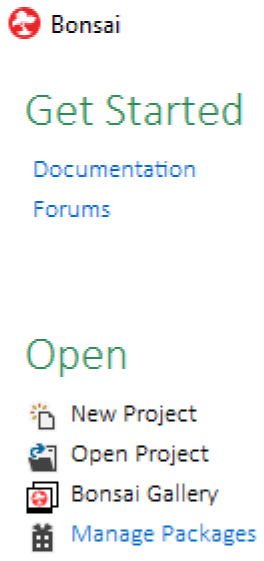
Neurophotometrics.Design
A Bonsai design library containing visualizer and editor classes for Neurophotometrics device sources.

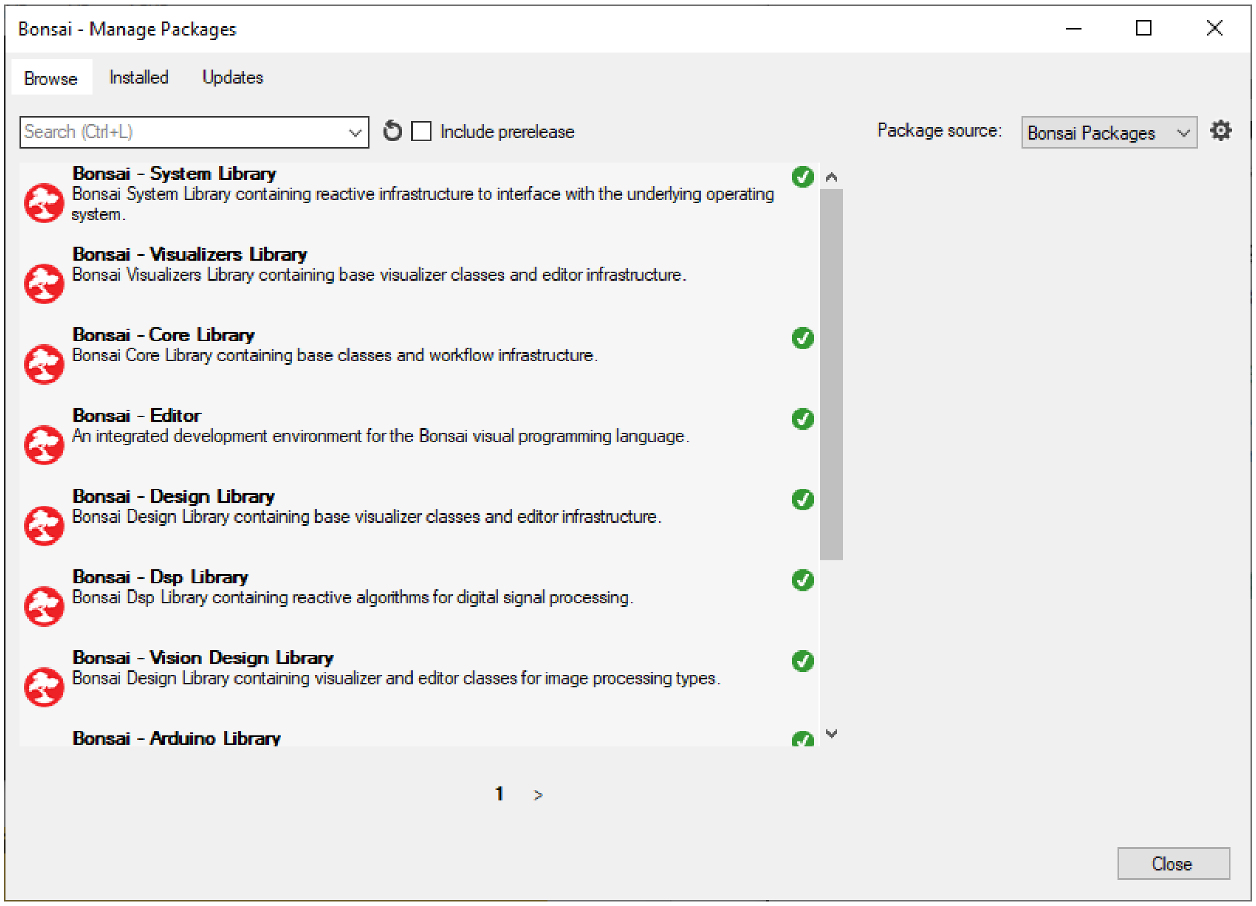
Created by: Neurophotometrics
Id: Neurophotometrics.Design
Version: 0.5.5
Last Published: 5/18/2022
Downloads: 992
View License Terms

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.




Bonsai - Manage Packages

Bonsai - System Library
Bonsai System Library containing reactive infrastructure to interface with the underlying operating system.

Bonsai - Visualizers Library
Bonsai Visualizers Library containing base visualizer classes and editor infrastructure.

Bonsai - Core Library
Bonsai Core Library containing base classes and workflow infrastructure.

Bonsai - Editor
An integrated development environment for the Bonsai visual programming language.

Bonsai - Design Library
Bonsai Design Library containing base visualizer classes and editor infrastructure.

Bonsai - Dsp Library
Bonsai Dsp Library containing reactive algorithms for digital signal processing.

Bonsai - Vision Design Library
Bonsai Design Library containing visualizer and editor classes for image processing types.

Bonsai - Arduino Library

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:

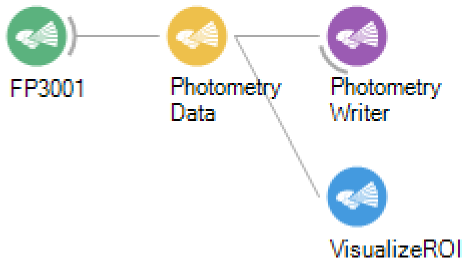
As Bonsai Packages are updated to newer versions, occasionally new features are not compatible with older Bonsai Software versions. To update the Bonsai Software, uninstall Bonsai then reinstall the latest stable version. Find “Bonsai” inside of the Windows search bar. Right click it and click “uninstall”. 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

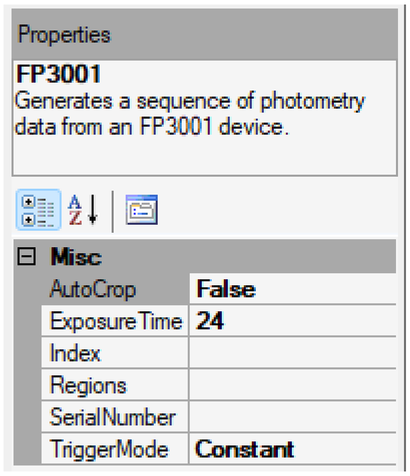
In this chapter, we develop the idea of a standard fiber photometry workflow. This workflow is used to record interleaved fiber photometry data, saving it to a *.csv* file. The "Standard Photometry" workflow not only provides a basic way of recording photometry data, but also acts as a base for more 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.

FP3001 System

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, frame flags, and activity data. The construction of the standard FP3001 photometry workflow is the exact same as the FP3002 workflow, except that the “FP3002” node is replaced with the “FP3001” node.



Begin configuration of the “FP3001” node by specifying the properties found within the property panel:


Properties

FP3001
Generates a sequence of photometry data from an FP3001 device.

Misc

AutoCrop	**False**
ExposureTime	**24**
Index	
Regions	
SerialNumber	
TriggerMode	**Constant**

AutoCrop: A boolean value used to specify whether or not the camera will automatically crop the incoming image. When set to “True” the “FP3001” node will crop the image to the smallest rectangle bounding all of the drawn ROIs. This act of cropping the image allows the camera on the system to run at faster frequencies.

ExposureTime: An integer value used to specify the exposure time of the internal camera of the FP3001 system. This value must agree with the FPS set on the driver box such that the exposure time is at least one millisecond less than the period of data acquisition. For example, when the FPS is set to 40Hz, the period of data acquisition is equal to $Period = \frac{1}{40Hz} \star \frac{1000ms}{1sec} = 25ms$. In this 25ms duration, the camera must go through its exposure time and its dead time. The dead time must be at least 1ms which means in this 40Hz case, the exposure time is recommended to be 24ms.

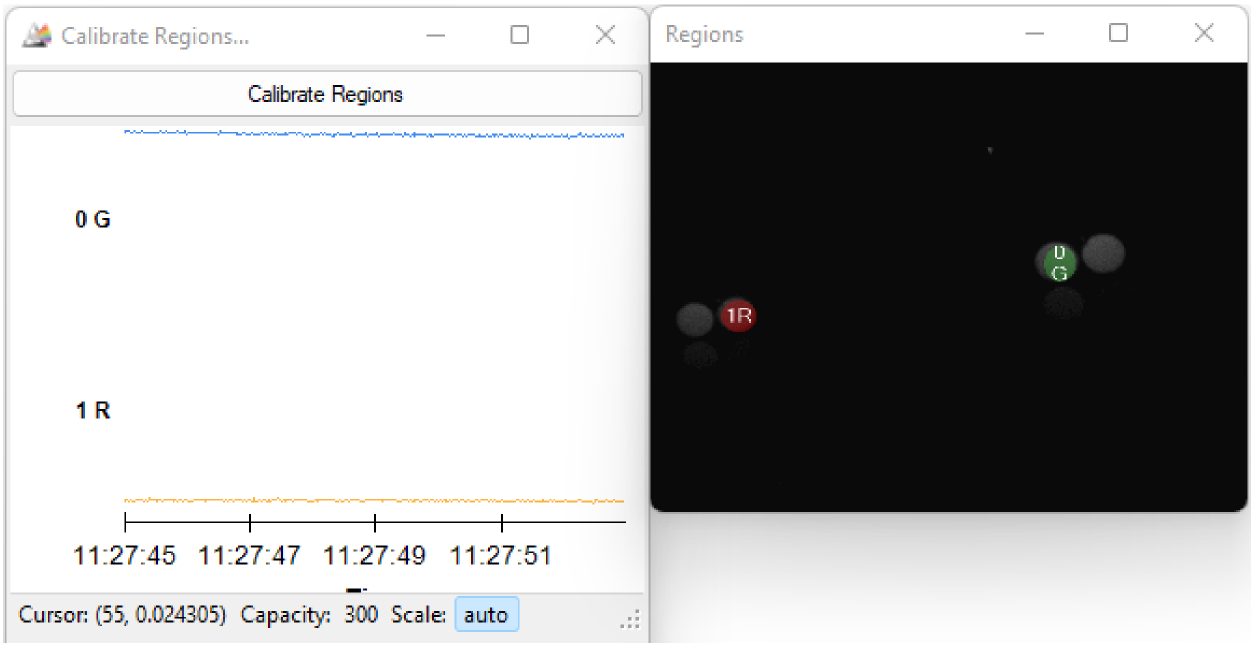
Index: An integer value used to specify which spinnaker camera to connect. This property is used to ensure that the “FP3001” node correctly connects to the FP3001 system and not a behavioral camera.

Regions: A custom data type property used to store the dimensions and locations of the user-defined ROIs. This property is only used to store this data and not used to define the ROIs. In order to define the ROIs, double click the “FP3001” node while the workflow is stopped and the system is connected.

Serial Number: A string value used to specify the serial number of the camera to connect. Similar to the “Index” property, this property can be used to ensure that the “FP3001” node correctly connects to the FP3001 system and not to a behavioral camera.

Trigger Mode: A dropdown menu used to specify the trigger sequence of the FP3001 system. This must agree with the driver box in order for the “FP3001” to assign the correct frame flags to the photometry data frames.

Once the properties shown above are specified, double click the FP3001 node to open a calibration window. Begin data acquisition on the driver box and 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. To draw ROIs, click the “Calibrate Regions” button to open a new window containing the camera image. In this new “Regions” window, you can draw regions by left clicking and dragging.

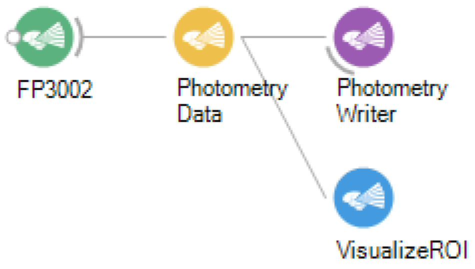


Cursor: (55, 0.024305) Capacity: 300 Scale: auto

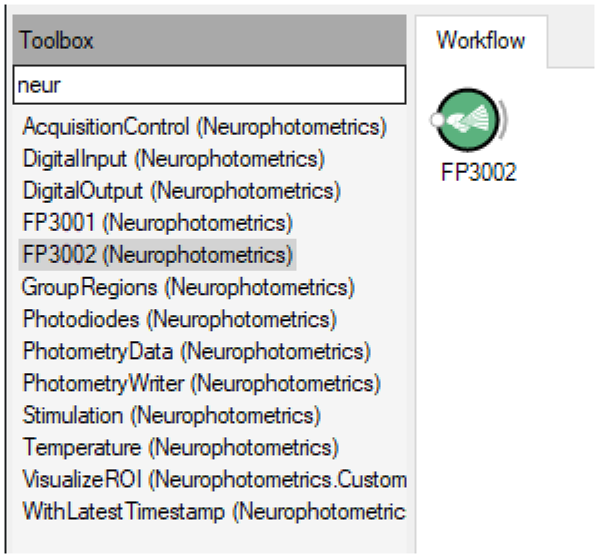
Within the “Regions” window you can move already drawn ROIs by left clicking them and dragging them to a new position. You can resize an existing ROI by right clicking it and dragging. While interacting with the “Regions” window be cautious about stray clicks within the window as they will draw ROI that are too small to see or select by right clicking. Whenever drawing ROIs, double check that the number of signals shown in the “Calibrate Regions…” window matches the desired number of drawn ROIs in the “Regions” window. 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 System

In order to have a standard workflow for recording from the FP3002 system we need to be able to connect to the system, process the data coming from it, and record it to a *.csv* file. We also include the ability to visualize the data in real time. The workflow presented here accomplishes all of these goals.



In order to construct this workflow, begin by adding an “FP3002” node to a new workflow. The “FP3002” node connects Bonsai to the FP3002 system and configures all of the system settings.

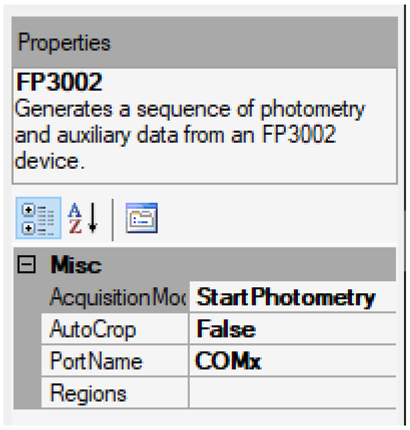


Toolbox

neur

AcquisitionControl (Neurophotometrics)
DigitalInput (Neurophotometrics)
DigitalOutput (Neurophotometrics)
FP3001 (Neurophotometrics)
FP3002 (Neurophotometrics)
GroupRegions (Neurophotometrics)
Photodiodes (Neurophotometrics)
PhotometryData (Neurophotometrics)
PhotometryWriter (Neurophotometrics)
Stimulation (Neurophotometrics)
Temperature (Neurophotometrics)
VisualizeROI (Neurophotometrics.Custom
WithLatestTimestamp (Neurophotometric

Click the “FP3002” node and find the “Properties” panel on the right side of the Bonsai workflow.

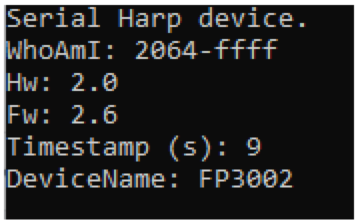

Properties

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

Misc

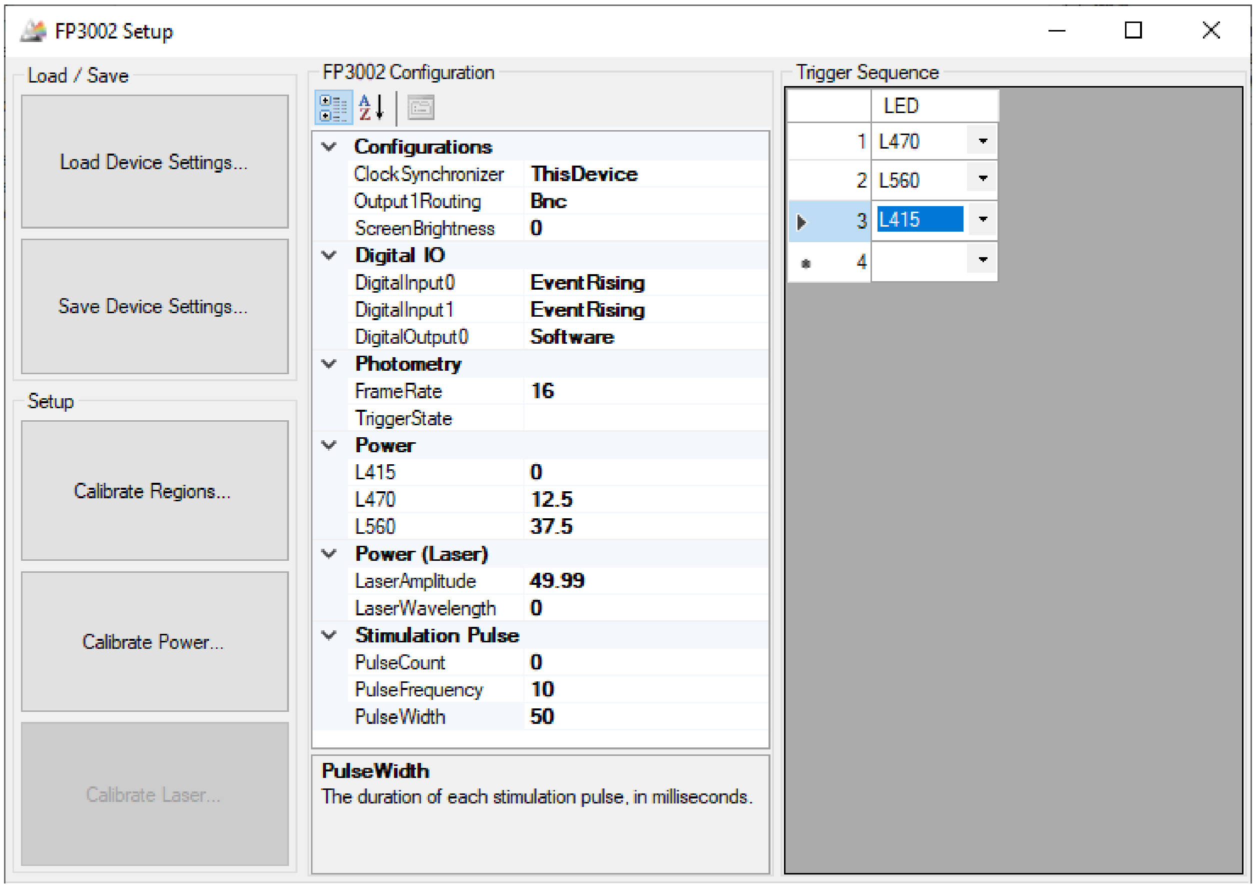
AcquisitionMoc	**StartPhotometry**
AutoCrop	**False**
PortName	**COMx**
Regions	

Connect Bonsai to the FP3002 system by setting the value of the “Port Name” property to the correct COM port. A proper connection will show the FP3002 system’s information 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.

In order to allow for higher camera capture frames rates, be sure to set the “AutoCrop” property to “True”. This will cause the camera to crop its image to the smallest bounding rectangle that contains all of the defined ROIs. Once Bonsai is connected to the FP3002 system through the “FP3002” node, double click the node to open the “FP3002 Setup” window. This is where the configuration settings of the system are located. For a standard photometry workflow, there are four settings that should be configured: Trigger Sequence, LED Power, Frame Rate, and Regions of Interest.



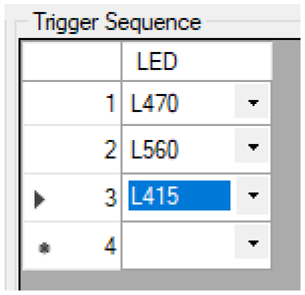
Configurations	
ClockSynchronizer	ThisDevice
Output1Routing	Bnc
ScreenBrightness	0
Digital IO	
DigitalInput0	EventRising
DigitalInput1	EventRising
DigitalOutput0	Software
Photometry	
FrameRate	16
TriggerState	
Power	
L415	0
L470	12.5
L560	37.5
Power (Laser)	
LaserAmplitude	49.99
LaserWavelength	0
Stimulation Pulse	
PulseCount	0
PulseFrequency	10
PulseWidth	50

PulseWidth
The duration of each stimulation pulse, in milliseconds.

	LED
1	L470
2	L560
3	L415
4	

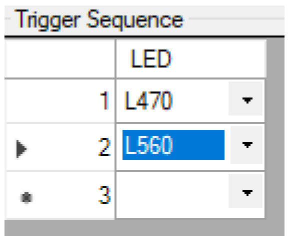
Trigger Sequence:

The “Trigger Sequence” panel of the “FP3002 Setup” window configures which LEDs are triggered and in what order. The last row in the panel does not affect the trigger sequence and is used to add new LEDs to the sequence.



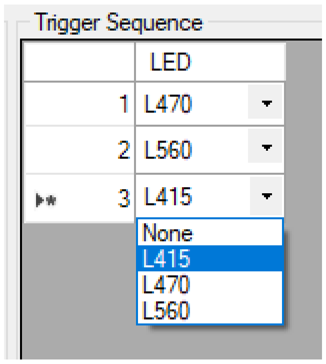
	LED
1	L470
2	L560
3	L415
4	

To delete an LED, select a row in the sequence and click the delete (Del) button on the keyboard.



	LED
1	L470
2	L560
3	

To add an LED, click the dropdown menu on the bottom row and select which LED to use.

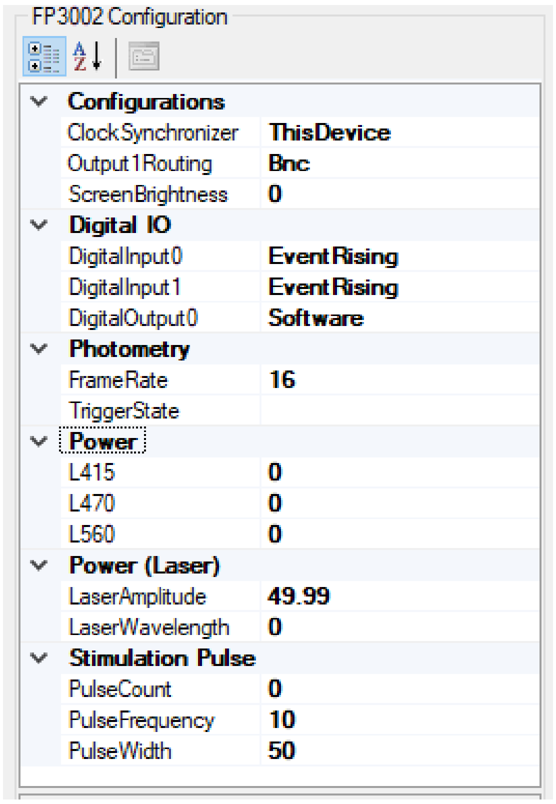


	LED
1	L470
2	L560
3	L415

To configure the order, click the drop down menu on each row and select where each LED goes in the sequence.

NEUROPHOTOMETRICS
The fiber photometry company

LED Power:
In the “FP3002 Configuration” panel of the “FP3002 Setup” window is a “Power” section. This section sets the percentage of power going to the LED when it is triggered in the trigger sequence.

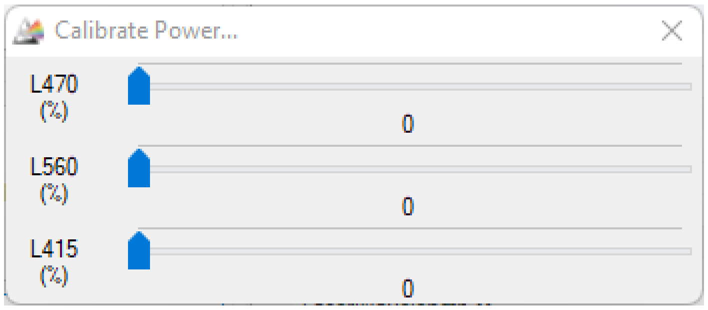

FP3002 Configuration

Configurations	
ClockSynchronizer	ThisDevice
Output1Routing	Bnc
ScreenBrightness	0
Digital IO	
DigitalInput0	EventRising
DigitalInput1	EventRising
DigitalOutput0	Software
Photometry	
FrameRate	16
TriggerState	
Power	
L415	0
L470	0
L560	0
Power (Laser)	
LaserAmplitude	49.99
LaserWavelength	0
Stimulation Pulse	
PulseCount	0
PulseFrequency	10
PulseWidth	50

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 50µW for 200µm fibers and approximately 120µW for 400µm 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 most experiments. For longer experiments (upwards of 5 hours), consider lowering the duty cycle of the LEDs and/or lowering the LED powers.

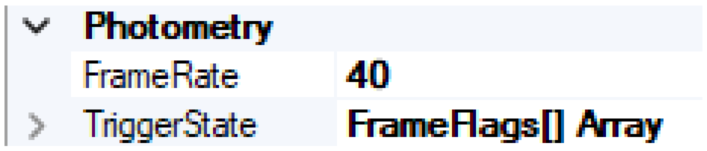
In order to configure the light power for an experiment, you must use the “Calibrate Power” tool in conjunction with a power meter. Click the “Calibrate Power” button found in the “Setup” panel of the “FP3002 Setup” window. Upon clicking the “Calibrate

Power" button, a "Calibrate Power" window will appear. This window contains a power scroll bar for each LED that allows you to manually adjust each LED's power while measuring the power of the light coming out of the ferrule.



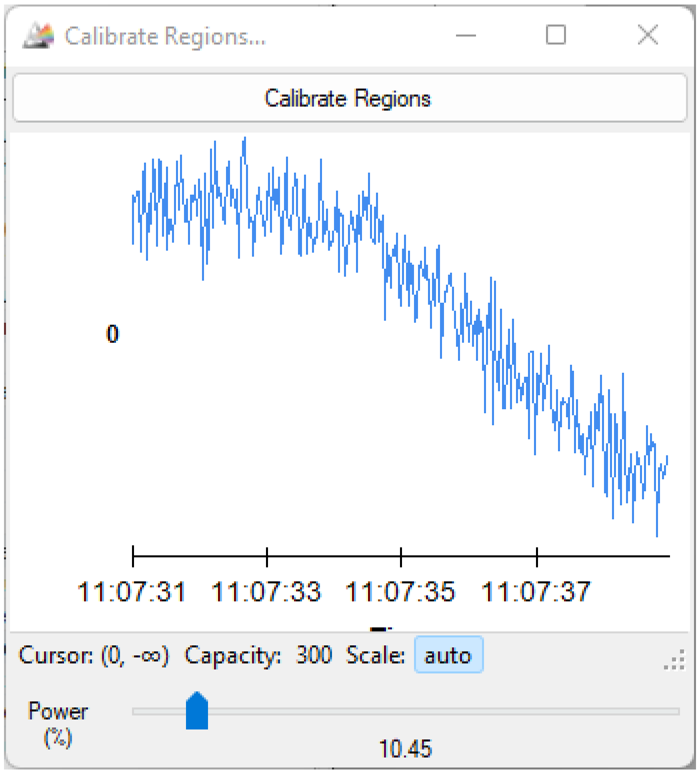
Once the appropriate percentage of power is found for each LED, close the "Calibrate Power" window and enter these values into the "FP3002 Configuration" panel.

Frame Rate:
The "Frame Rate" property specifies the frequency at which photometry data frames are generated. This value has units of "Hertz" and directly determines the frame rate of the internal camera. Indirectly, this value also determines the frequency of each LED since the current LED in the trigger sequence transitions every camera frame. This value ranges from 16-200Hz, however for higher frame rate, the "Auto Crop" property in the "FP3002" node's property panel must be set to "True". The frame rate value will be dictated by the response kinetics of the sensor that is to be observed. The frame rate must be set to accommodate the response kinetics of a given sensor (e.g. 20 Hz for GCaMP6) to capture both the excitation and emission timings in a trigger sequence. Please look up the paper where the sensor in use was published to see these kinetics. For *most* of the newer sensors, 30Hz (per wavelength) is the minimum and a safe starting point. Please note, several of the new GRABs will require >60Hz per wavelength to properly visualize the kinetics.



Photometry	
FrameRate	40
TriggerState	FrameFlags[] Array

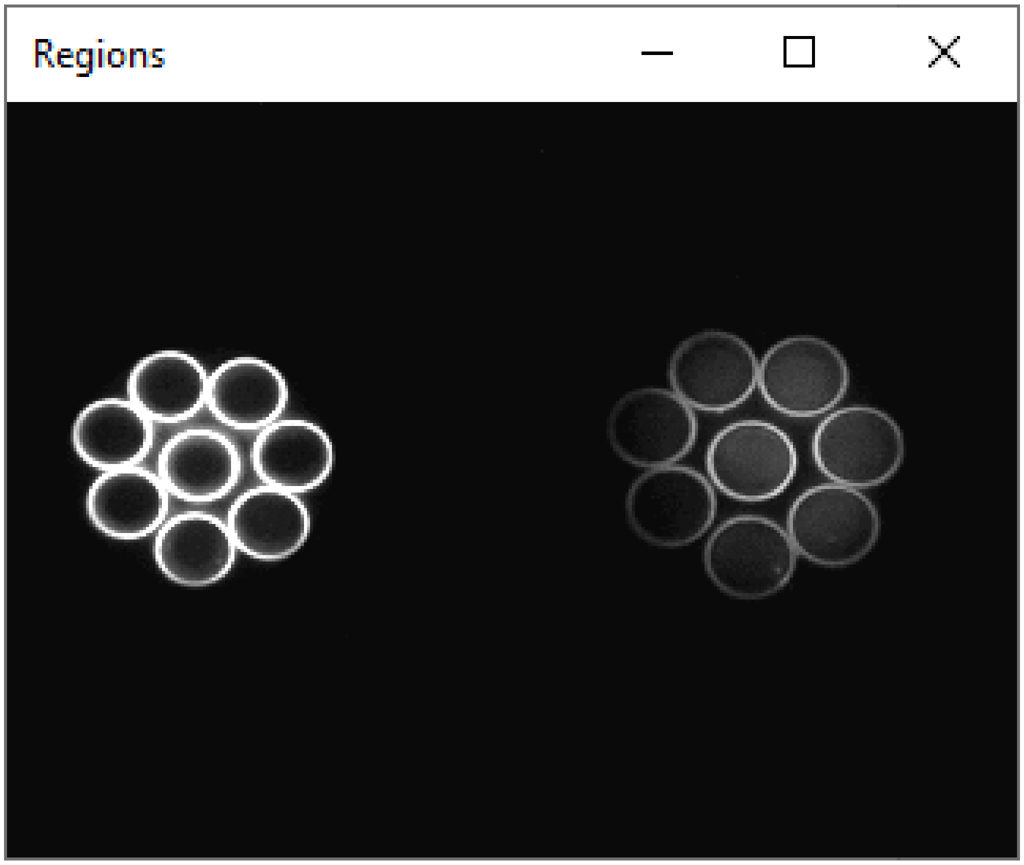
Regions of Interest (ROIs):
In order to define the ROIs, click the “Calibrate Regions…” button within the “FP3002 Setup” window. This will open a new window called “Calibrate Regions…” which displays a sample signal for each defined ROI. If no ROIs are defined yet, it will still show a signal that samples the whole image. When this window is opened, the system will be in calibration mode such that the frame rate is set to 40Hz and the trigger sequence consists of only the L470 LED. Here you can manually adjust the power of the L470 LED to help see the images of the fibers during alignment and defining of the ROIs.



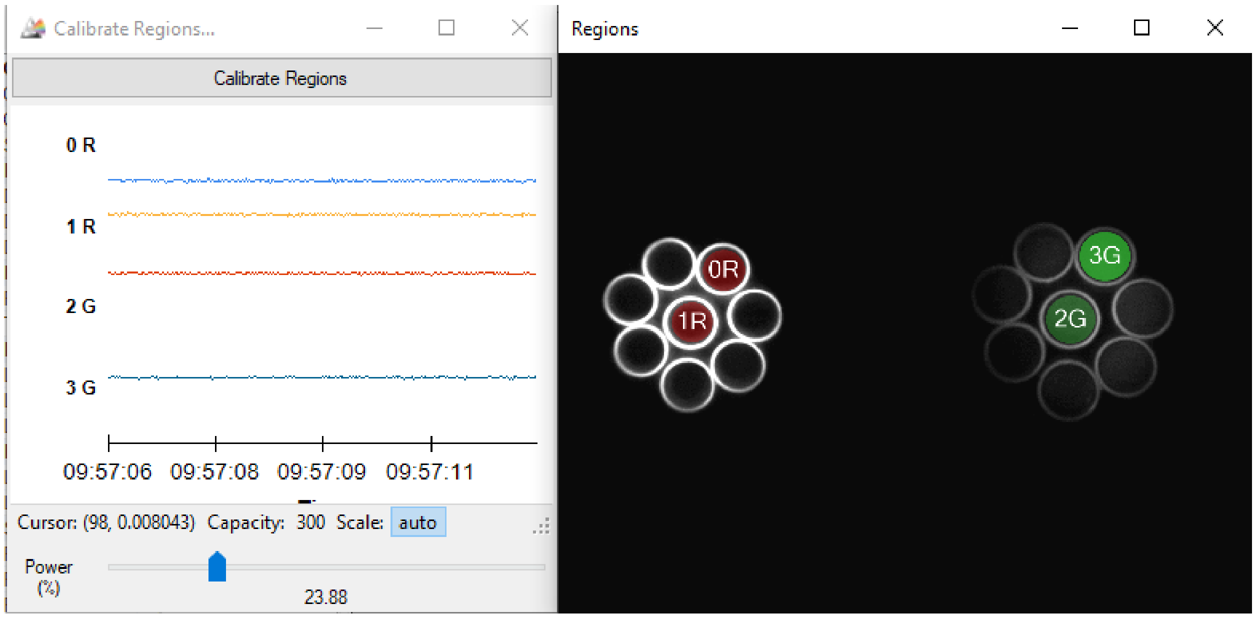
Cursor: (0, -∞) Capacity: 300 Scale: auto

Power (%) 10.45

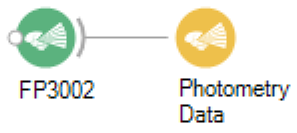
In this window, click the “Calibrate Regions” button to open a new window containing the image coming from the camera. This new window is called the “Regions” window and is used to both align to a patch cord and to draw ROIs over the fibers of the patch cord. For help aligning to a patch cord please see the “Connecting and Aligning My Patch Cord” section of the “Hardware Manual”.



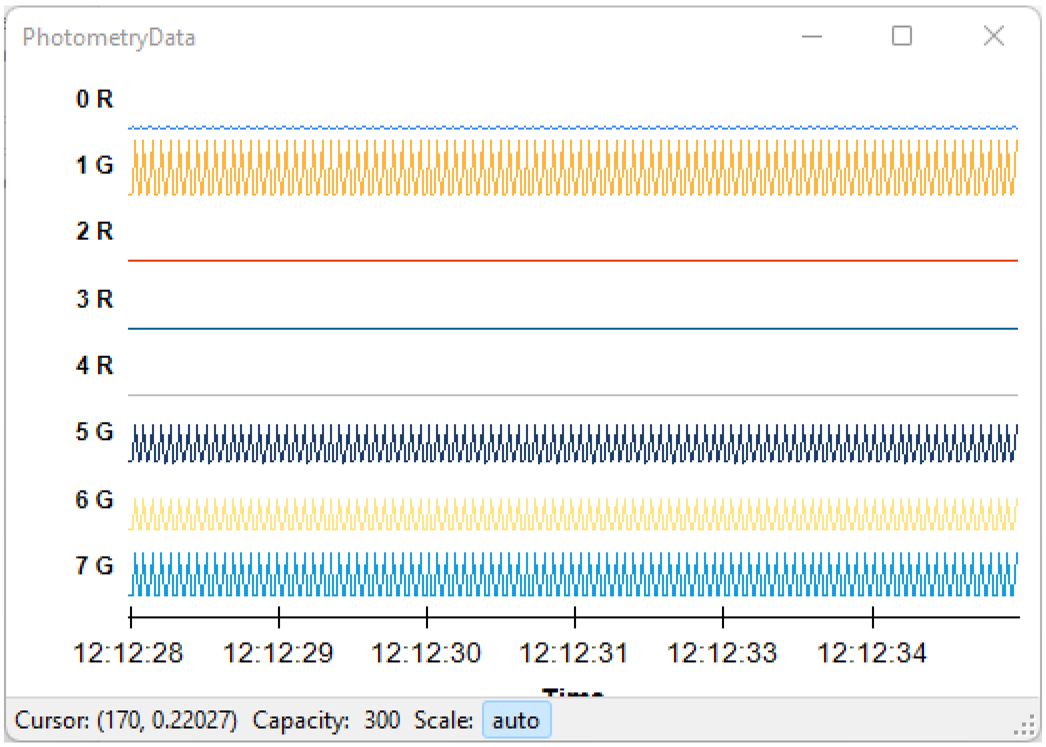
Once the patch cord is aligned and focused, left click and drag to draw an ROI on a fiber. You can move the ROI by left clicking and dragging an already existing ROI. To resize an ROI, right click and drag on an already existing ROI. In this window, be careful about stray clicks because it is possible to accidentally draw small ROIs that are barely visible. A simple test to determine whether or not extraneous ROIs have been drawn is to verify that the number of signals in the "Calibrate Regions…" window matches the desired number of ROIs. If an extra ROI is present, use the "Tab" key within the "Regions" window to cycle the selected region until the extra ROI is selected. Then, press the "Del" key to delete it. Once the ROIs are drawn, both of these windows and the "FP3002 Setup" window can be closed and the rest of the workflow can be configured.



The “FP3002” node deals with controlling the FP3002 system and bringing messages from the system into Bonsai for processing and recording. This node outputs data of type “Bonsai.Harp.HarpMessage”. This data type is not immediately writable to storage and requires some amount of processing. The “Photometry Data” node is used to process the data coming from the “FP3002” node.

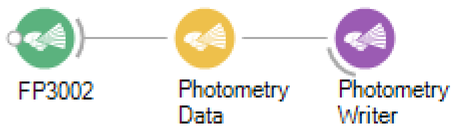


The “Photometry Data” node processes the data coming from the “FP3002” node by filtering out any data not related to photometry data. Then it converts the Harp message containing the photometry data into a custom data type called “Photometry Data Frame”. This “Photometry Data Frame” contains an image, frame counter, system timestamp, frame flag, and activity data for each ROI. The “Photometry Data” node has an associated visualizer that can be viewed by double clicking the node while the workflow is running.



This visualizer contains a running plot of the interleaved signals coming from each ROI. The toolbar on the bottom allows you to specify the capacity and the scale of the plots. In the image above, each plot is specified to contain 300 data points at any given time and each plot is set to autoscale. You can also manually set the minimum and maximum of the y-axes in the “Scale” section. This toolbar also displays the cursor position where the x value ranges from one to the capacity and represents the (one-based) index of the data point closest to the cursor. The y value is the average pixel value of that data point. This cursor position is updated when the mouse cursor moves. For a visualizer with more customization options, please see the “Visualize ROI” section below or visit the “Visualize ROI” node in the glossary

Once the data coming from the “FP3002” node is processed using the “Photometry Data” node, it is ready to be written to storage using the “Photometry Writer” node.



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.

<u>Column 1, Frame Counter</u>:

Provides a frame number for each photometry data frame. This frame number is zero based where the zeroth frame is a null frame

<u>Column 2, Timestamp</u>:

The timestamp generated by the system for each frame. This timestamp has units of seconds since the system turned on.

<u>Column 3, LED State</u>:

Indicates which LED, if any, were on for any particular frame. Here “1” indicates the L415, “2” indicates the “L470”, “4” indicates the “L560”, and “7” indicates a null frame (no LEDs).

Column 4, Stimulation:

A boolean value that represents whether stimulation is occurring during this frame. This is NOT used to determine the Laser state, please see the "Chapter 5: Stimulation" or the "Digital IOs" node's entry in the "Appendix I: Node Glossary" for more information on how to record the laser's state.

Column 5/6, Output 0/1:

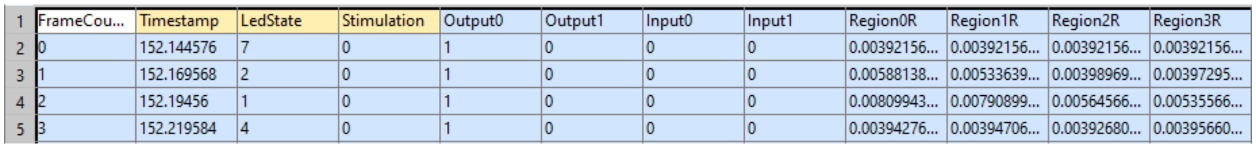
A boolean value that represents the state of the digital output ports during this frame. If you are sending digital outputs at a rate different from the photometry frame rate, please see the "Digital IOs" node's entry in the "Appendix I: Node Glossary" for more information on how to record the digital output port state.

Column 7/8, Input 0/1:

A boolean value that represents the state of the digital input ports during this frame. For higher precision recording of the digital input ports, please see "Digital IOs" node's entry in the "Appendix I: Node Glossary" for more information on how to record the digital input port state.

Column 9+, Region Data:

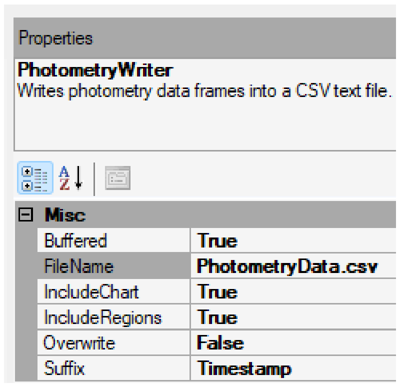
These are the columns where the relative fluorescence data will appear. Each pre-defined ROI will have its own column. This is the raw data: average pixel intensity per ROI per frame, normalized from 0 to 1.



1	FrameCou...	Timestamp	LedState	Stimulation	Output0	Output1	Input0	Input1	Region0R	Region1R	Region2R	Region3R
2	0	152.144576	7	0	1	0	0	0	0.00392156...	0.00392156...	0.00392156...	0.00392156...
3	1	152.169568	2	0	1	0	0	0	0.00588138...	0.00533639...	0.00398969...	0.00397295...
4	2	152.19456	1	0	1	0	0	0	0.00809943...	0.00790899...	0.00564566...	0.00535566...
5	3	152.219584	4	0	1	0	0	0	0.00394276...	0.00394706...	0.00392680...	0.00395660...

NEUROPHOTOMETRICS
The fiber photometry company

The “Photometry Writer” node has a variety of configurable properties. Three of the properties it has in common with the “Csv Writer” node: “File Name”, “Overwrite”, “Suffix”. You can specify the “File Name” property by either double clicking the node while the workflow is stopped, or clicking the “...” in the “File Name” text box. Be sure to specify the file extension as “.csv” in the filename. The “Overwrite” property allows the software to overwrite any files of the same name as specified in the “File Name” property. The “Suffix” property allows you to keep the same file name for multiple experiments by appending a unique suffix to the file name. You can either specify this unique suffix to be an integer value or as a date-time value. The “Include Chart” and “Include Regions” options allow you to generate a chart of the photometry data collected during the experiment and an image showing the labeled regions.

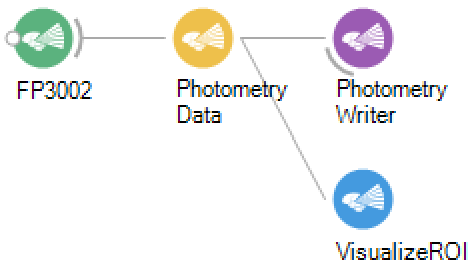

Properties

PhotometryWriter
Writes photometry data frames into a CSV text file.

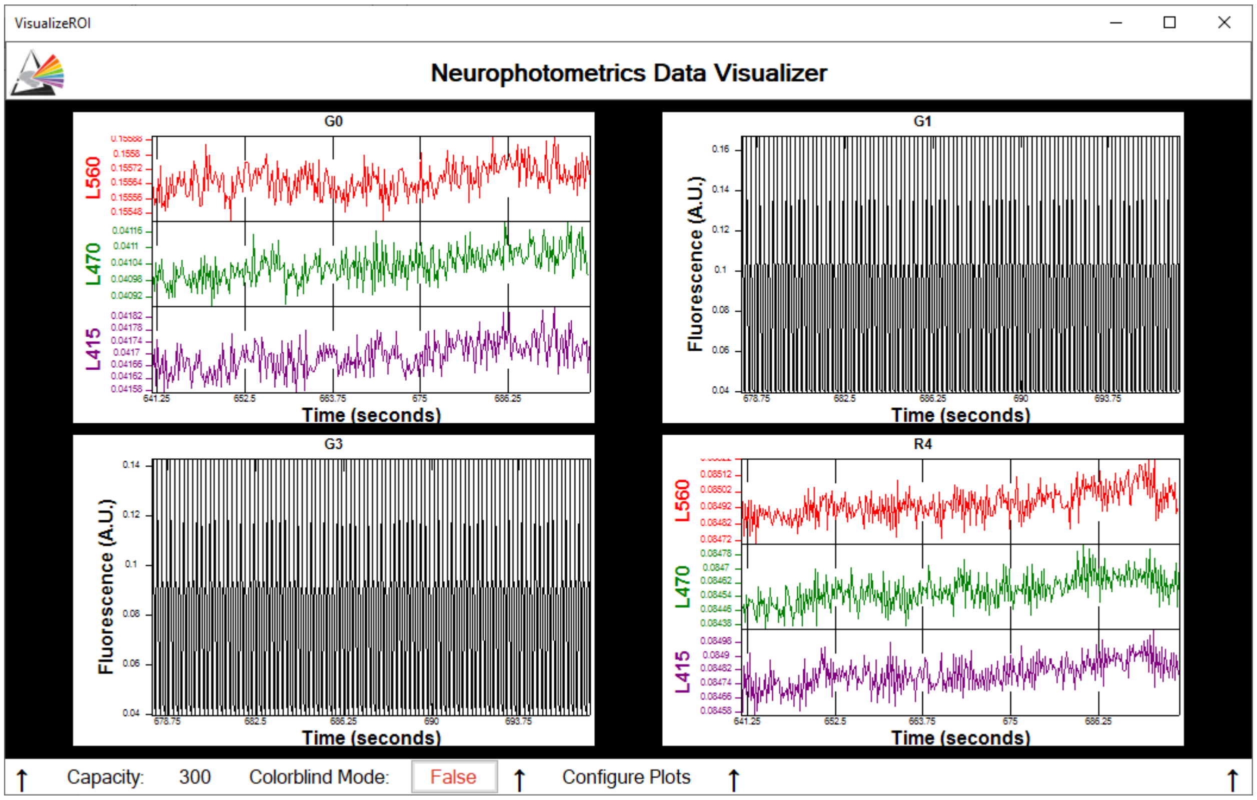
Misc

Buffered	True
FileName	PhotometryData.csv
IncludeChart	True
IncludeRegions	True
Overwrite	False
Suffix	Timestamp

With the combination of the “FP3002”, “Photometry Data”, and “Photometry Writer” nodes a basic fiber photometry workflow is complete. However, it is highly recommended to make use of the “Visualize ROI” node. This node provides a customizable visualization of the photometry data being passed through the workflow. Connect the “Photometry Data” node to the “Visualize ROI” node in parallel with the “Photometry Writer” node.



The “VisualizeROI” node allows for the data from the “Photometry Data” node to be displayed in the form of rolling plots. It has an automated layout that will maximize the size of each plot based on the number of visible plots and the dimensions of the visualizer window. This node has a user interface (UI) that allows for the user to adjust the configuration settings of each ROI’s plot. The user has control over which plots are visible, which are deinterleaved, and which are autoscaled. When an ROI’s plot is deinterleaved, the user also has control over which LED plots are visible and which are autoscaled. This new node also allows the user to toggle ON/OFF colorblind mode, creating a more colorblind friendly color palette for deinterleaved plots. The “Visualize ROI” node also gives the option to change the capacity of the plots, showing more or less data points per window. For more information on the “Visualize ROI” node please visit its entry in the “Appendix I: Node Glossary”.



Chapter 3: Synchronization

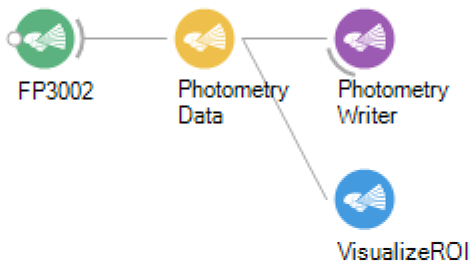
This chapter explores the concept of synchronization of data streams. Here we define synchronization as the ability to compare the timestamps of one data stream to another. With this definition, our goal of these sections is to generate data from multiple sources, including the FP3002 system, and create methods for determining the relative time between events of different data streams.

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 within Bonsai. This is the most commonly used technique, however its limitation is that it requires all data sources to have support within Bonsai. When trying to synchronize with data sources without Bonsai support, 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. The hardware synchronization techniques presented in this chapter also possess a limitation that data sources 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.

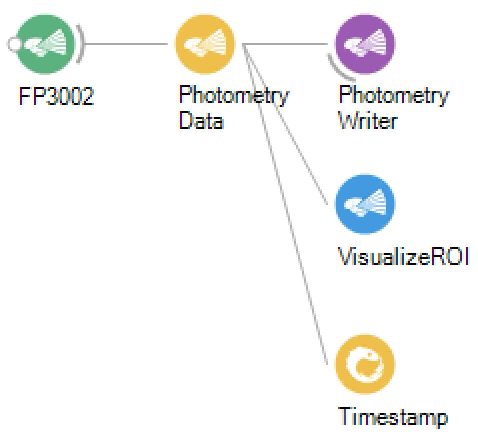
With these three techniques, most devices can be synchronized with the fiber photometry data stream using Bonsai. However, if a device is unsupported by Bonsai, does not have +5V TTL signals, and possesses no LED indicators, then more complex hardware synchronization methods will have to be implemented.

Software

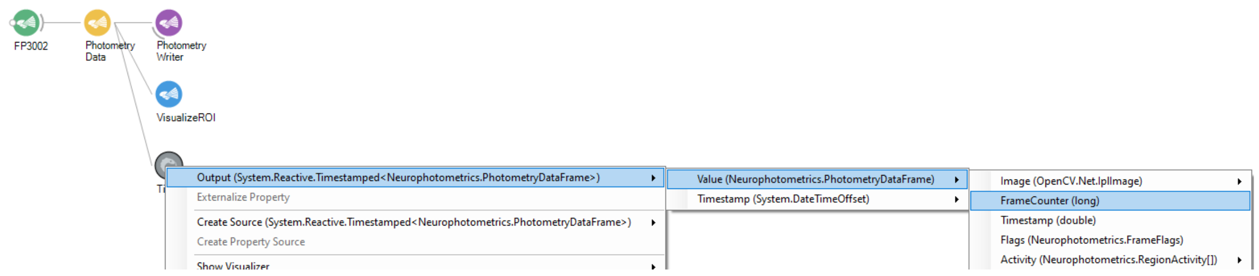
In many experiments, data is generated by multiple devices. When these devices have support within Bonsai, their data streams can be easily synchronized using built-in Bonsai nodes. Let's begin our discussion with the standard photometry workflow.



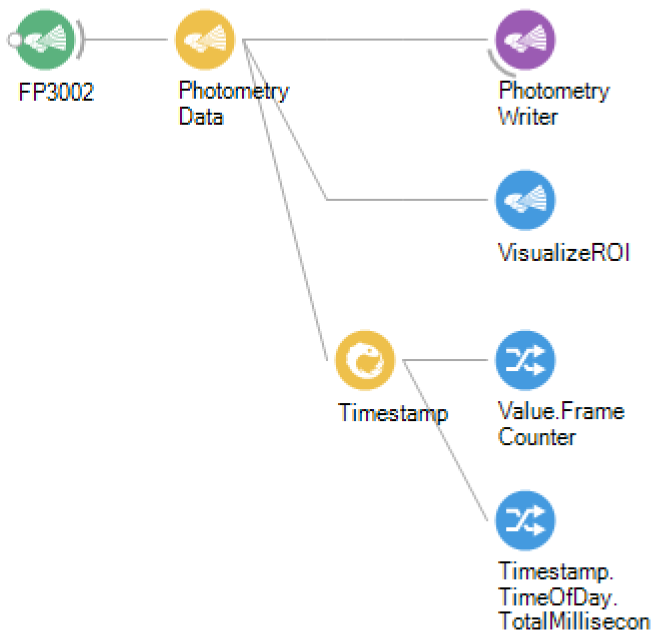
In this photometry data stream, the FP3002 system generates a timestamp with great precision. The clock used to generate this timestamp is reset every time the system goes through a power cycle. This means that the units of time in the .csv file produced by the "Photometry Writer" will be as seconds since the system turned on. Other data streams will not have immediate access to the clock on the FP3002 system. In order to make the timestamps of the photometry data stream comparable to timestamps of another data stream, we need to share the same clock. An easy way to do this while still keeping the precision of the system clock's timestamp is to also timestamp every photometry data frame with the computer's clock. The computer's clock will be accessible to all data streams allowing them to be synchronized. To add a computer clock timestamp to the photometry data stream, add a "Timestamp" node after the "Photometry Data" node, parallel to the "Photometry Writer" and "Visualize ROI" nodes.



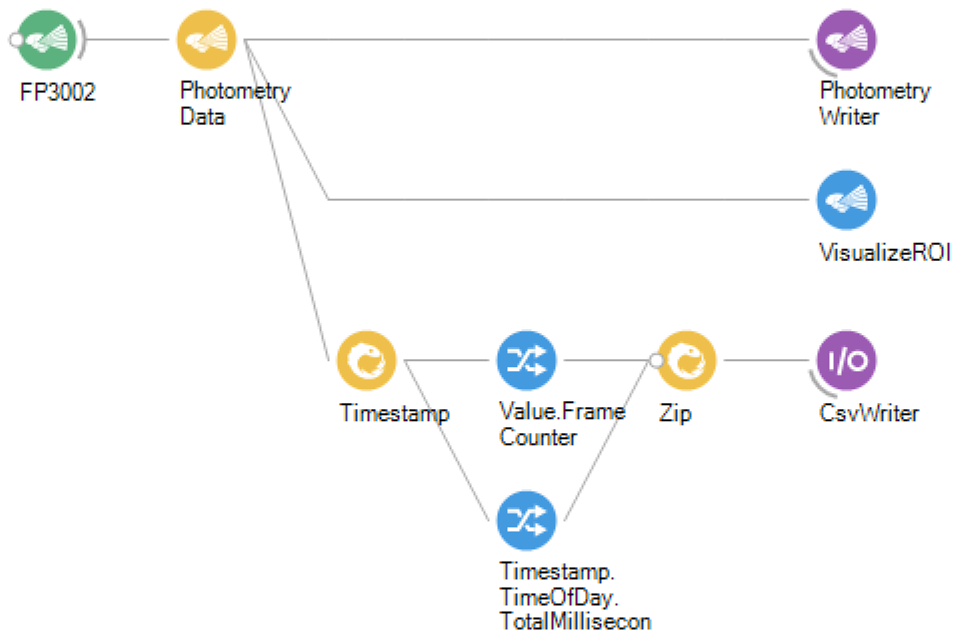
The “Timestamp” node will generate data of type “DateTimeOffset”. This is a powerful .NET structure that contains both DateTime data and TimeSpan data. It also allows you to output data coming from the “Photometry Data” node. First, let's output the frame number from the “Timestamp” node by right clicking it and selecting “Output → Value → FrameCounter”.


FP3002
Photometry Data
Photometry Writer
VisualizeROI
Output (System.Reactive.Timestamped<Neurophotometrics.PhotometryDataFrame>)
Externalize Property
Create Source (System.Reactive.Timestamped<Neurophotometrics.PhotometryDataFrame>)
Create Property Source
Show Visualizer
Value (Neurophotometrics.PhotometryDataFrame)
Timestamp (System.DateTimeOffset)
Image (OpenCV.Net.IplImage)
FrameCounter (long)
Timestamp (double)
Flags (Neurophotometrics.FrameFlags)
Activity (Neurophotometrics.RegionActivity[])

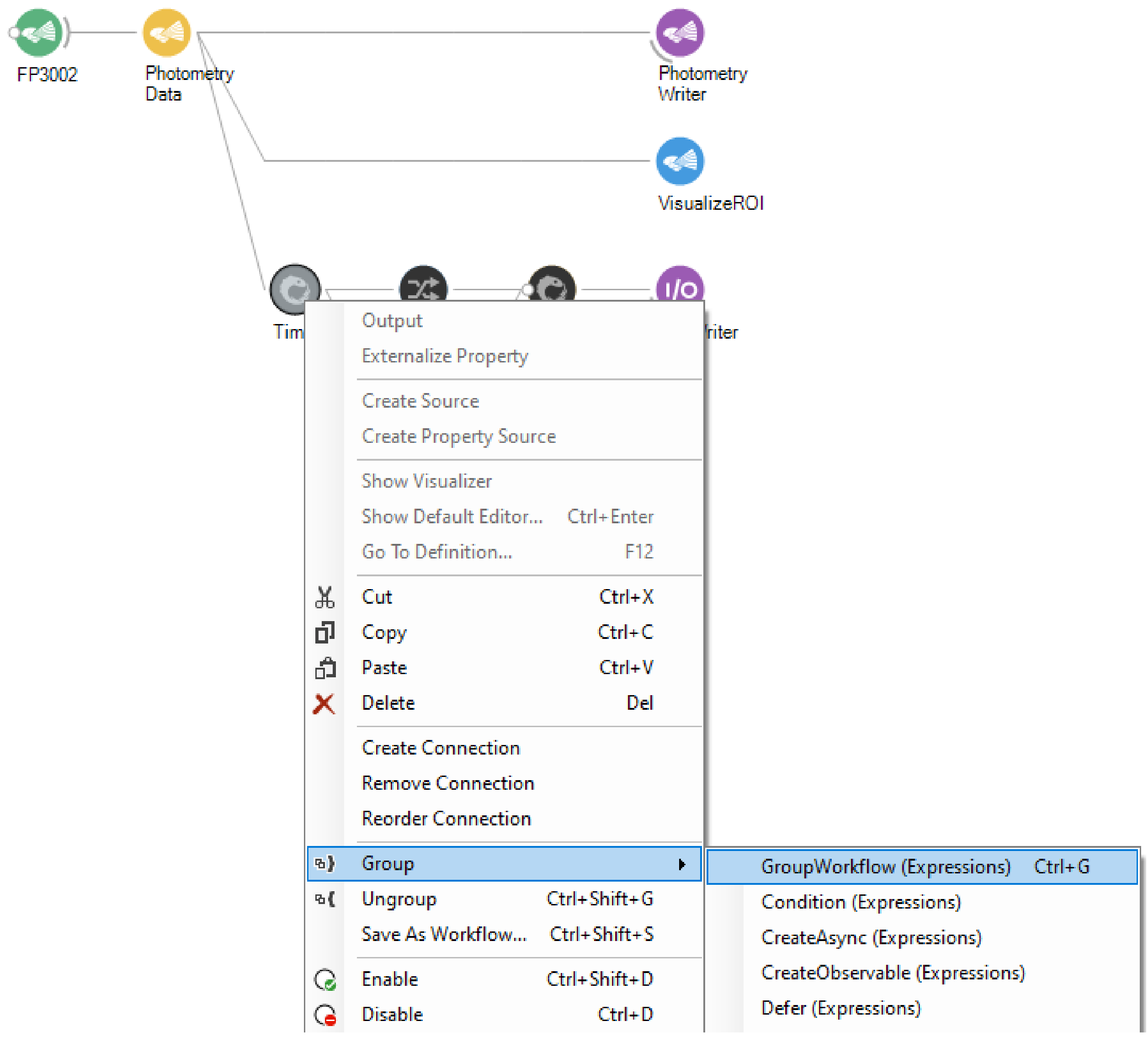
Then we will also output the desired timestamp. Here we have many options for the type of timestamp. One of the most commonly used timestamps is the time of day, total milliseconds. This will generate a timestamp with units of milliseconds since midnight in the computer’s time zone. To output this type of timestamp, right click the “Timestamp” node and select “Output → Timestamp → TimeOfDay → TotalMilliseconds”.



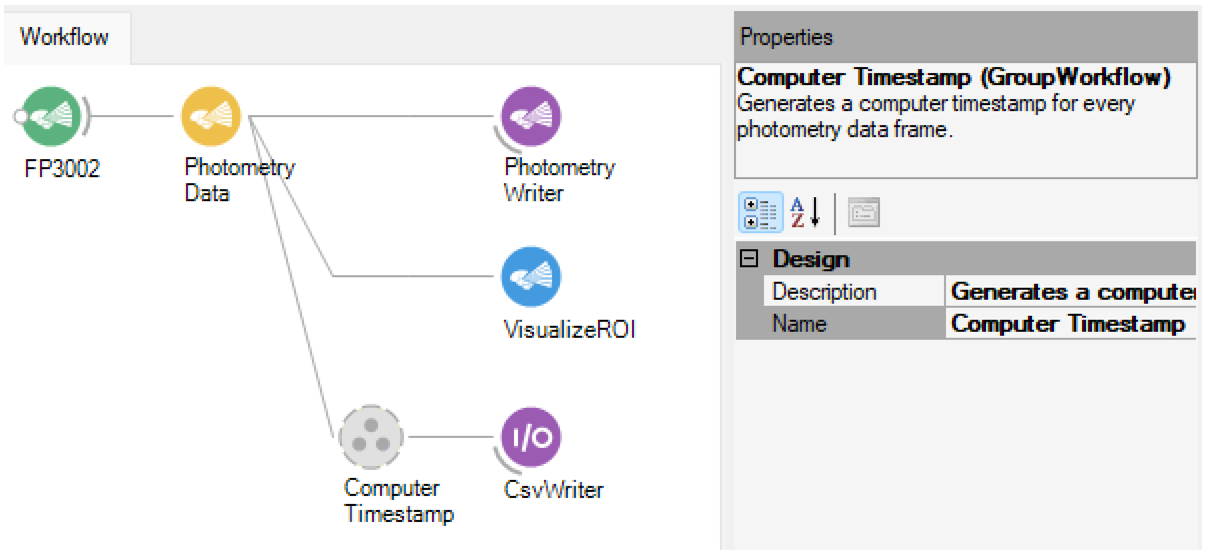
Now, we can combine these two outputs using a "Zip" node and connect the "Zip" node to a "CsvWriter" node. This way the photometry data stream will produce a seconds *.csv* file containing the computer timestamp of every frame. This new *.csv* file is readily alignable to the *.csv* file generated by the "Photometry Writer" node using the frame numbers in both files.



To add a bit of organization to this workflow, we can select all of the computer timestamp nodes, from the "Timestamp" node to the "Zip" node, and group them into a single grouped workflow. Do this by right clicking the selected nodes and clicking "Group → GroupWorkflow".



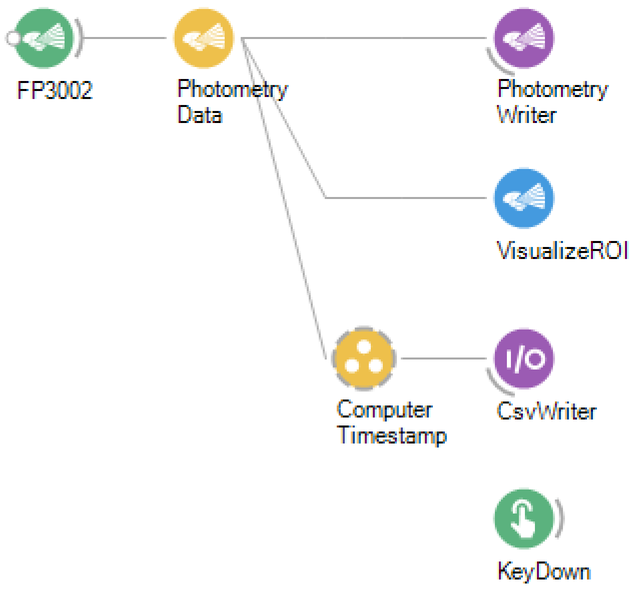
Be sure to name and provide a description of the new grouped workflow. You can do this within the properties panel of the grouped workflow.



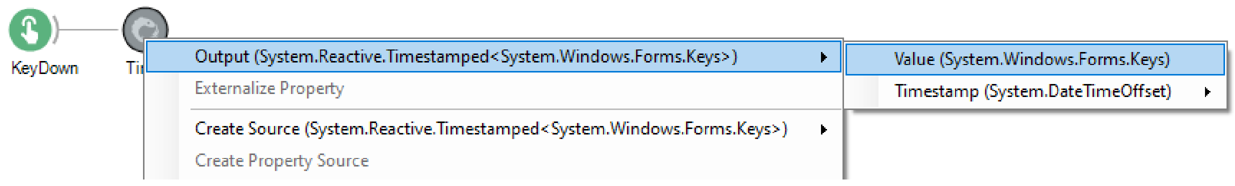
Computer Timestamp (GroupWorkflow)
Generates a computer timestamp for every photometry data frame.

Design	
Description	Generates a compute
Name	Computer Timestamp

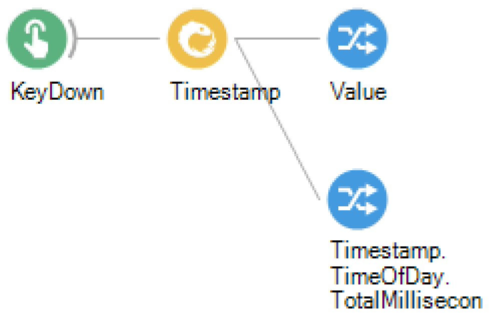
Now that every photometry data frame is timestamped using both the system's clock and the computer's clock, we can now timestamp every other data stream using the computer's clock. Since the computer's clock is being shared, all of the computer timestamps generated for every data stream are comparable to each other. Let's show this with a behavioral camera data stream and a keystroke data stream. First, create a keystroke source node using the "KeyDown" node.



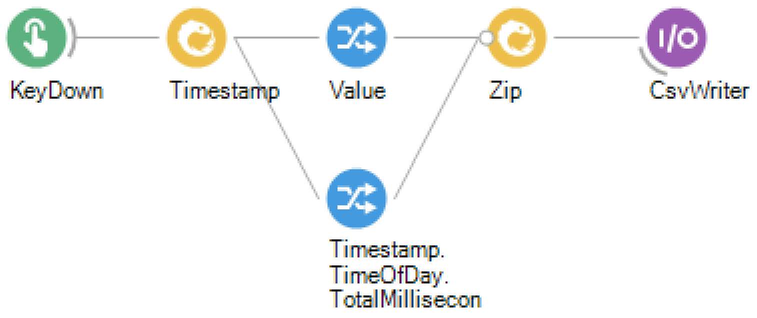
Connect the “KeyDown” node to a “Timestamp” node. Then right click the “Timestamp” node and output the incoming value.


KeyDown
Ti
Output (System.Reactive.Timestamped<System.Windows.Forms.Keys>)
Externalize Property
Create Source (System.Reactive.Timestamped<System.Windows.Forms.Keys>)
Create Property Source
Value (System.Windows.Forms.Keys)
Timestamp (System.DateTimeOffset)

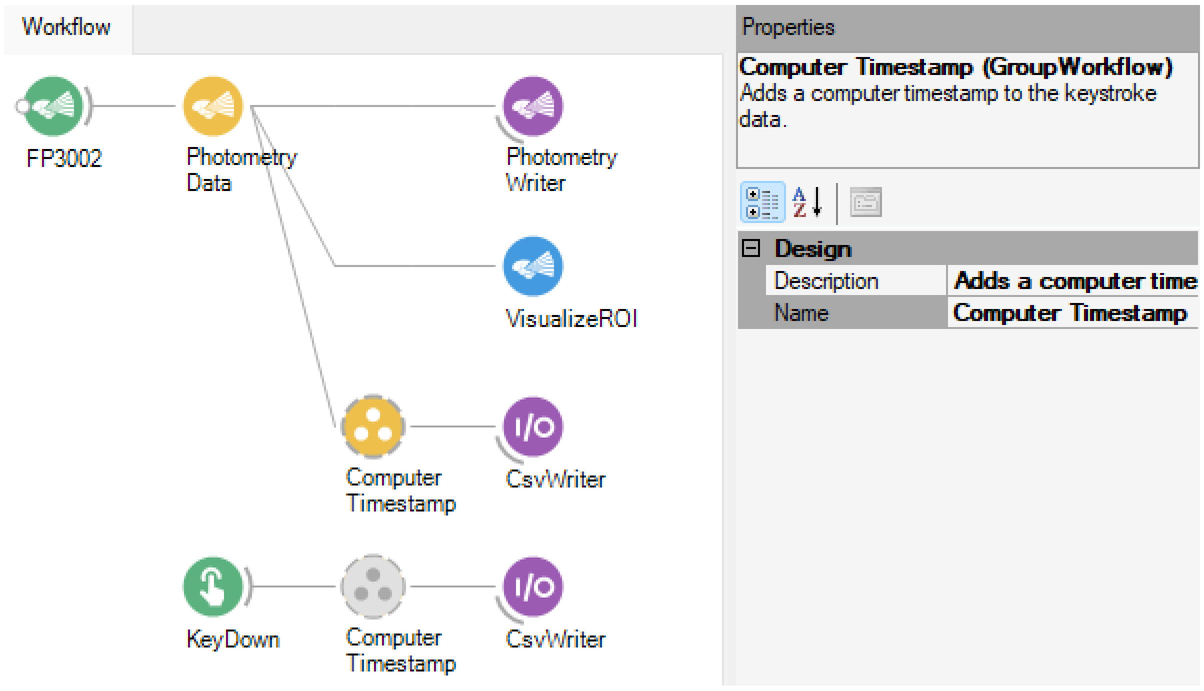
Then output the same type of timestamp that was used in the photometry data stream. In the example we used time of day, total milliseconds so we will use that here as well.



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 “Csv Writer” in the photometry data stream.



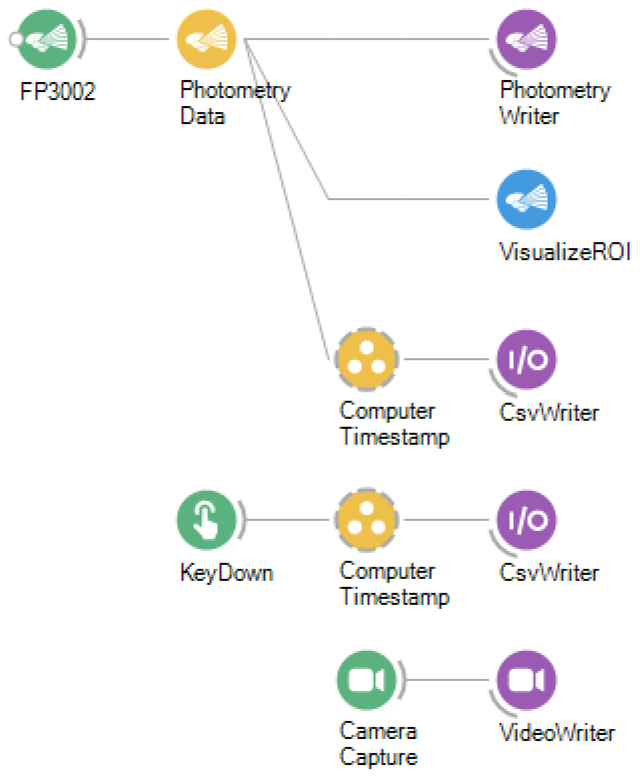
Here we will group together the computer timestamp nodes again to add a level of organization.



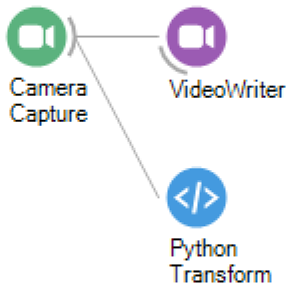
Computer Timestamp (GroupWorkflow)
Adds a computer timestamp to the keystroke data.

Design	
Description	Adds a computer time
Name	Computer Timestamp

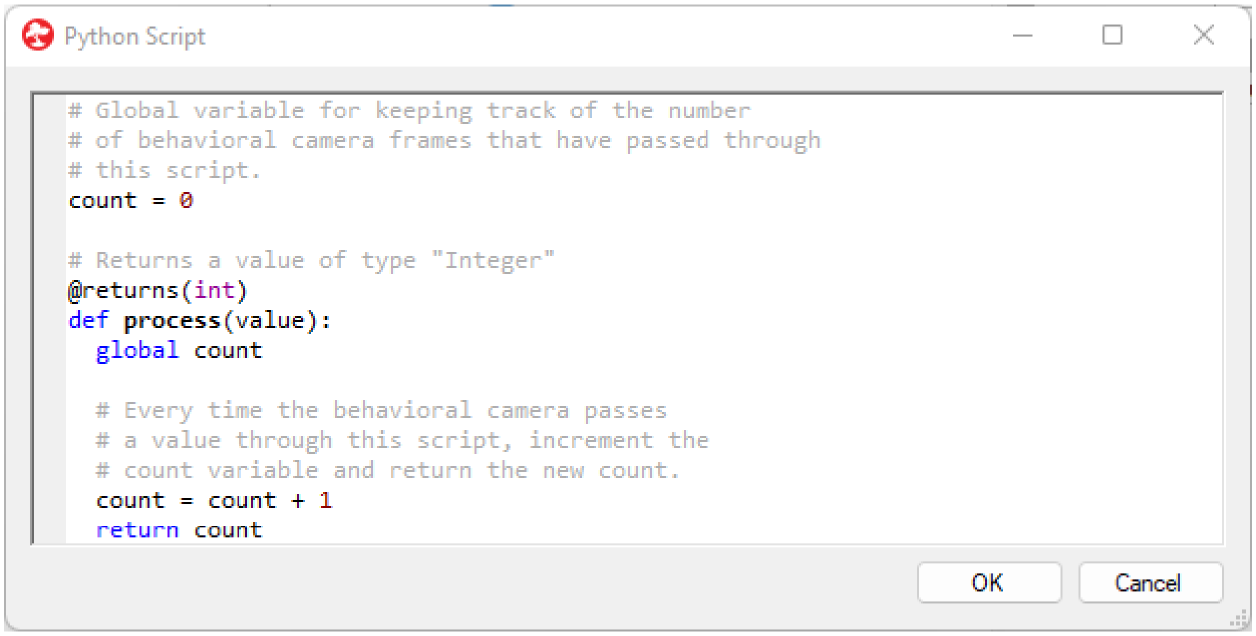
Next, let's add a behavioral camera data stream. This data stream will have to be treated differently because it makes use of the “Video Writer” node, meaning we cannot simply add a timestamp to the image data.



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.

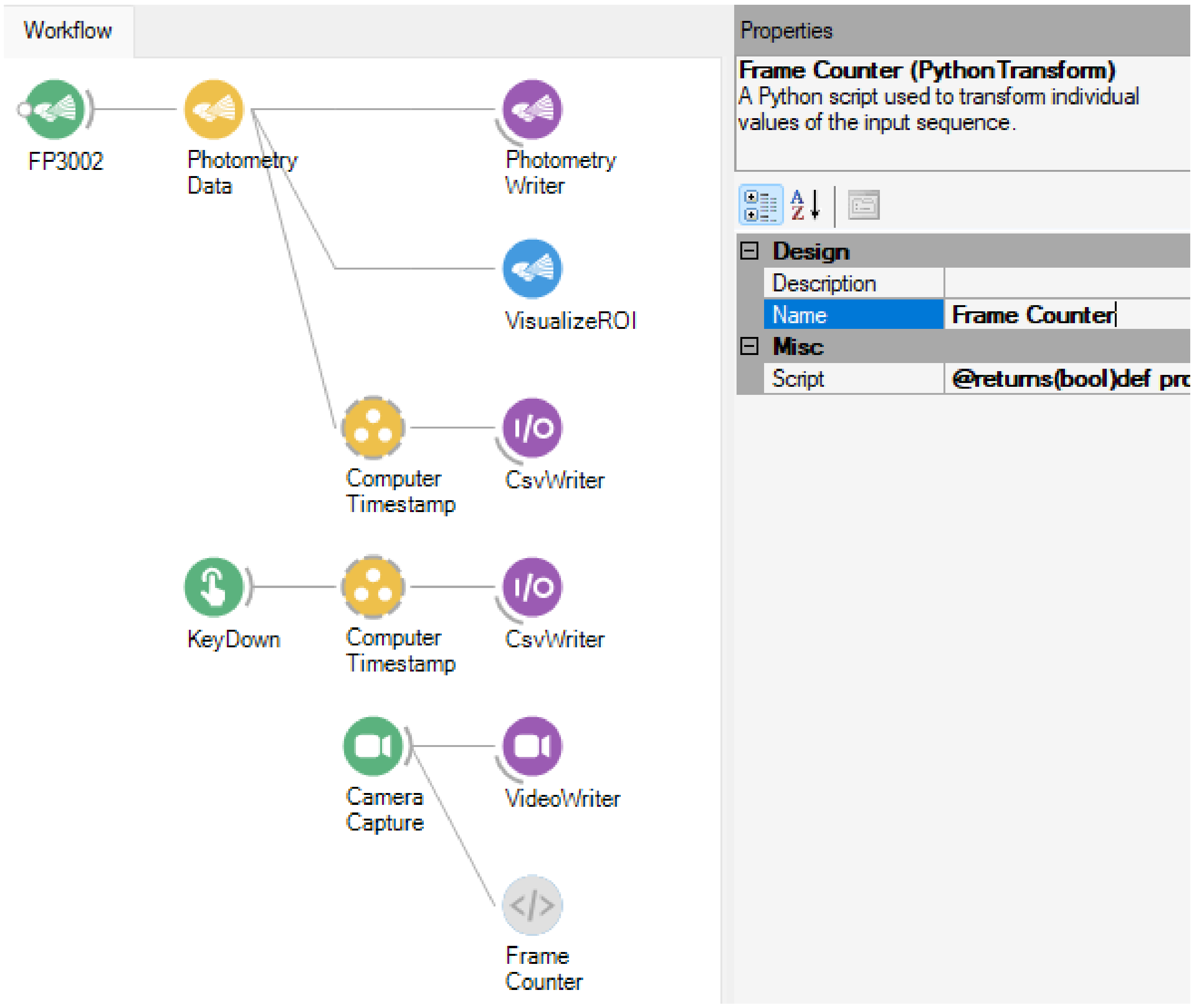


This "Python Transform" node will contain a python script that will count the number of behavior camera frames that have been generated since the start of the workflow. Declare a global variable to keep track of the number of frames that have occurred, then increment it every frame and return the new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



```
# 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
```

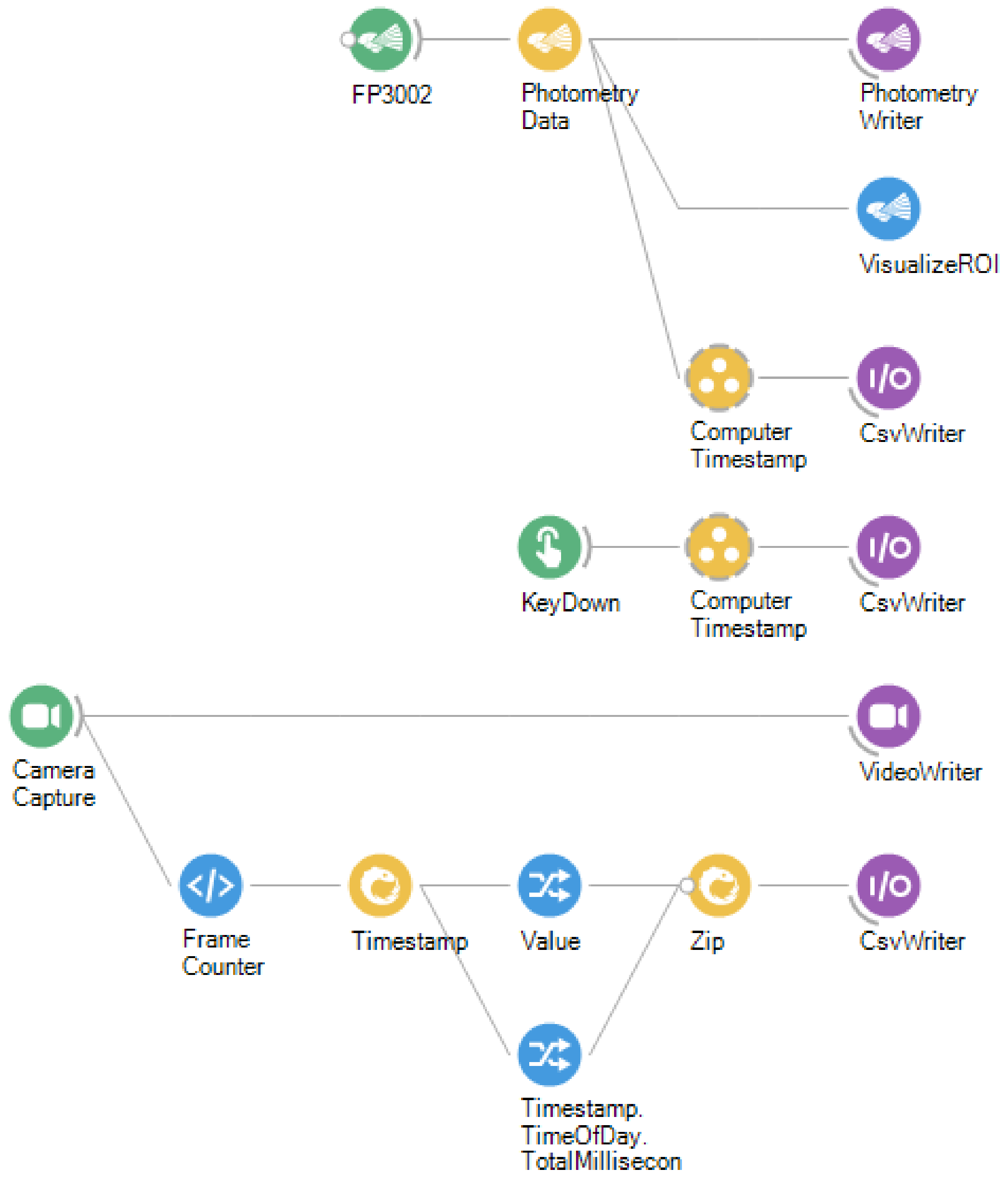
We can also provide a name for the python script from the "Python Transform" node's property panel.



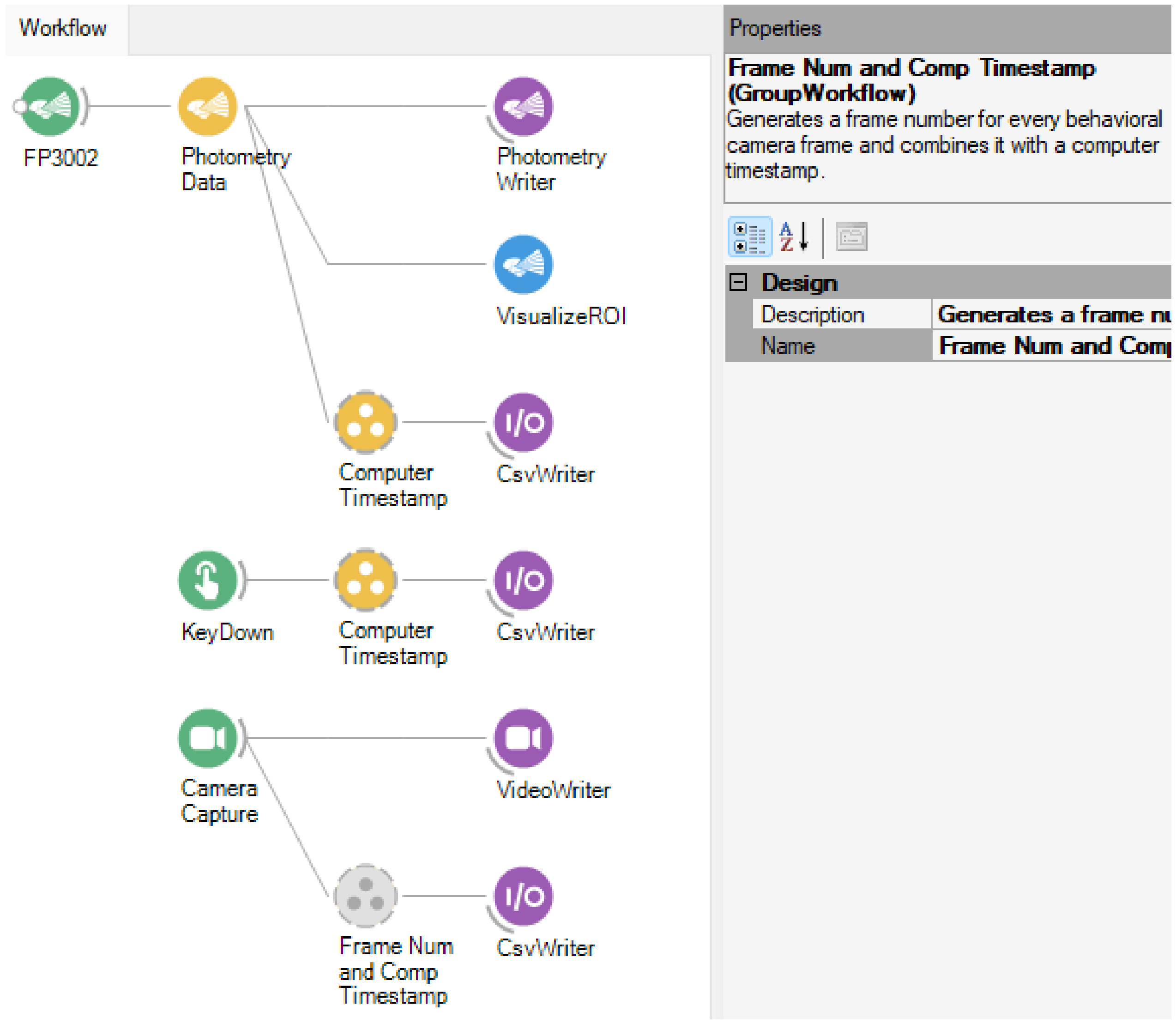
Frame Counter (PythonTransform)
A Python script used to transform individual values of the input sequence.

Design	
Description	
Name	Frame Counter
Misc	
Script	@returns(bool)def pro

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.



Frame Num and Comp Timestamp (GroupWorkflow)
Generates a frame number for every behavioral camera frame and combines it with a computer timestamp.

Design	
Description	Generates a frame nu
Name	Frame Num and Comp

We now have three major data streams. The photometry data stream will save two *.csv* files, one containing the photometry data timestamped using the system's clock and the other containing the photometry data frame number timestamped using 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

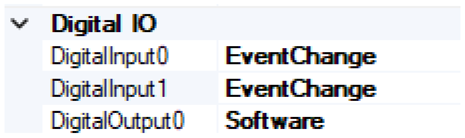
There are a variety of devices used within fiber photometry experiments that are not currently supported by a NuGet package usable within Bonsai. This adds a level of complexity for synchronizing with the photometry data stream. While designing an experiment there are some key factors that will influence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

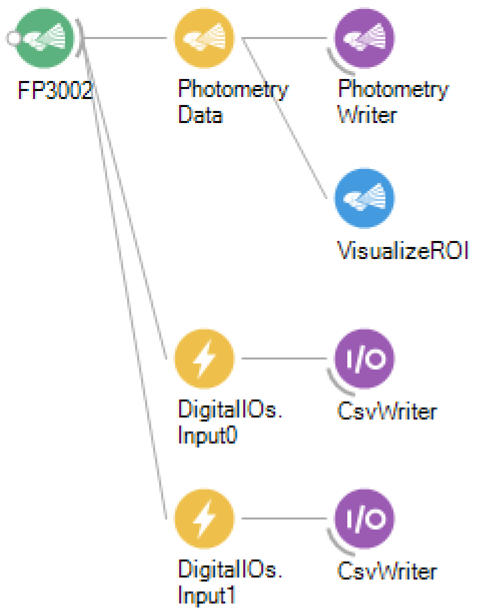
- One FP3002 system.
- One lever switch that produces a +5V digital voltage when triggered.
- One lickometer that produces a +5V digital voltage when licked.

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 are specified to "Event Change".



Digital IO	
DigitalInput0	EventChange
DigitalInput1	EventChange
DigitalOutput0	Software

With the digital input ports configured and connected in this way, we can use “Digital IOs” nodes to record and timestamp the signals from both external sensors using the system’s internal clock.

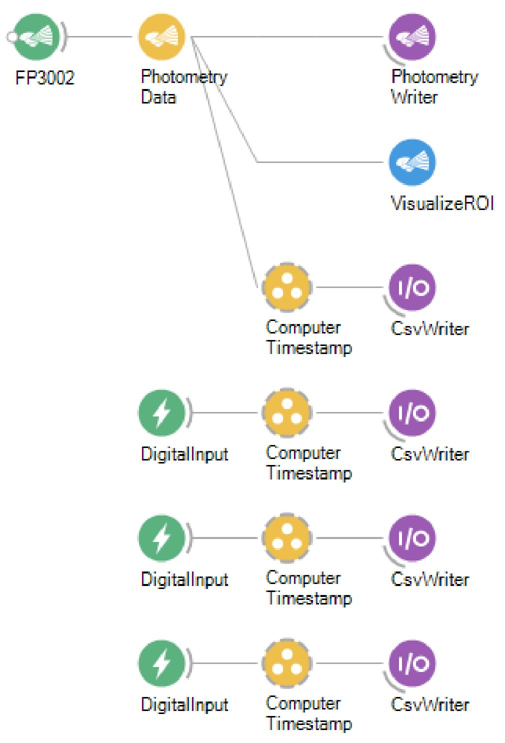


This example showcases that up to two external sensors can be synchronized with the photometry data stream so long as they output +5V digital signals. In the following example, suppose we had an additional level switch. Here we have three external sensors: two lever switches and the lickometer, all of 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:

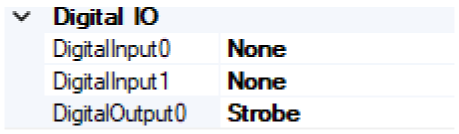
1. Visit the GitHub link: https://github.com/firmata/arduino
2. Download Zip (in the "Code" drop down menu in the upper right).
3. Open up Arduino IDE.
4. Click Sketch -> Include Library -> Add .ZIP Library
5. Click on the .ZIP file you just downloaded (should be in the downloads folder)
6. Open the "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.

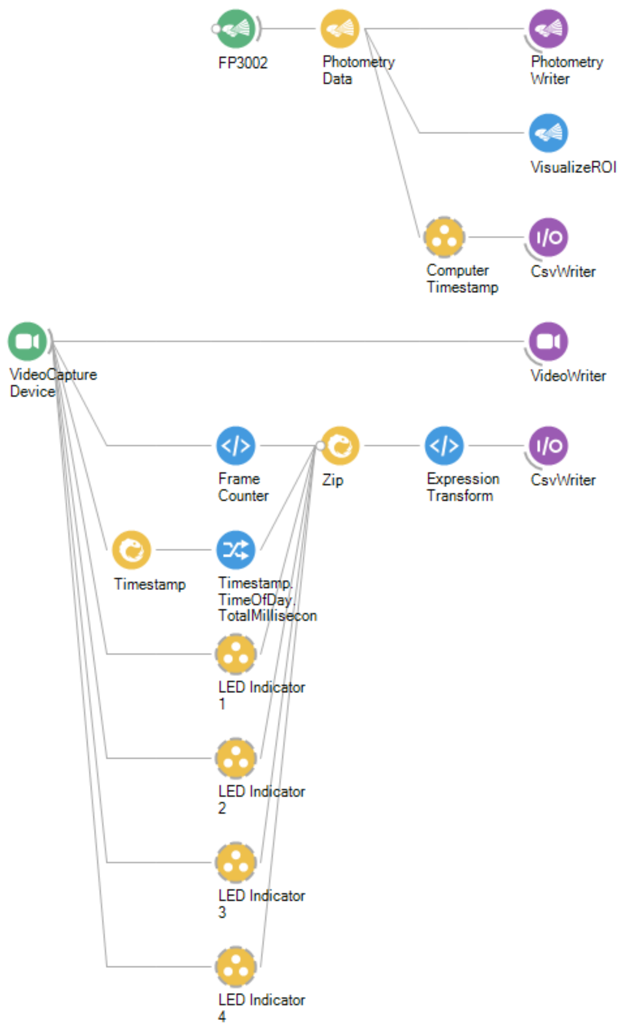


Digital IO	
DigitalInput0	None
DigitalInput1	None
DigitalOutput0	Strobe

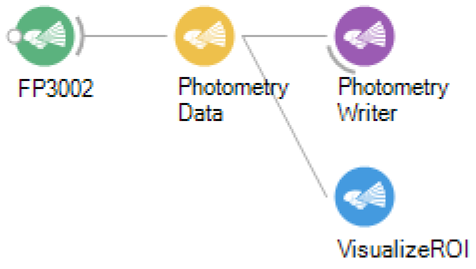
For more information on the options for configuring the digital input and output ports of the FP3002 system please visit the “FP3002” entry in the “Appendix I: Node Glossary”

Machine Vision

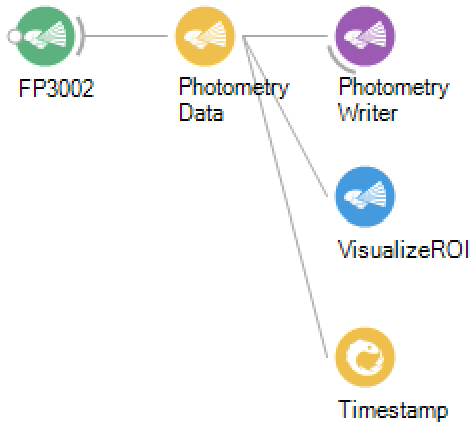
There are times when a device contains no support within Bonsai and is not compatible with +5V TTL signals. Most of the time, these devices have their own proprietary software for data acquisition. This can cause difficulty for synchronizing data sets produced by Bonsai with data sets produced by a different 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.



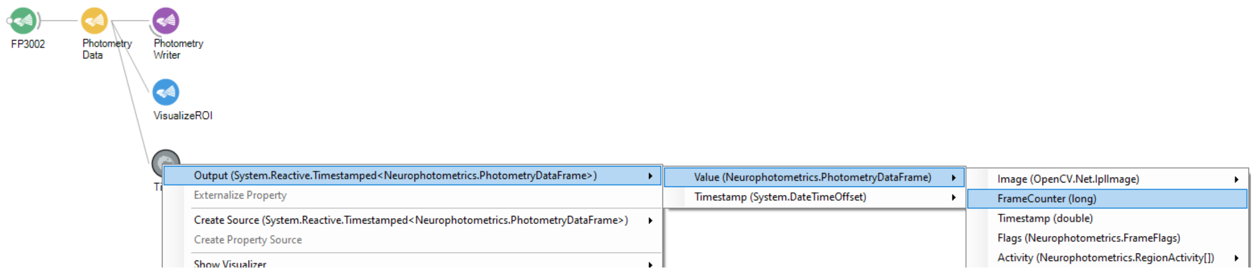
This workflow will timestamp each photometry data frame using the computer's clock, while using basic image processing techniques to read the state of the LED indicators using an external camera. To construct this workflow, begin with the standard photometry workflow.



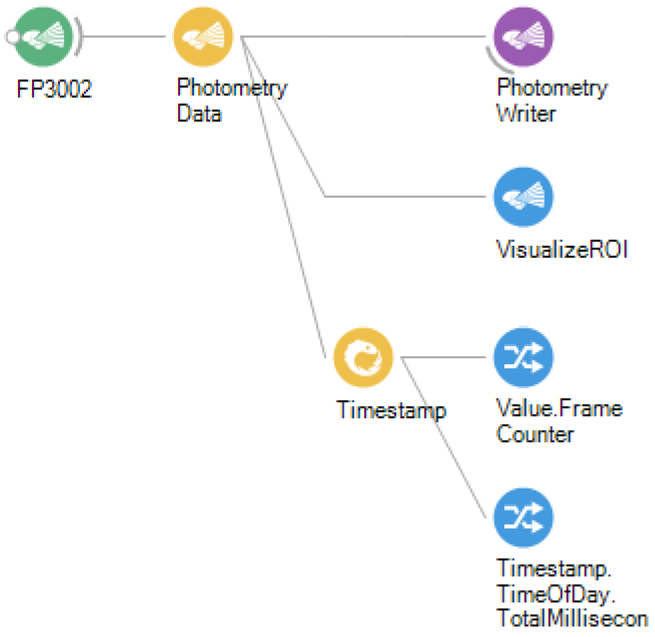
To add a computer clock timestamp to the photometry data stream, add a "Timestamp" node after the "Photometry Data" node, parallel to the "Photometry Writer" and "Visualize ROI" nodes.



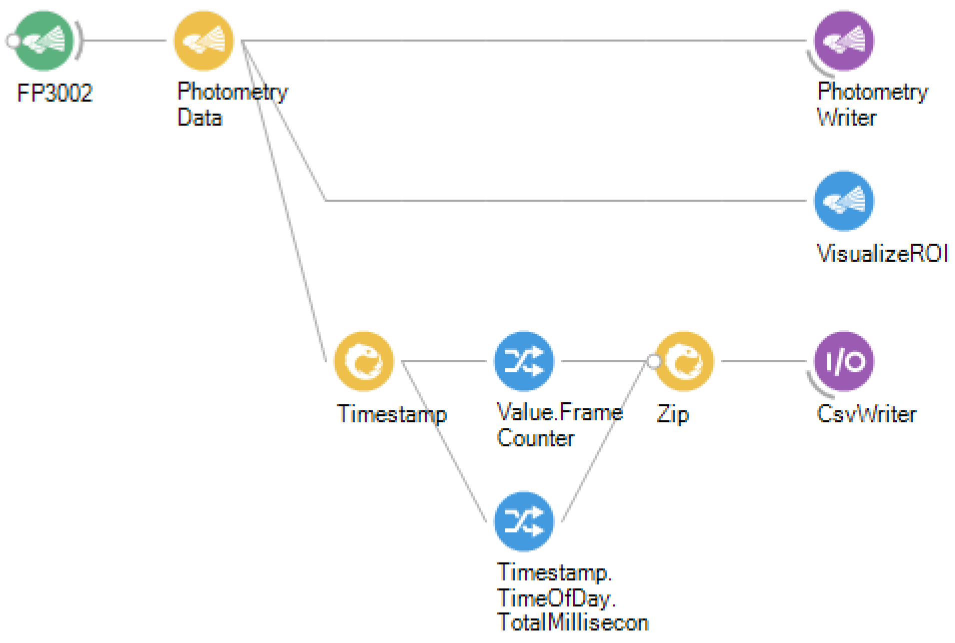
The "Timestamp" node will generate data of type "DateTimeOffset". This is a powerful .NET struct that contains both DateTime data and TimeSpan data. It also allows you to output data coming from the "Photometry Data" node. First, let's output the frame number from the "Timestamp" node by right clicking it and selecting "Output → Value → FrameCounter".


FP3002
Photometry Data
Photometry Writer
VisualizeROI
Output (System.Reactive.Timestamped<Neurophotometrics.PhotometryDataFrame>)
Externalize Property
Create Source (System.Reactive.Timestamped<Neurophotometrics.PhotometryDataFrame>)
Create Property Source
Show Visualizer
Value (Neurophotometrics.PhotometryDataFrame)
Timestamp (System.DateTimeOffset)
Image (OpenCV.Net.IplImage)
FrameCounter (long)
Timestamp (double)
Flags (Neurophotometrics.FrameFlags)
Activity (Neurophotometrics.RegionActivity[])

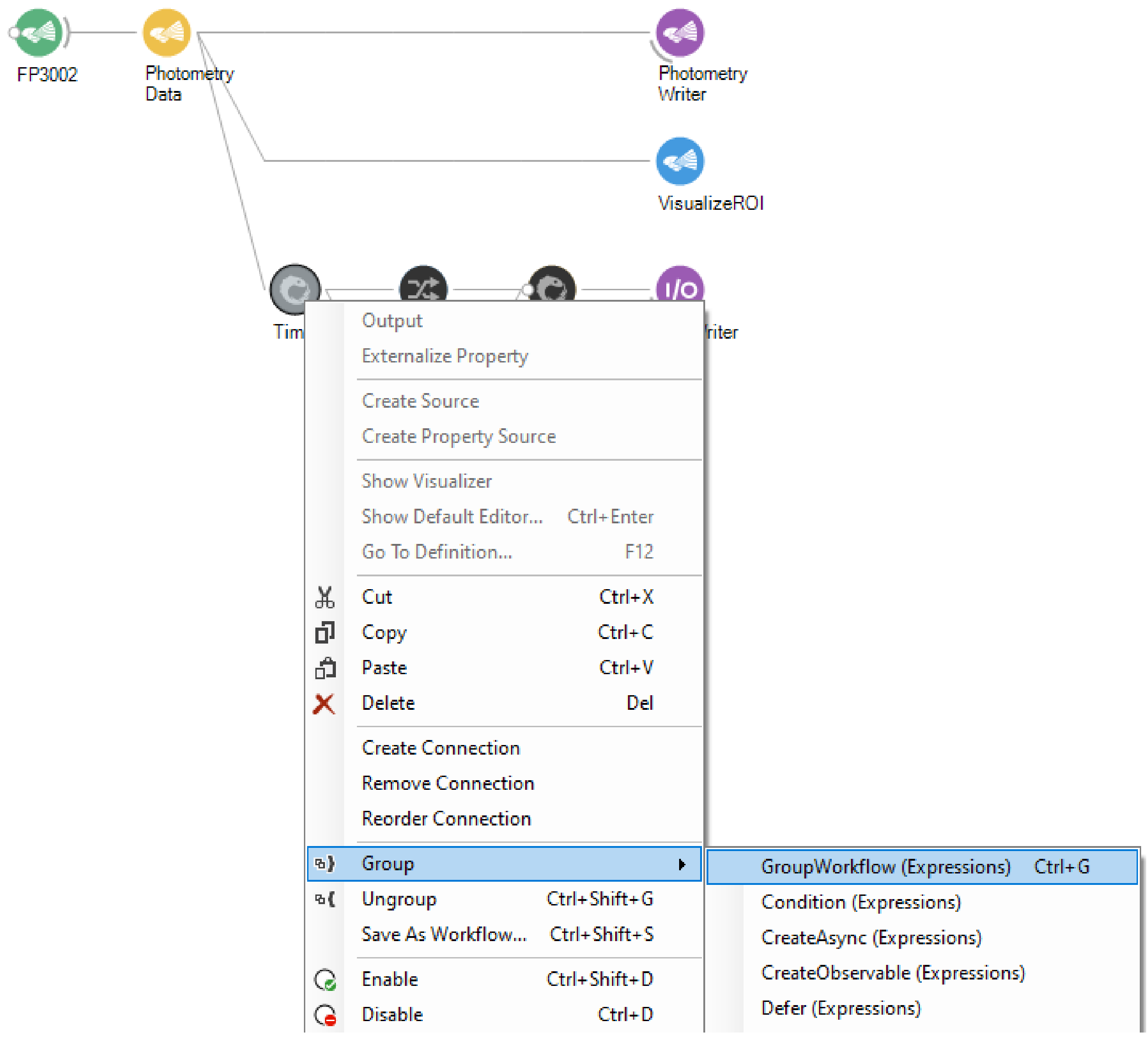
Then we will also output the desired timestamp. Here we have many options for the type of timestamp. One of the most commonly used timestamps is the time of day, total milliseconds. This will generate a timestamp with units of milliseconds since midnight in the computer's time zone. To output this type of timestamp, right click the "Timestamp" node and select "Output → Timestamp → TimeOfDay → TotalMilliseconds".



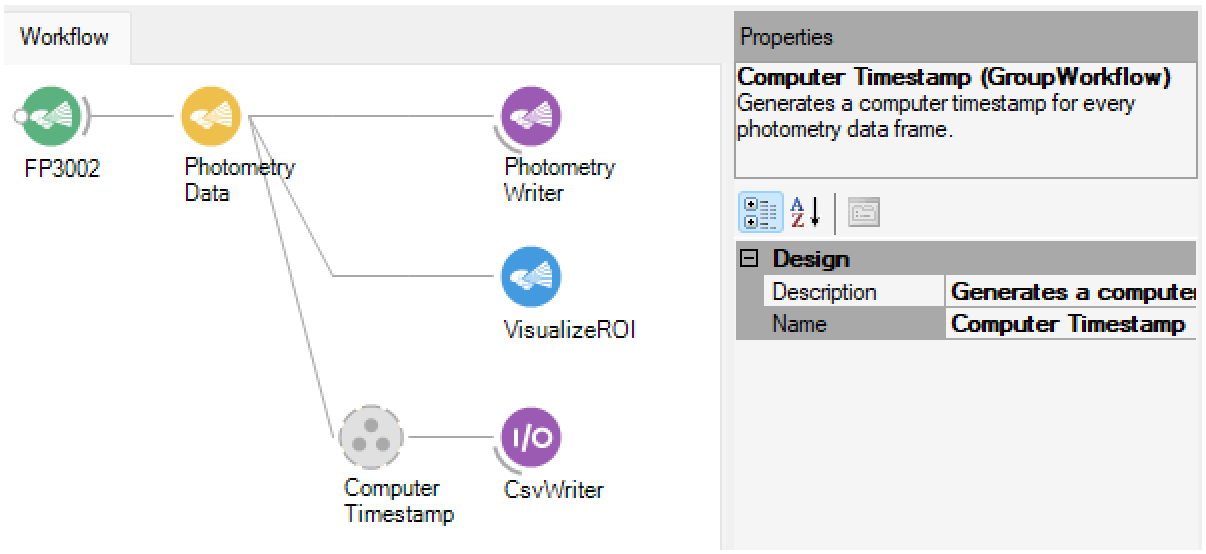
Now, we can combine these two outputs using a "Zip" node and connect the "Zip" node to a "CsvWriter" node. This way the photometry data stream will produce a seconds *.csv* file containing the computer timestamp of every frame. This new *.csv* file is readily alignable to the *.csv* file generated by the "Photometry Writer" node using the frame numbers in both files.



To add a bit of organization to this workflow, we can select all of the computer timestamp nodes, from the “Timestamp” node to the “Zip” node, and group them into a single grouped workflow. Do this by right clicking the selected nodes and clicking “Group → GroupWorkflow”.



Be sure to name and provide a description of the new grouped workflow. You can do this within the properties panel of the grouped workflow.

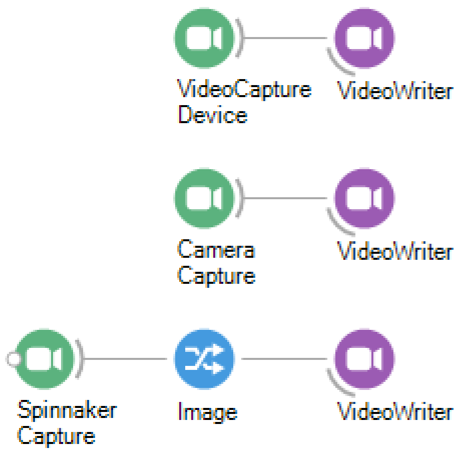


Computer Timestamp (GroupWorkflow)
Generates a computer timestamp for every photometry data frame.

Description	Generates a compute
Name	Computer Timestamp

Now that the photometry data stream is constructed and organized, 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".

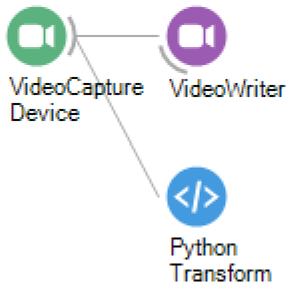
Create an external camera data stream by connecting the desired source node to a "Video Writer" node.

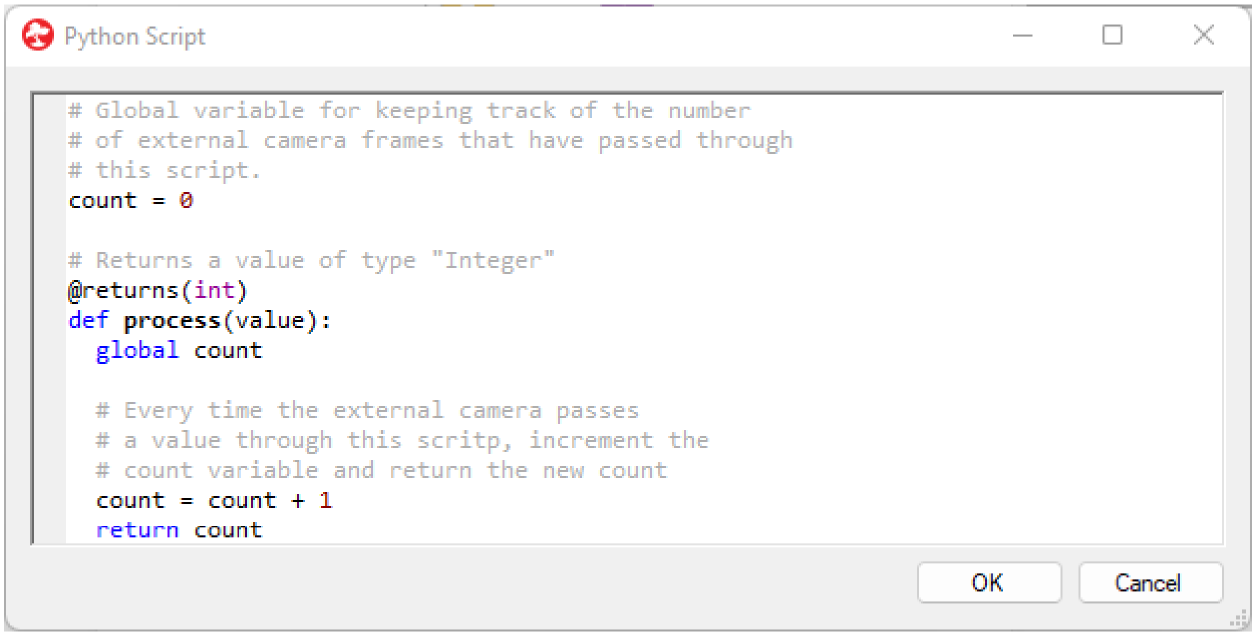


Some configuration is available for the capture nodes and the "Video Writer" node. All three capture nodes have the option to specify the camera's index. For the "Camera Capture" and "Video Capture Device" nodes, the internal camera on the "FP3002" system will not be recognized, so if only one external 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.

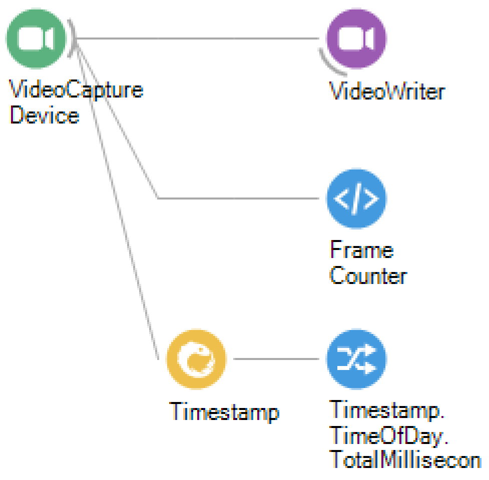
With the source node and the video writer configured, it is time to implement our machine vision algorithm for tracking the state of the LED indicators. We will begin by generating a frame number for each external camera frame. This process will be done in parallel to the “Video Writer” node. Insert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



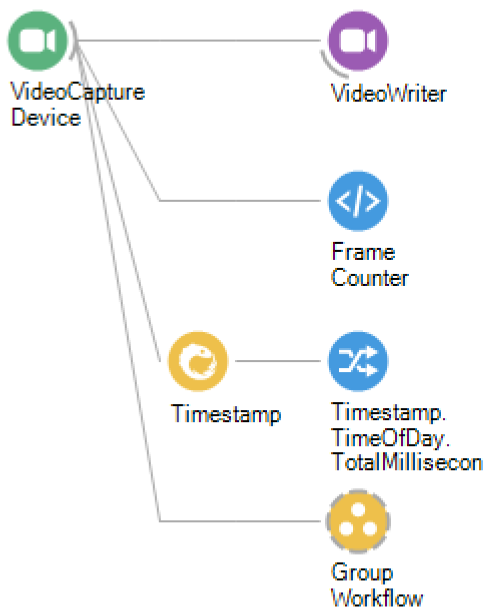


```
# Global variable for keeping track of the number
# of external camera frames that have passed through
# this script.
count = 0

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

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

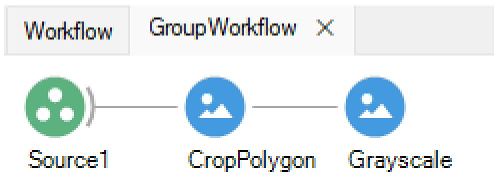
It is useful to rename the “Python Transform” node to indicate its purpose. Here we will rename it to “Frame Counter”. Parallel to the “Frame Counter” node we will implement a computer timestamp so that we can synchronize this data stream with the photometry data stream. Here we will use a “Timestamp” node that only outputs the time of day, total milliseconds timestamp. This is slightly different from the method we used to timestamp the photometry data frame since we will be combining more than just the frame number and the timestamp together.



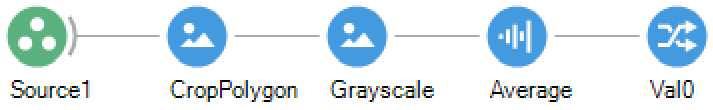
Next we will create the image processing algorithm for tracking the state of a single LED indicator. This algorithm will crop the incoming image to a single LED indicator, convert the cropped image to grayscale, find the average pixel value of the grayscale image, and output a boolean value indicating 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.



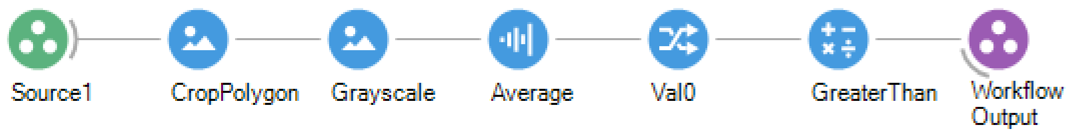
Inside of the grouped workflow, begin the image processing algorithm with a “Crop Polygon” node followed by a “Grayscale” node. This will generate a grayscale image, cropped to a single LED indicator.



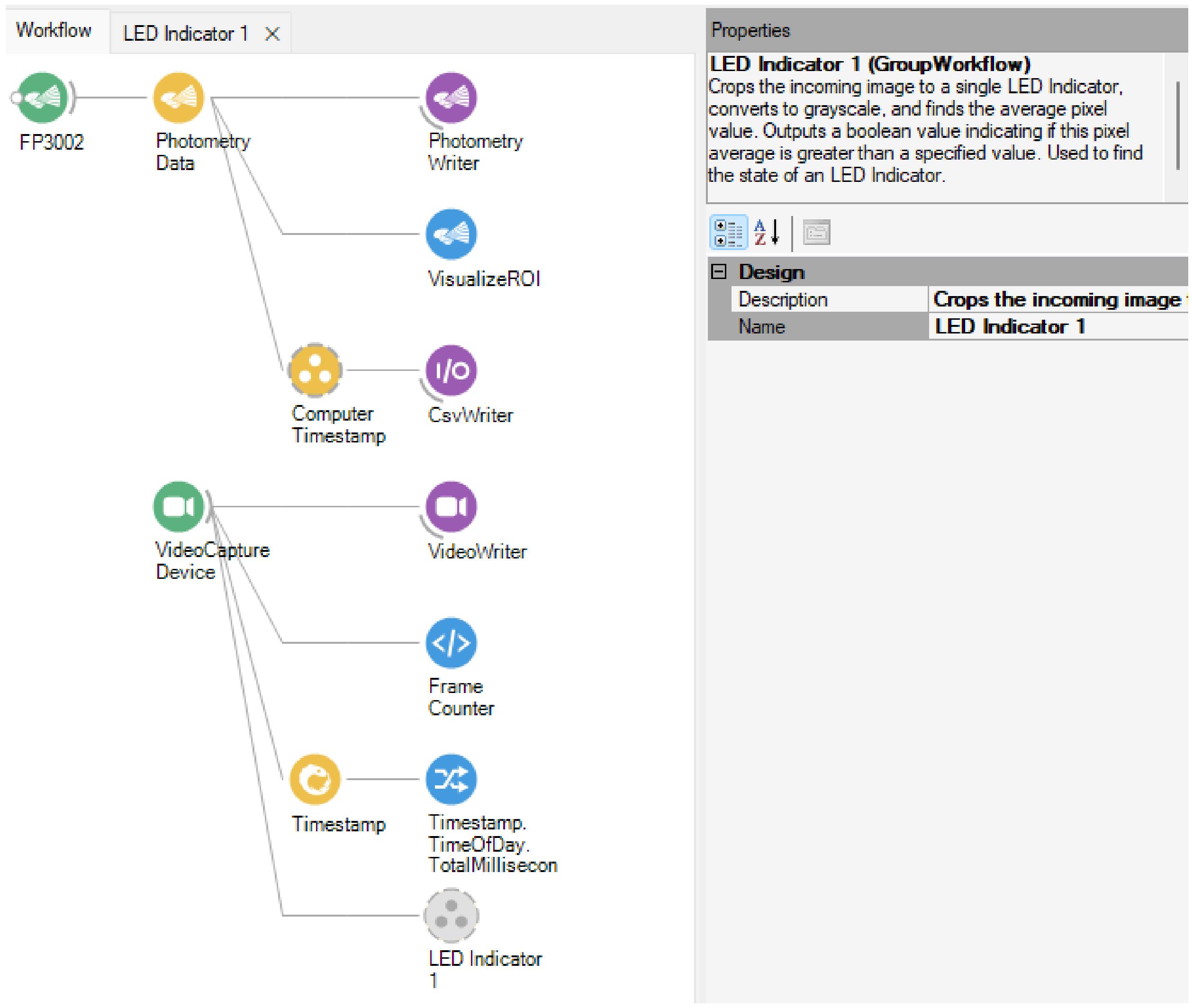
After the “Grayscale” node, add an “Average” node and output the “Val0” element from it. This will average the pixel values and output the blue component. Since the incoming image is grayscale, all color components are equal. Thus, outputting the “Val0” element works to convert the data type from “OpenCV.Net.Scalar”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



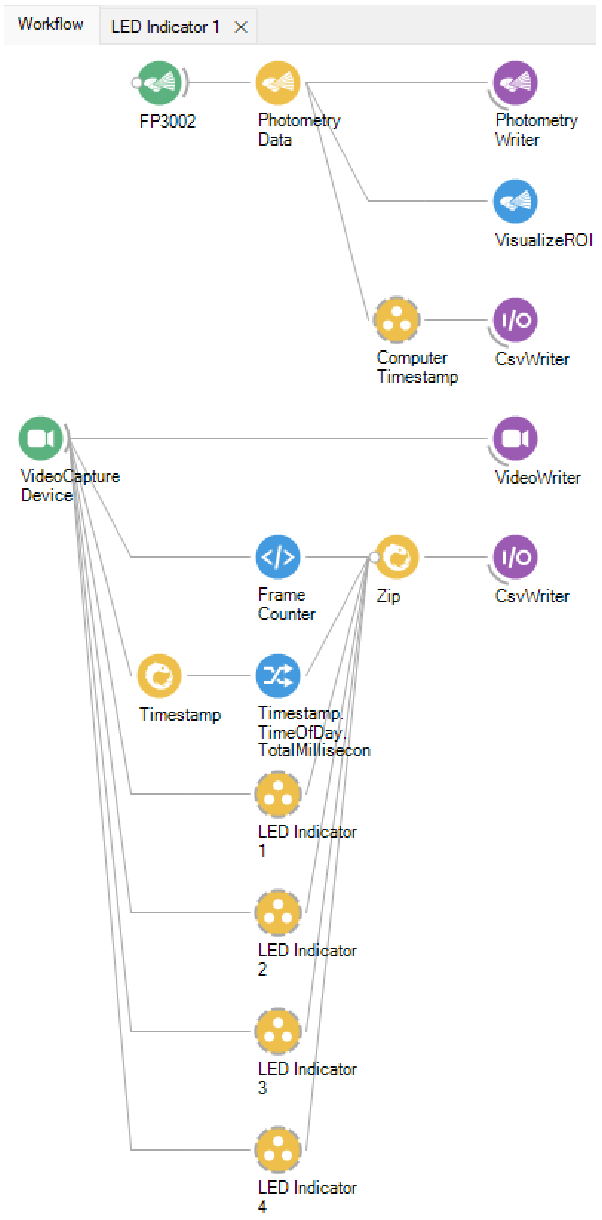
Be sure to give the grouped workflow a unique name and a description.



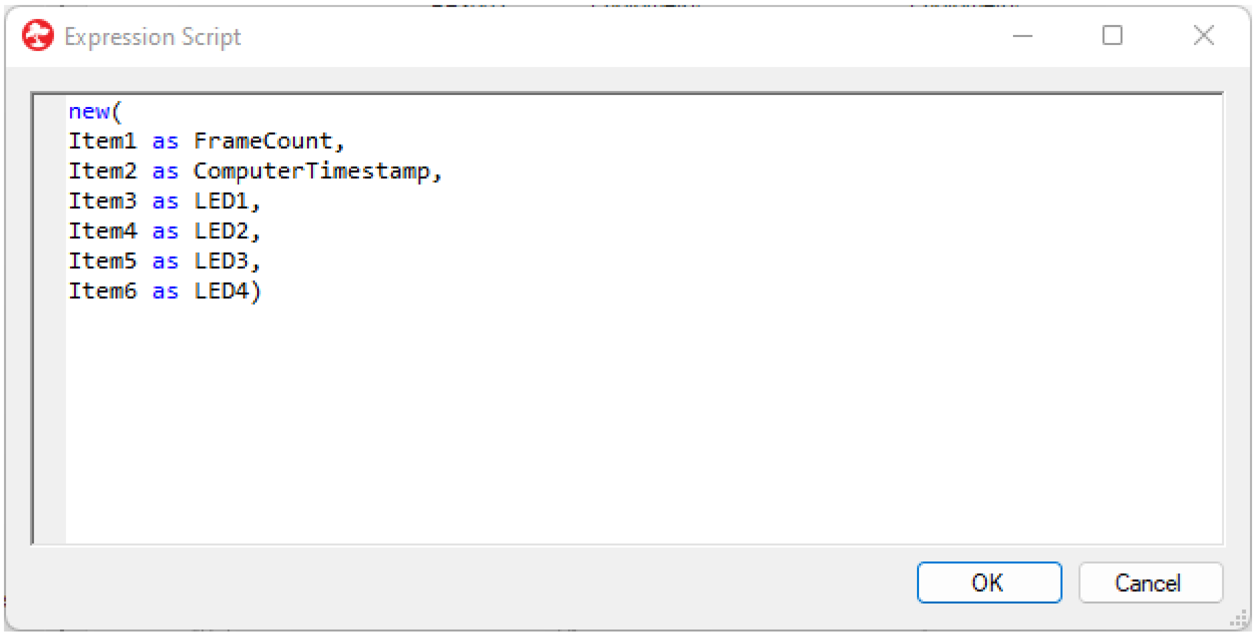
LED Indicator 1 (GroupWorkflow)
Crops the incoming image to a single LED Indicator, converts to grayscale, and finds the average pixel value. Outputs a boolean value indicating if this pixel average is greater than a specified value. Used to find the state of an LED Indicator.

Design	
Description	Crops the incoming image
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.

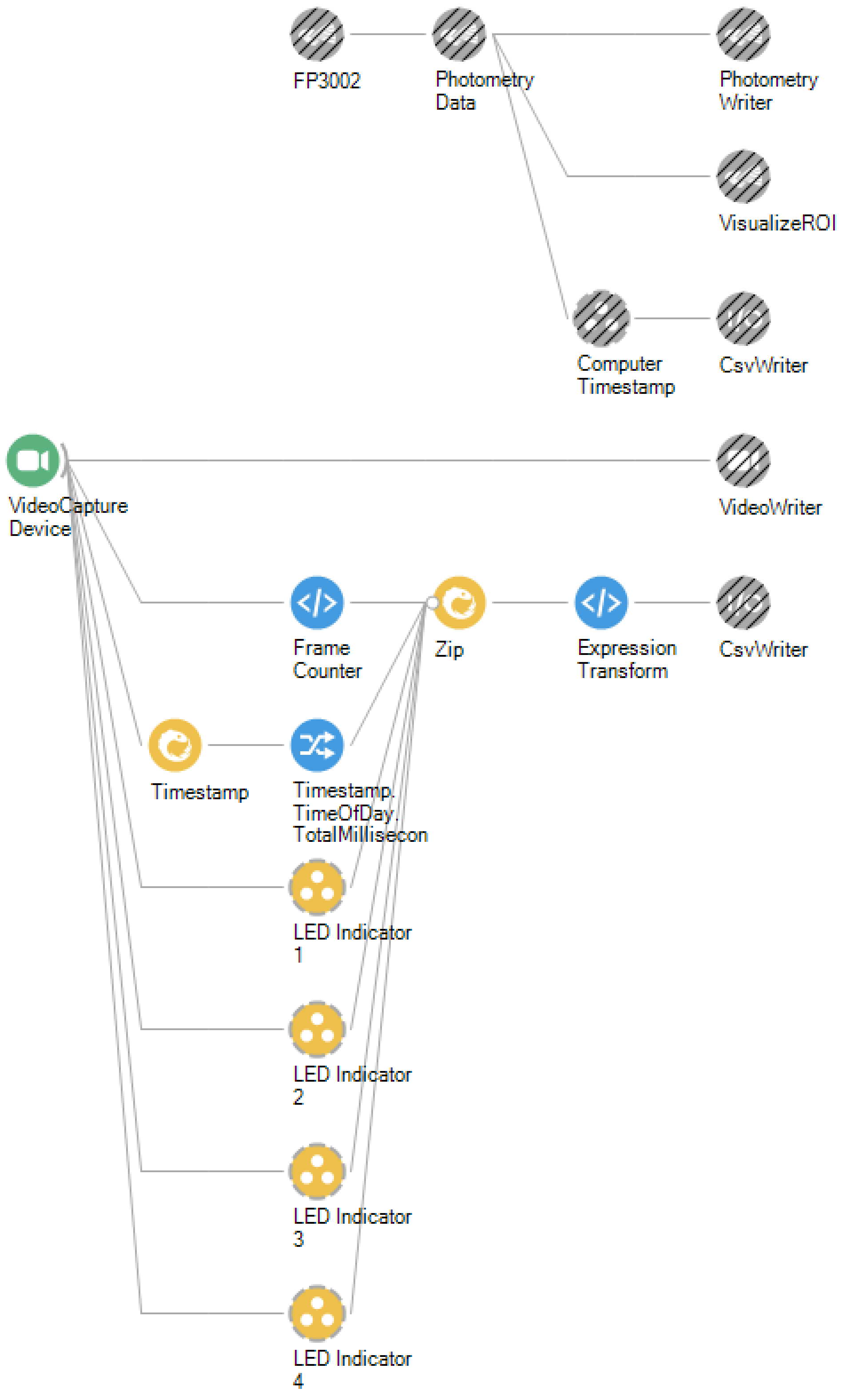


We can also add headers to the .csv file by inserting an “Expression Transform” node immediately after the “Zip” node with the following script. Be sure that the “Include Header” property of the “Csv Writer” node is set to “True”.

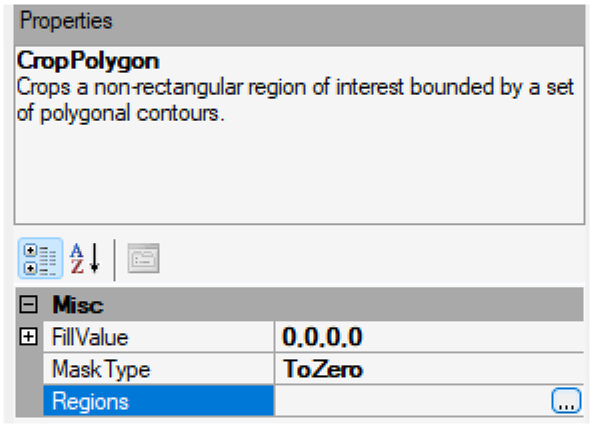


```
new(
Item1 as FrameCount,
Item2 as ComputerTimestamp,
Item3 as LED1,
Item4 as LED2,
Item5 as LED3,
Item6 as LED4)
```

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”.


FP3002
Photometry Data
Photometry Writer
VisualizeROI
Computer Timestamp
CsvWriter
VideoCapture Device
VideoWriter
Frame Counter
Zip
Expression Transform
CsvWriter
Timestamp
Timestamp.TimeOfDay.TotalMillisecon
LED Indicator 1
LED Indicator 2
LED Indicator 3
LED Indicator 4

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.

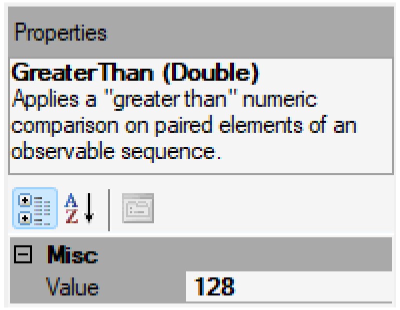

Properties

CropPolygon
Crops a non-rectangular region of interest bounded by a set of polygonal contours.

Misc

FillValue	0,0,0,0
MaskType	ToZero
Regions	...

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.


Properties

GreaterThan (Double)
Applies a "greater than" numeric comparison on paired elements of an observable sequence.

Misc

Value	128

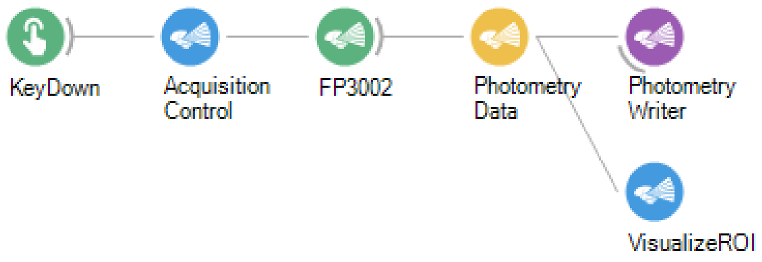
Once all of the “Crop Polygon” and “Greater Than” nodes are configured, stop the workflow and enable all of the nodes by pressing “CTRL + A”, to select everything, followed by “CTRL + Shift + D”, to enable everything selected.

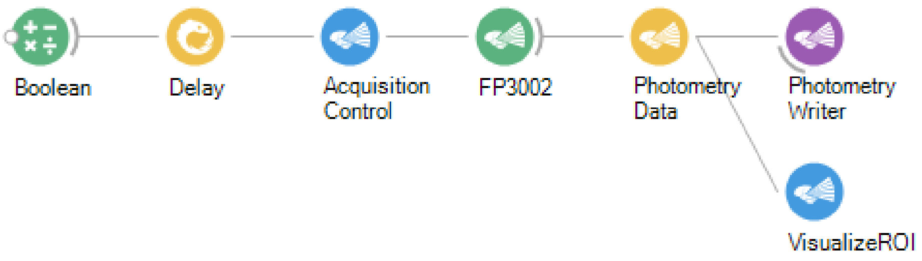
Chapter 4: Data Acquisition

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.

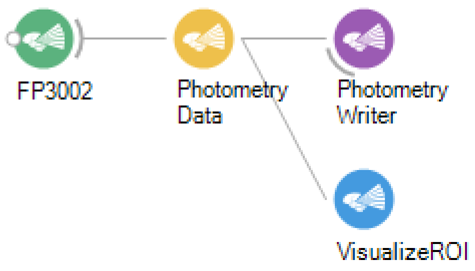
Delayed Start

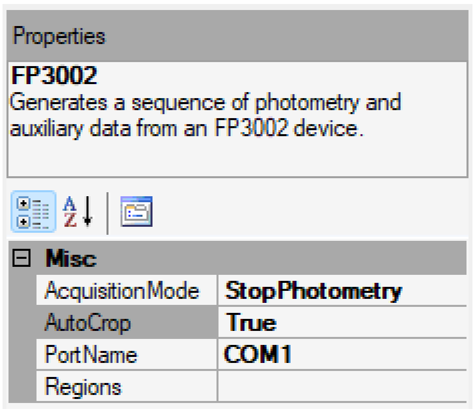
The “Standard Photometry” workflow can be expanded to allow for a delayed start to data acquisition. This example workflow makes use of the “Acquisition Control” node and a software trigger to command the FP3002 system to start data acquisition a period of time after the Bonsai workflow has been started. 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".





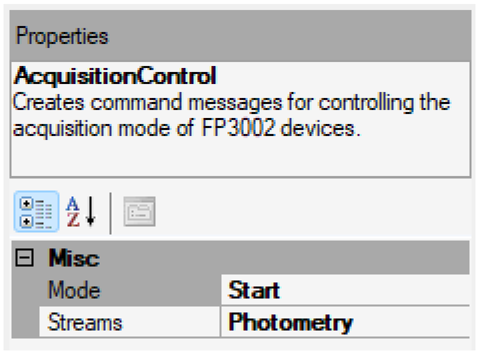
Properties

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

Misc

AcquisitionMode	**StopPhotometry**
AutoCrop	**True**
PortName	**COM1**
Regions	

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.



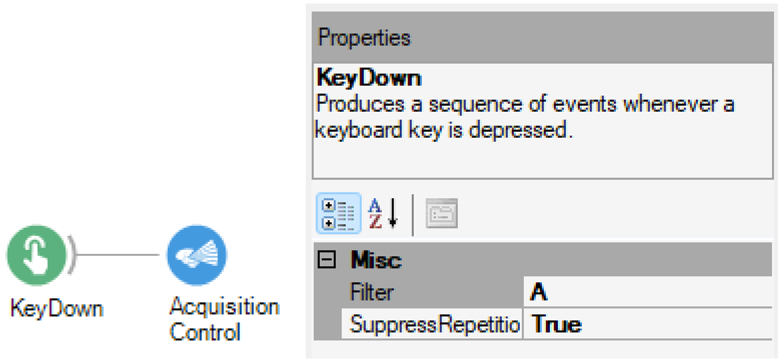
Properties

AcquisitionControl
Creates command messages for controlling the acquisition mode of FP3002 devices.

Misc

Mode	**Start**
Streams	**Photometry**

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.

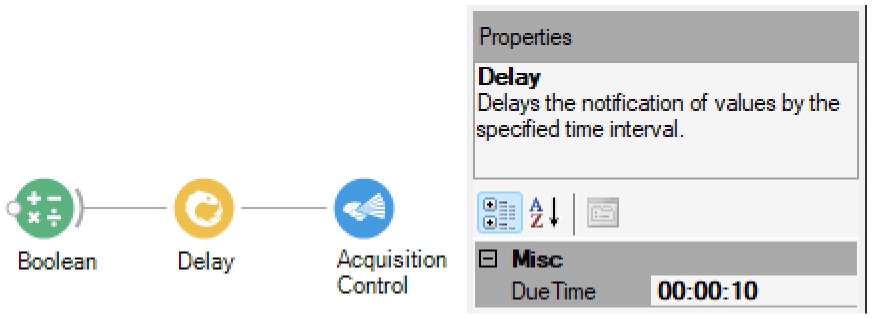


KeyDown
Produces a sequence of events whenever a keyboard key is depressed.

Misc

Filter	A
SuppressRepetitio	True

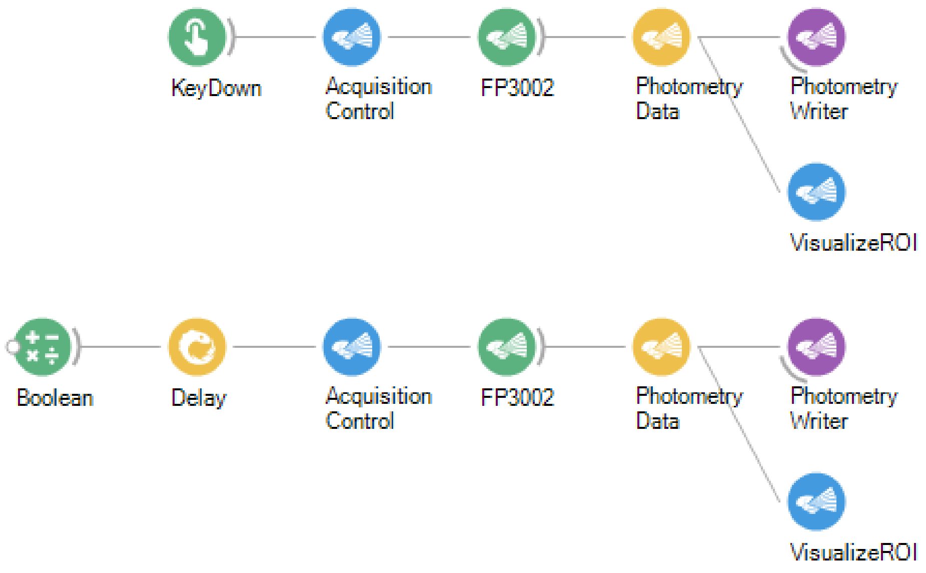
If instead, the desired delay to data acquisition is to be a precise amount of time since the workflow has started, a “Boolean” node followed by a “Delay (Reactive)” node as the software trigger would be more appropriate. The “Delay (Reactive)” node delays the notification of values by the specified 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



Delay
Delays the notification of values by the specified time interval.

Misc
DueTime 00:00:10

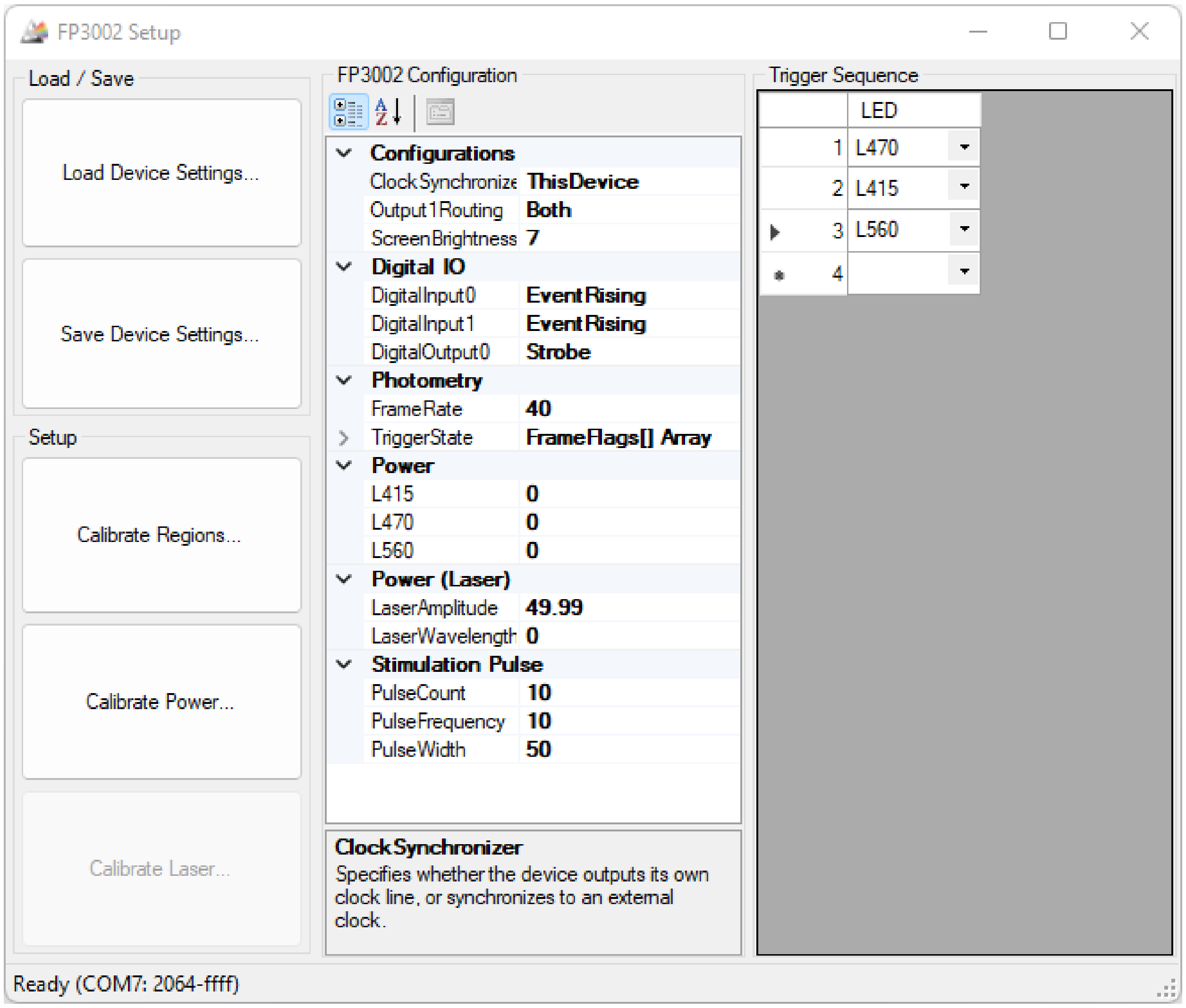
Once the desired software trigger is connected to the “Acquisition Control” node, the “Acquisition Control” node must be connected to the input of the “FP3002” node to complete the workflow. This will allow the “Start Photometry” command, created by the “Acquisition Control” node, to be sent to the “FP3002”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



NEUROPHOTOMETRICS
The fiber photometry company

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.



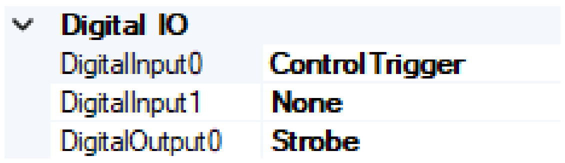
Configurations	
ClockSynchronize	ThisDevice
Output1Routing	Both
ScreenBrightness	7
Digital IO	
DigitalInput0	EventRising
DigitalInput1	EventRising
DigitalOutput0	Strobe
Photometry	
FrameRate	40
TriggerState	FrameFlags[] Array
Power	
L415	0
L470	0
L560	0
Power (Laser)	
LaserAmplitude	49.99
LaserWavelength	0
Stimulation Pulse	
PulseCount	10
PulseFrequency	10
PulseWidth	50

ClockSynchronizer
Specifies whether the device outputs its own clock line, or synchronizes to an external clock.

	LED
1	L470
2	L415
3	L560
4	

Ready (COM7: 2064-ffff)

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

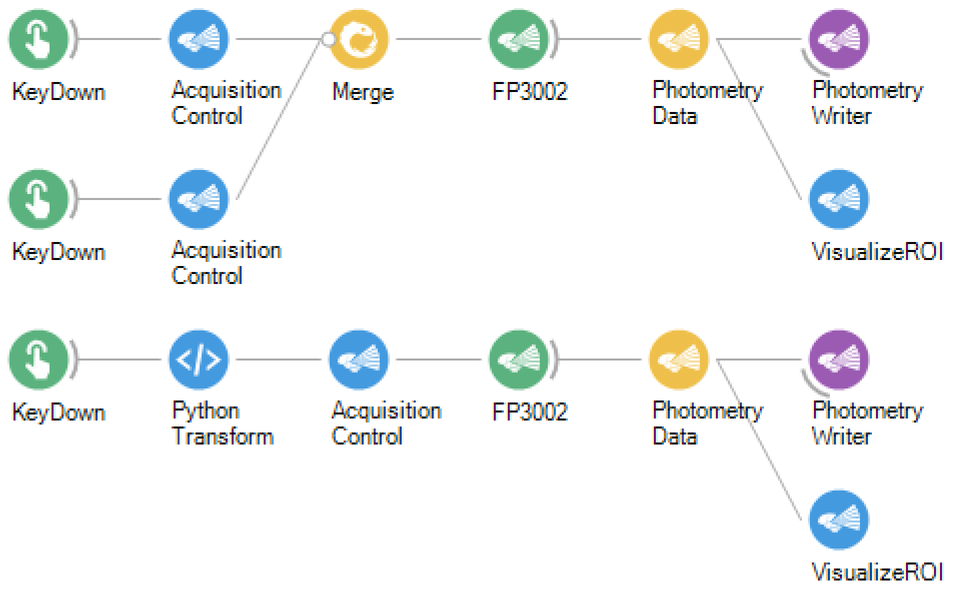


Digital IO	
DigitalInput0	Control Trigger
DigitalInput1	None
DigitalOutput0	Strobe

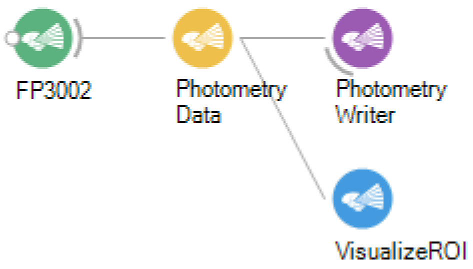
This will allow the +5V TTL signal on the specified port to control the data acquisition of the FP3002 system. While the TTL signal is HIGH, the system will be acquiring data frames and while the TTL signal is LOW, the system will not be acquiring data frames.

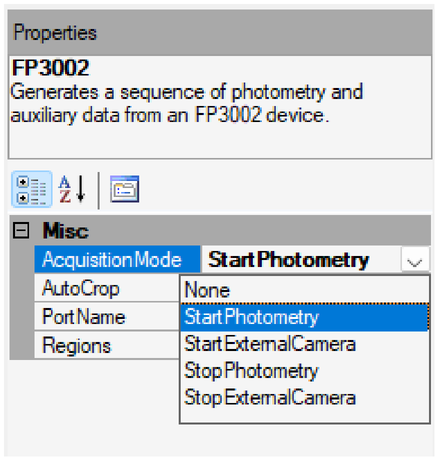
Manual Control

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”.



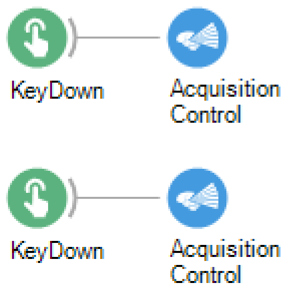

Properties

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

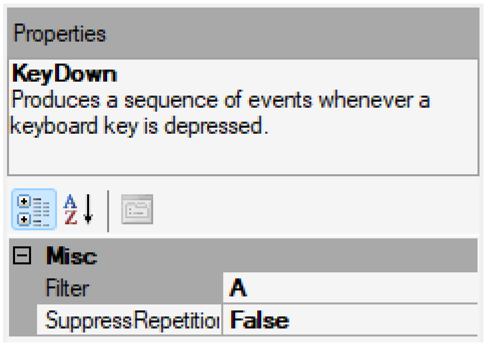
Misc

AcquisitionMode	**StartPhotometry**
AutoCrop	
PortName	
Regions	

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.

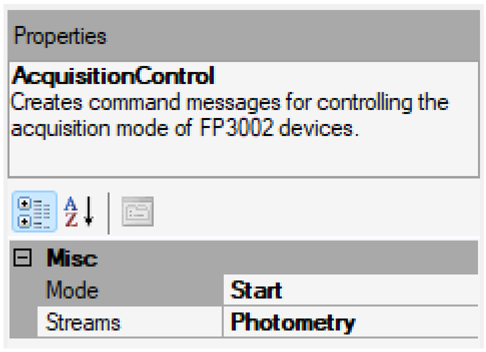


Properties

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

Misc

Filter	A
SuppressRepetitio	False

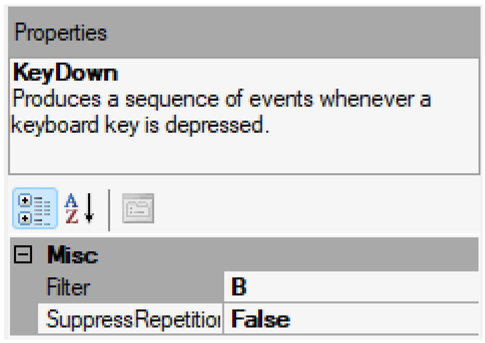


Properties

AcquisitionControl
Creates command messages for controlling the acquisition mode of FP3002 devices.

Misc

Mode	Start
Streams	Photometry

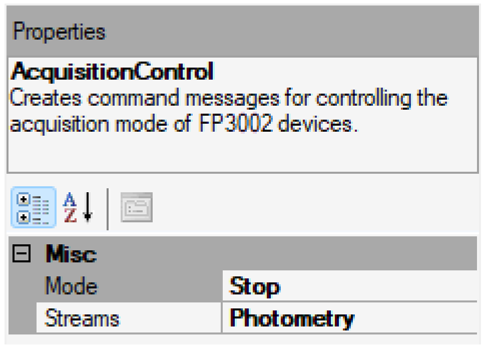


Properties

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

Misc

Filter	B
SuppressRepetitio	False



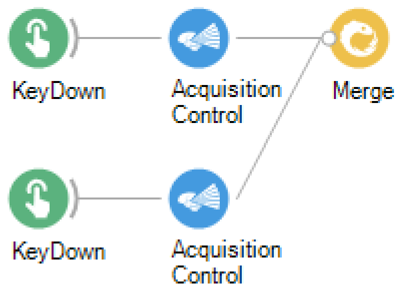
Properties

AcquisitionControl
Creates command messages for controlling the acquisition mode of FP3002 devices.

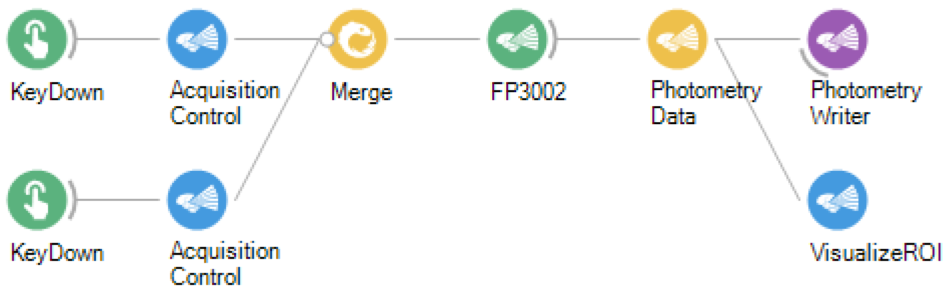
Misc

Mode	Stop
Streams	Photometry

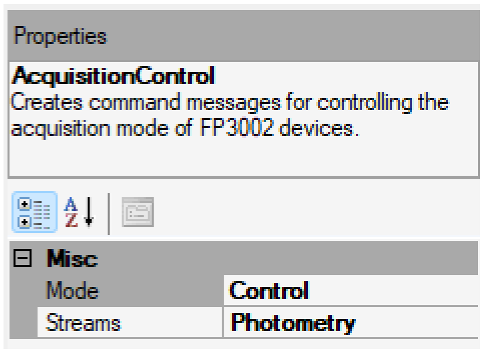
Currently, there are two data streams for producing commands to be sent to the "FP3002" node, one producing the "Start Photometry" command when the "A" key is pressed and the other producing the "Stop Photometry" command when the "B" key is pressed. These two data streams need to be converted to a single 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.

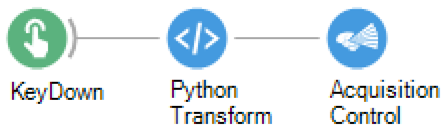

Properties

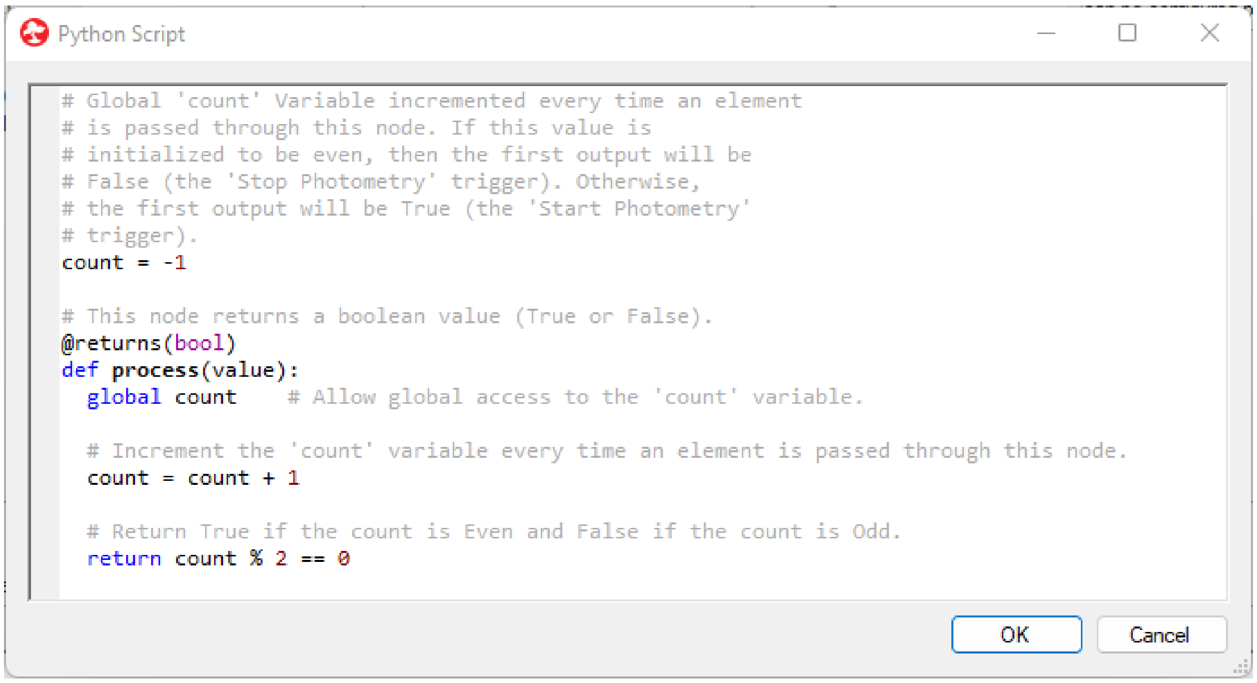
AcquisitionControl
Creates command messages for controlling the acquisition mode of FP3002 devices.

Misc

Mode	**Control**
Streams	**Photometry**

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.



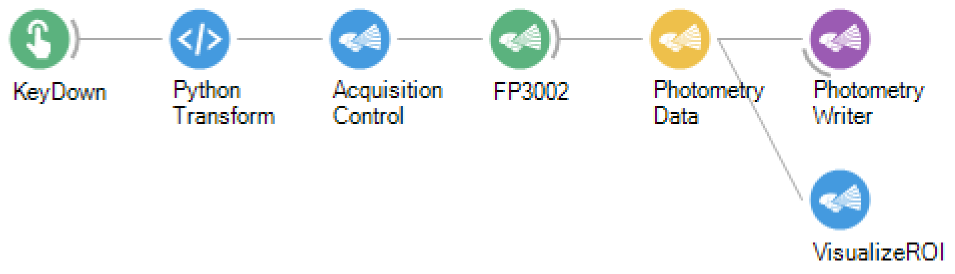


```
# Global 'count' Variable incremented every time an element
# is passed through this node. If this value is
# initialized to be even, then the first output will be
# False (the 'Stop Photometry' trigger). Otherwise,
# the first output will be True (the 'Start Photometry'
# trigger).
count = -1

# 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
```

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

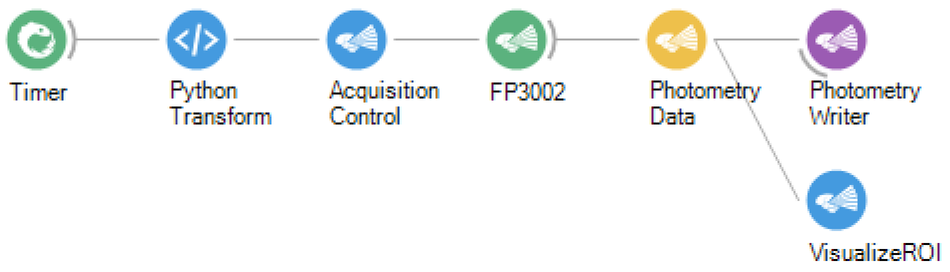
$count = -1$ to $count = 0$. Previously, we set the “Acquisition Mode” of the “FP3002” node to specify whether the FP3002 system should begin data acquisition on the start of the Bonsai workflow, or to wait until the first “Start Photometry” command. The “Acquisition Mode” property and the initialization 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.

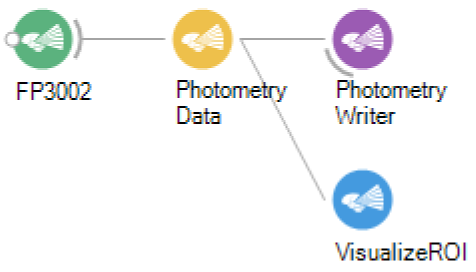


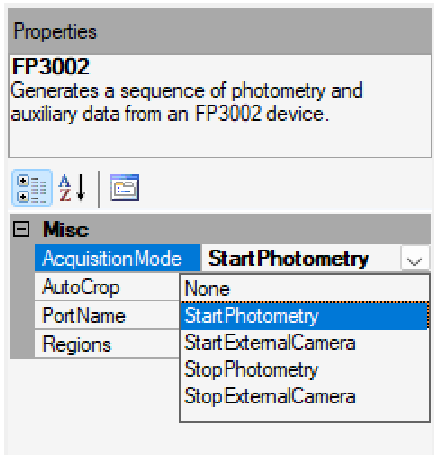
Basic Periodic Control

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”.

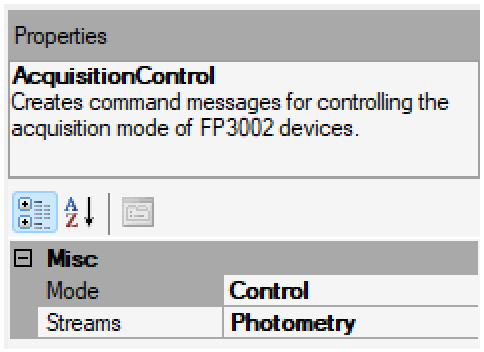



Properties
FP3002
Generates a sequence of photometry and auxiliary data from an FP3002 device.

Misc
AcquisitionMode: **StartPhotometry**
AutoCrop
PortName
Regions

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.

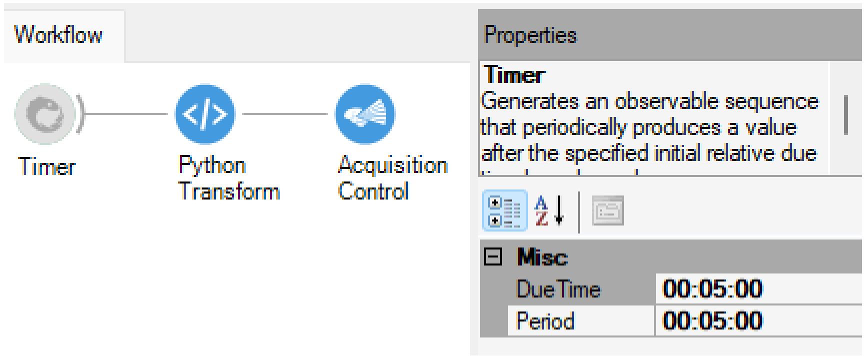

Properties

AcquisitionControl
Creates command messages for controlling the acquisition mode of FP3002 devices.

Misc

Mode	Control
Streams	Photometry

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.



Properties

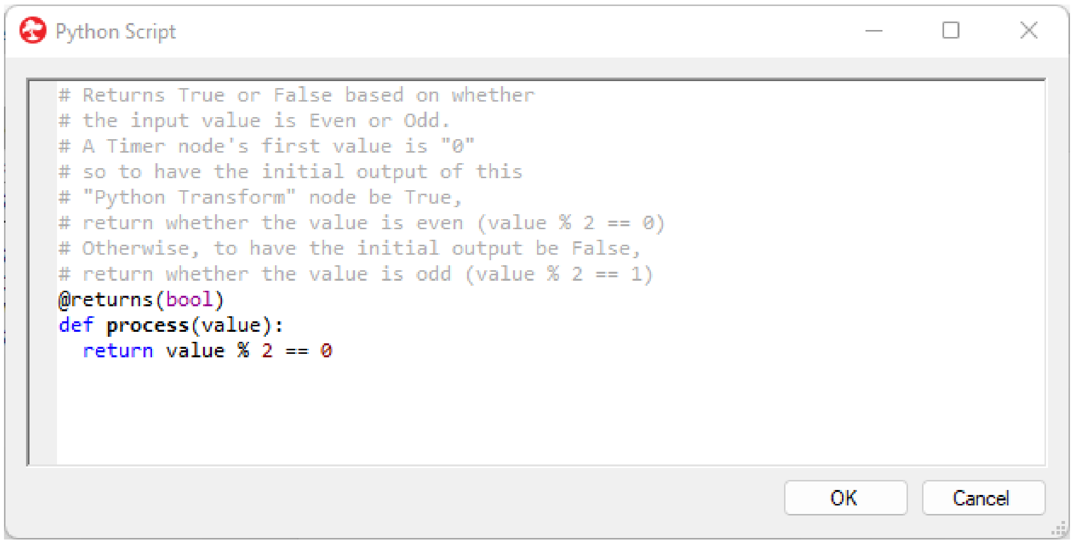
Timer
Generates an observable sequence that periodically produces a value after the specified initial relative due

Misc

DueTime	00:05:00
Period	00:05:00

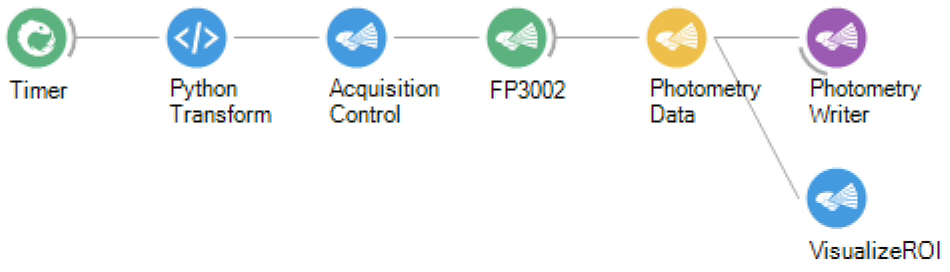
To properly configure the “Timer” node, set the “Period” property to be the duration of time for data acquisition to occur each cycle. Since the duty cycle of this particular workflow is 50%, this will also be the period of time that data is not being acquired during each cycle. Then, set the “Due Time” 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.



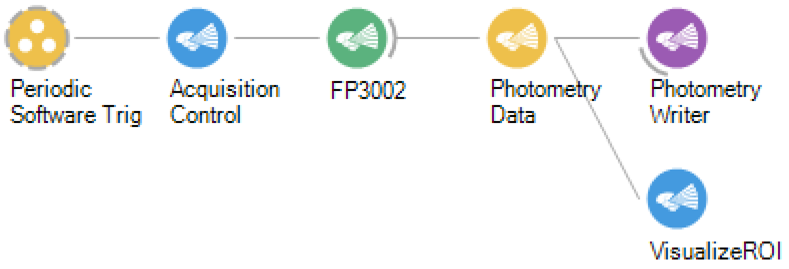
```
# Returns True or False based on whether
# the input value is Even or Odd.
# A Timer node's first value is "0"
# so to have the initial output of this
# "Python Transform" node be True,
# return whether the value is even (value % 2 == 0)
# Otherwise, to have the initial output be False,
# return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

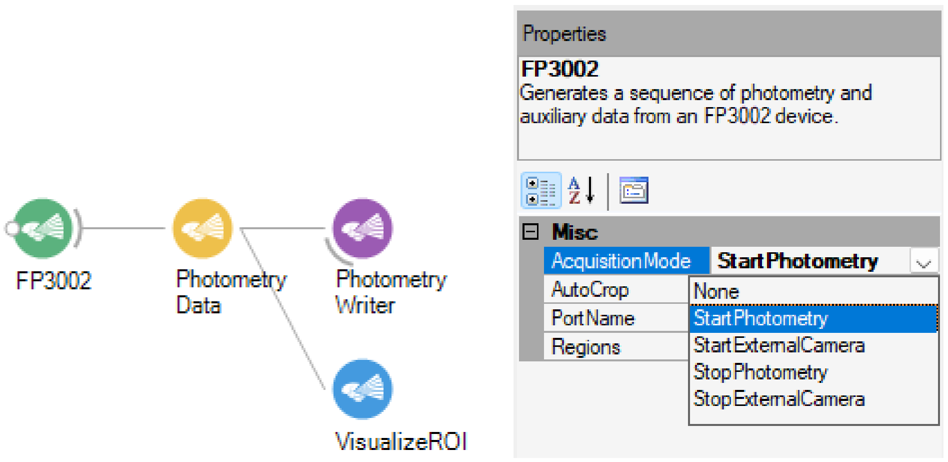


Periodic Control, Variable Duty Cycle

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.



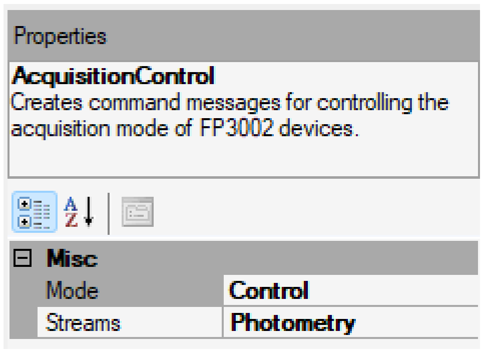
To construct this workflow, begin with the standard photometry workflow, setting the "Acquisition Mode" property of the "FP3002" node to the desired initial acquisition state. If data acquisition should begin with the start of the Bonsai workflow, select "Start Photometry". Otherwise, if data acquisition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



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

AcquisitionMode: StartPhotometry

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.

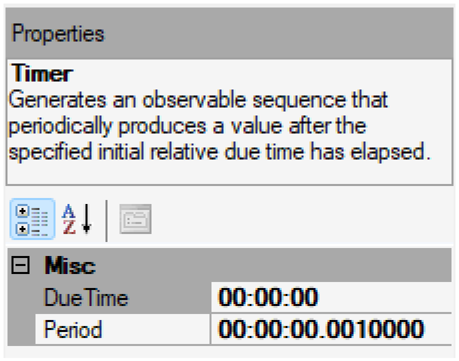

Properties

AcquisitionControl
Creates command messages for controlling the acquisition mode of FP3002 devices.

Misc

Mode	Control
Streams	Photometry

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.

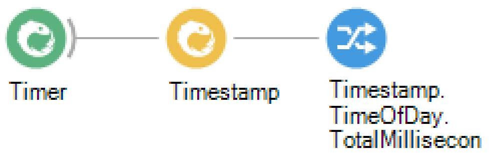

Properties

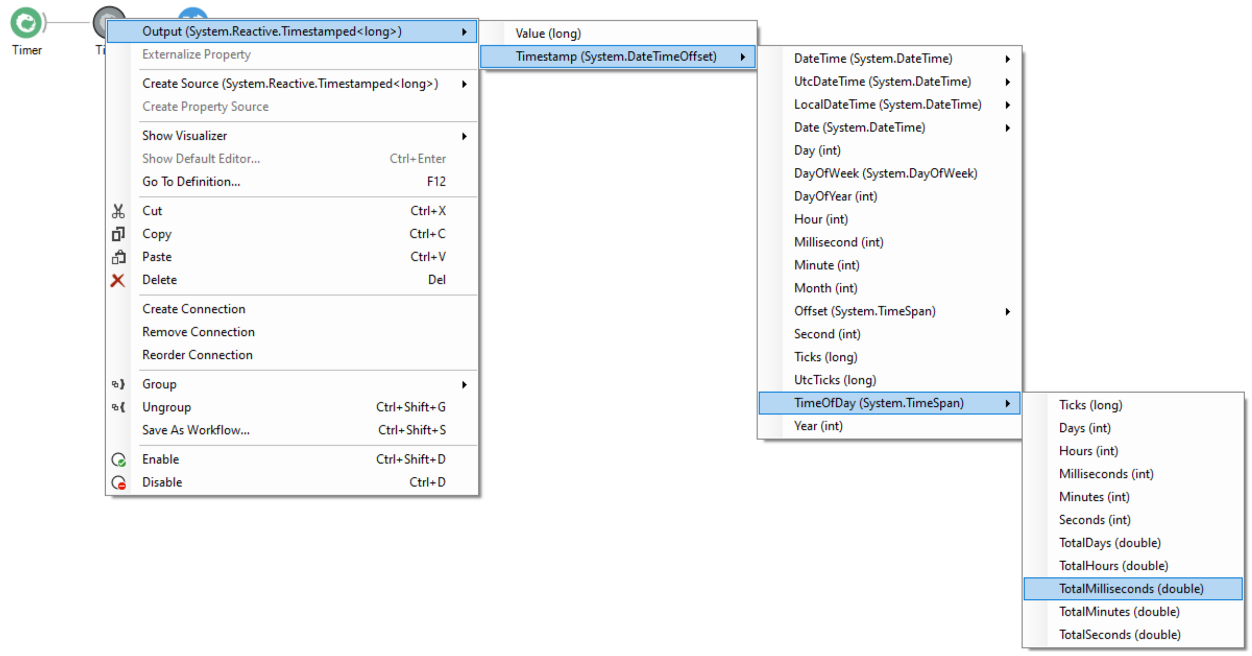
Timer
Generates an observable sequence that periodically produces a value after the specified initial relative due time has elapsed.

Misc

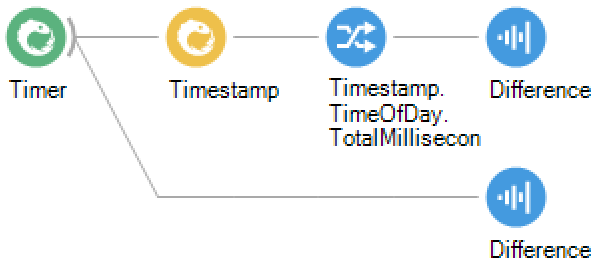
DueTime	00:00:00
Period	00:00:00.0010000

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”.

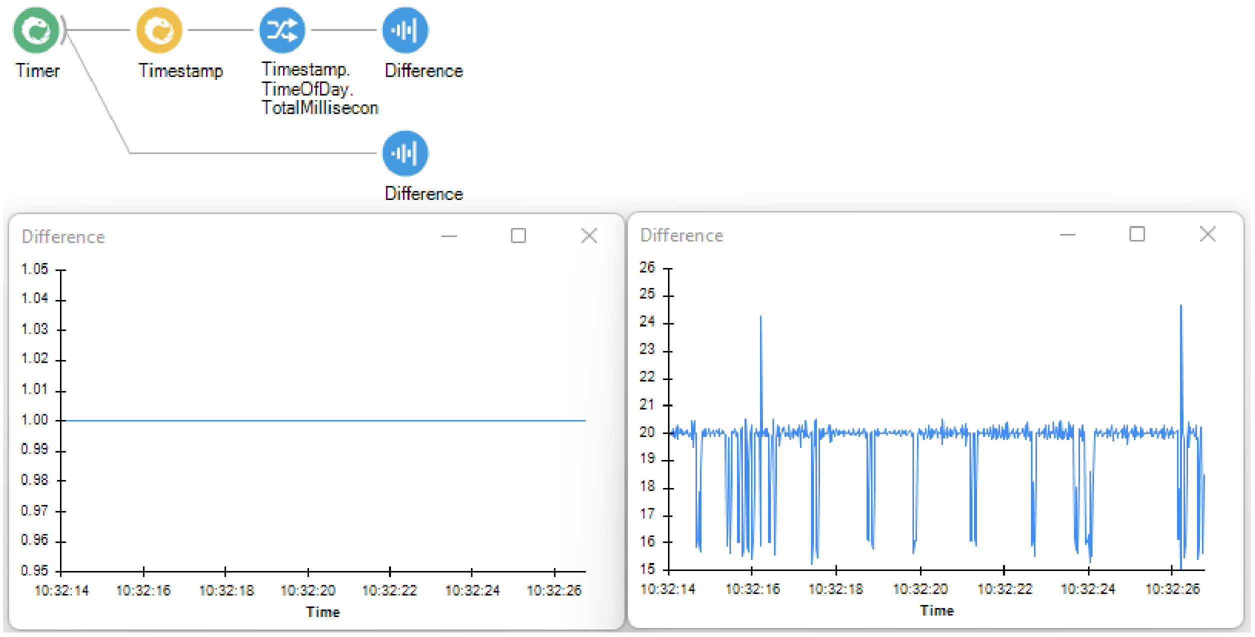




To highlight the importance of actually timestamping the output of the “Timer” node that has a low “Period”, let us use a “Difference” node on the “Timer” and on the “Timestamp.TimeOfDay.TotalMilliseconds” nodes.

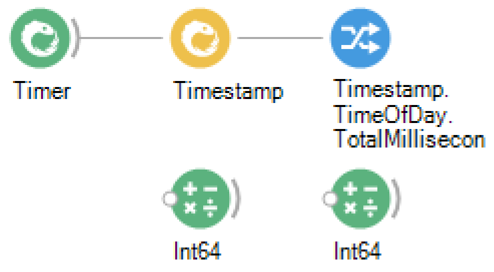


When running the above code, the “Difference” node connected to the “Timer” will show the theoretical amount of milliseconds between outputs of the “Timer”. Meanwhile, the “Difference” node connected to the “Timestamp.TimeOfDay.TotalMilliseconds” node will show the observed amount of milliseconds between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



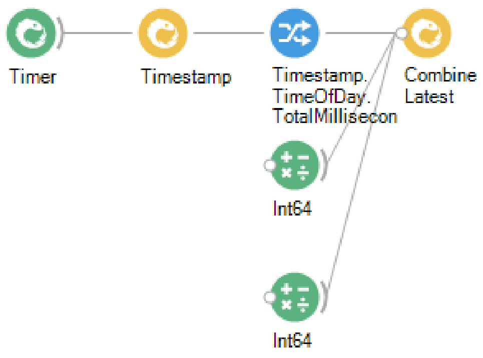
The left plot shows that theoretically, only one millisecond passed between outputs of the “Timer” node, but the right plot shows the actual time between outputs is closer to 20ms. The actual rate at which values are output from the “Timer” configured with a 1ms period is dependent on many factors, so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

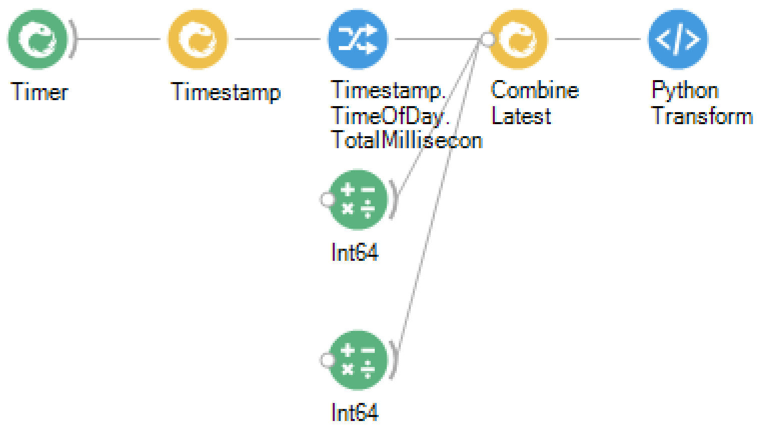


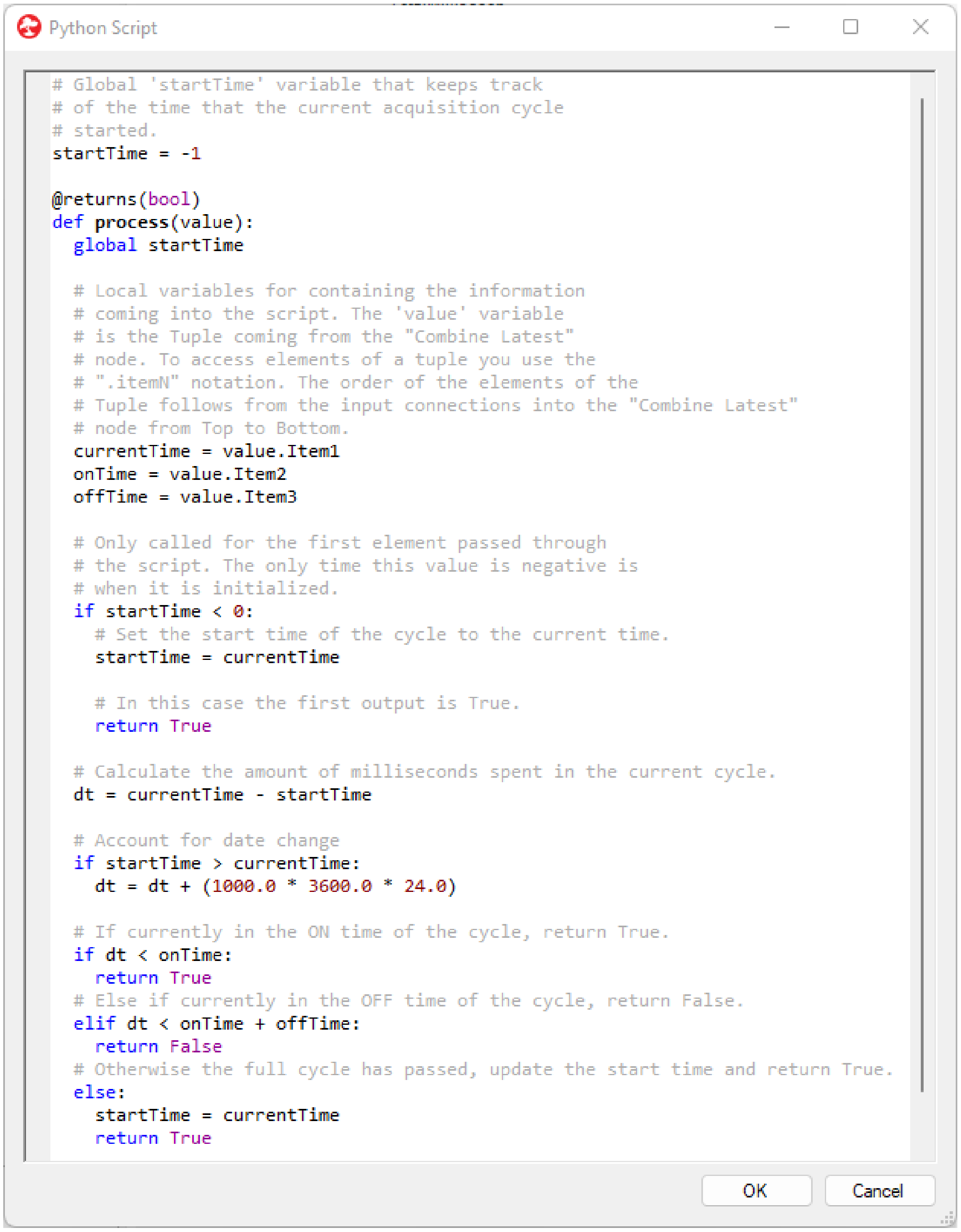
NEUROPHOTOMETRICS
The fiber photometry company

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.



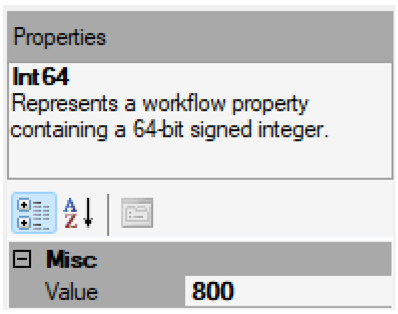
Now that we have one data stream containing all the information we need, we can connect a “Python Transform” node after the “Combine Latest” node to process the information, outputting the appropriate boolean value. In this python script, we will keep track of each time a new acquisition cycle starts, 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.



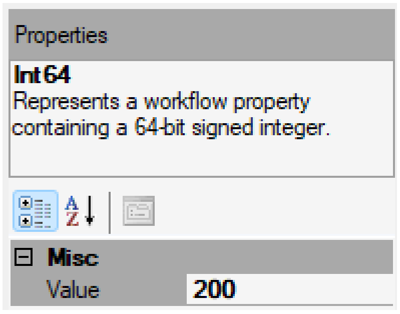


```
# Global 'startTime' variable that keeps track
# of the time that the current acquisition cycle
# started.
startTime = -1

@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
```

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.

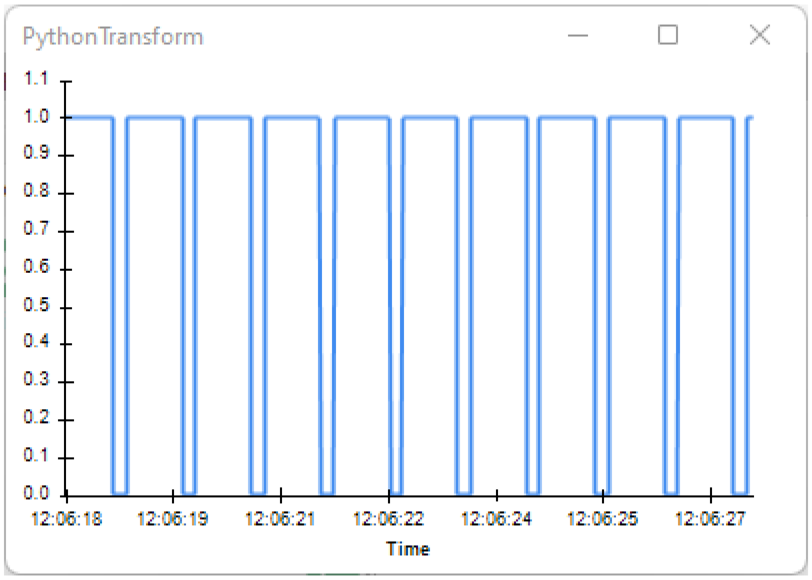

Properties
Int64
Represents a workflow property containing a 64-bit signed integer.

Misc	
Value	800

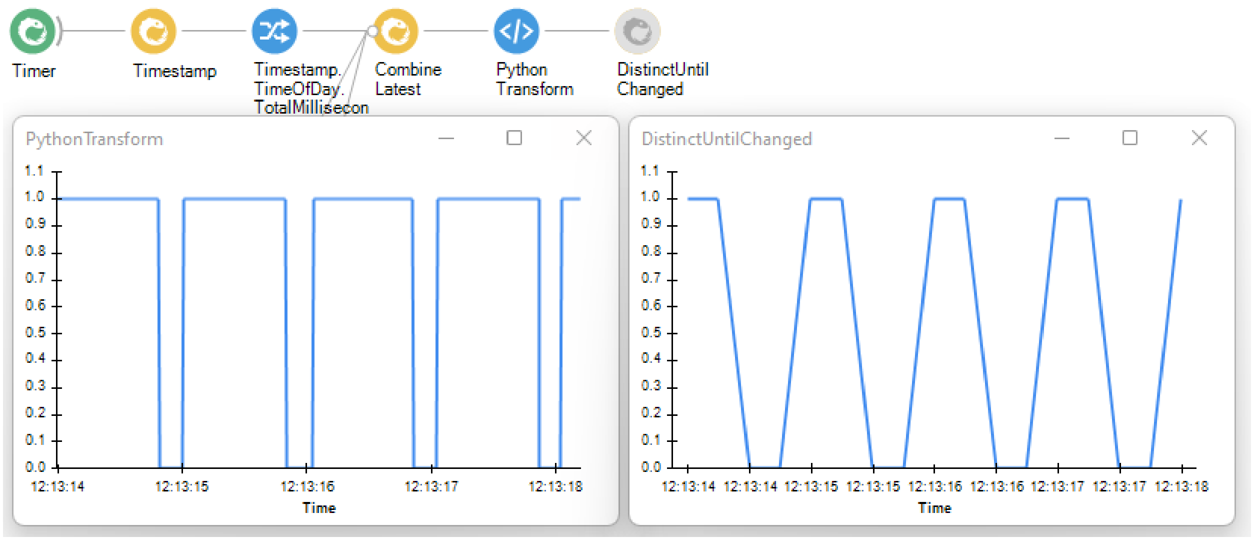

Properties
Int64
Represents a workflow property containing a 64-bit signed integer.

Misc	
Value	200

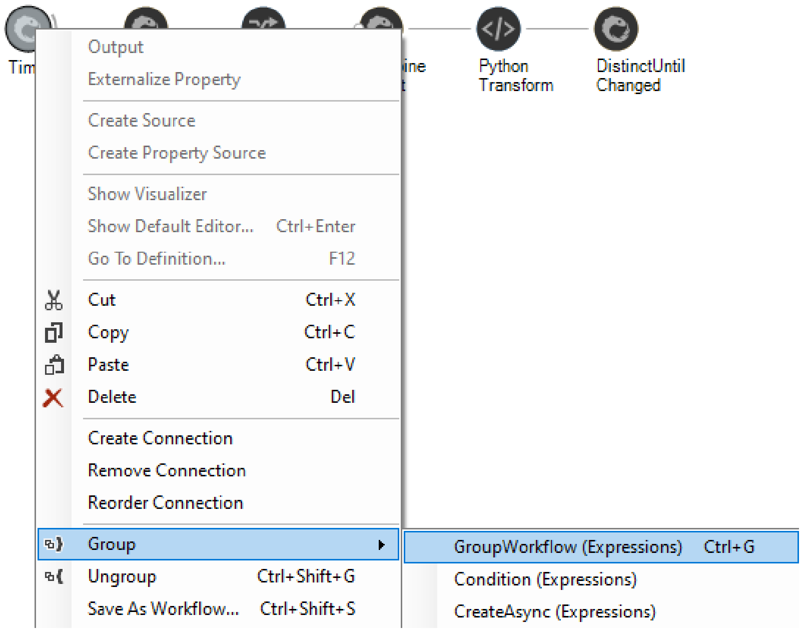
Once these values are set, run this software trigger section of the workflow by disabling all other nodes in the workflow and clicking start. Open the visualizer for the "Python Transform" node to check if our boolean signal appears correct.



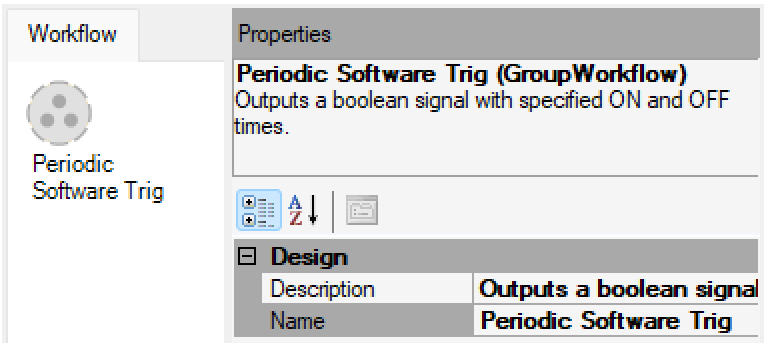
Here, the “Python Transform” periodically outputs True values for 800ms and False values for 200ms, as intended. However, this “Python Transform” outputs a value every time the timer’s data stream produces a new value. We saw above that this occurs every ~20ms. If this were connected to the “Acquisition 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.



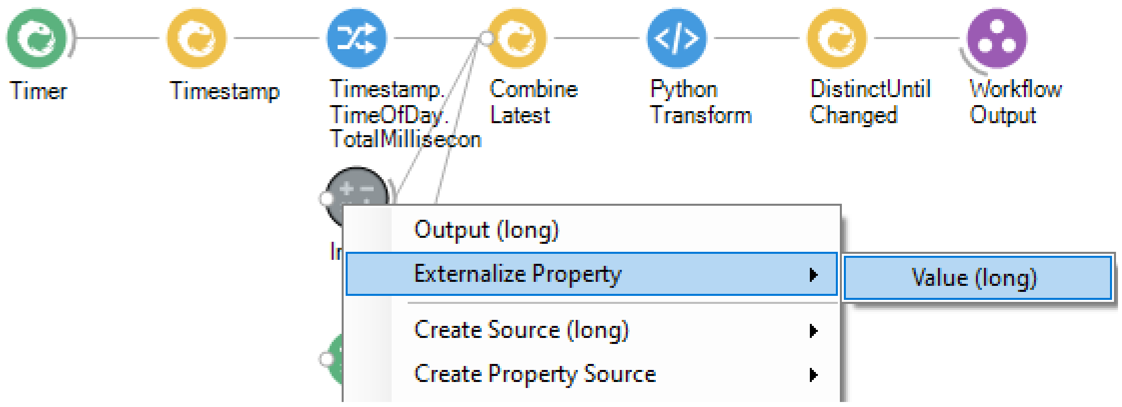
The properties panel of the “Group Workflow” node allows you to name it and provide a description of the encapsulated process.



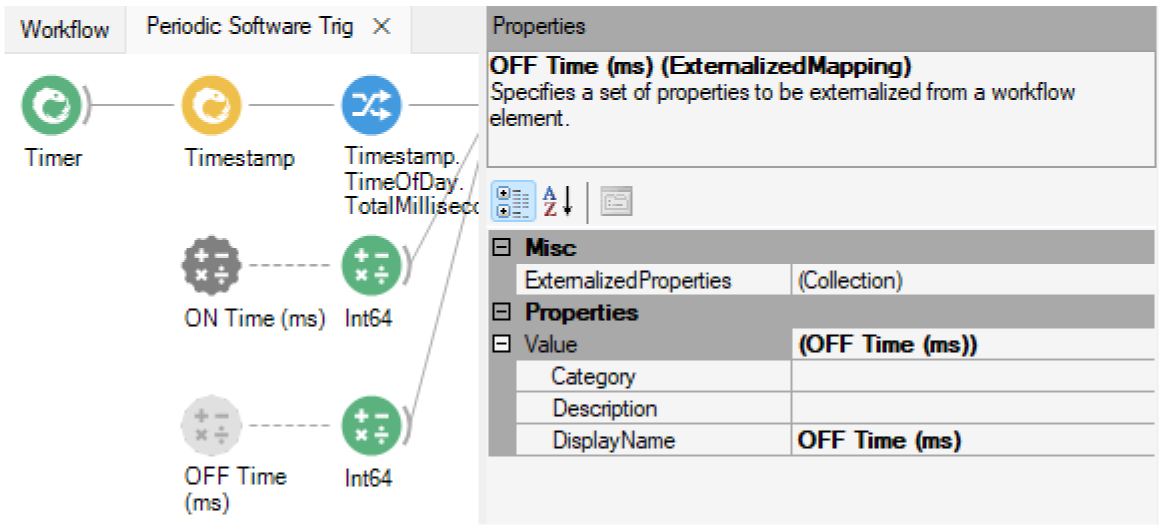
Periodic Software Trig (GroupWorkflow)
Outputs a boolean signal with specified ON and OFF times.

Design	
Description	**Outputs a boolean signal**
Name	**Periodic Software Trig**

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”.



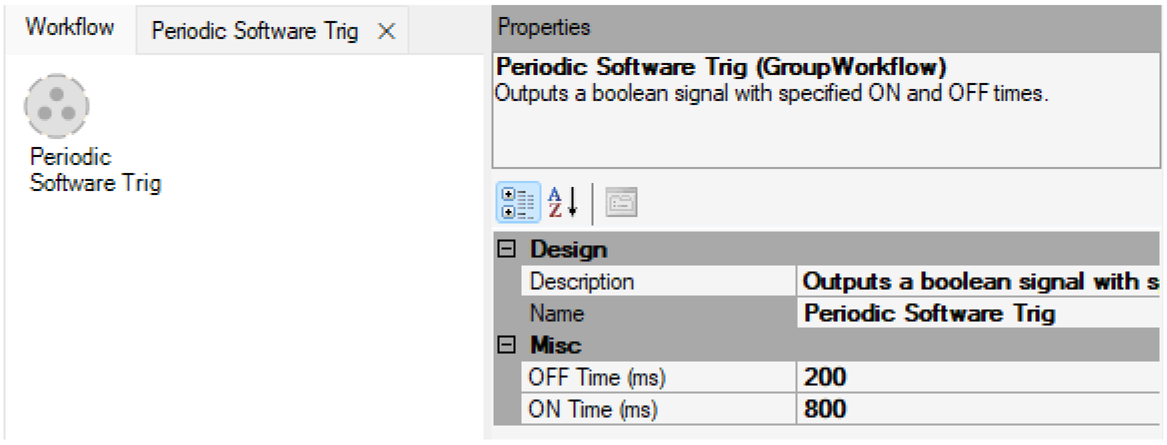
This will add “Value (ExternalizedMapping)” nodes to the inputs of the “Int64” nodes and make these properties accessible from the properties panel of the grouped workflow. Both of these “Value” nodes need to have unique “DisplayNames”. To name them, select the node, then expand the “Value” section of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



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

Misc	
ExternalizedProperties	(Collection)
Properties	
Value	**(OFF Time (ms))**
Category	
Description	
DisplayName	**OFF Time (ms)**

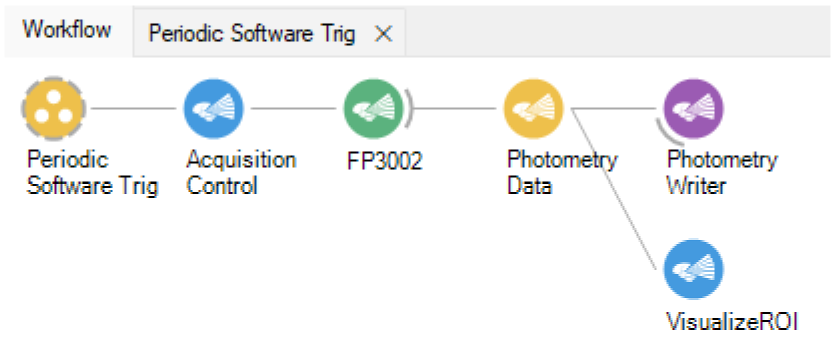
With these externalized properties added, the ON and OFF times can be specified from the properties panel of the grouped workflow.



Periodic Software Trig (GroupWorkflow)
Outputs a boolean signal with specified ON and OFF times.

Design	
Description	**Outputs a boolean signal with s**
Name	**Periodic Software Trig**
Misc	
OFF Time (ms)	**200**
ON Time (ms)	**800**

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.

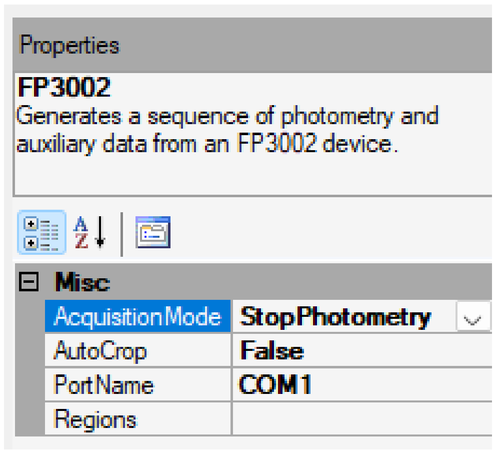


Periodic Control, Delayed Start

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".

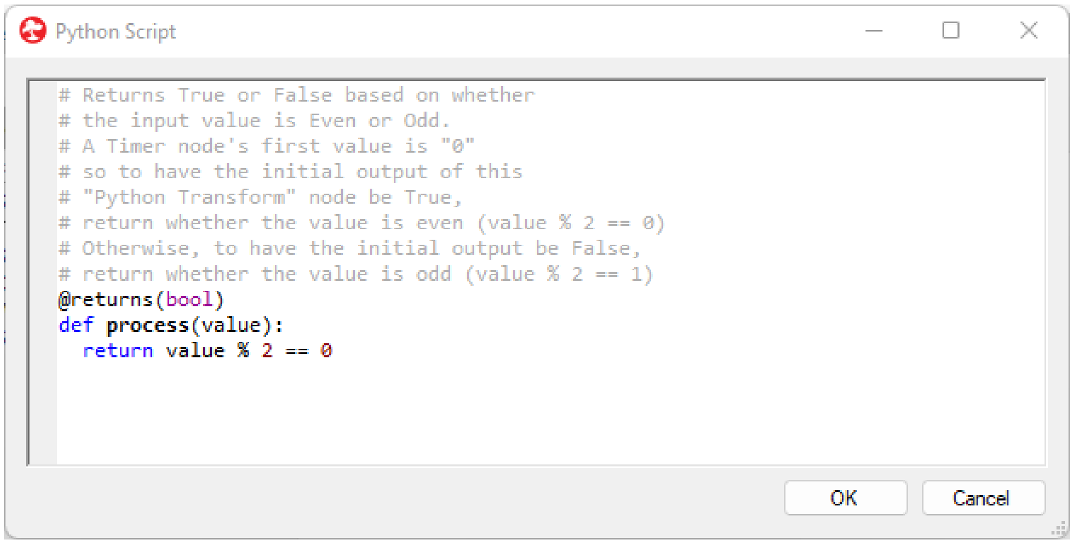


FP3002
Generates a sequence of photometry and auxiliary data from an FP3002 device.

Misc

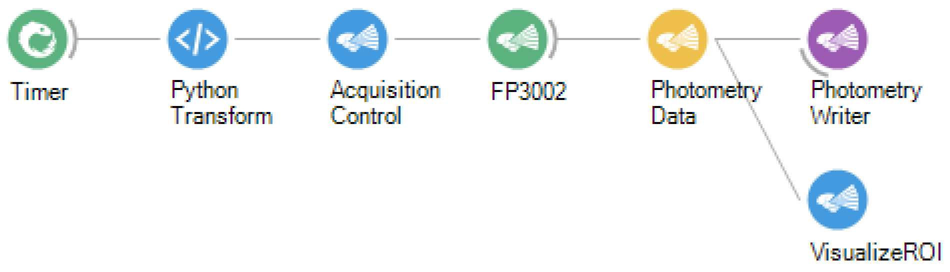
AcquisitionMode	StopPhotometry
AutoCrop	False
PortName	COM1
Regions	

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 “ $value \% 2 \; == \; 0$”.

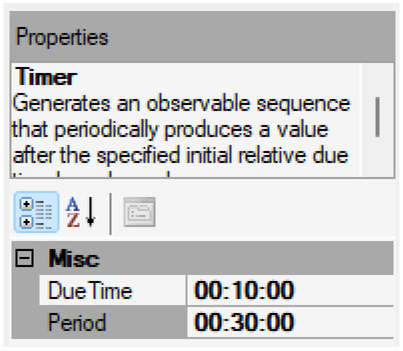


```
# Returns True or False based on whether
# the input value is Even or Odd.
# A Timer node's first value is "0"
# so to have the initial output of this
# "Python Transform" node be True,
# return whether the value is even (value % 2 == 0)
# Otherwise, to have the initial output be False,
# return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

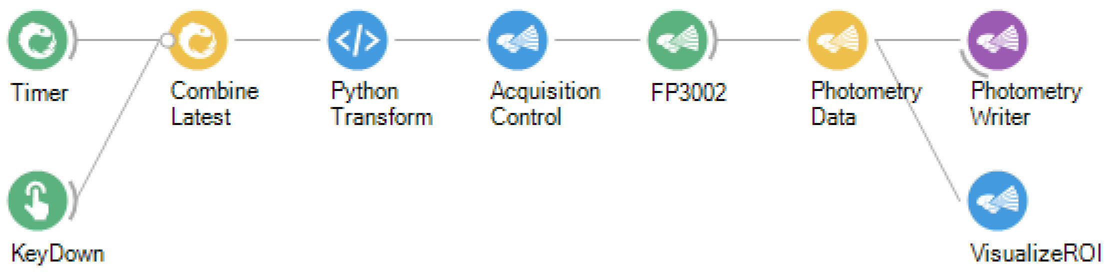

Properties
Timer
Generates an observable sequence that periodically produces a value after the specified initial relative due

Misc

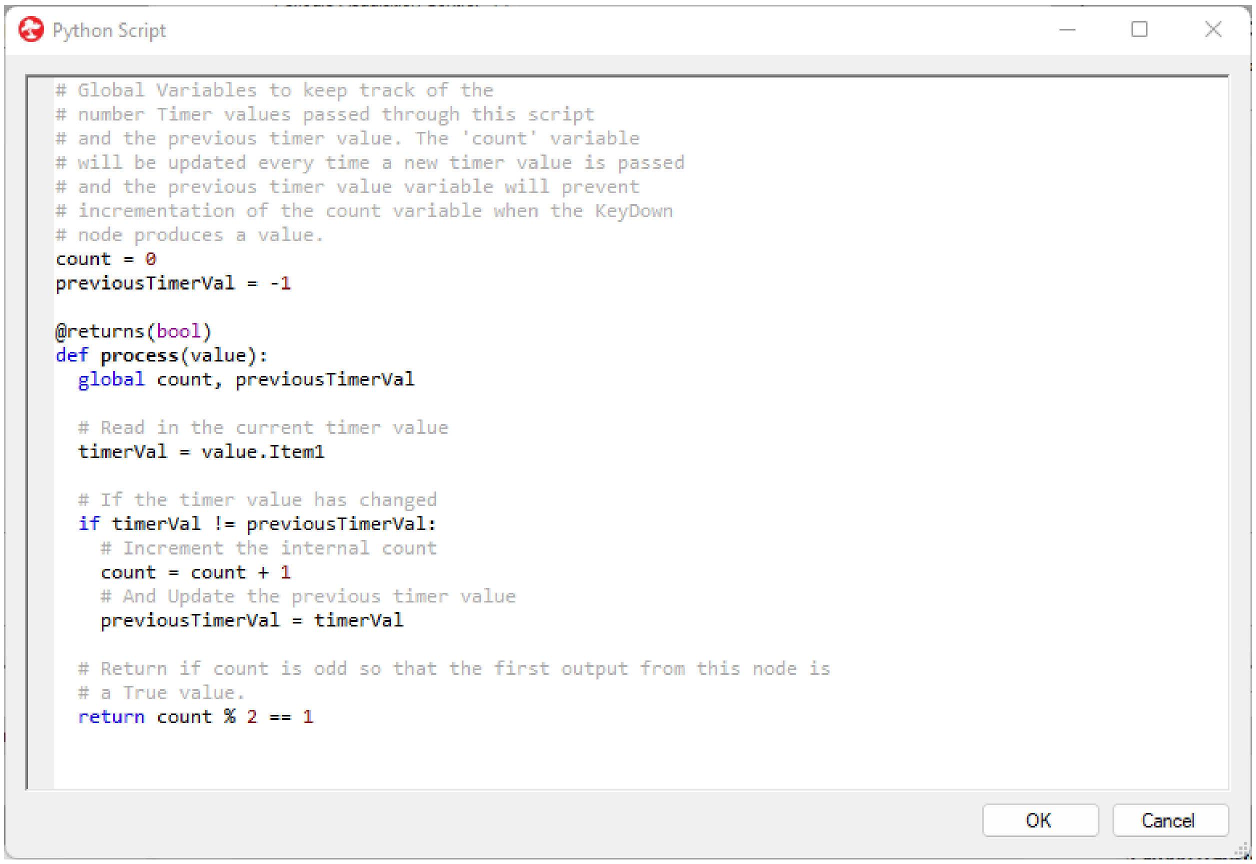
DueTime	00:10:00
Period	00:30:00

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 that the "Due Time" property of the "Timer" node 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.

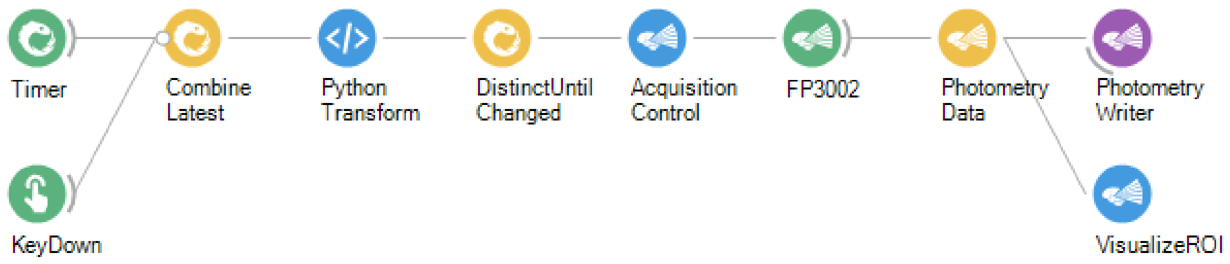


Now, some minor changes to the “Python Transform” node need to be made. Currently, the python script assumes that the input is an integer value, but now its input is a Tuple. Also, the “Python Transform” outputs whether or not the value from the “Timer” node is even or odd. However, with the delayed 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.



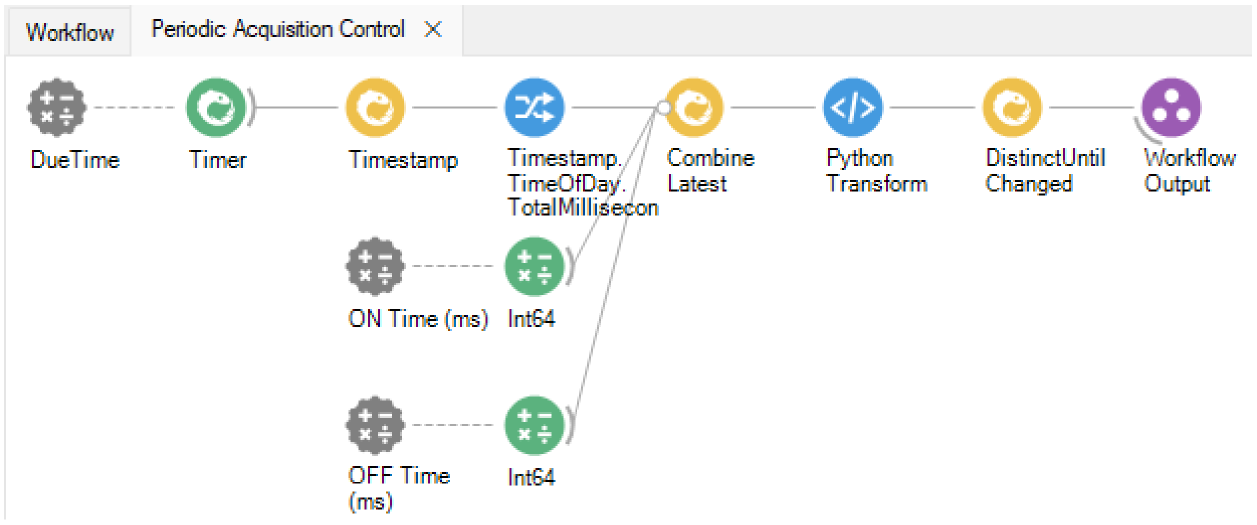
```
# Global Variables to keep track of the
# number Timer values passed through this script
# and the previous timer value. The 'count' variable
# will be updated every time a new timer value is passed
# and the previous timer value variable will prevent
# incrementation of the count variable when the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

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.



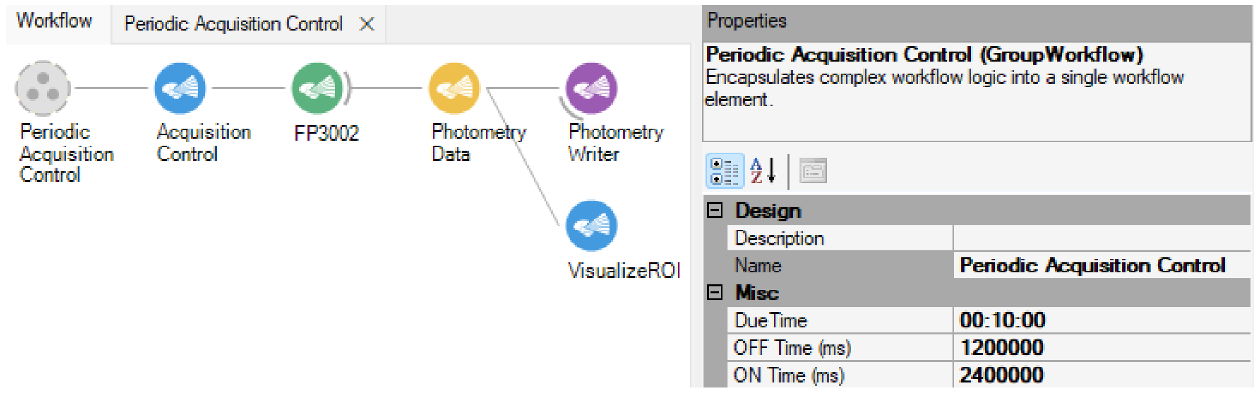
Periodic Control, Variable Duty Cycle, Automatic Delayed Start

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"



NEUROPHOTOMETRICS
The fiber photometry company

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 10minute delay. Once started it will alternate between 40minutes of data acquisition and 20minutes of no data acquisition.

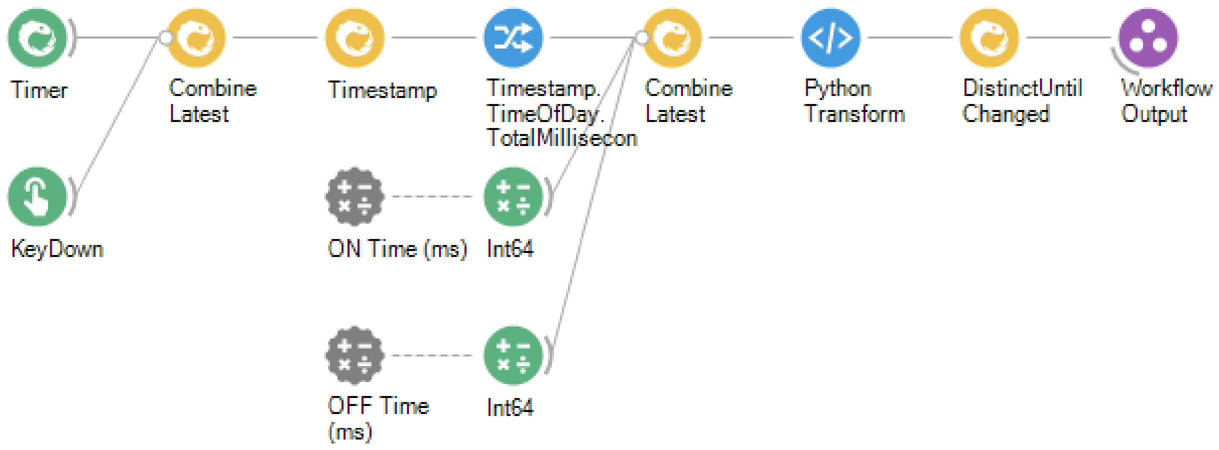


Periodic Acquisition Control (GroupWorkflow)
Encapsulates complex workflow logic into a single workflow element.

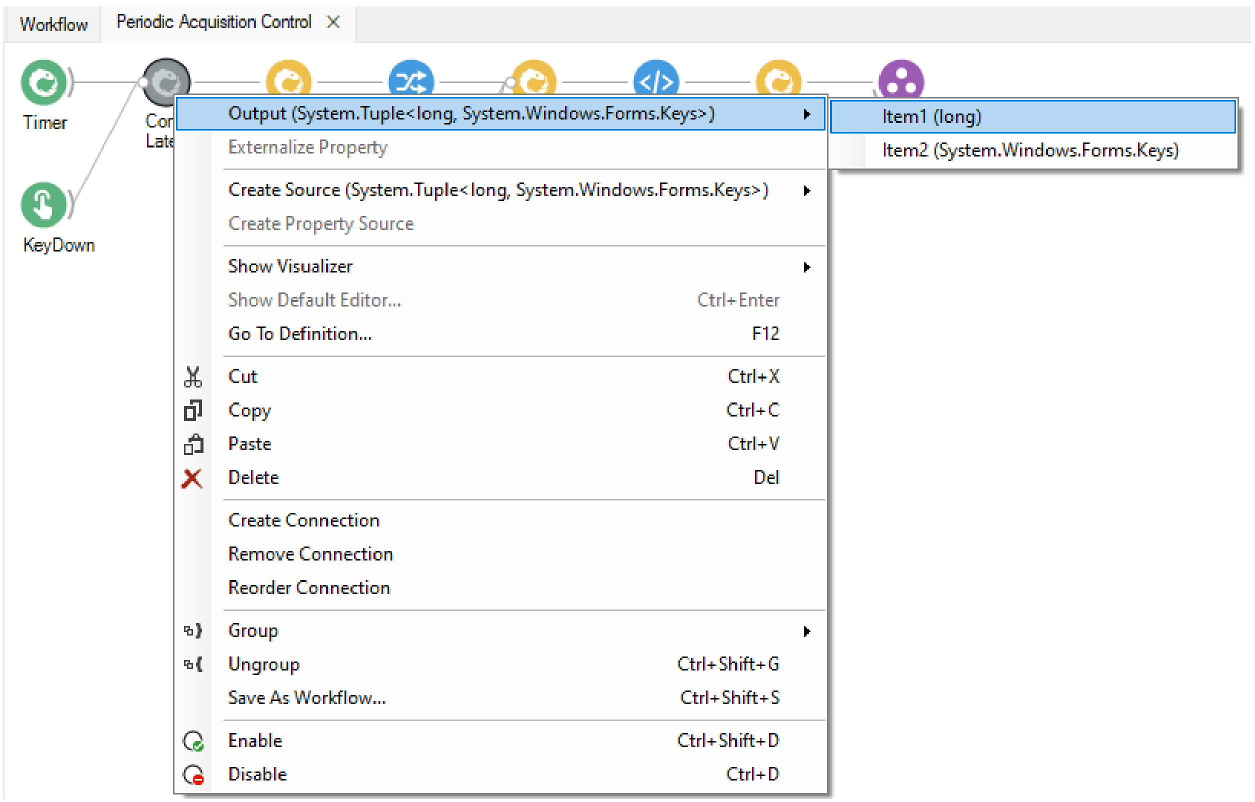
Design	
Description	
Name	**Periodic Acquisition Control**
Misc	
DueTime	**00:10:00**
OFF Time (ms)	**1200000**
ON Time (ms)	**2400000**

Periodic Control, Variable Duty Cycle, Manual Delayed Start

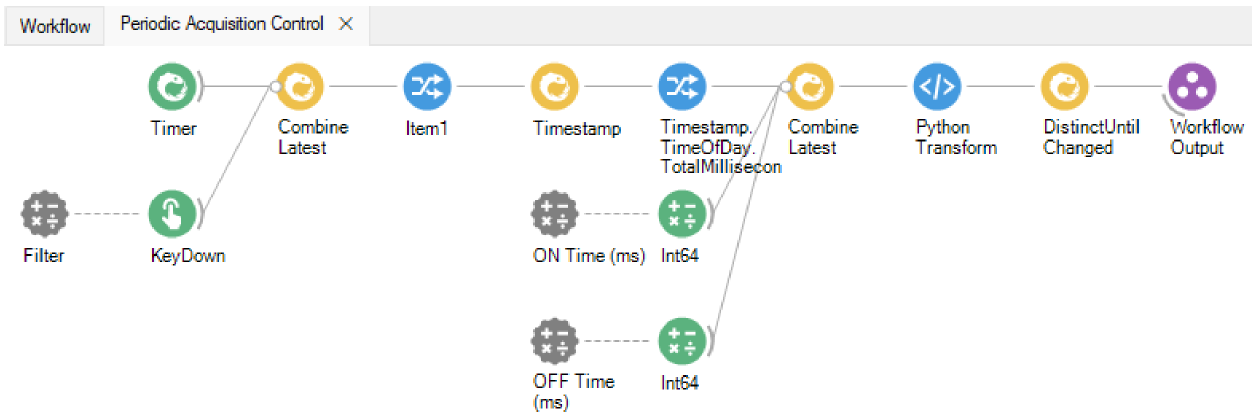
To add a manual delay to this workflow, instead of specifying the “Due Time” property, we will use a “Combine Latest” node to combine the “Timer” node and a “KeyDown” node. This way the “PythonTransform” node will not receive its first value until the specified key is pressed.

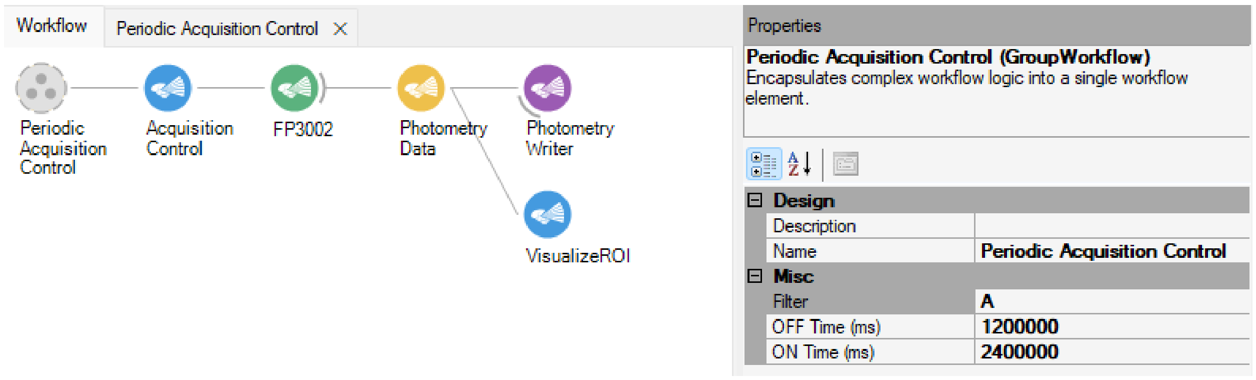


Now, we have two “Combine Latest” nodes in the same data stream leading up to the “Python Transform” node. This adds a little complexity to the data types going into python script. We can fix this data type mix-match either within the python script or within the workflow. In this case, the easiest way 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.





Periodic Acquisition Control (GroupWorkflow)
Encapsulates complex workflow logic into a single workflow element.

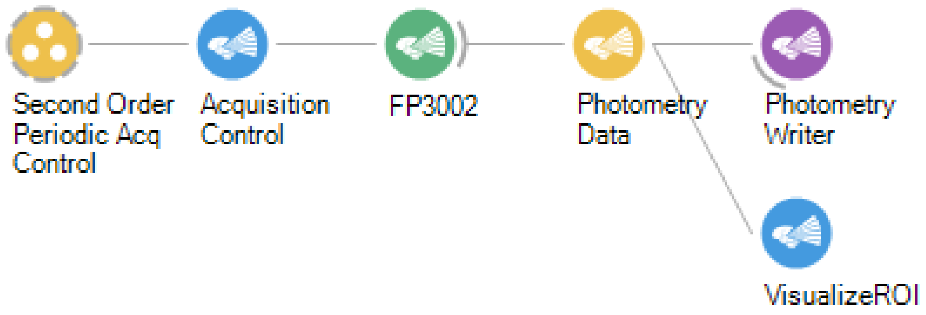
Design	
Description	
Name	**Periodic Acquisition Control**
Misc	
Filter	**A**
OFF Time (ms)	**1200000**
ON Time (ms)	**2400000**

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.

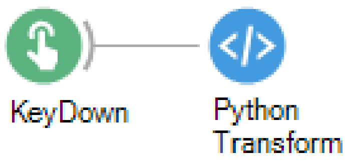
Similar to the previous data acquisition workflows, these second order periodic control workflows can be separated into three sections: the software trigger, the acquisition control node, and the standard photometry section. Our starting point in constructing this workflow will be the “Periodic Control, 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”.



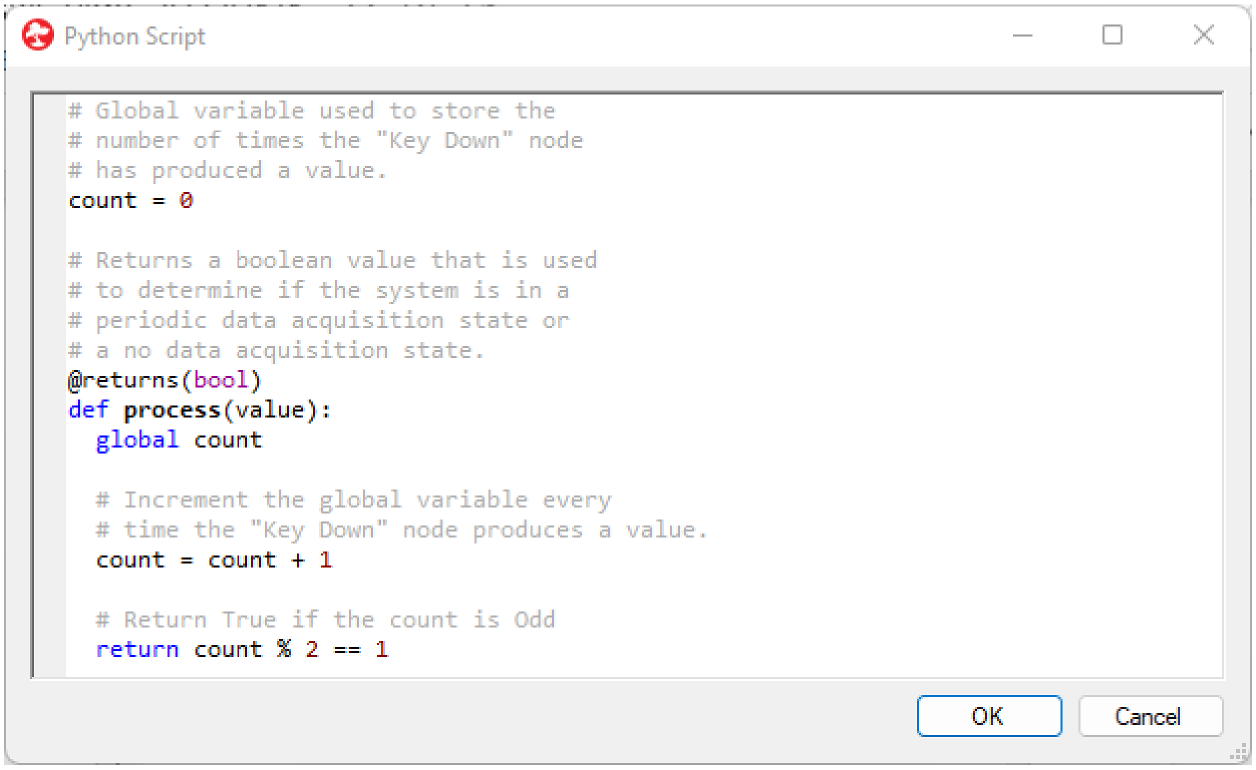
To change this first order periodic control workflow to a second order periodic control workflow, all we need to do is reconfigure the software trigger logic.

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.



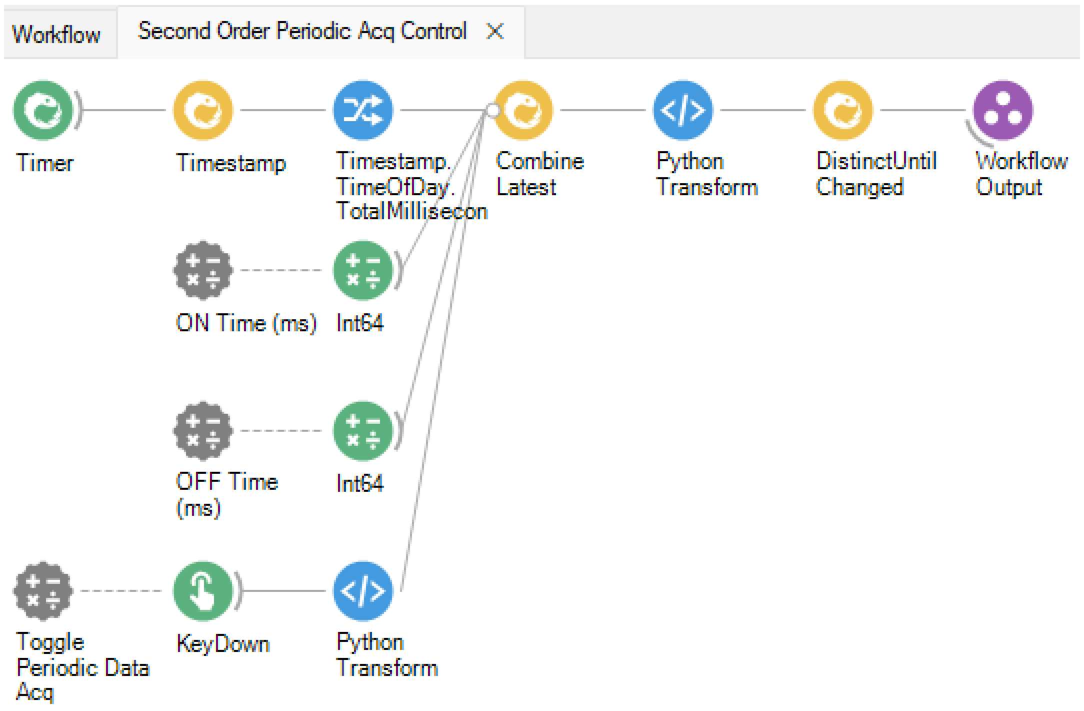
The python script in this new “Python Transform” node will contain an internal counter and output whether the key was pressed an Even or Odd number of times. We will use the output of this “Python Transform” node to dictate whether the system is in a periodic acquisition control state or a no data acquisition 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
```

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.



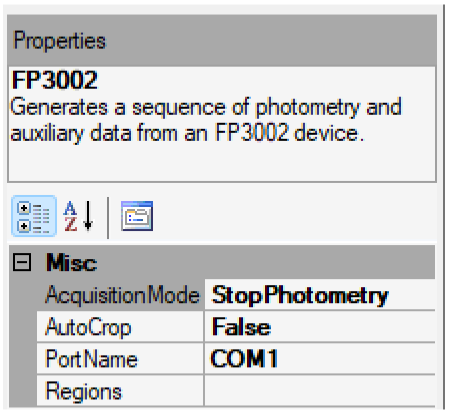
With some additional python logic in the “Python Transform” node immediately after the “Combine Latest” node, we will finish constructing our software trigger. The boolean value dictating whether the system should be in a periodic data acquisition state or a no data acquisition state will be contained 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.

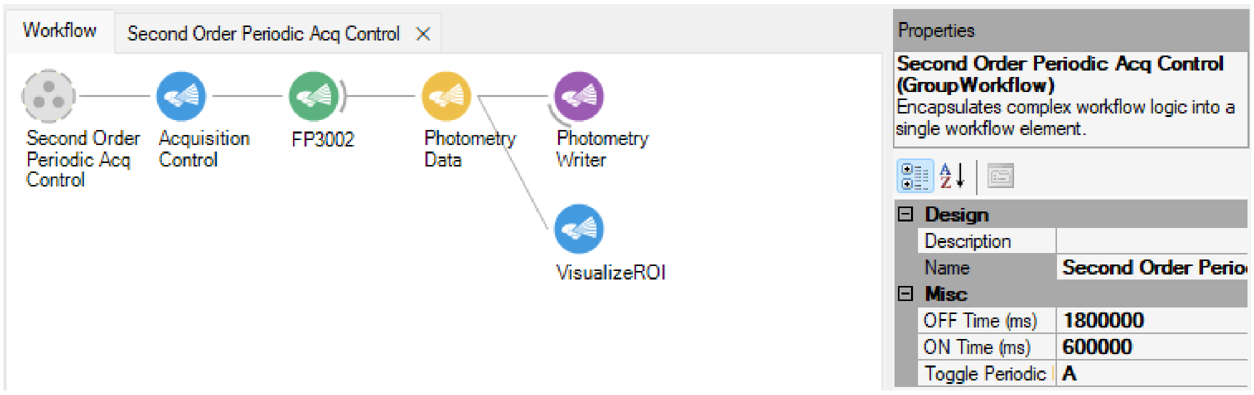

Properties

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

Misc

AcquisitionMode	StopPhotometry
AutoCrop	False
PortName	COM1
Regions	

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.

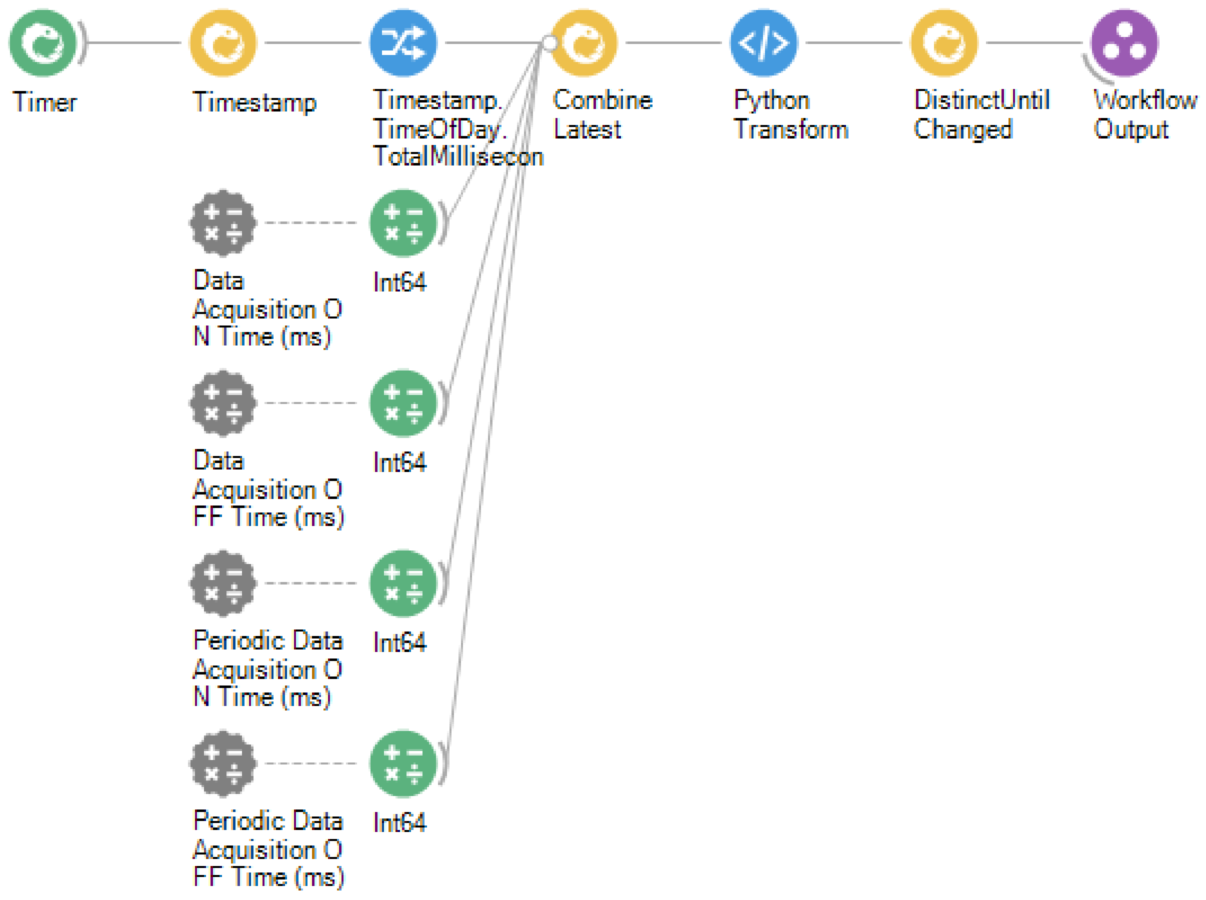


Second Order Periodic Acq Control (GroupWorkflow)
Encapsulates complex workflow logic into a single workflow element.

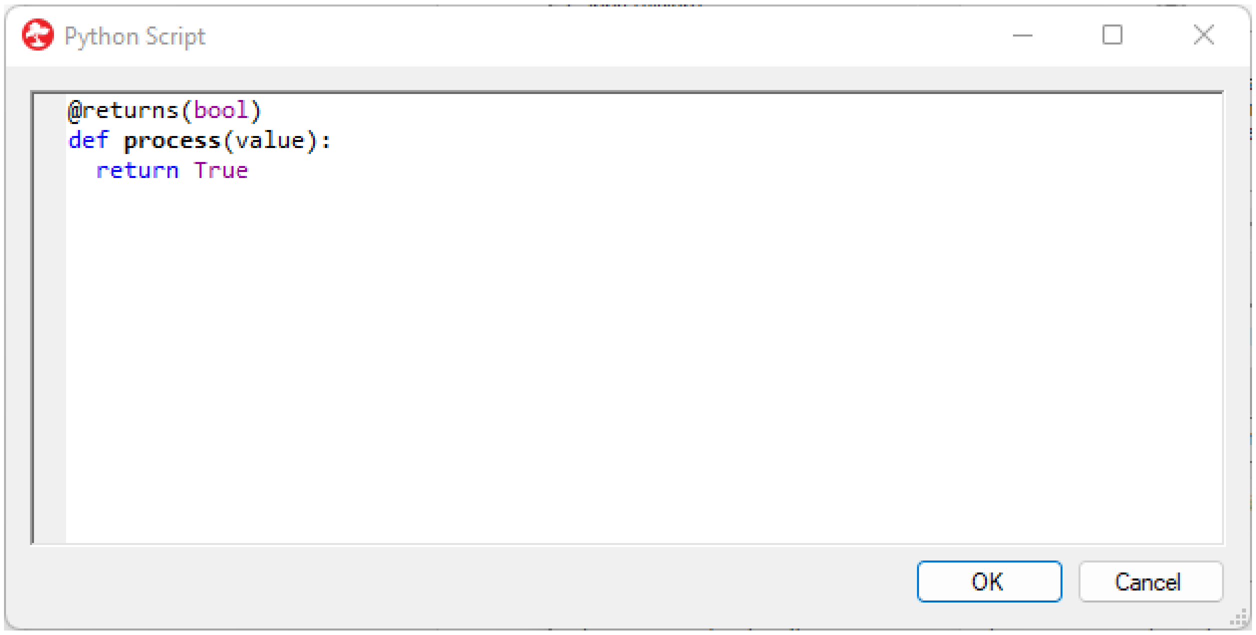
Design	
Description	
Name	Second Order Perio
Misc	
OFF Time (ms)	1800000
ON Time (ms)	600000
Toggle Periodic	A

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.


Timer
Timestamp
Timestamp.TimeOfDay.TotalMillisecon
Combine Latest
Python Transform
DistinctUntil Changed
Workflow Output
Data Acquisition ON Time (ms)
Int64
Data Acquisition OFF Time (ms)
Int64
Periodic Data Acquisition ON Time (ms)
Int64
Periodic Data Acquisition OFF Time (ms)
Int64

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.

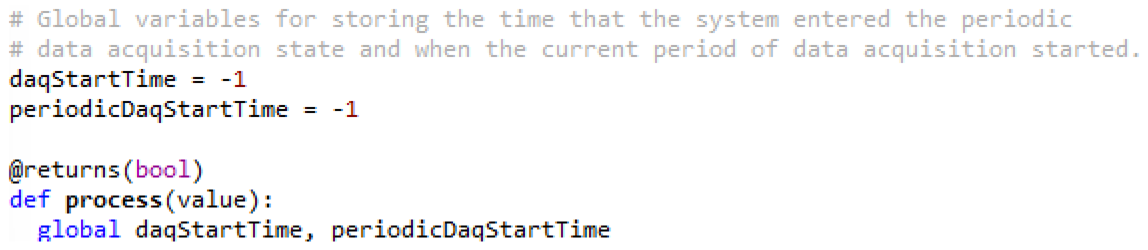


```
@returns(bool)
def process(value):
  return True
```

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.

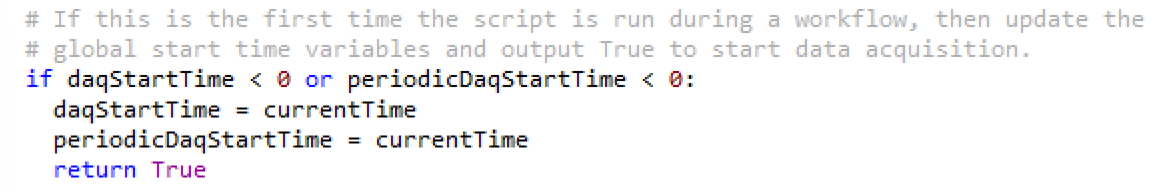
```
@returns(bool)
def process(value):

  # 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.



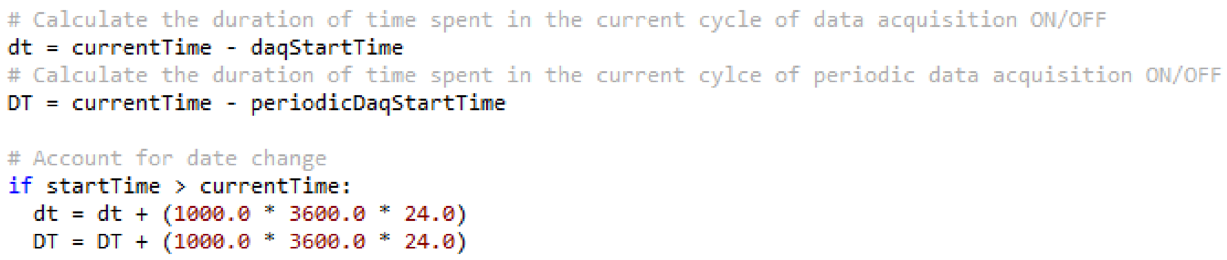
```
# Global variables for storing the time that the system entered the periodic
# data acquisition state and when the current period of data acquisition started.
daqStartTime = -1
periodicDaqStartTime = -1

@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 cylce 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
```

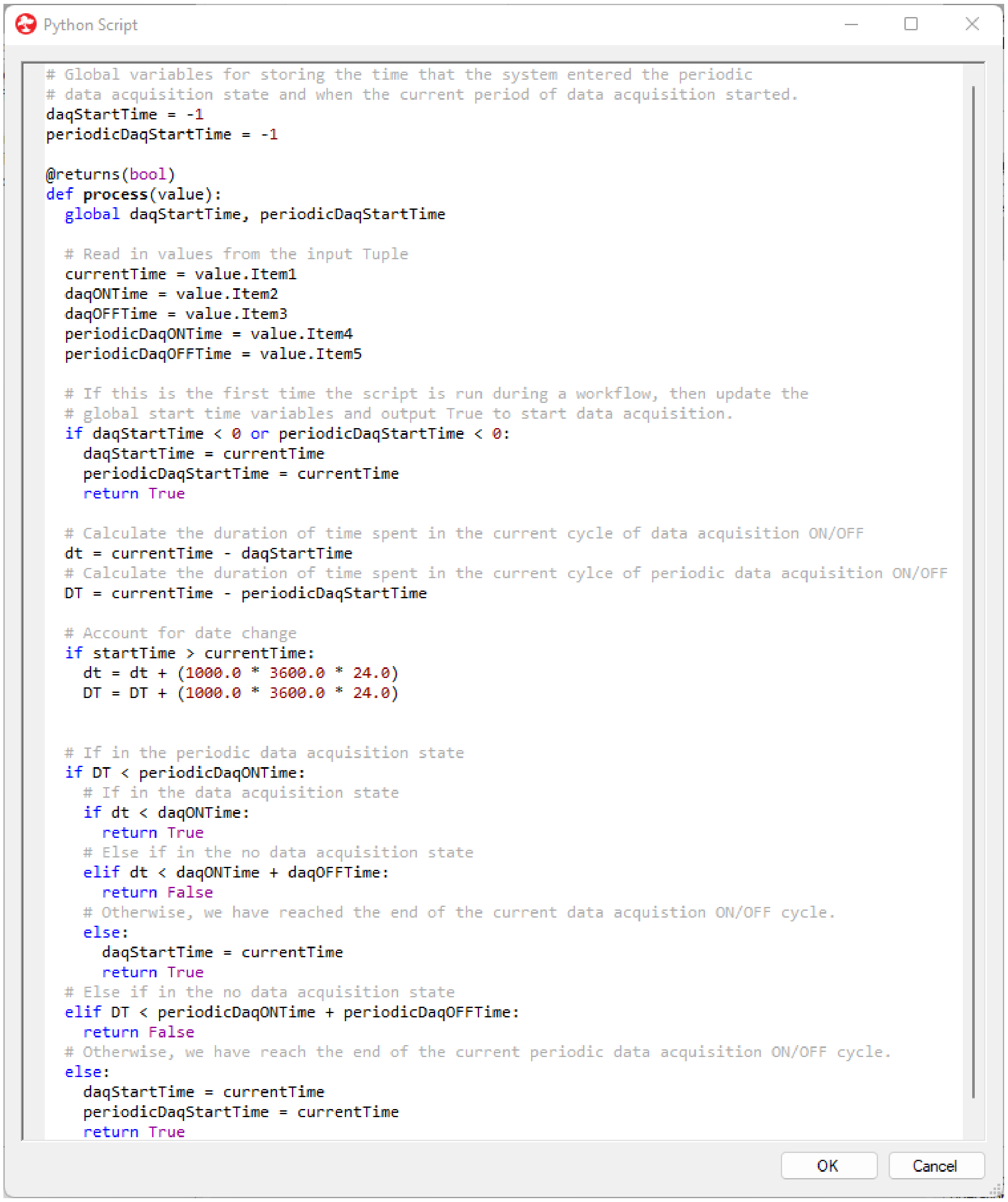
In the if portion of the if, elif, else statement, we need to include logic for alternating between data acquisition ON/OFF states. This logic will be quite similar to our logic for alternating between periodic data acquisition ON/OFF states. So we can add another if, elif, else statement here to determine 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 acquistion 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.

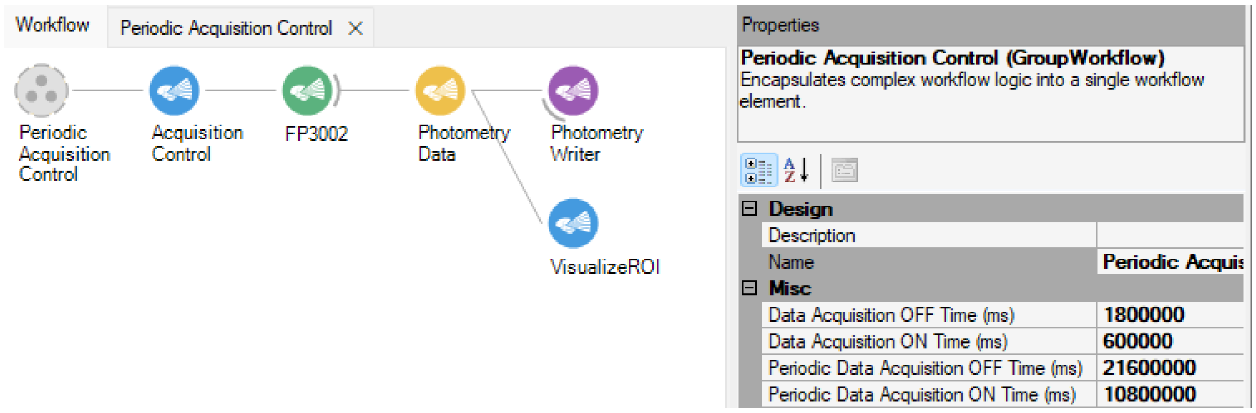
```
# If in the periodic data acquisition state
if DT < periodicDaqONTime:
  # If in the data acquisition state
  if dt < daqONTime:
    return True
  # Else if in the no data acquisition state
  elif dt < daqONTime + daqOFFTime:
    return False
  # Otherwise, we have reached the end of the current data acquistion 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:



```
# 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 cylce 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 acquistion 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
```

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.



Periodic Acquisition Control (GroupWorkflow)
Encapsulates complex workflow logic into a single workflow element.

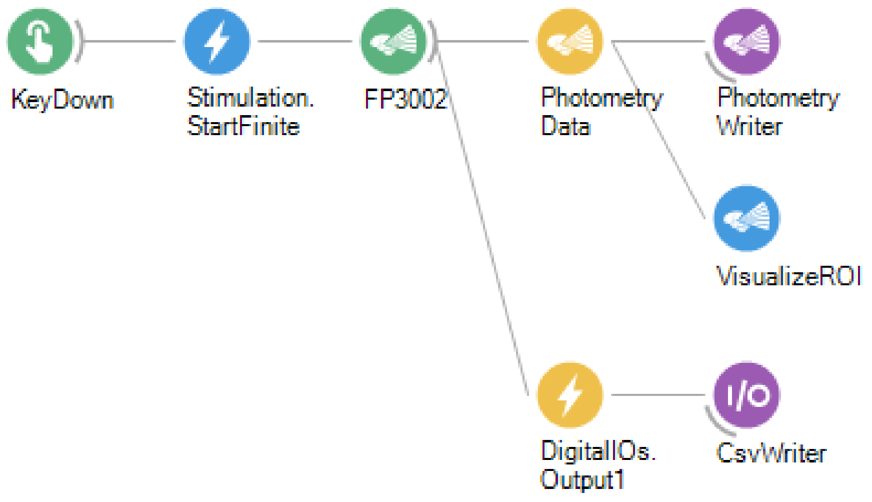
Design	
Description	
Name	**Periodic Acquis**
Misc	
Data Acquisition OFF Time (ms)	**1800000**
Data Acquisition ON Time (ms)	**600000**
Periodic Data Acquisition OFF Time (ms)	**21600000**
Periodic Data Acquisition ON Time (ms)	**10800000**

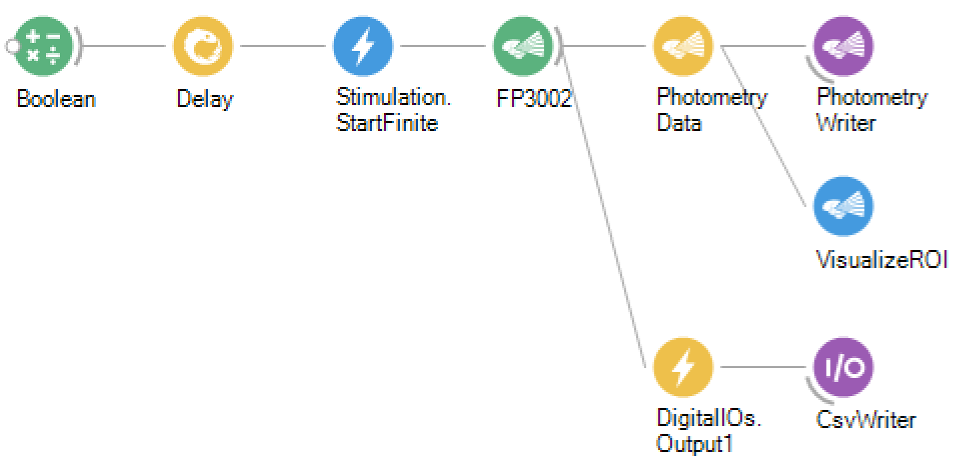
Chapter 5: Stimulation

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.

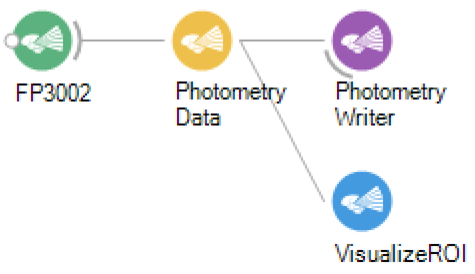
Delayed Start

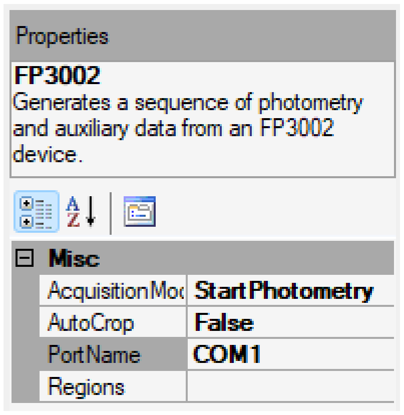
The “Standard Photometry” workflow can be expanded to allow for stimulation to be triggered after a duration of time has passed since the workflow has been started. This example workflow makes use of the “Stimulation” node and a software trigger to command the FP3002 system to start stimulation. The 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” .



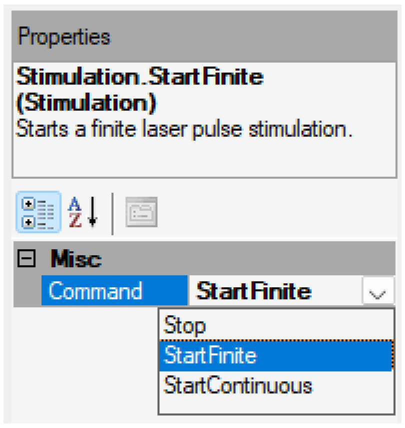


FP3002
Generates a sequence of photometry and auxiliary data from an FP3002 device.

Misc

AcquisitionMo	**StartPhotometry**
AutoCrop	**False**
PortName	**COM1**
Regions	

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.

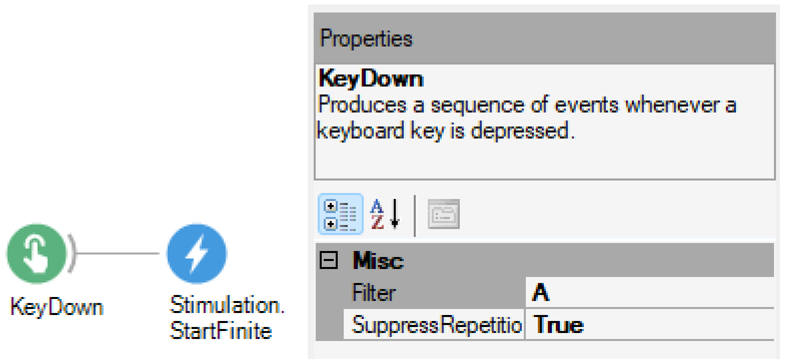

Properties
Stimulation.StartFinite (Stimulation)
Starts a finite laser pulse stimulation.

Misc
Command: **StartFinite**

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.



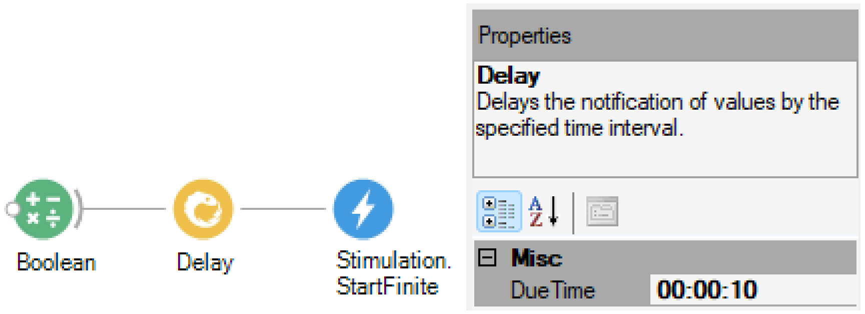
KeyDown
Produces a sequence of events whenever a keyboard key is depressed.

Misc

Filter	A
SuppressRepetitio	True

Automated Trigger:

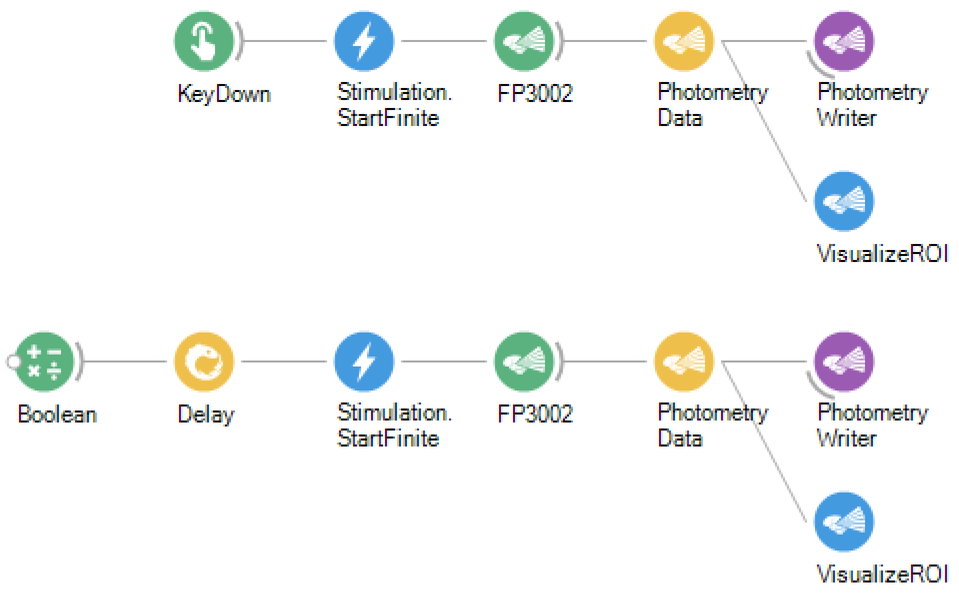
If instead, the desired delay to stimulation is to be a precise amount of time since the workflow has started, a “Boolean” node followed by a “Delay (Reactive)” node as the software trigger would be more appropriate. The “Delay (Reactive)” node delays the notification of values by the specified time 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.



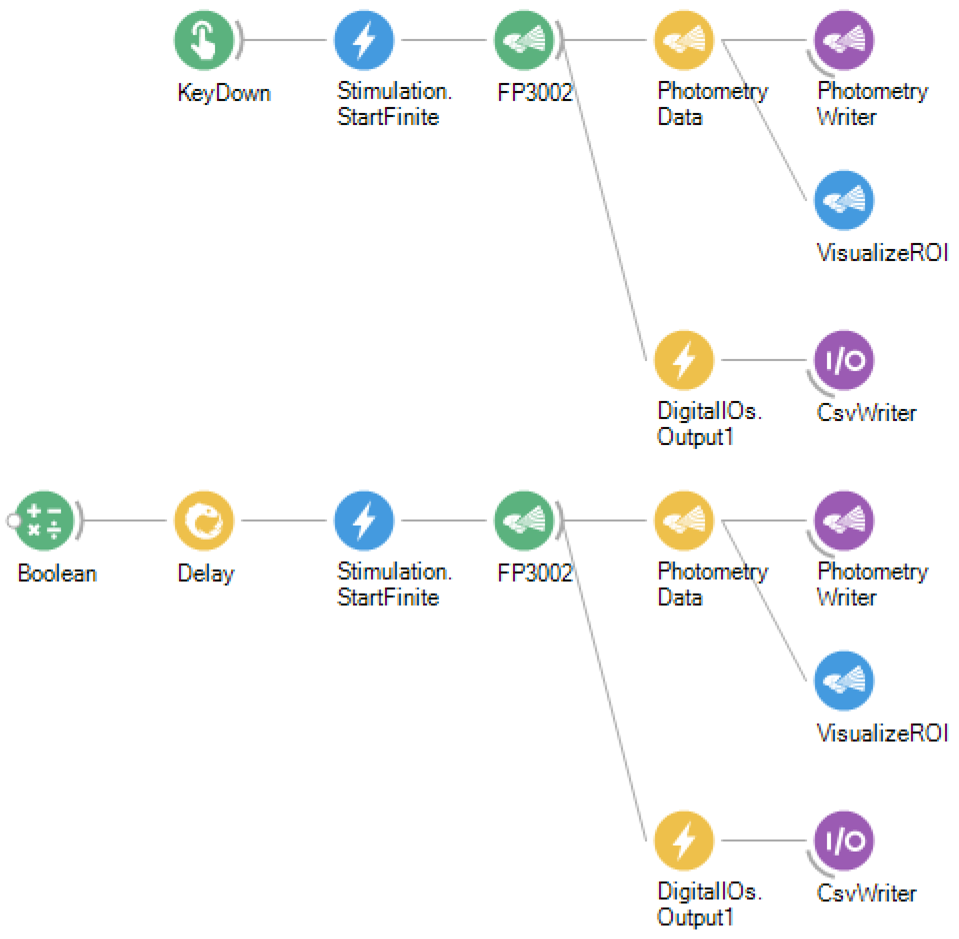
Delay
Delays the notification of values by the specified time interval.

Misc
DueTime 00:00:10

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

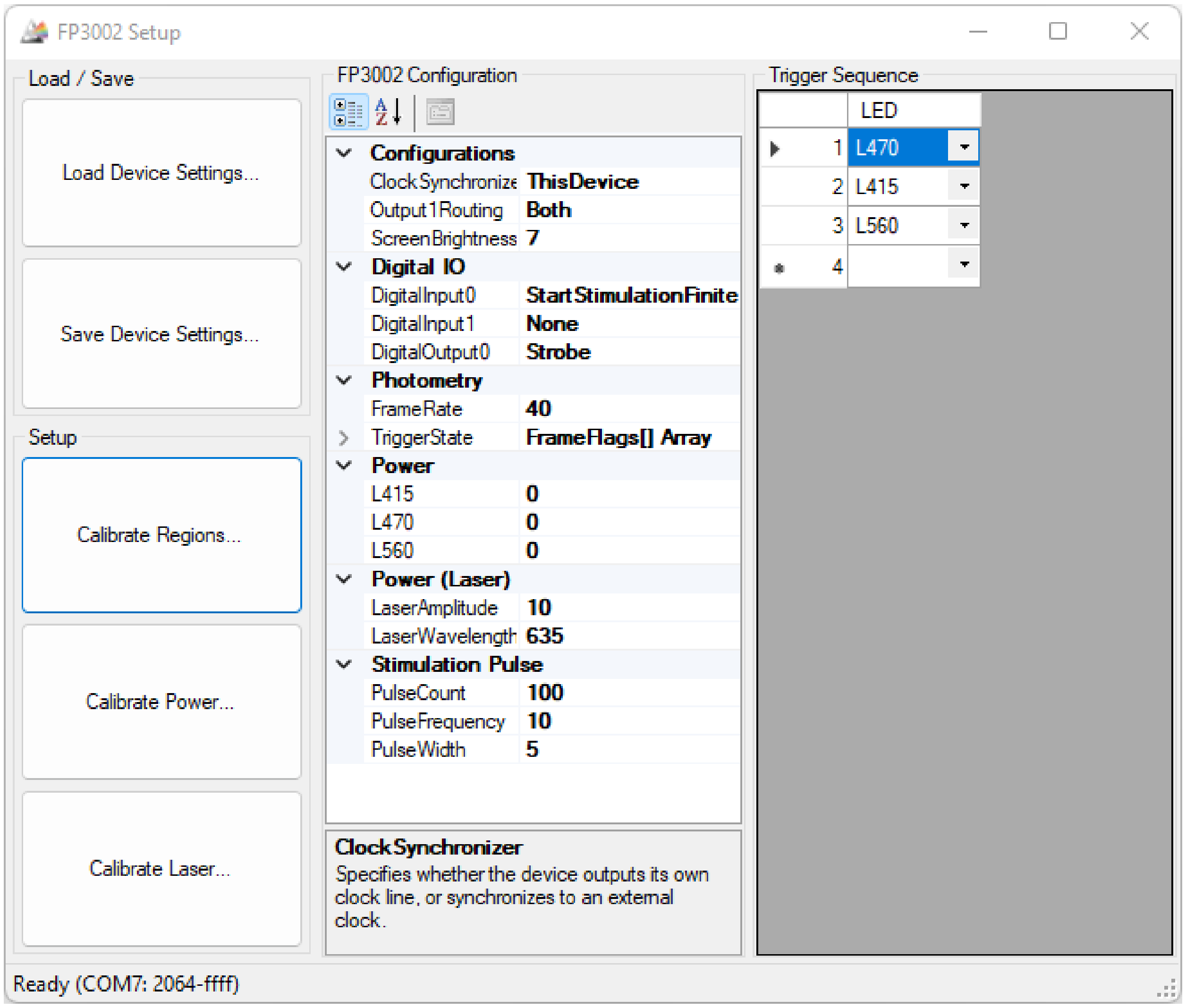


The above workflows successfully triggers a delayed start to stimulation during a photometry experiment. However, there is an important last step for constructing any stimulation workflow. The “FP3002” node knows precisely when the laser changes state. We can gain access to that information by using 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 “Type” property of the “Digital IOs” node to “Output 1”. Then set the “Include Timestamp” property 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.



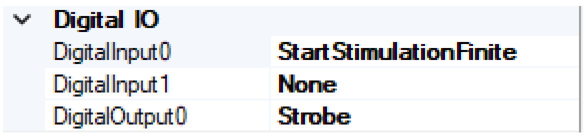
Configurations	
ClockSynchronize	ThisDevice
Output1Routing	Both
ScreenBrightness	7
Digital IO	
DigitalInput0	StartStimulationFinite
DigitalInput1	None
DigitalOutput0	Strobe
Photometry	
FrameRate	40
TriggerState	FrameFlags[] Array
Power	
L415	0
L470	0
L560	0
Power (Laser)	
LaserAmplitude	10
LaserWavelength	635
Stimulation Pulse	
PulseCount	100
PulseFrequency	10
PulseWidth	5

ClockSynchronizer
Specifies whether the device outputs its own clock line, or synchronizes to an external clock.

	LED
1	L470
2	L415
3	L560
4	

Ready (COM7: 2064-ffff)

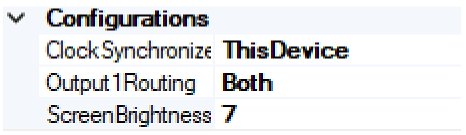
In the “Digital IOs” section, configure either the “Digital Input 0” or “Digital Input 1” setting to be “Start Stimulation Finite” or “Start Stimulation Continuous”.



Digital IO	
DigitalInput0	StartStimulationFinite
DigitalInput1	None
DigitalOutput0	Strobe

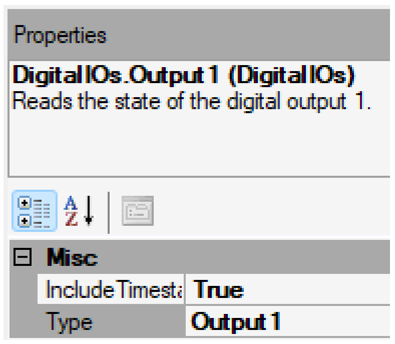
This will allow the +5V TTL signal on the specified port to control the stimulation of the FP3002 system. While the TTL signal is HIGH, the system will be stimulating and while the TTL signal is LOW, the system will not be stimulating. The pulse train parameters can be configured within the “Power (Laser)” and “Stimulation Pulse” sections. 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. You will notice that the “Start Stimulation Interleaved” setting is an option. However, this option is currently disabled in the FP3002 system’s firmware until future updates.

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.



Configurations	
ClockSynchronize	ThisDevice
Output1Routing	Both
ScreenBrightness	7

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 to have the “Include Timestamp” property set to “True” and the “Type” property set to “Output 1”.



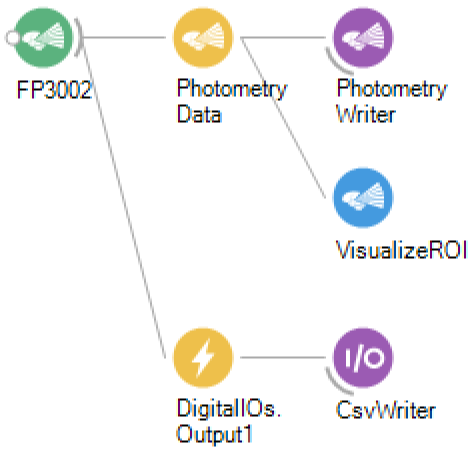
Properties

DigitalIOs.Output1 (DigitalIOs)
Reads the state of the digital output 1.

Misc

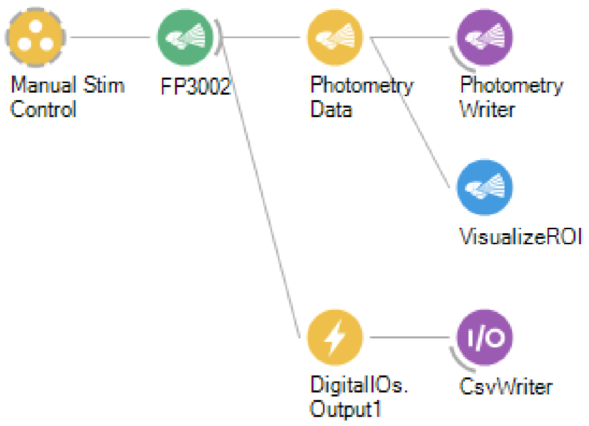
IncludeTimest	**True**
Type	**Output1**

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.

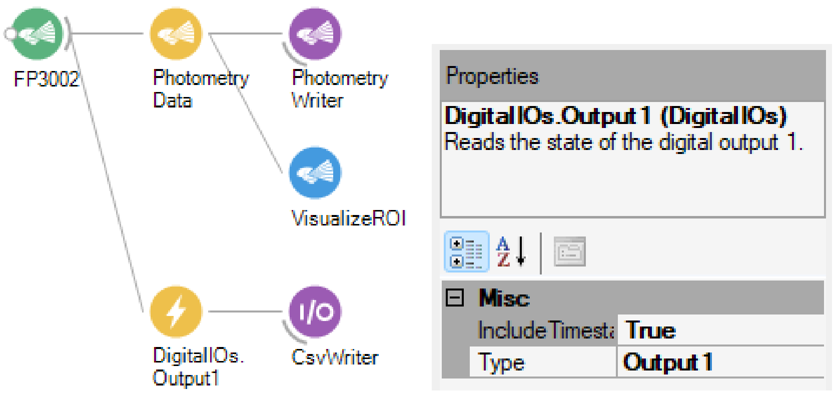


Manual Control

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 “Type” and “Include Timestamp” properties of the “Digital IOs” node to “Output 1” and “True” respectively. The “Digital IOs” node should also be connected to a “Csv Writer” in order to save the laser state signal to a *.csv* file.



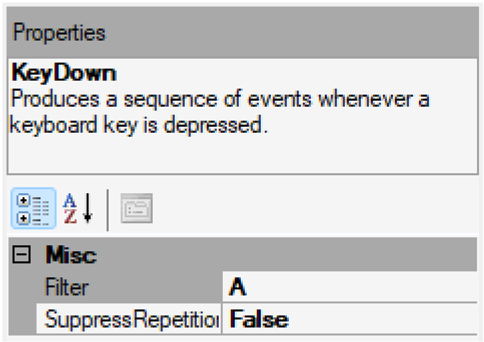
DigitalIOs.Output1 (DigitalIOs)
Reads the state of the digital output 1.

Misc

IncludeTimest	**True**
Type	**Output 1**

Case #1: Separate Keys

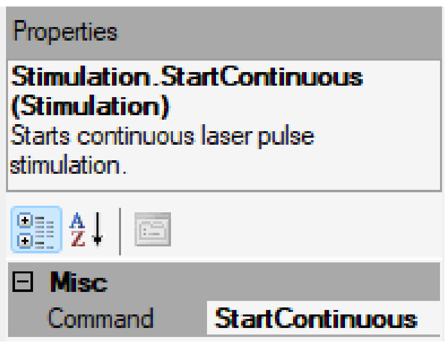
For the case in which separate keys will be used for the “Stop Stimulation” and “Start Stimulation” commands, pair two “Stimulation” nodes with two “Key Down” nodes. One “Stimulation” node should have a “Command” property of “Stop” and the other should have a “Command” property of “Start Finite” or “Start 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.


Properties

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

Misc

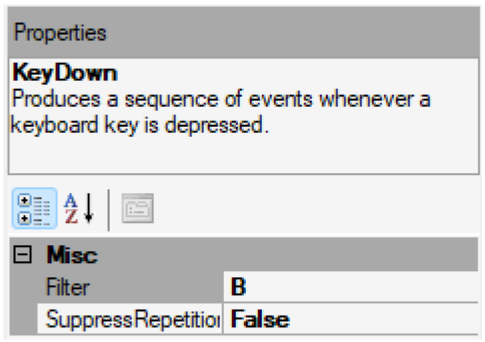
Filter	**A**
SuppressRepetitio	**False**


Properties

Stimulation.StartContinuous (Stimulation)
Starts continuous laser pulse stimulation.

Misc

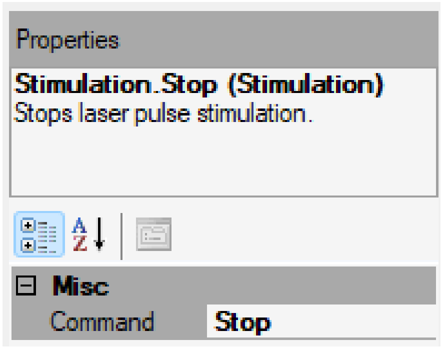
Command	**StartContinuous**


Properties

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

Misc

Filter	**B**
SuppressRepetitio	**False**

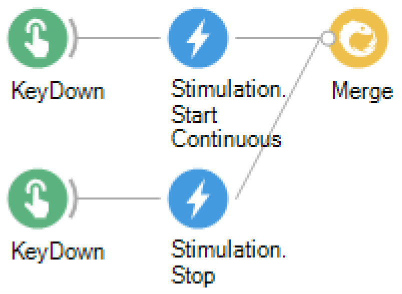

Properties

Stimulation.Stop (Stimulation)
Stops laser pulse stimulation.

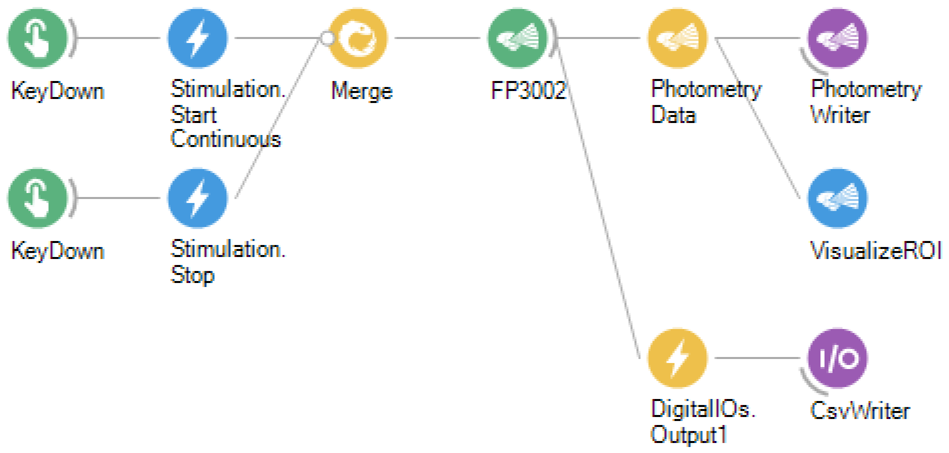
Misc

Command	**Stop**

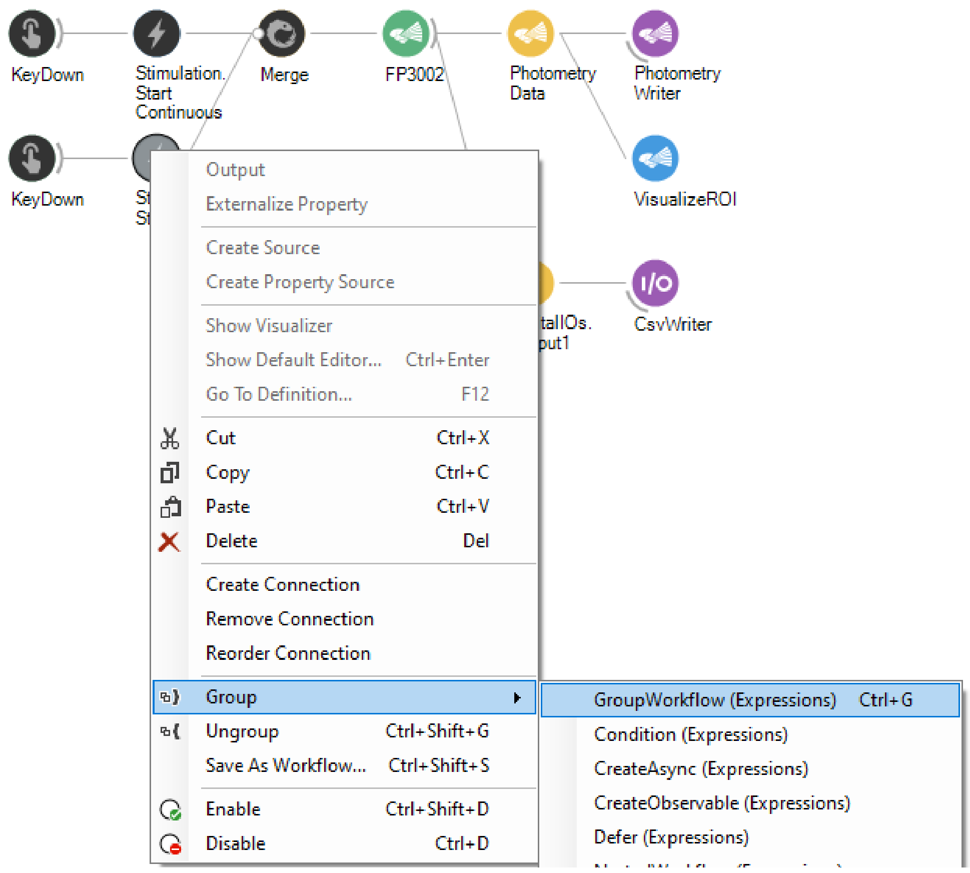
Currently, there are two data streams for producing commands to be sent to the "FP3002" node. These two data streams need to be converted to a single data stream without combining elements from both data streams into a different data type. For situations like this, the "Merge" node is ideal for converting 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.



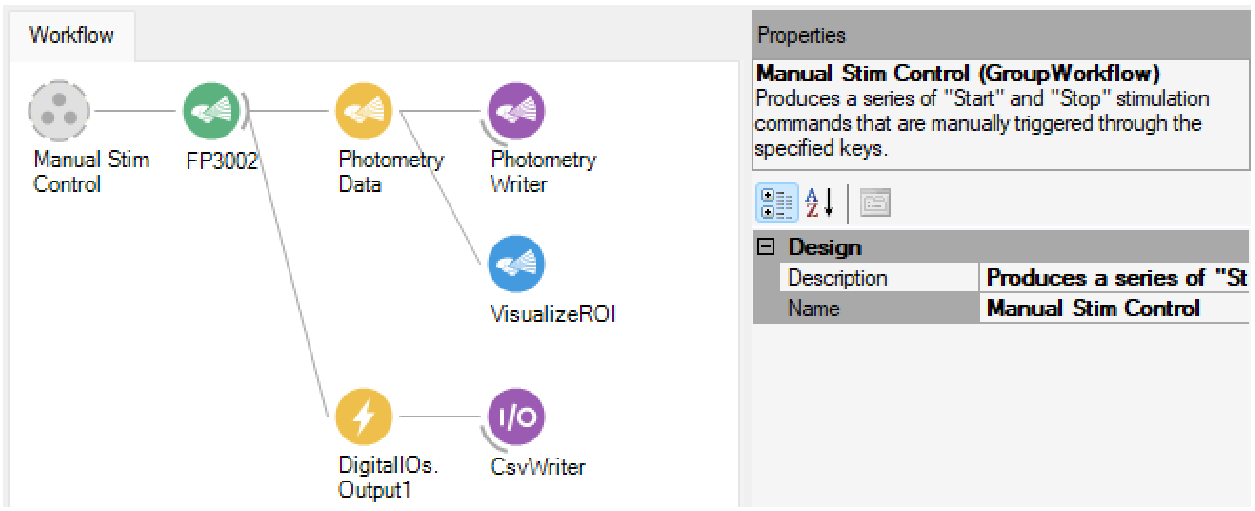
Now that the commands controlling stimulation are merged together into a single data stream, the merged data stream can be connected to the input of the "FP3002" node so that the commands can be sent to the FP3002 system after the workflow has been started.



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".



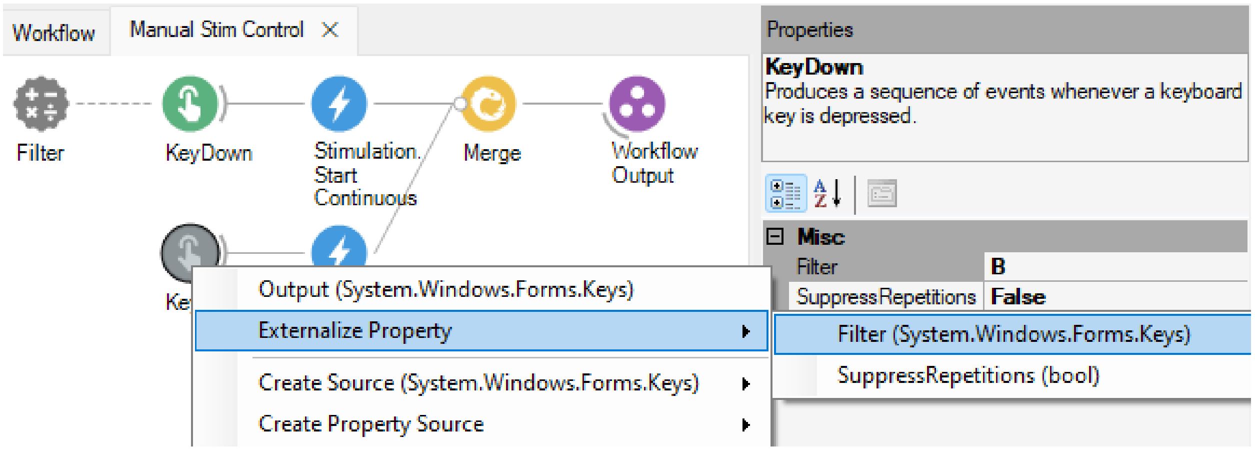
This will encapsulate the stimulation control nodes into a single grouped workflow that the user can name and provide a description for.



Manual Stim Control (GroupWorkflow)
Produces a series of "Start" and "Stop" stimulation commands that are manually triggered through the specified keys.

Design	
Description	Produces a series of "St
Name	Manual Stim Control

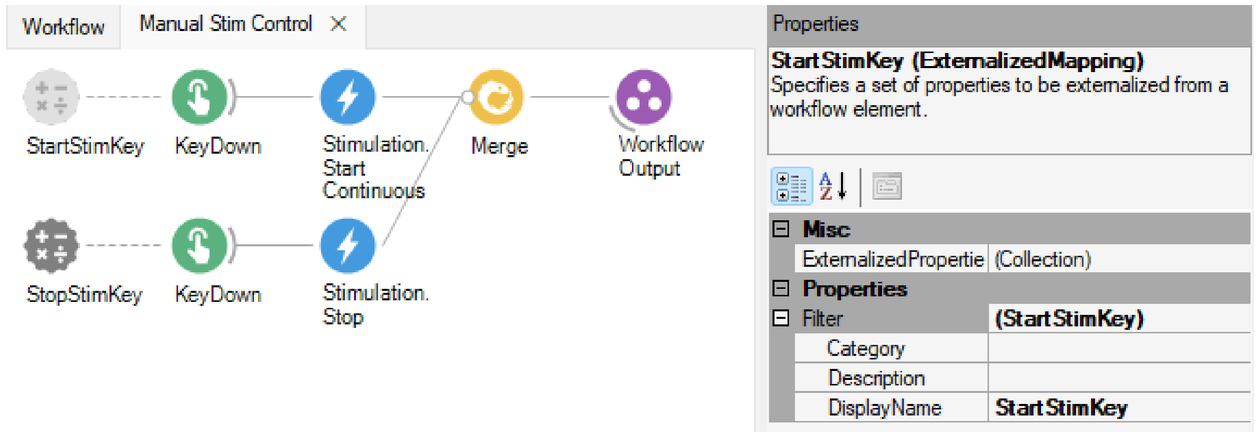
In order to make the "Filter" properties easily accessible by the user, open the grouped workflow by double clicking it. Then right click each "Key Down" node and select "Externalize Property → Filter". This will create an "Externalized Mapping" node for each "Key Down" node.


Workflow
Manual Stim Control
Filter
KeyDown
Stimulation. Start Continuous
Merge
Workflow Output
Ke
Properties
Output (System.Windows.Forms.Keys)
Externalize Property
Create Source (System.Windows.Forms.Keys)
Create Property Source
Filter (System.Windows.Forms.Keys)
SuppressRepetitions (bool)

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

Misc	
Filter	B
SuppressRepetitions	False

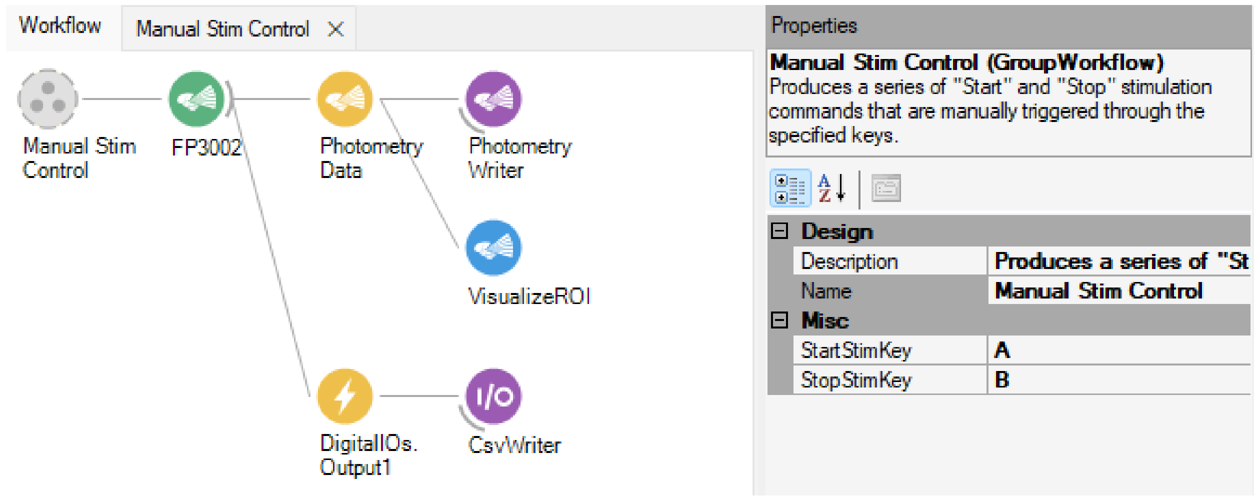
These need to have unique names, so be sure to expand the “Filter” property of the “Externalized Mapping” nodes and specify unique and informative “Display Name” properties for both nodes. In the example, the filters are named “Start Stim Key” and “Stop Stim Key”.



StartStimKey (ExternalizedMapping)
Specifies a set of properties to be externalized from a workflow element.

Misc	
ExternalizedPropertie	(Collection)
Properties	
Filter	(StartStimKey)
Category	
Description	
DisplayName	StartStimKey

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.

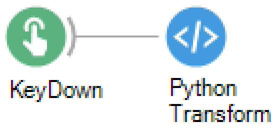


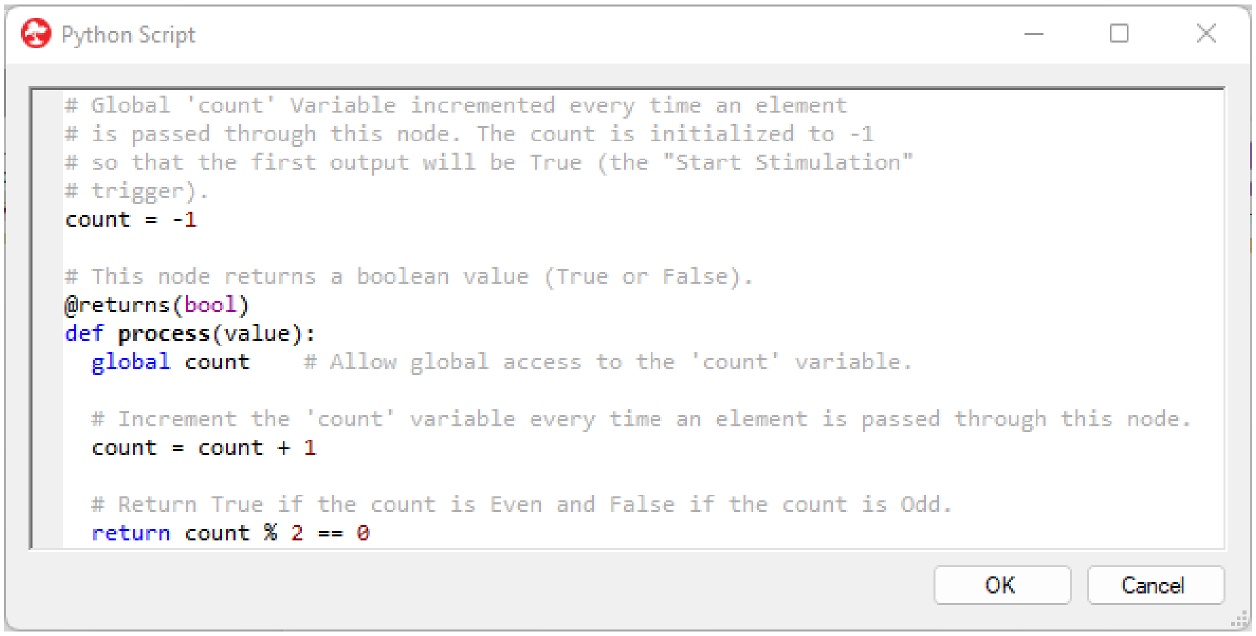
Manual Stim Control (GroupWorkflow)
Produces a series of "Start" and "Stop" stimulation commands that are manually triggered through the specified keys.

Design	
Description	Produces a series of "St
Name	Manual Stim Control
Misc	
StartStimKey	A
StopStimKey	B

Case #2: Togglable

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.



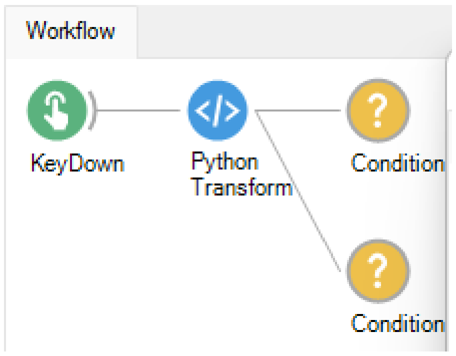


```
# Global 'count' Variable incremented every time an element
# is passed through this node. The count is initialized to -1
# so that the first output will be True (the "Start Stimulation"
# trigger).
count = -1

# 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
```

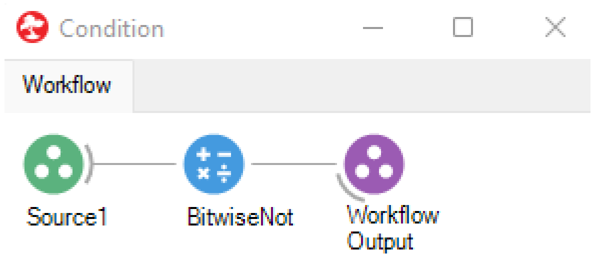
This python script initializes the ‘count’ variable to ‘-1’ when the Bonsai workflow is started. Then every time a value is passed through the node, the count variable is incremented. The script outputs $count \% 2 \;==\; 0$ which is True if the count is Even and False if the count is Odd.

NEUROPHOTOMETRICS
The fiber photometry company

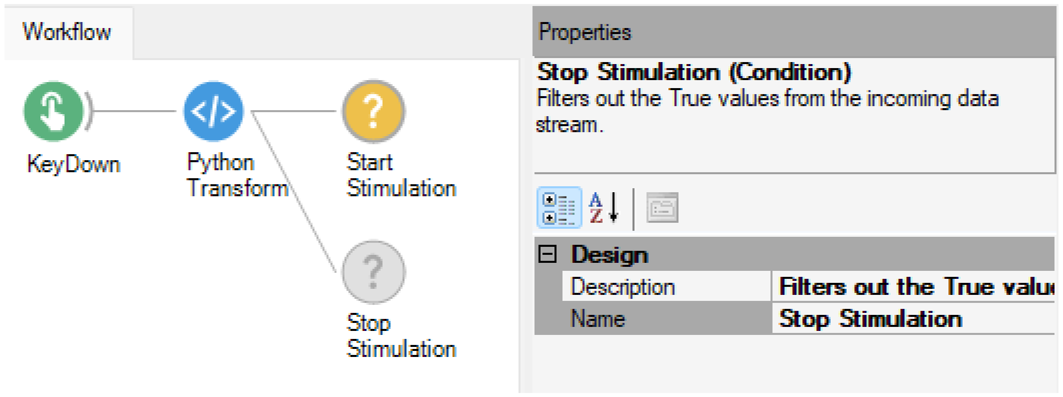
Once the software trigger is configured to alternate between True and False we need to have the True values send the “Start Stimulation” command to the “FP3002” node and we need the False values to send the “Stop Stimulation” command. In order to accomplish this, we need to separate the boolean values 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.

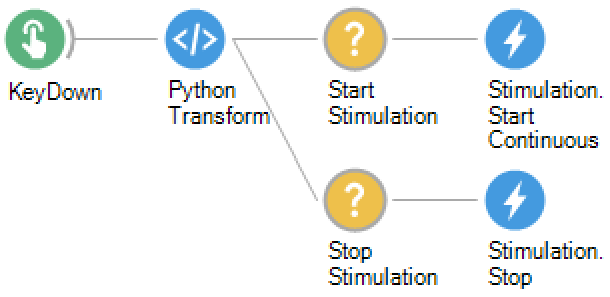


Stop Stimulation (Condition)
Filters out the True values from the incoming data stream.

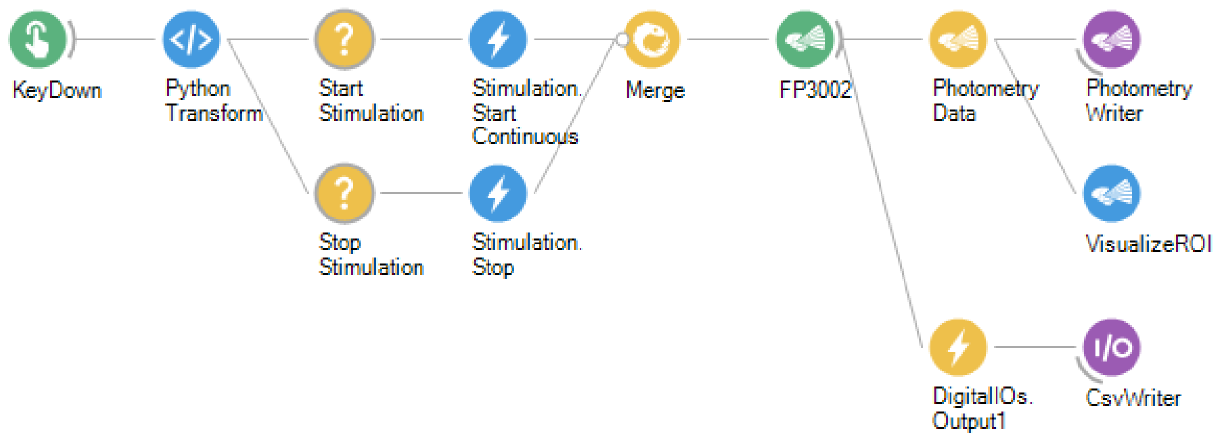
Design

Description	Filters out the True valu
Name	Stop Stimulation

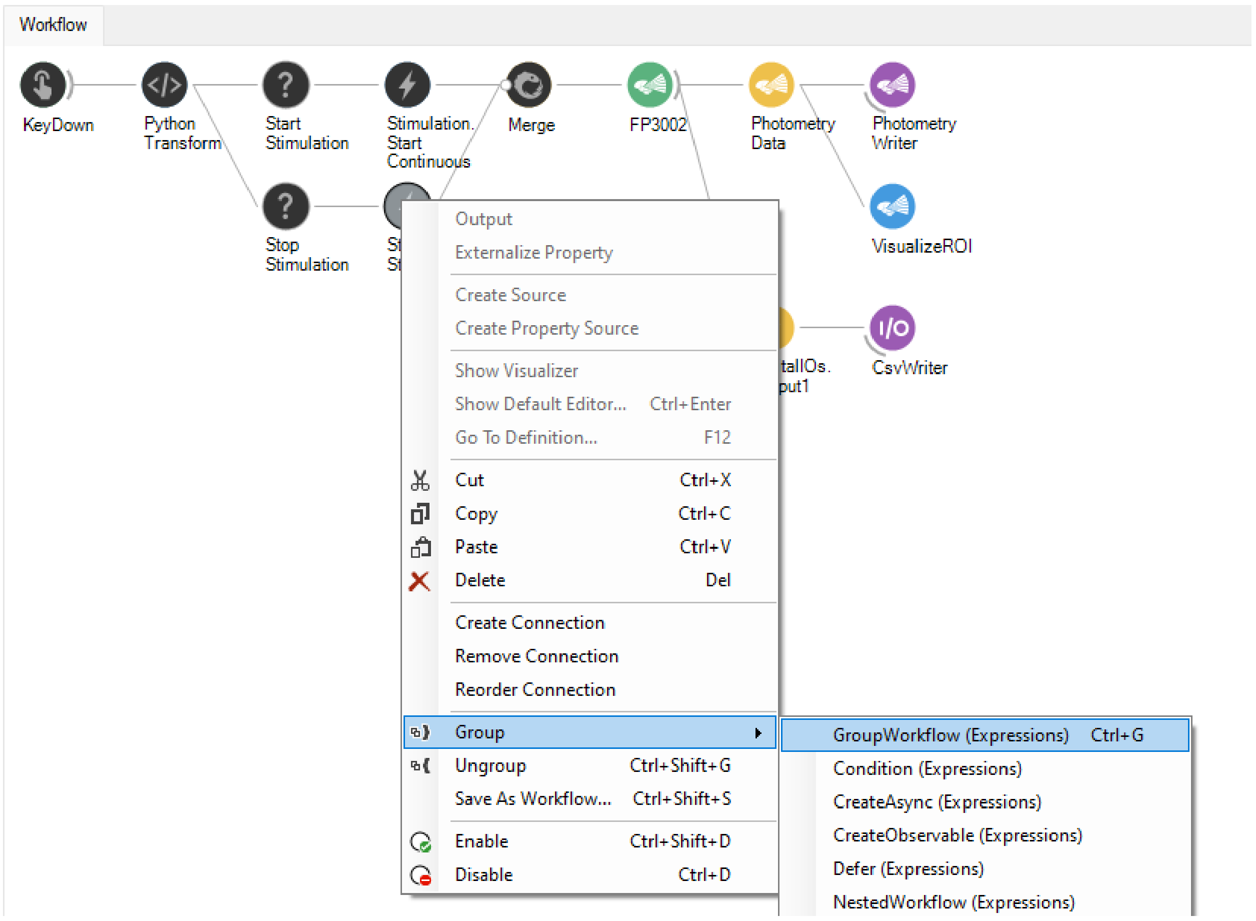
Now that we have the boolean values separated into two data streams, connect “Stimulation” nodes to each of the “Condition” nodes. Configure the “Stimulation” nodes such that the one receiving True values outputs a “Start Stimulation” command, and the other outputs the “Stop Stimulation” command.



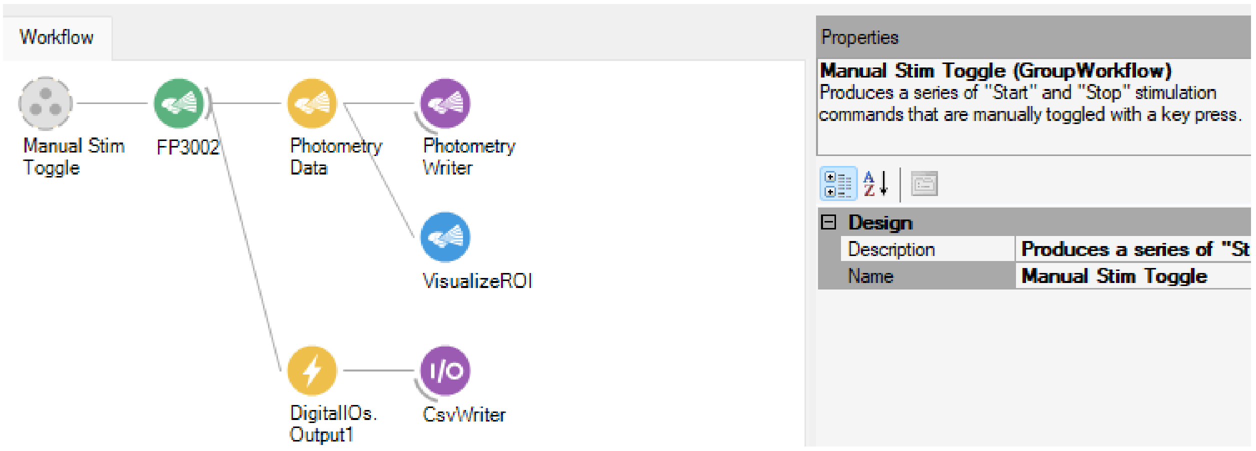
We now have to convert these two data streams outputting stimulation commands into a single data stream that outputs the most recent stimulation command. To do this, connect both of the “Stimulation” nodes to a single “Merge” node. This “Merge” node will output the most recent command produced by the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



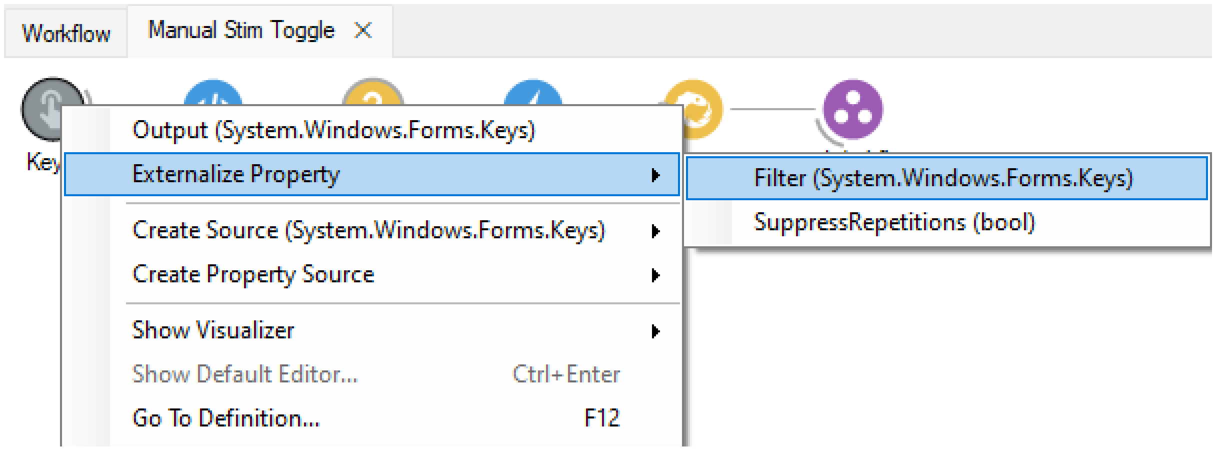
This will encapsulate the stimulation control nodes into a single grouped workflow that the user can name and provide a description for.



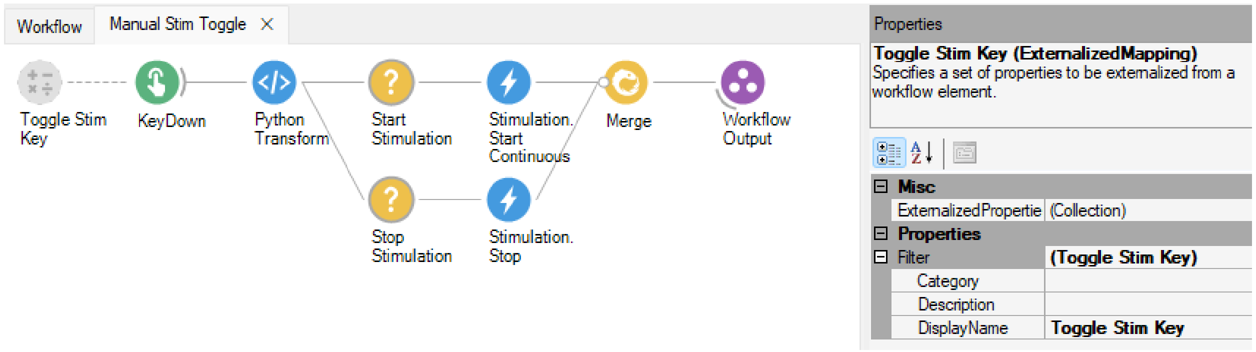
Manual Stim Toggle (GroupWorkflow)
Produces a series of "Start" and "Stop" stimulation commands that are manually toggled with a key press.

Description	Produces a series of "St
Name	Manual Stim Toggle

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.


Workflow
Manual Stim Toggle
Key
Output (System.Windows.Forms.Keys)
Externalize Property
Filter (System.Windows.Forms.Keys)
SuppressRepetitions (bool)
Create Source (System.Windows.Forms.Keys)
Create Property Source
Show Visualizer
Show Default Editor... Ctrl+Enter
Go To Definition... F12

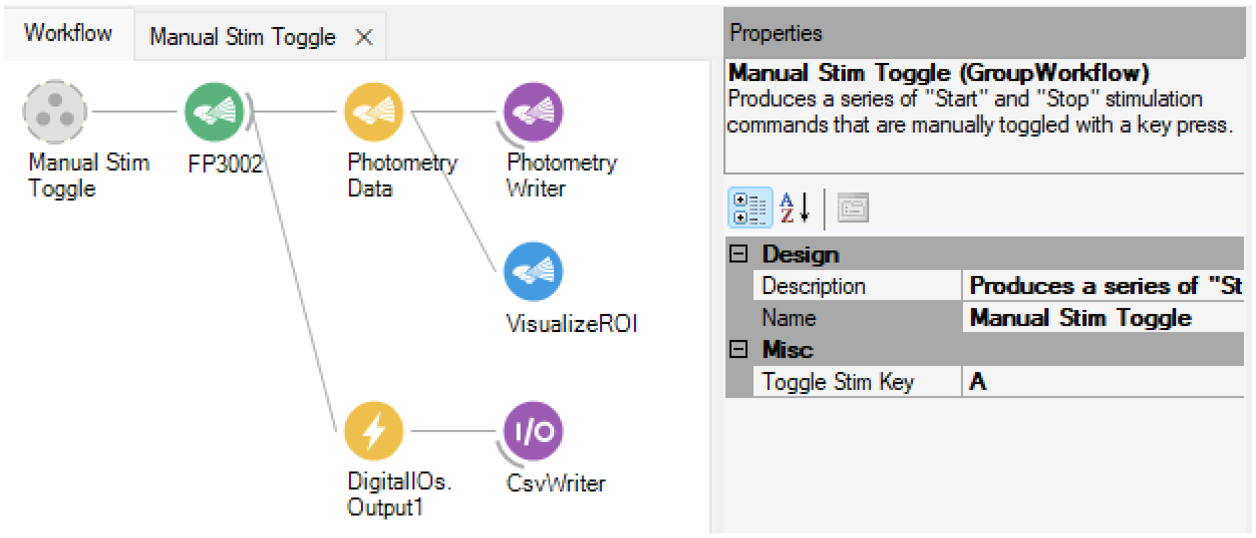
You can specify the display name of the "Filter" property by expanding the "Filter" property of the "Externalized Mapping" node and specifying a unique and informative "Display Name" property. In the example, the filter property is named "Toggle Stim Key".



Toggle Stim Key (ExternalizedMapping)
Specifies a set of properties to be externalized from a workflow element.

Misc	
ExternalizedPropertie	(Collection)
Properties	
Filter	**(Toggle Stim Key)**
Category	
Description	
DisplayName	**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.

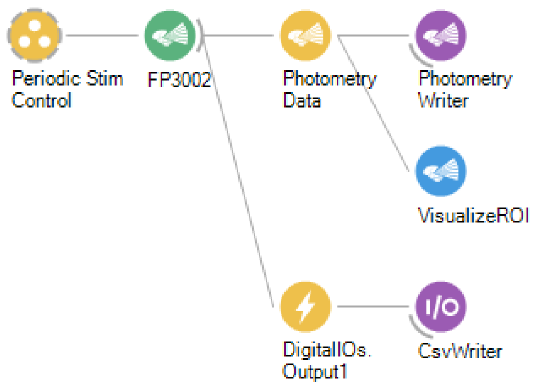


Manual Stim Toggle (GroupWorkflow)
Produces a series of "Start" and "Stop" stimulation commands that are manually toggled with a key press.

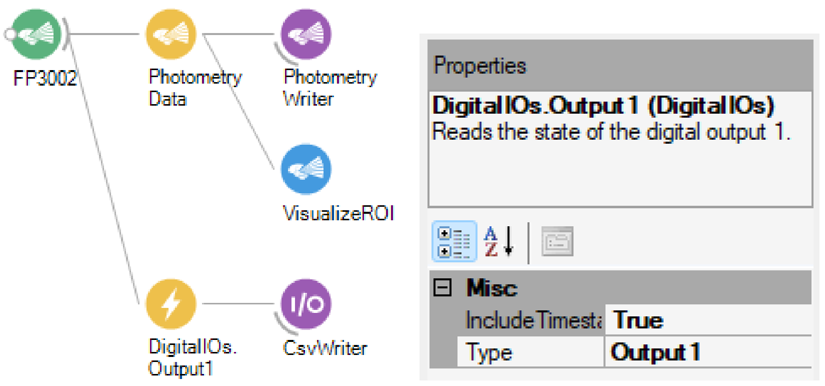
Design	
Description	**Produces a series of "St**
Name	**Manual Stim Toggle**
Misc	
Toggle Stim Key	**A**

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.



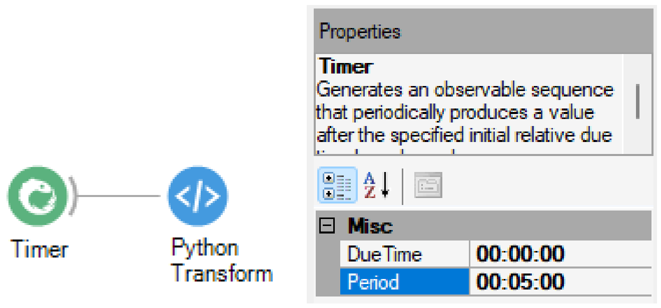
To construct this workflow, begin with the "Standard Photometry" workflow and add the "Digital IOs" node in parallel to the "Photometry Data" node. This way the workflow will be able to record both the photometry data and the laser state. Be sure to set the "Type" and "Include Timestamp" properties of the "Digital IOs" node to "Output 1" and "True" respectively. The "Digital IOs" node should also be connected to a "Csv Writer" in order to actually save the laser state signal to a *.csv* file.



DigitalIOs.Output1 (DigitalIOs)
Reads the state of the digital output 1.

IncludeTimesta	True
Type	Output1

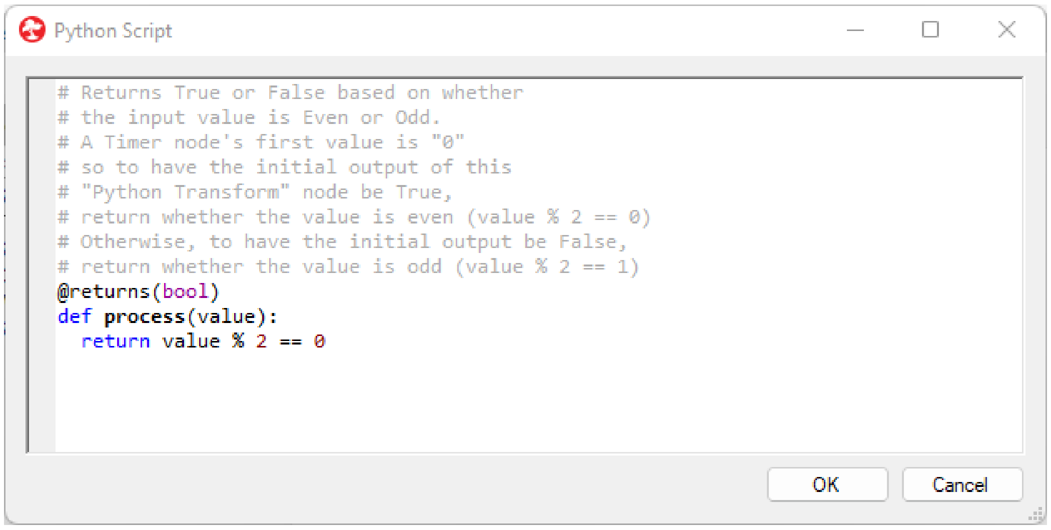
The rest of the workflow consists of logic for controlling the production of the stimulation commands. One way to construct the software trigger logic is to connect a “Timer” node to a “Python Transform” node. The “Timer” node can be configured to output an incremented value after a specified time interval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



Timer
Generates an observable sequence that periodically produces a value after the specified initial relative due

Misc
DueTime 00:00:00
Period 00:05:00

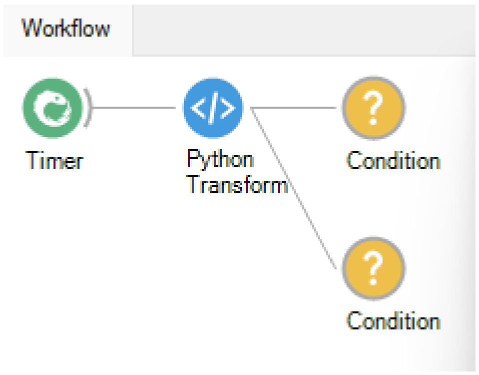
To properly configure the “Timer” node, set the “DueTime” to zero, unless a delay to the first command is desired. Then specify the “Period” property to be the duration of time for stimulation to occur each cycle. Since the duty cycle of this particular workflow is 50%, this will also be the period of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



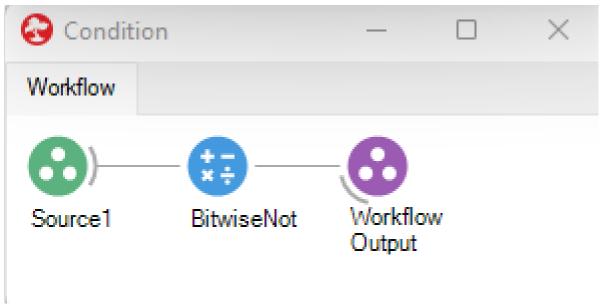
```
# Returns True or False based on whether
# the input value is Even or Odd.
# A Timer node's first value is "0"
# so to have the initial output of this
# "Python Transform" node be True,
# return whether the value is even (value % 2 == 0)
# Otherwise, to have the initial output be False,
# return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

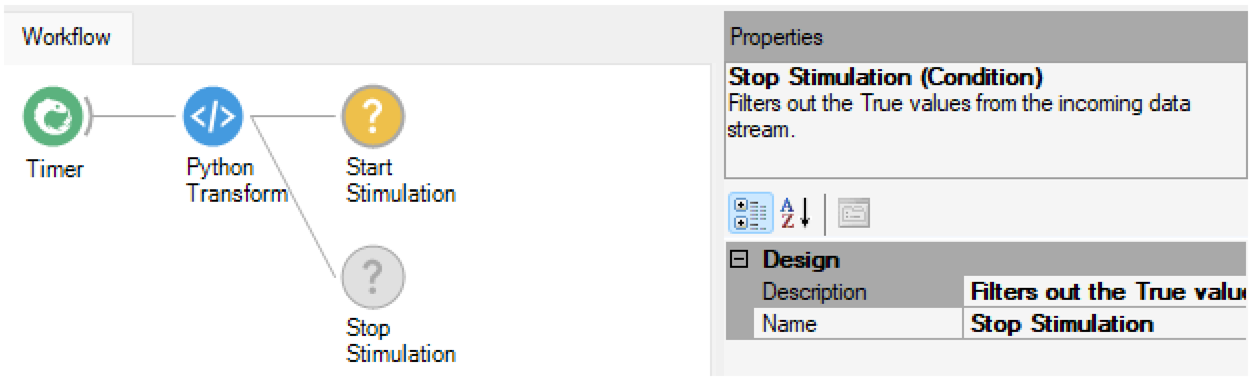
Once the software trigger is configured to alternate between True and False we need to have the True values send the "Start Stimulation" command to the "FP3002" node and we need the False values to send the "Stop Stimulation" command. In order to accomplish this, we need to separate the boolean values 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.



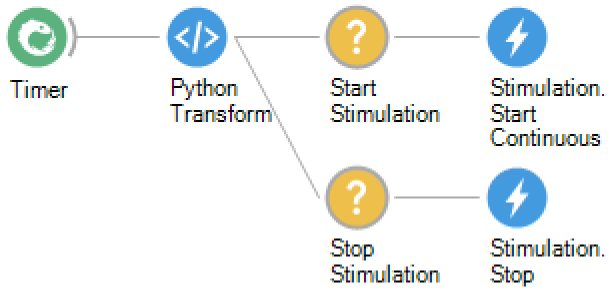
The “Condition” nodes also give the option to set the name and description of the node. It is useful to utilize these options to keep the logic organized.



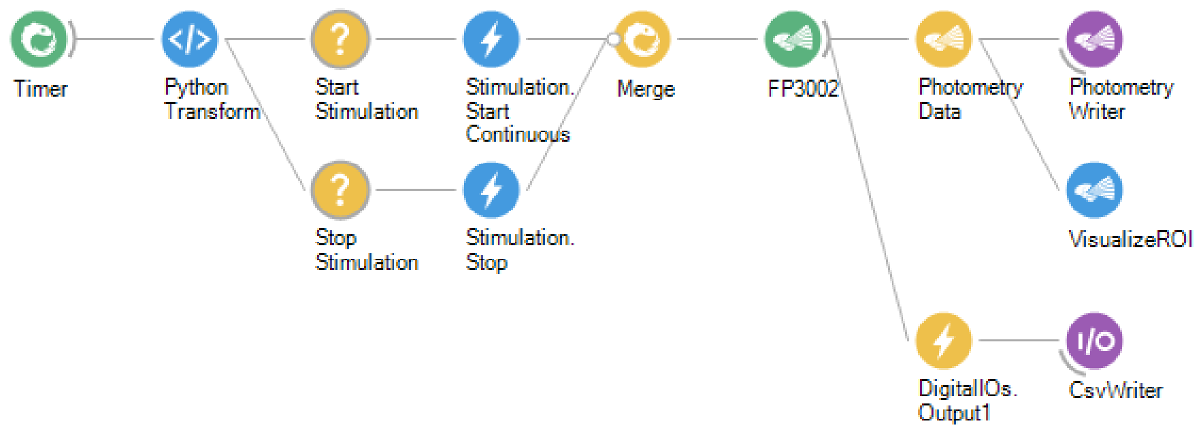
Stop Stimulation (Condition)
Filters out the True values from the incoming data stream.

Description	Filters out the True valu
Name	Stop Stimulation

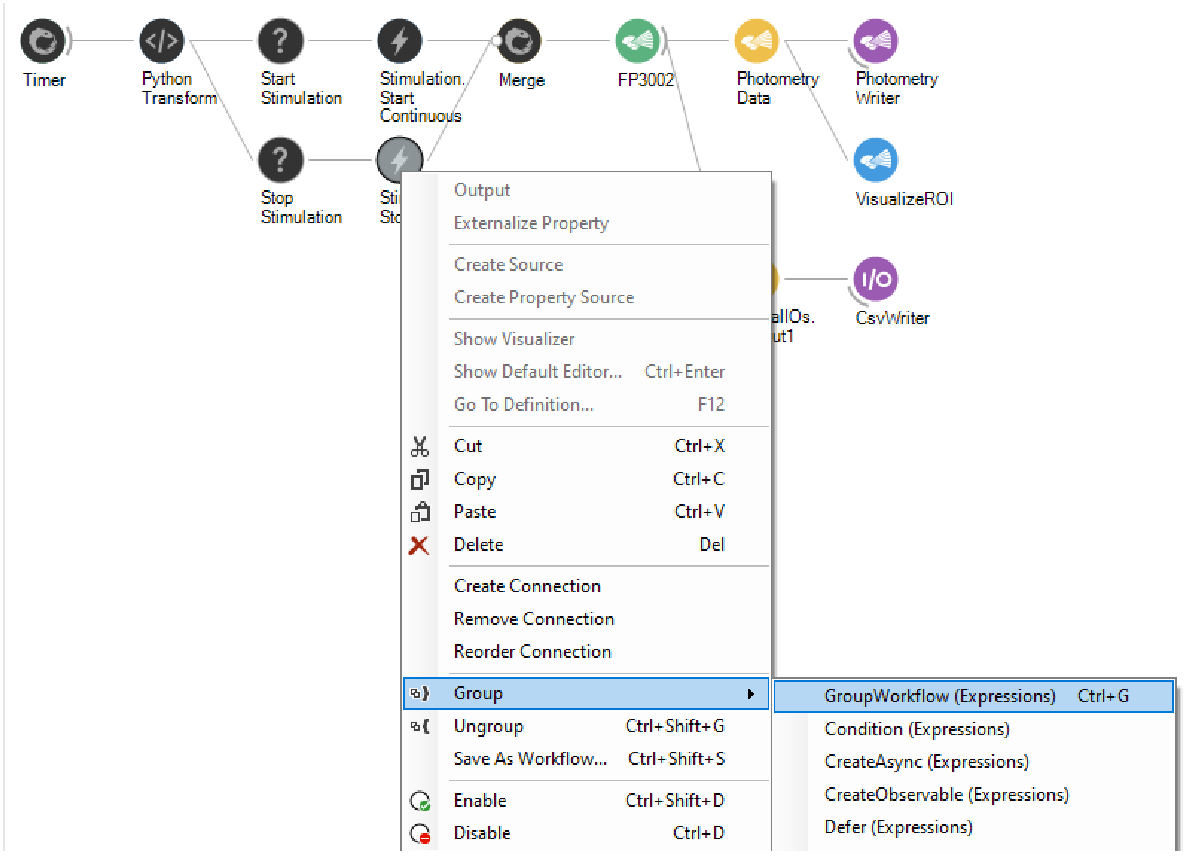
Now that we have the boolean values separated into two data streams, connect “Stimulation” nodes to each of the “Condition” nodes. Configure the “Stimulation” nodes such that the one receiving True values outputs a “Start Stimulation” command, and the other outputs the “Stop Stimulation” command.



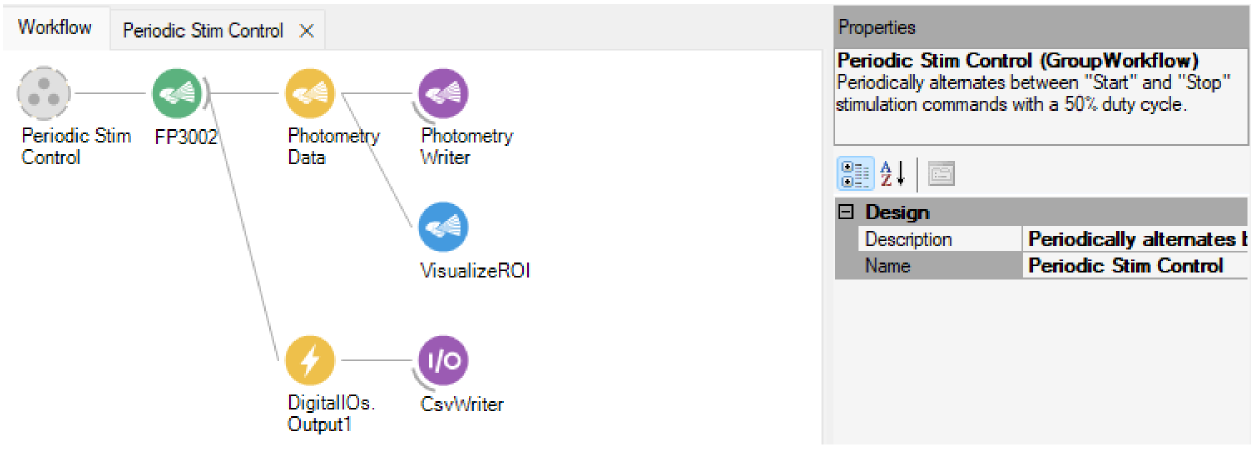
We now have to convert these two data streams outputting stimulation commands into a single data stream that outputs the most recent stimulation command. To do this, connect both of the "Stimulation" nodes to a single "Merge" node. This "Merge" node will output the most recent command produced by the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



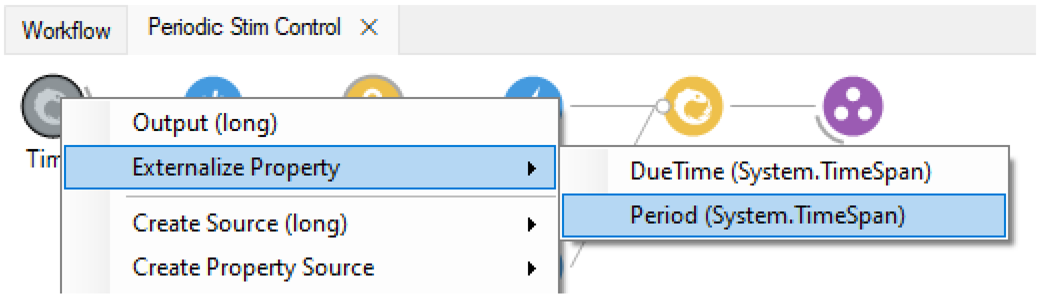
This will encapsulate the stimulation control nodes into a single grouped workflow that the user can name and provide a description for.



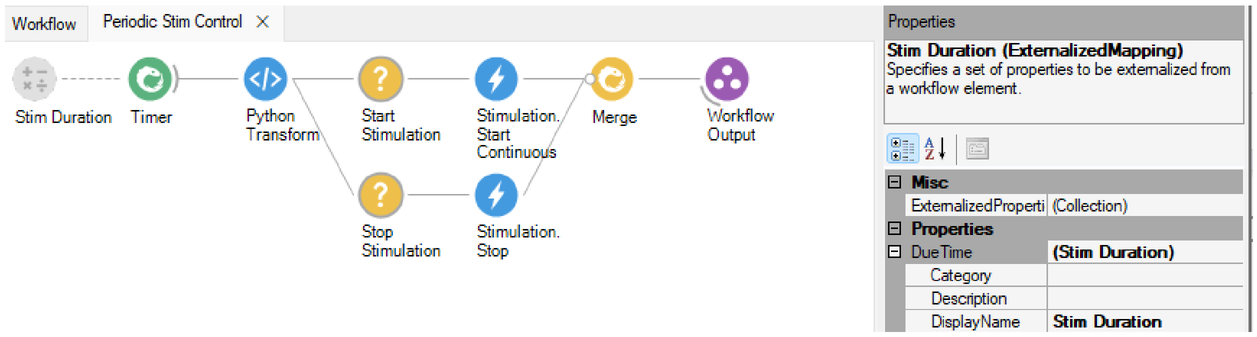
Periodic Stim Control (GroupWorkflow)
Periodically alternates between "Start" and "Stop" stimulation commands with a 50% duty cycle.

Description	Periodically alternates b
Name	Periodic Stim Control

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.



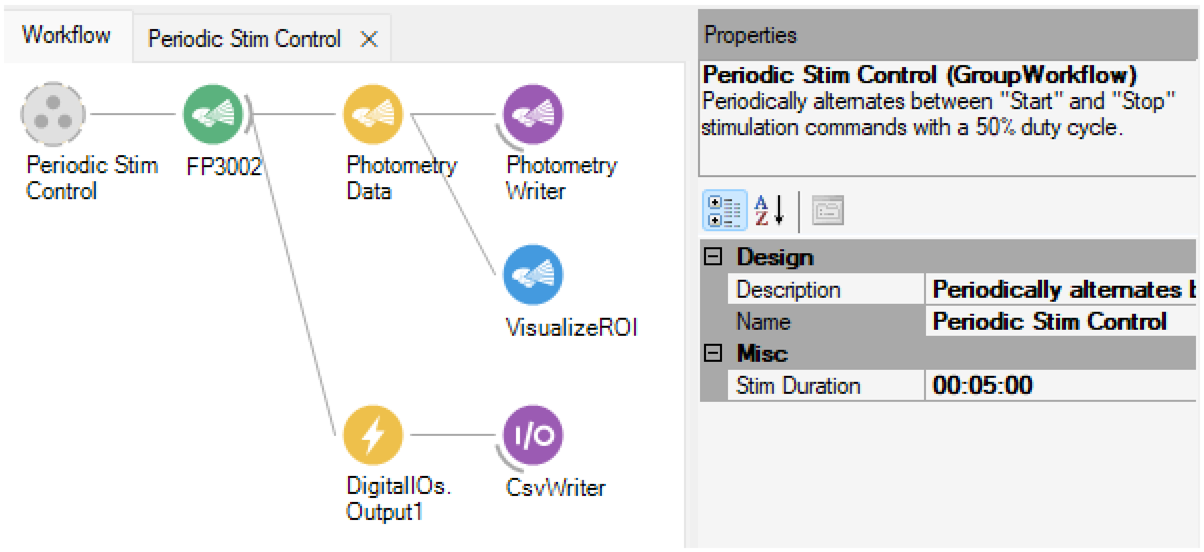
You can specify the display name of the “Period” property by expanding the “Period” property of the “Externalized Mapping” node and specifying a unique and informative “Display Name” property. In the example, the due time property is named “Stim Duration”.



Stim Duration (ExternalizedMapping)
Specifies a set of properties to be externalized from a workflow element.

Misc	
ExternalizedProperti	(Collection)
Properties	
DueTime	**(Stim Duration)**
Category	
Description	
DisplayName	**Stim Duration**

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



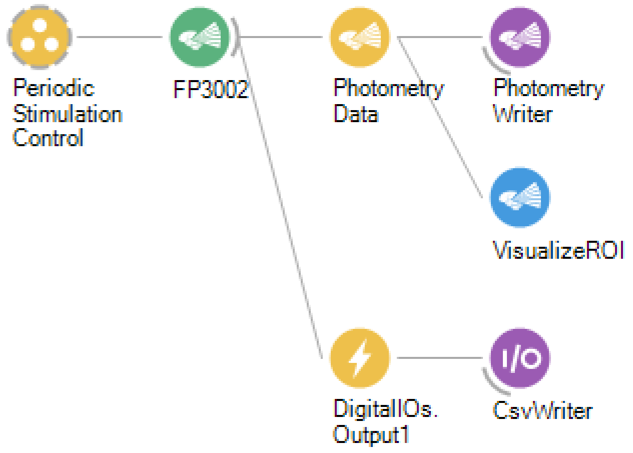
Periodic Stim Control (GroupWorkflow)
Periodically alternates between "Start" and "Stop" stimulation commands with a 50% duty cycle.

Design	
Description	**Periodically alternates b**
Name	**Periodic Stim Control**
Misc	
Stim Duration	**00:05:00**

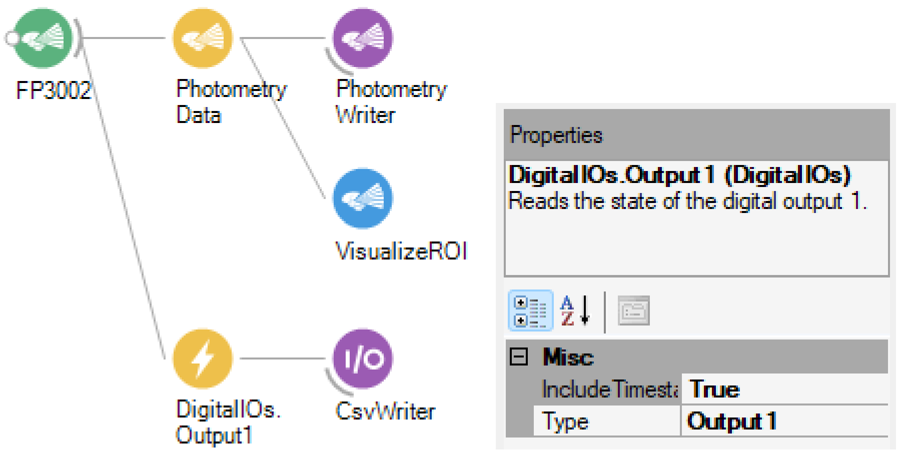
NEUROPHOTOMETRICS
The fiber photometry company

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 “Type” and “Include Timestamp” properties of the “Digital IOs” node to “Output 1” and “True” respectively. The “Digital IOs” node should also be connected to a “CsvWriter” in order to actually save the laser state signal to a *.csv* file.

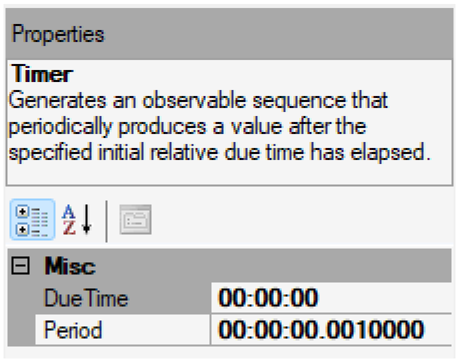


DigitalIOs.Output1 (DigitalIOs)
Reads the state of the digital output 1.

Misc

IncludeTimest	**True**
Type	**Output 1**

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.



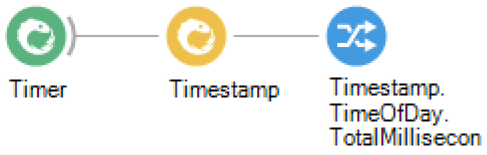
Properties

Timer
Generates an observable sequence that periodically produces a value after the specified initial relative due time has elapsed.

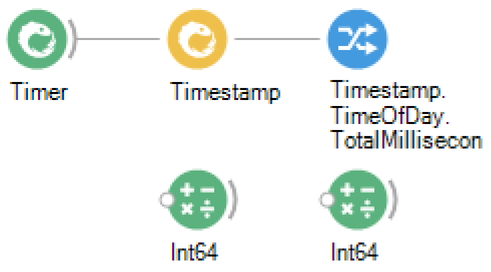
Misc

DueTime	00:00:00
Period	00:00:00.0010000

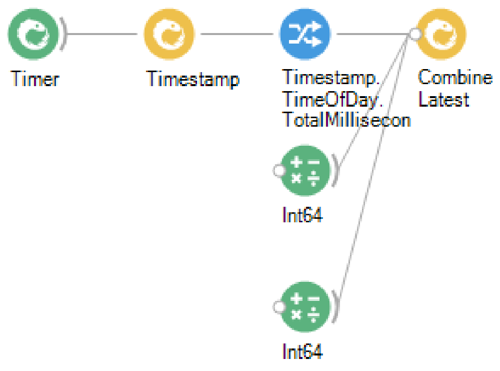
The “Timer” node does not have the precision to output a value every 1ms so it is advised to actually timestamp the values coming from this node using the computer’s timestamp. To do this, connect a “Timestamp” node after the “Timer” node. Then select the output of the “Timestamp” node to be “Time Of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



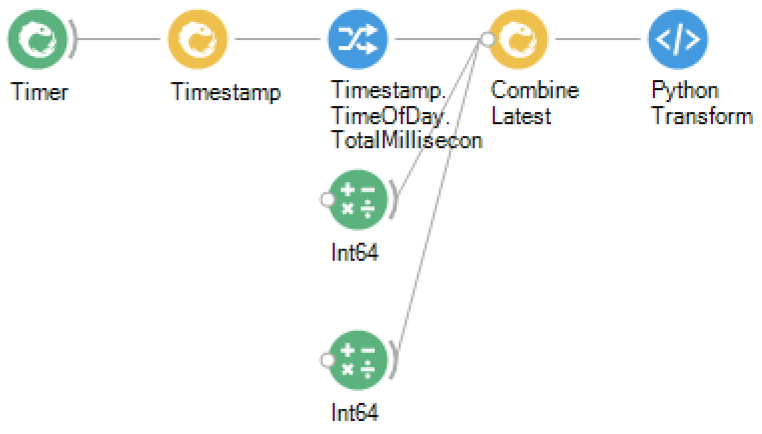
The goal for this software trigger is to alternate between outputting True and False values, where the user can specify how long to stay in each state. With this in mind, we need a way to store the duration spent stimulating versus not stimulating. This is easily done by inserting two “Int64” nodes into 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.

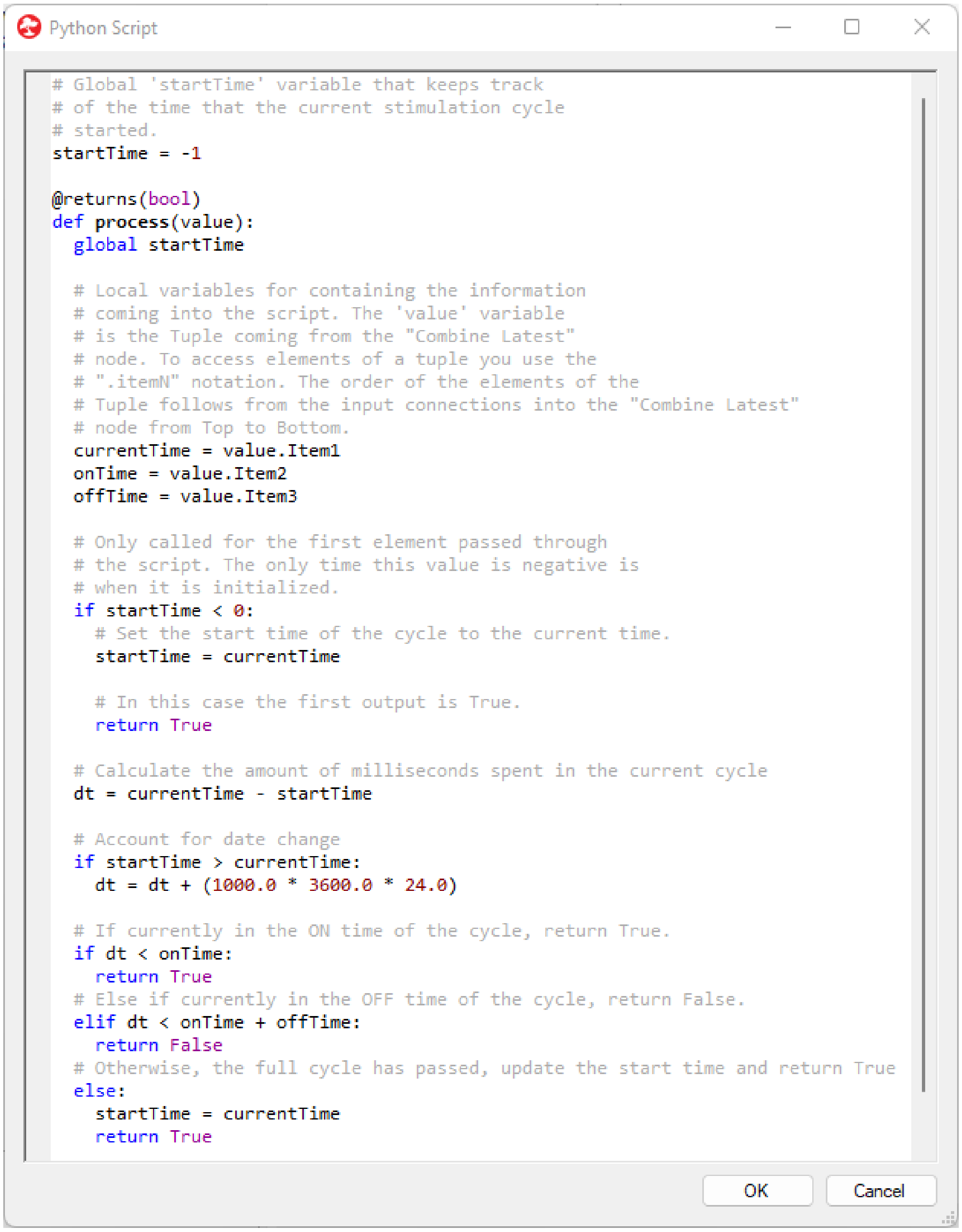


With these three data streams, we have all the information we need to pass through a python script to periodically output boolean values with the desired duty cycle. However, to get this information into a “Python Transform” node, these three data streams must be combined. In this particular case, connecting 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.



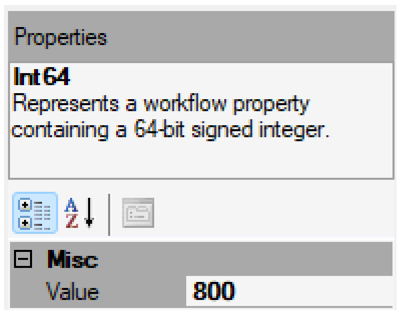
Now that we have one data stream containing all the information we need, we can connect a “Python Transform” node after the “Combine Latest” node to process the information and output the appropriate boolean value. In this python script, we will keep track of each time a new stimulation cycle starts, 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.

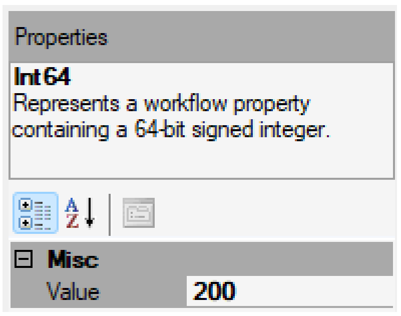




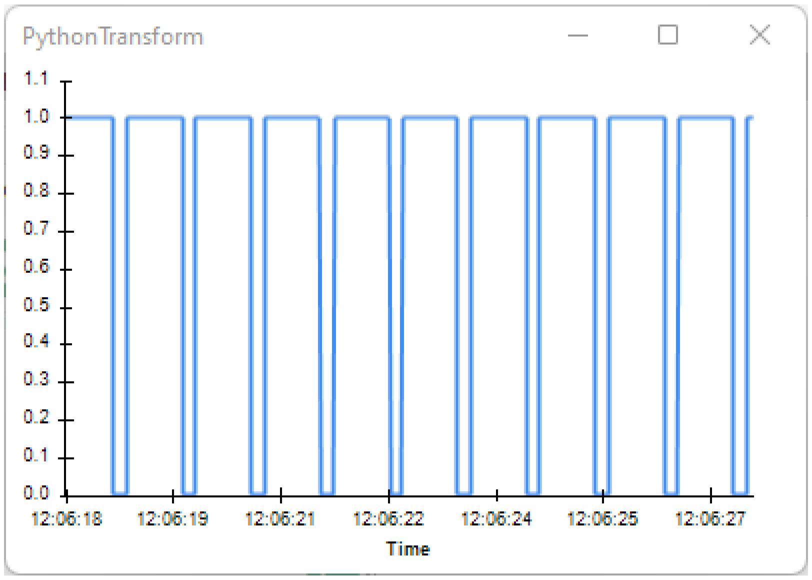
```python
# Global 'startTime' variable that keeps track
# of the time that the current stimulation cycle
# started.
startTime = -1

@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
```

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.

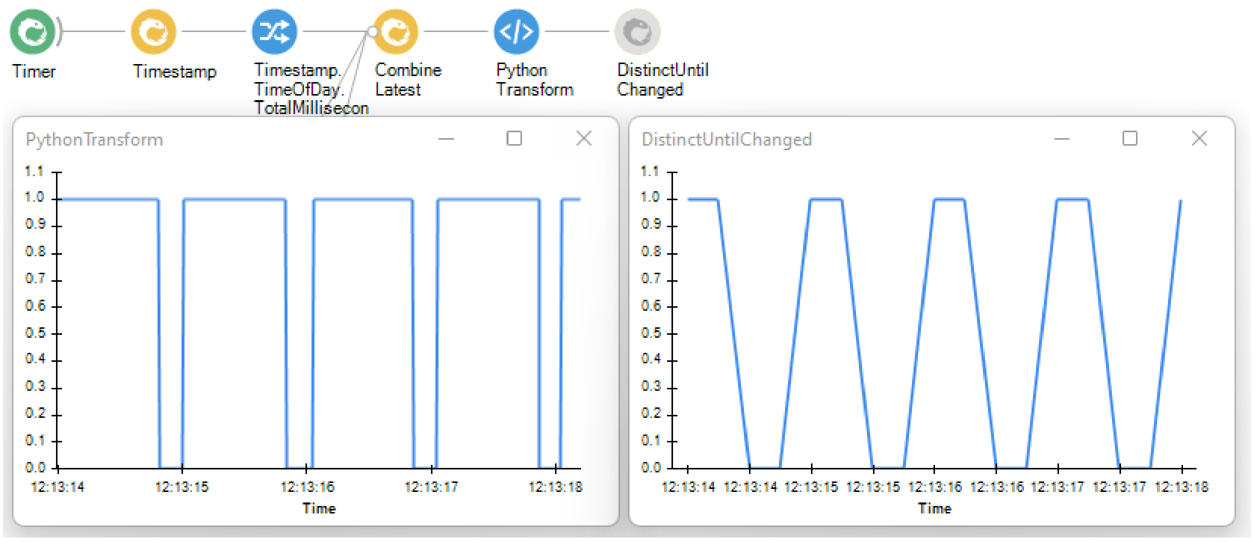

Properties
Int64
Represents a workflow property containing a 64-bit signed integer.
Misc
Value 800


Properties
Int64
Represents a workflow property containing a 64-bit signed integer.
Misc
Value 200

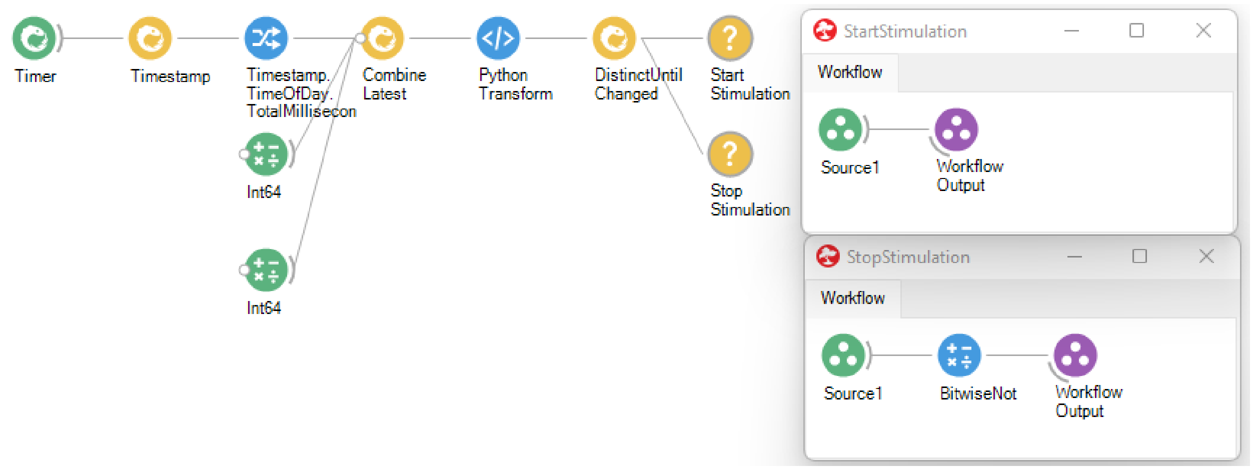
Once these values are set, run this software trigger section of the workflow and open the visualizer for the “Python Transform” node to check if our boolean signal appears correct.



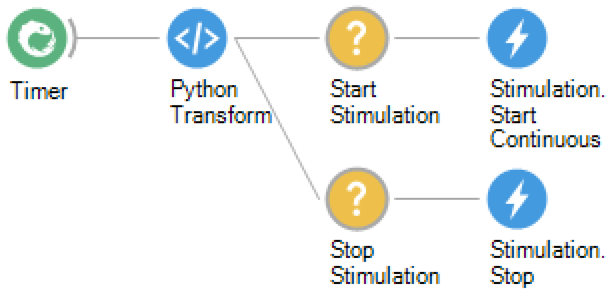
The above signal appears correct, the “Python Transform” successfully outputs True values for 800ms then False values for 200ms and repeats itself. However, this “Python Transform” outputs a value every time the timer’s data stream has a new value. If this were connected to our “Stimulation” node logic, 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.



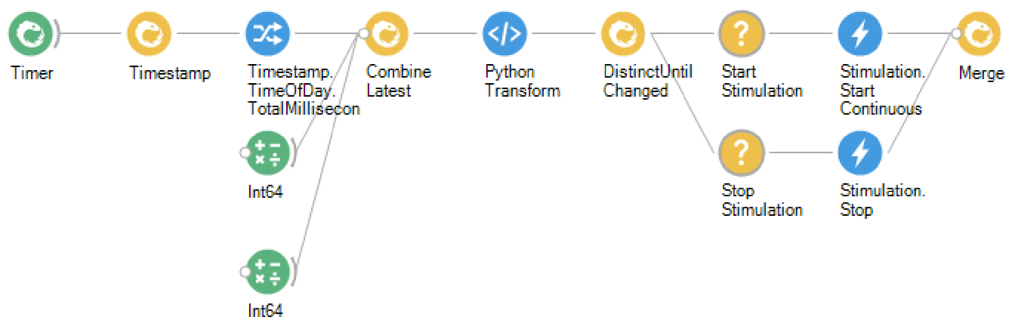
Once the software trigger is configured to alternate between True and False we need to have the True values send the “Start Stimulation” command to the “FP3002” node and we need the False values to send the “Stop Stimulation” command. Here is where we implement the “Condition” nodes to split the incoming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



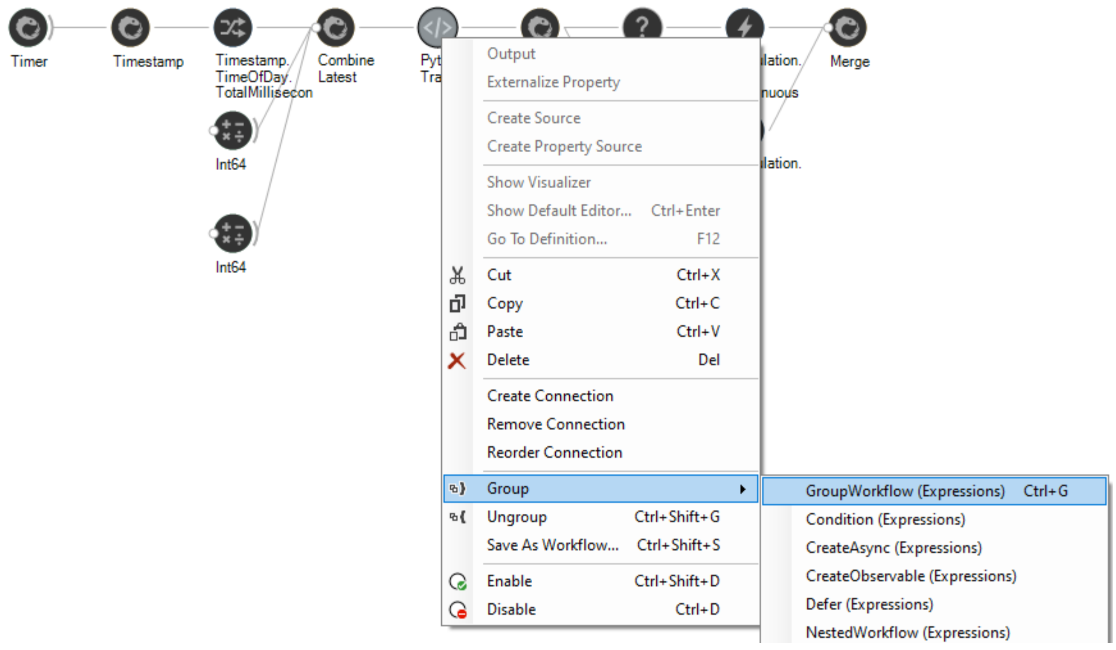
Now that we have the boolean values separated into two data streams, connect “Stimulation” nodes to each of the “Condition” nodes. Configure the “Stimulation” nodes such that the one receiving True values outputs a “Start Stimulation” command, and the other outputs the “Stop Stimulation” command.



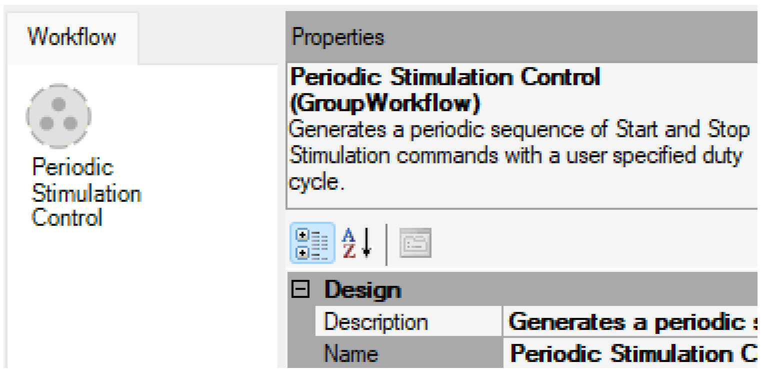
We now have to convert these two data streams outputting stimulation commands into a single data stream that outputs the most recent stimulation command. To do this, connect both of the “Stimulation” nodes to a single “Merge” node. This “Merge” node will output the most recent command produced by the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



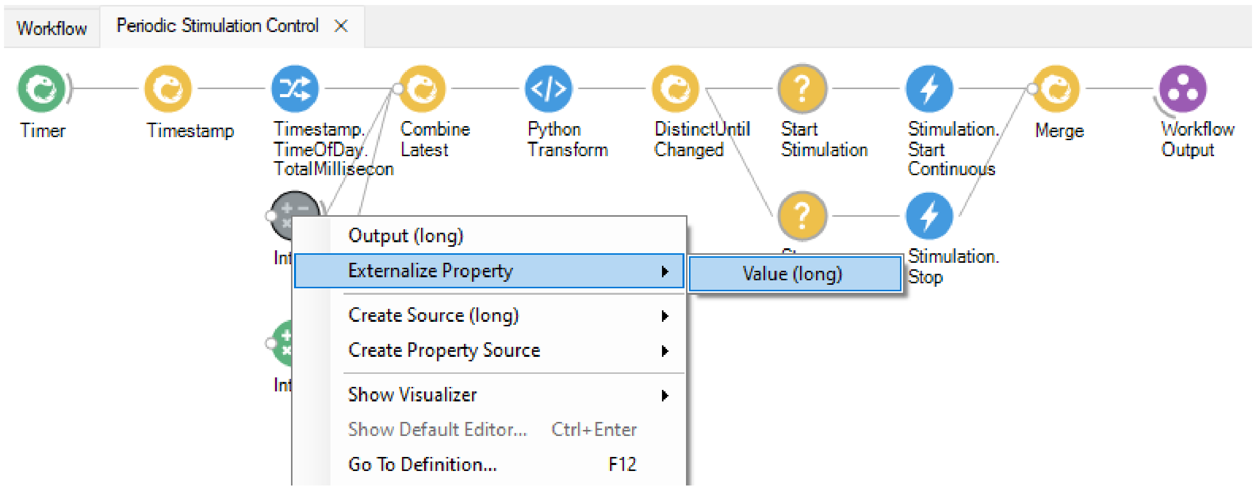
The properties panel of the “Group Workflow” node allows you to name it and provide a description of the encapsulated process.



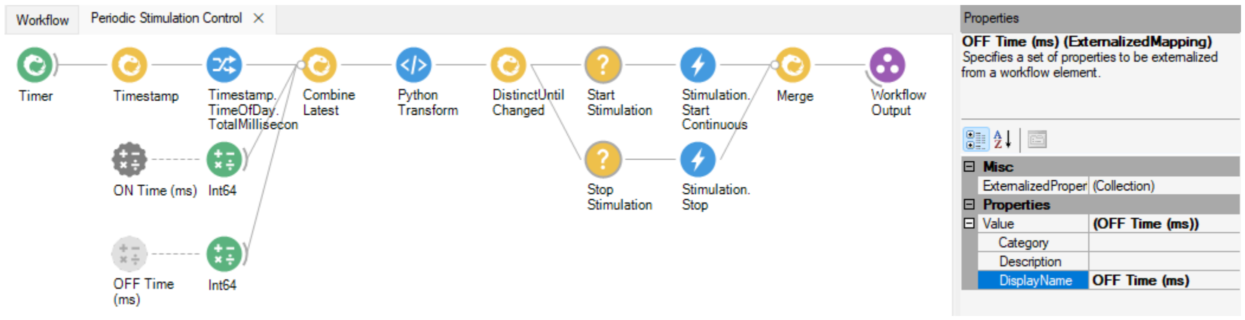
Periodic Stimulation Control (GroupWorkflow)
Generates a periodic sequence of Start and Stop Stimulation commands with a user specified duty cycle.

Design	
Description	Generates a periodic s
Name	Periodic Stimulation C

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”.



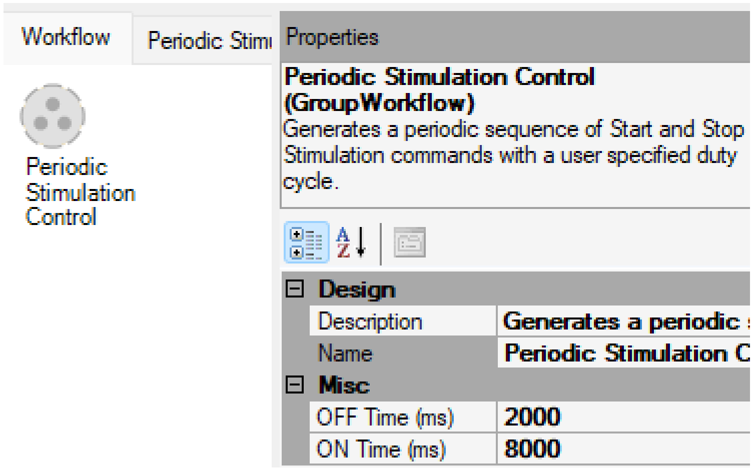
This will add “Value (ExternalizedMapping)” nodes to the inputs of the “Int64” nodes. Both of these “Value” nodes need to have unique “DisplayNames”. To name them, select the node, then expand the “Value” section of the properties panel and set the “Display Name”.



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

Misc	
ExternalizedProper	(Collection)
Properties	
Value	**(OFF Time (ms))**
Category	
Description	
DisplayName	**OFF Time (ms)**

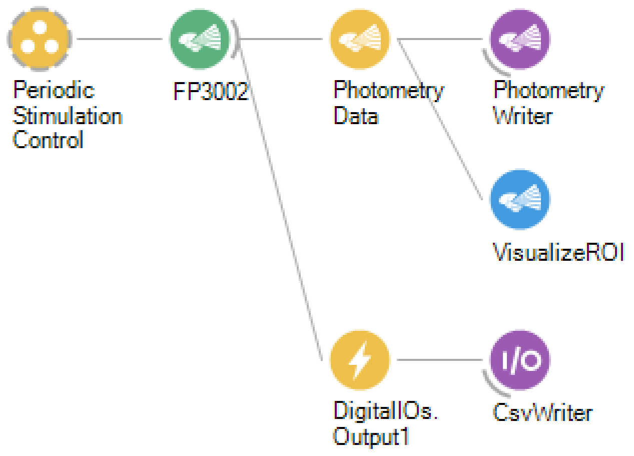
With these externalized properties added, the ON and OFF times can be specified from the properties panel of the grouped workflow.



Periodic Stimulation Control (GroupWorkflow)
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.



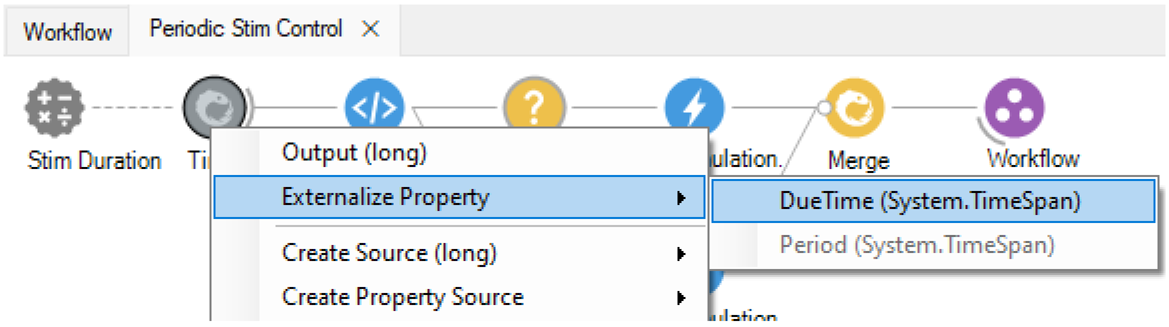
NEUROPHOTOMETRICS
The fiber photometry company

Periodic Control, Delayed Start

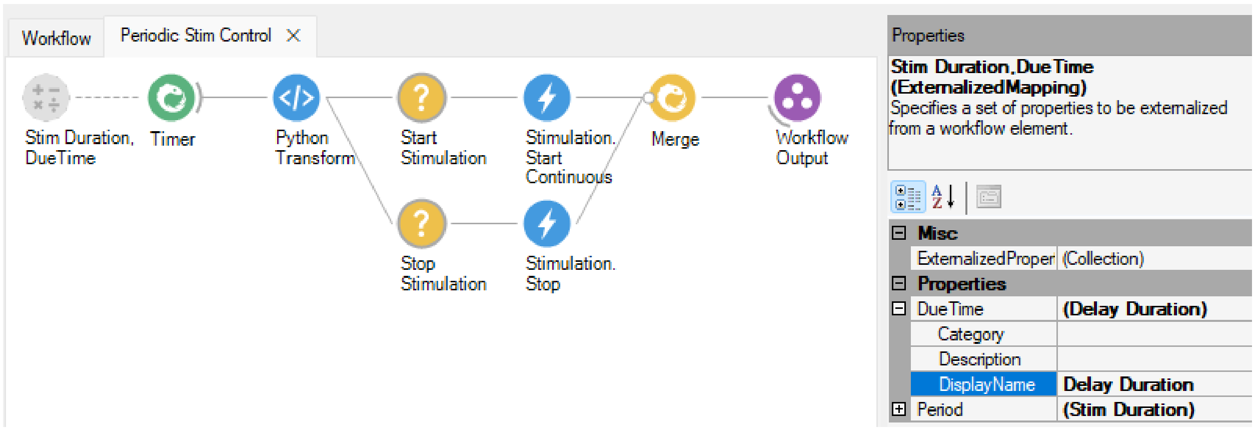
The previous workflows discussing methods for periodic control over stimulation can be expanded to include a delayed start to periodic control. This technique allows for the user to manually or automatically trigger periodic stimulation after a duration of time has passed since the Bonsai workflow has 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.



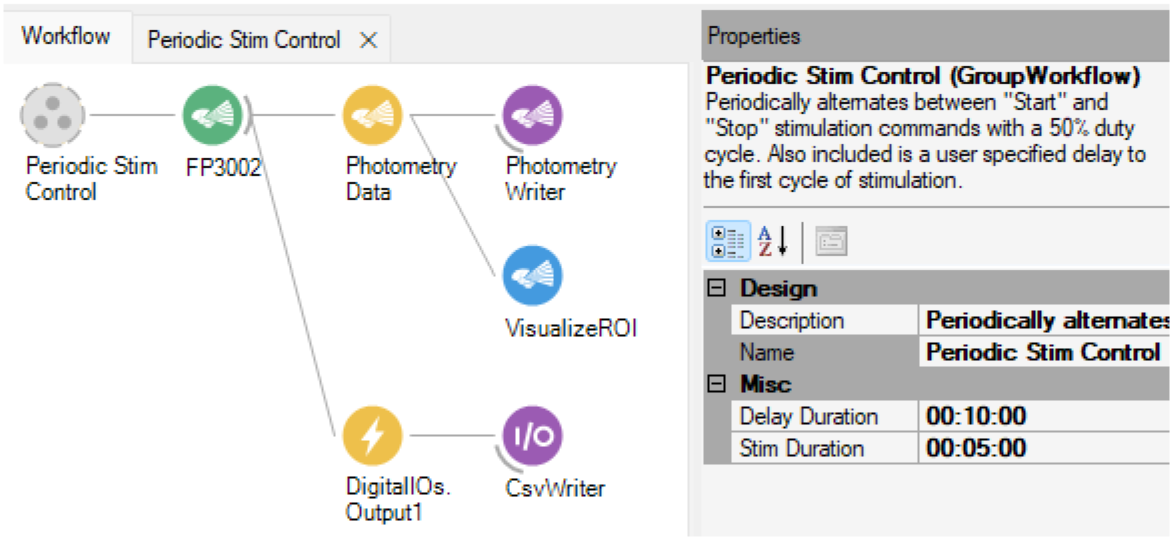
Set the “Display Name” of the externalized “Due Time” property to something unique and informative.



Stim Duration,DueTime (ExternalizedMapping)
Specifies a set of properties to be externalized from a workflow element.

Misc	
ExternalizedProper	(Collection)
Properties	
DueTime	**(Delay Duration)**
Category	
Description	
DisplayName	**Delay Duration**
Period	**(Stim Duration)**

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.

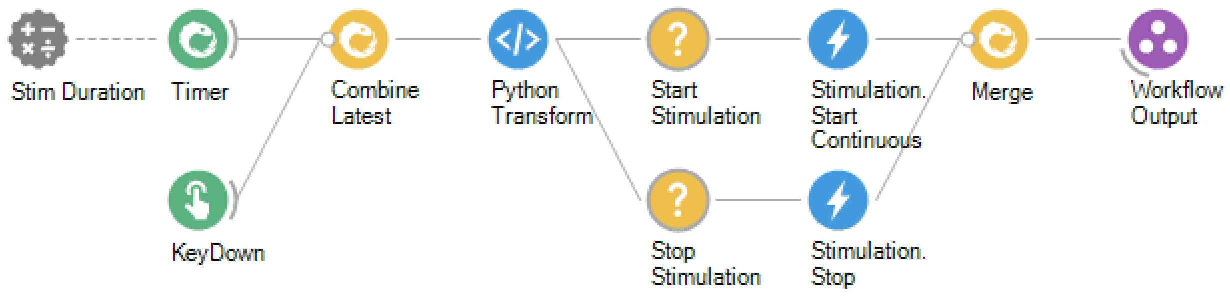


Periodic Stim Control (GroupWorkflow)
Periodically alternates between "Start" and "Stop" stimulation commands with a 50% duty cycle. Also included is a user specified delay to the first cycle of stimulation.

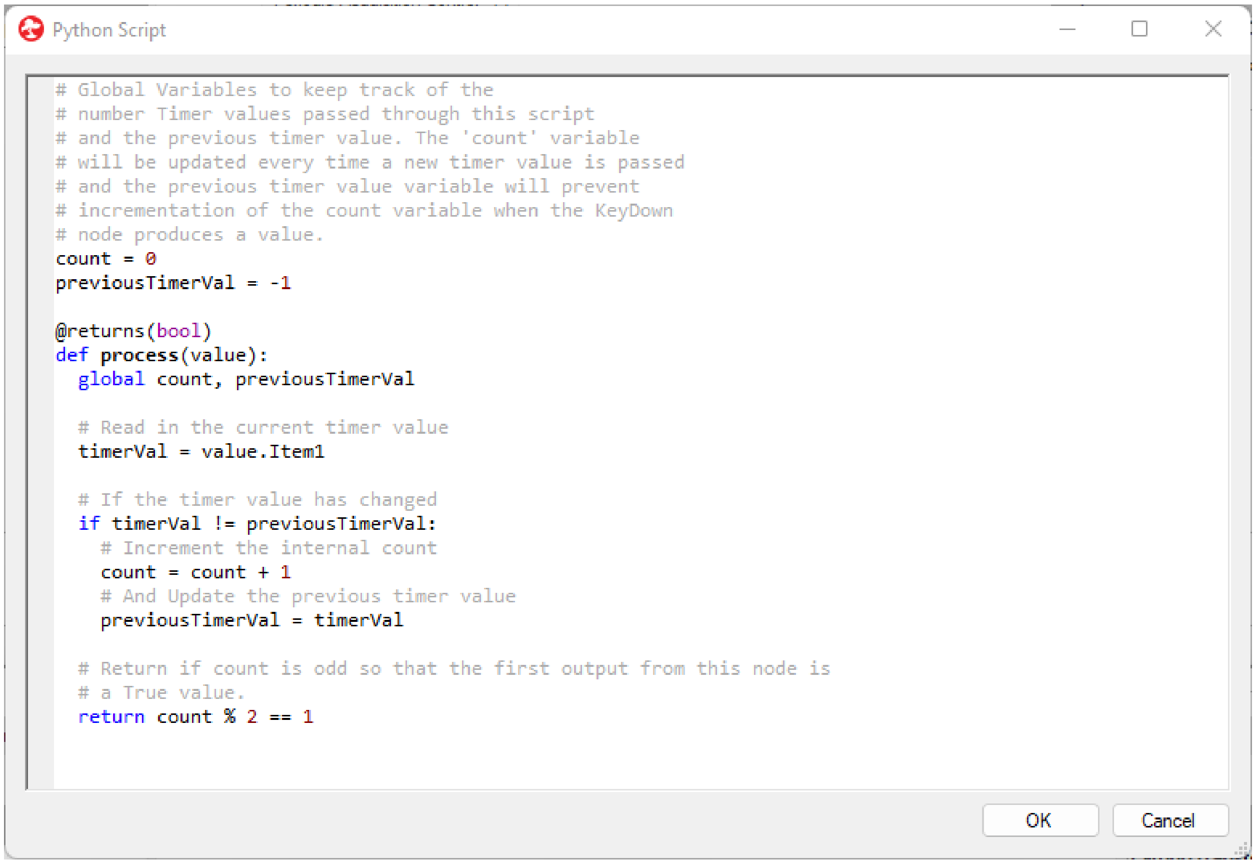
Design	
Description	**Periodically alternates**
Name	**Periodic Stim Control**
Misc	
Delay Duration	**00:10:00**
Stim Duration	**00:05:00**

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.



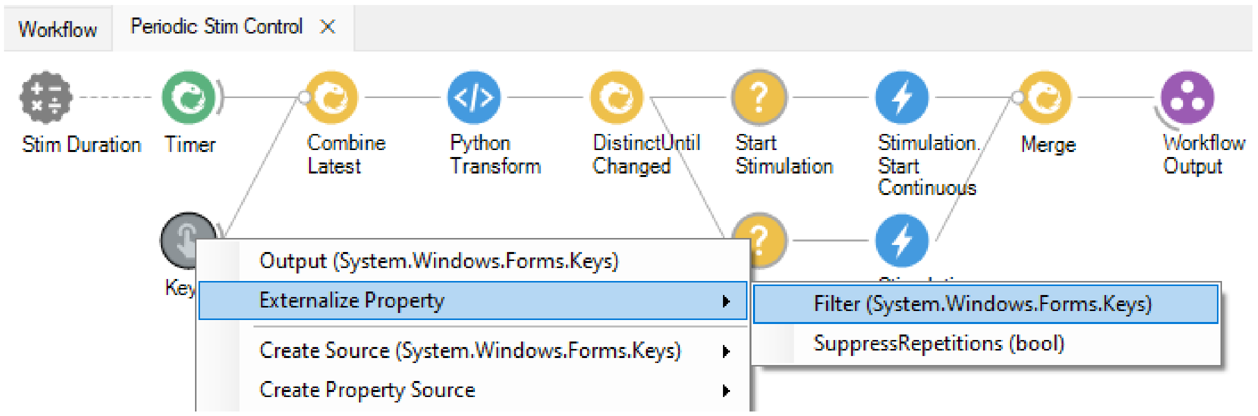
Now, some minor changes to the “Python Transform” node need to be made. Currently, the python script assumes that the input is an integer value, but now its input is a Tuple. Also, the “Python Transform” outputs whether or not the value from the “Timer” node is even or odd. However, with the delayed 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.



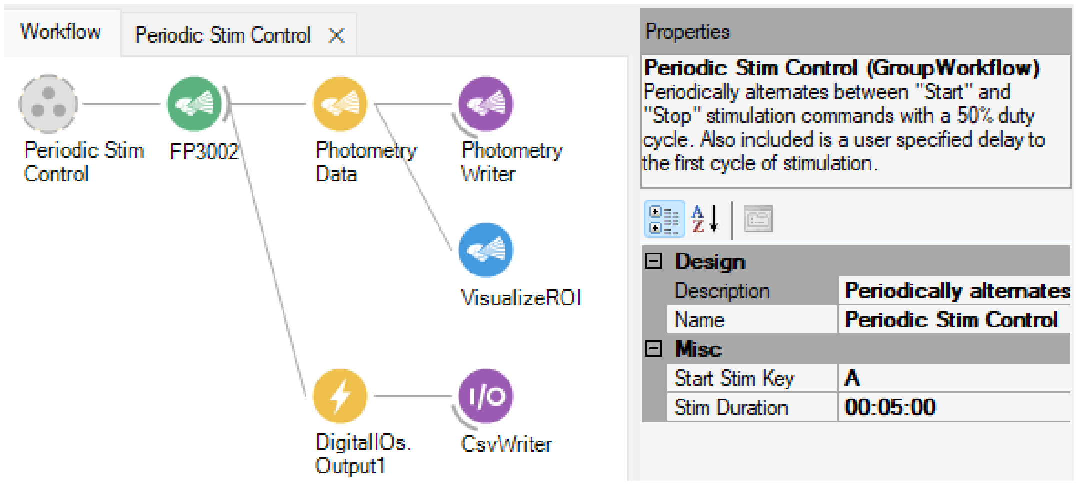
```
# Global Variables to keep track of the
# number Timer values passed through this script
# and the previous timer value. The 'count' variable
# will be updated every time a new timer value is passed
# and the previous timer value variable will prevent
# incrementation of the count variable when the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

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.

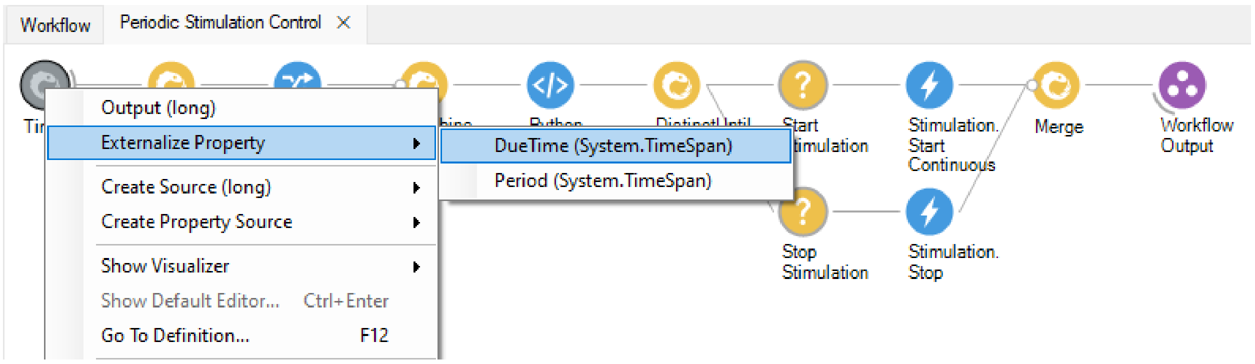


Periodic Stim Control (GroupWorkflow)
Periodically alternates between "Start" and "Stop" stimulation commands with a 50% duty cycle. Also included is a user specified delay to the first cycle of stimulation.

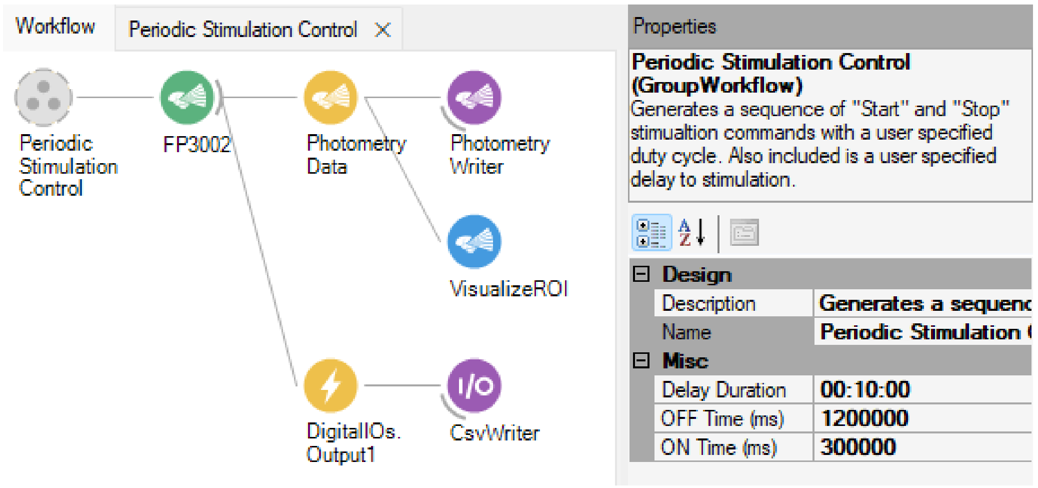
Design	
Description	Periodically alternates
Name	Periodic Stim Control
Misc	
Start Stim Key	A
Stim Duration	00:05:00

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.

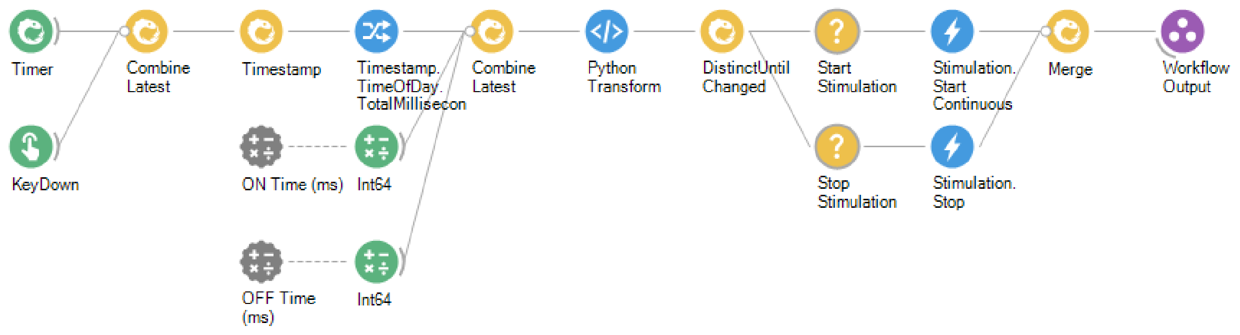


Periodic Stimulation Control (GroupWorkflow)
Generates a sequence of "Start" and "Stop" stimualtion commands with a user specified duty cycle. Also included is a user specified delay to stimulation.

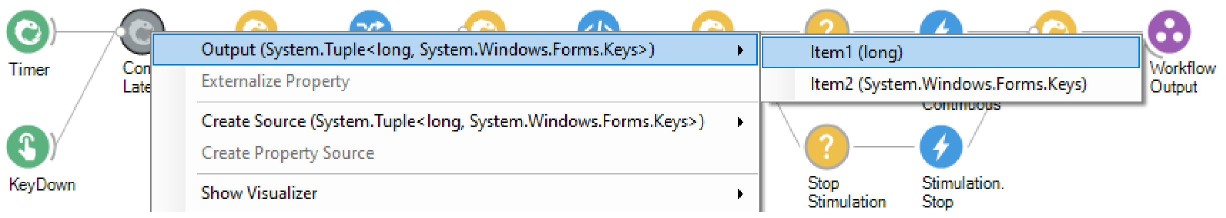
Design	
Description	Generates a sequenc
Name	Periodic Stimulation (
Misc	
Delay Duration	00:10:00
OFF Time (ms)	1200000
ON Time (ms)	300000

Periodic Control, Variable Duty Cycle, Manual Delayed Start

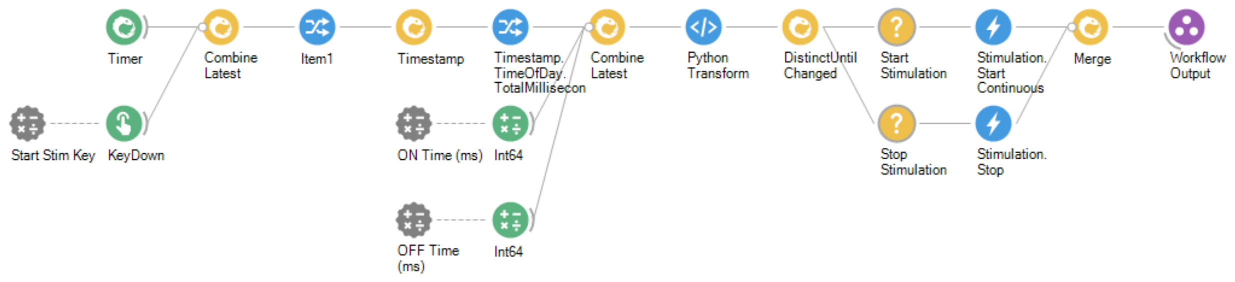
To add a manual delay to this workflow, instead of specifying the “Due Time” property, we will use a “Combine Latest” node to combine the “Timer” node and a “KeyDown” node. This way the “PythonTransform” node will not receive its first value until the specified key is pressed.



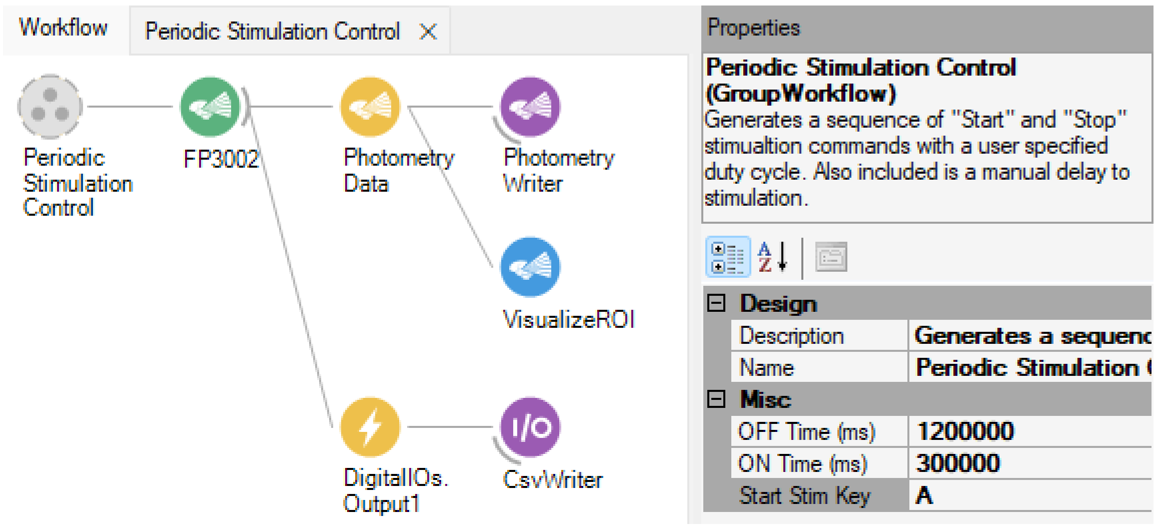
Now, we have two “Combine Latest” nodes in the same data stream leading up to the “Python Transform” node. This adds a little complexity to the data types going into python script. We can fix this data type mix-match either within the python script or within the workflow. In this case, the easiest way 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”.


Timer
KeyDown
Output (System.Tuple<long, System.Windows.Forms.Keys>)
Externalize Property
Create Source (System.Tuple<long, System.Windows.Forms.Keys>)
Create Property Source
Show Visualizer
Item1 (long)
Item2 (System.Windows.Forms.Keys)
Stop Stimulation
Stimulation.Stop
Workflow Output

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.



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

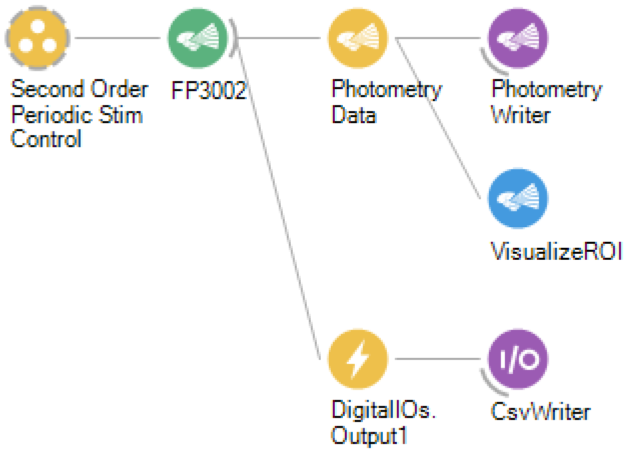
Design	
Description	**Generates a sequenc**
Name	**Periodic Stimulation**
Misc	
OFF Time (ms)	**1200000**
ON Time (ms)	**300000**
Start Stim Key	**A**

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.

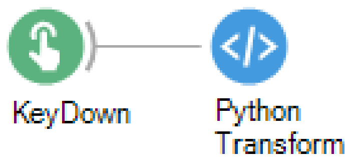
Similar to the previous data acquisition workflows, these second order periodic control workflows can be separated into three sections: the software trigger, the generation of stimulation commands, and the standard photometry section with the addition of the "Digital IOs" node to record the laser state. 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".



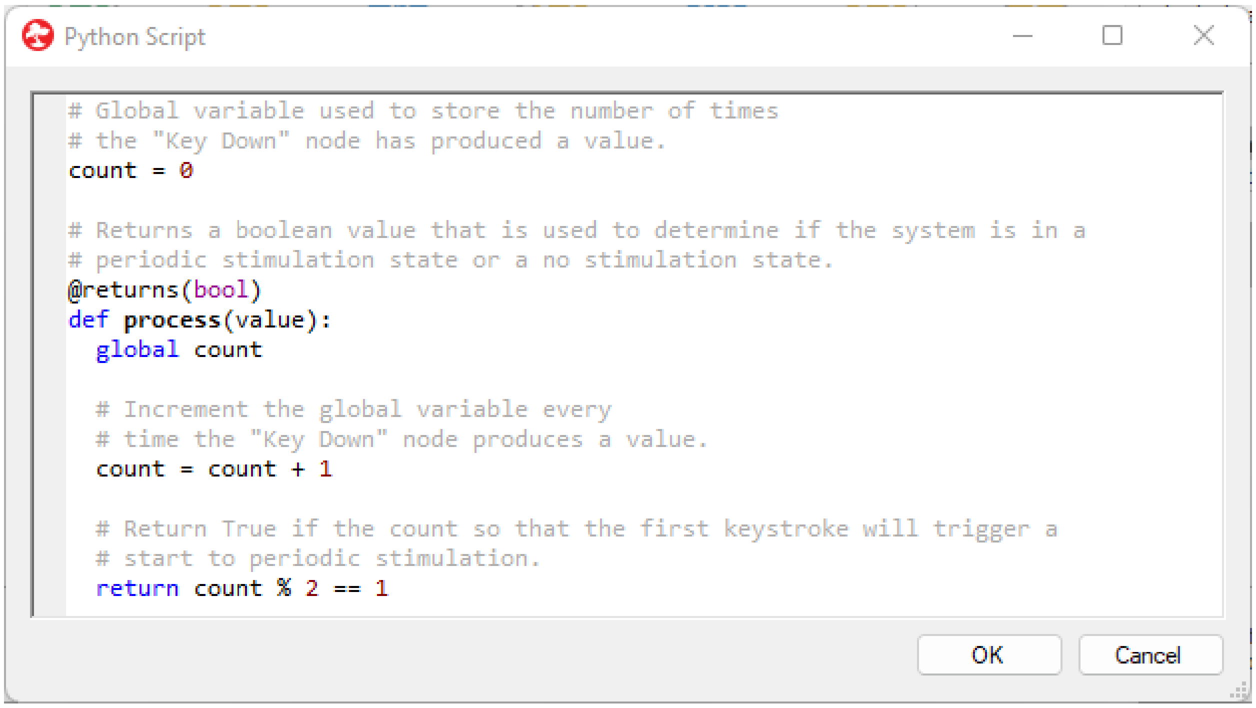
To change this first order periodic control workflow to a second order periodic control workflow, all we need to do is reconfigure the software trigger logic.

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.



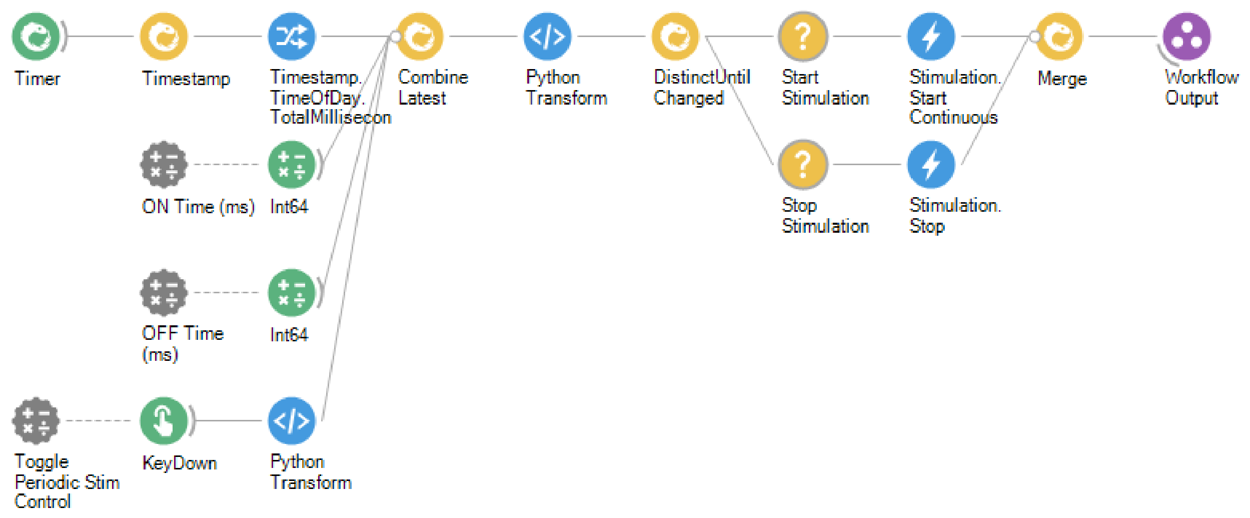
The python script in this new "Python Transform" node will contain an internal counter and output whether the key was pressed an Even or Odd number of times. We will use the output of this "Python Transform" node to dictate whether the system is in a periodic stimulation control state or a no stimulation 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.



```
# 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
```

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.



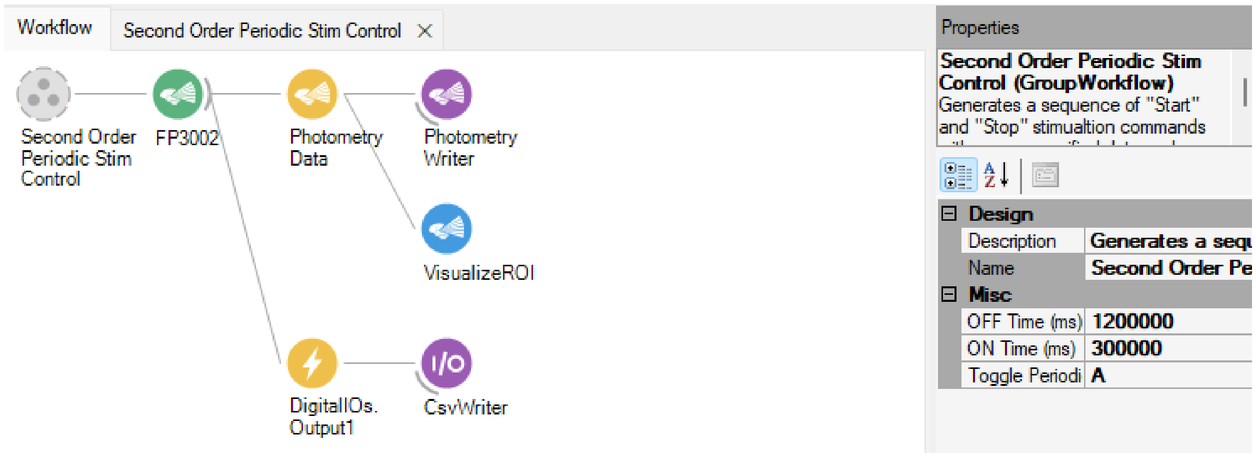
With some additional python logic in the “Python Transform” node immediately after the “Combine Latest” node, we will finish constructing our software trigger. The boolean value dictating whether the system should be in a periodic stimulation state or a no stimulation state will be contained in the fourth 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.

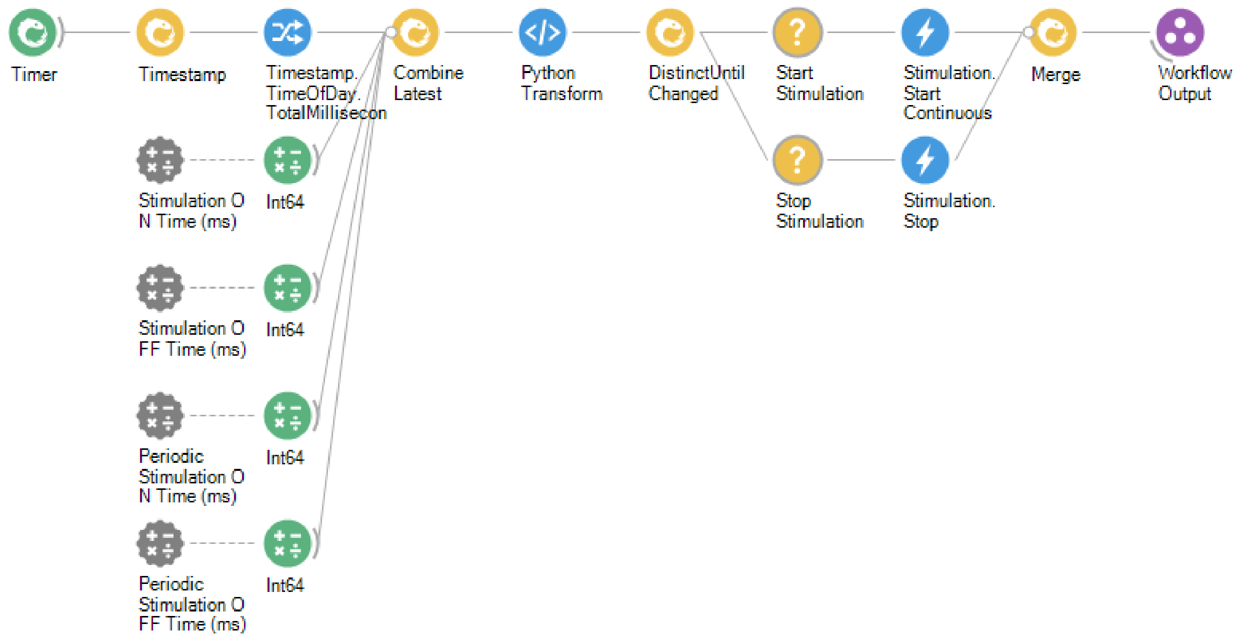


Second Order Periodic Stim Control (GroupWorkflow)
Generates a sequence of "Start" and "Stop" stimualtion commands

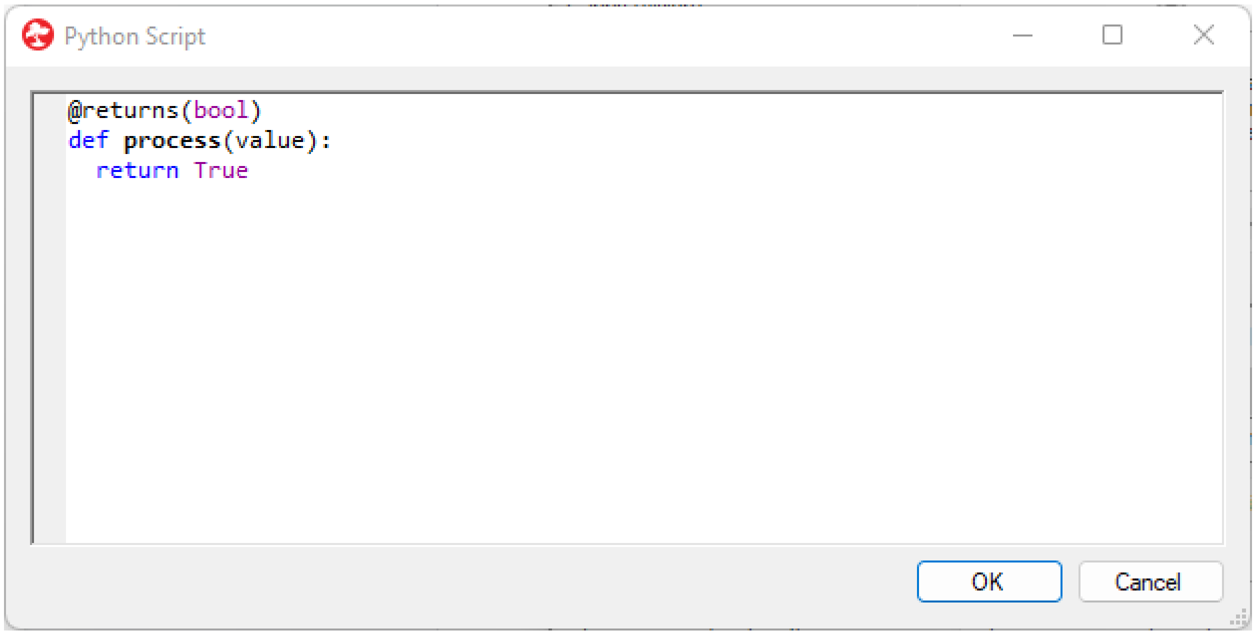
Design	
Description	Generates a sequ
Name	Second Order Pe
Misc	
OFF Time (ms)	1200000
ON Time (ms)	300000
Toggle Periodi	A

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.



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

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
```

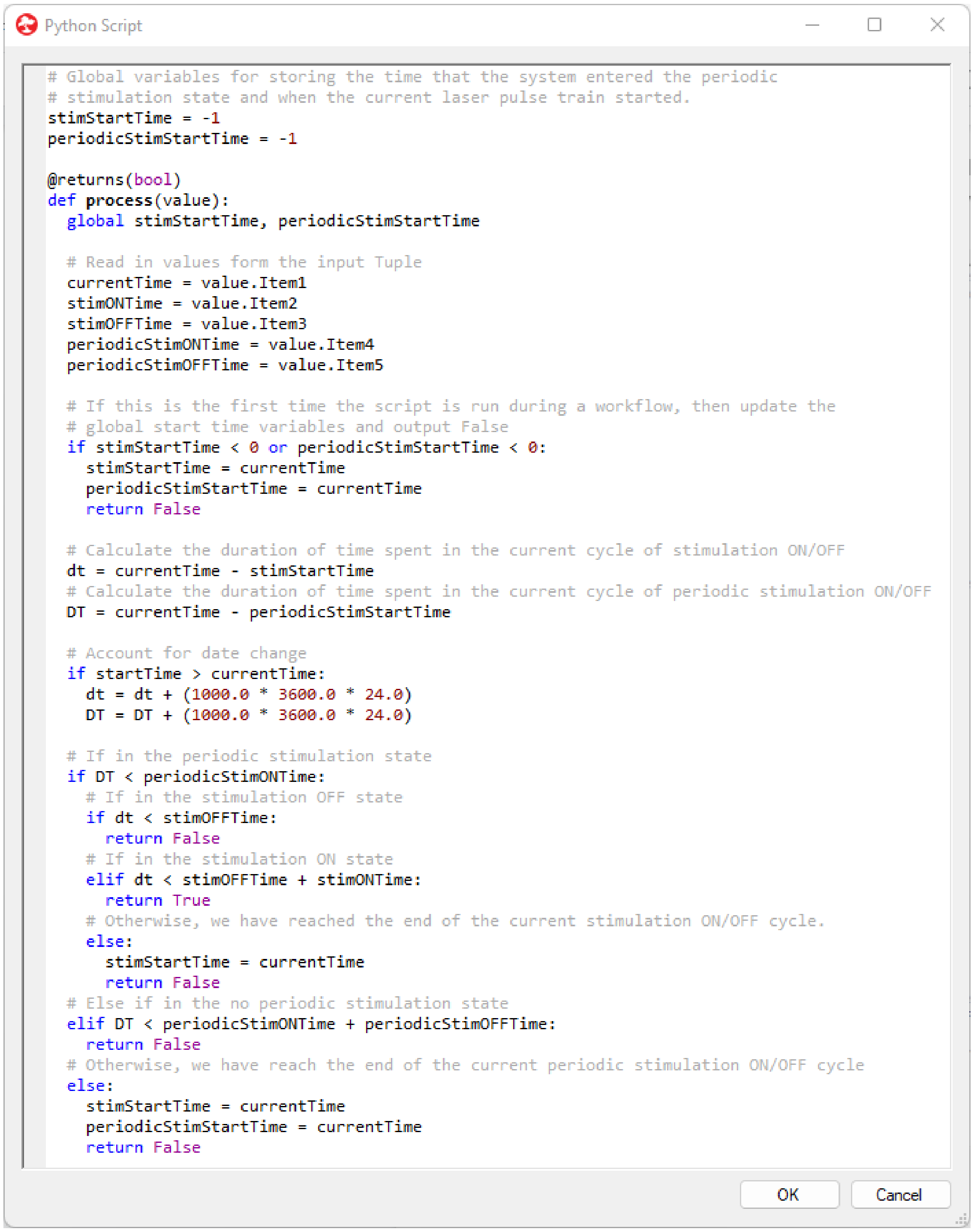
In the if portion of the if, elif, else statement, we need to include logic for alternating between stimulation ON/OFF states. This logic will be quite similar to our logic for alternating between periodic stimulation ON/OFF states. So we can add another if, elif, else statement here to determine if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

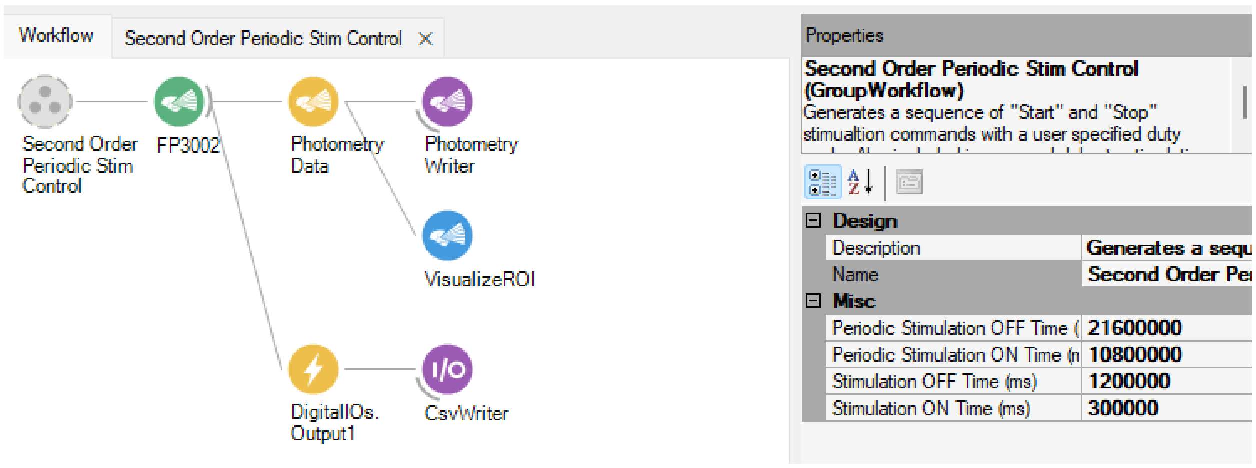
```
# If in the periodic stimulation state
if DT < periodicStimONTime:
  # If in the stimulation OFF state
  if dt < stimOFFTime:
    return False
  # If in the stimulation ON state
  elif dt < stimOFFTime + stimONTime:
    return True
  # Otherwise, we have reached the end of the current stimulation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



```
# 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
```

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.



Second Order Periodic Stim Control (GroupWorkflow)
Generates a sequence of "Start" and "Stop" stimualtion commands with a user specified duty

Design	
Description	Generates a sequ
Name	Second Order Pe
Misc	
Periodic Stimulation OFF Time (	21600000
Periodic Stimulation ON Time (n	10800000
Stimulation OFF Time (ms)	1200000
Stimulation ON Time (ms)	300000

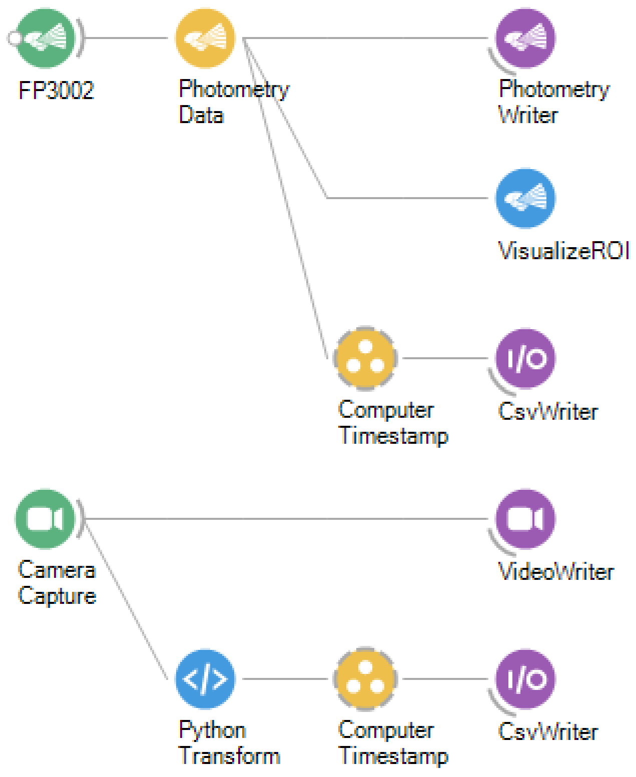
Chapter 6: Machine Vision

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.

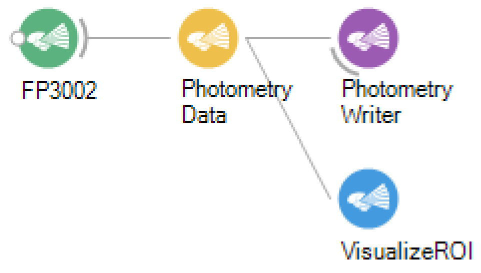
Once we establish the ability to conduct animal tracking in multi-chamber enclosures, we transition to the concept of “Closed-Loop” experiments. We use this term to describe fiber photometry experiments that are influenced by the actions of animal being observed. In this chapter, we describe how to use 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

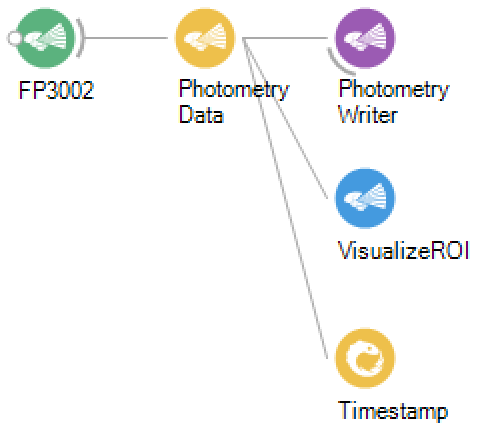
For experiments that require post-hoc behavioral analysis, it is possible to construct Bonsai workflows with external behavioral cameras synchronized with the photometry data. This particular workflow will save four files: the standard photometry dataset (.*csv*), the photometry frame numbers timestamped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



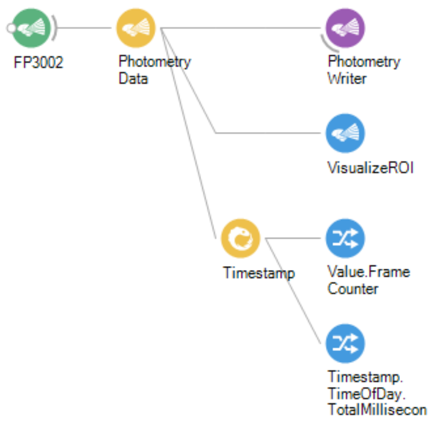
This workflow consists of two parallel data streams. One deals with processing and recording the photometry data, while the other processes and records the information from the external behavioral camera. In order to construct the photometry data stream, begin with the "Standard Photometry" workflow.



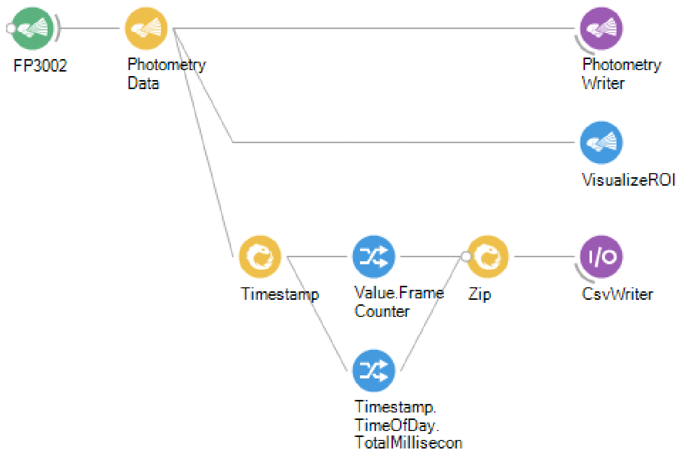
This will record the photometry data, timestamped using the internal clock of the FP3002 system. The data set produced by the "Standard Photometry" workflow needs to be synchronized with the behavioral camera's datasets. To prepare for synchronizing the two parallel data streams, we need to timestamp every photometry data frame using a clock that both data streams have access to. In this case, we will use the computer's clock to timestamp every photometry frame and behavioral frame. Insert a "Timestamp" node after the "Photometry Data" node, in parallel with the "Photometry Writer" and "Visualize ROI" nodes.



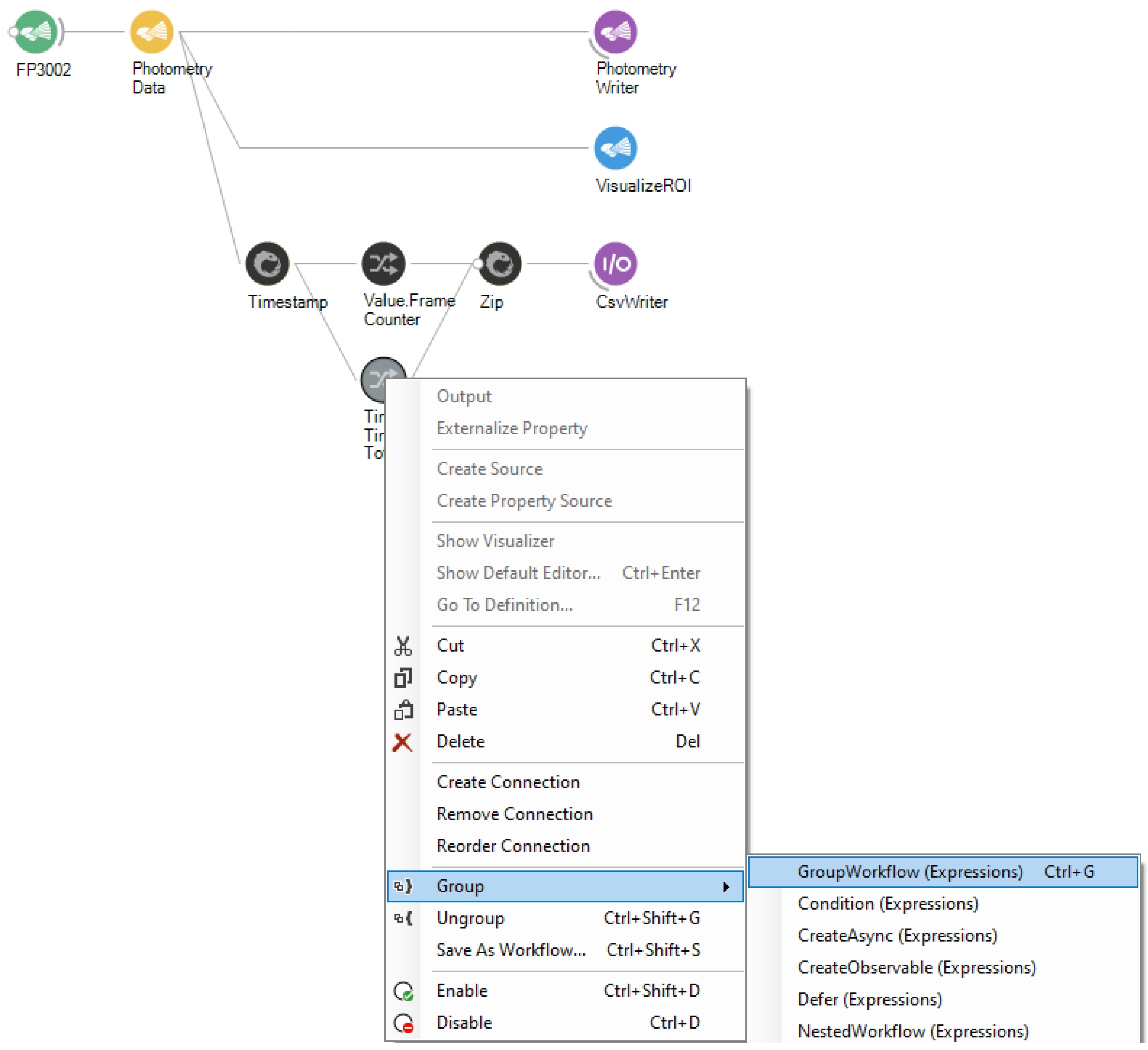
This “Timestamp” node will allow the computer to timestamp every frame count element of every photometry data frame. We need to configure the output of the “Timestamp” node to specifically output only the frame count and the computer’s timestamp. Do this by right clicking the “Timestamp” node and selecting “Output → Value → FrameCounter”. Then right click the “Timestamp” node and select “Output → Timestamp → TimeOfDay → TotalMilliseconds”.



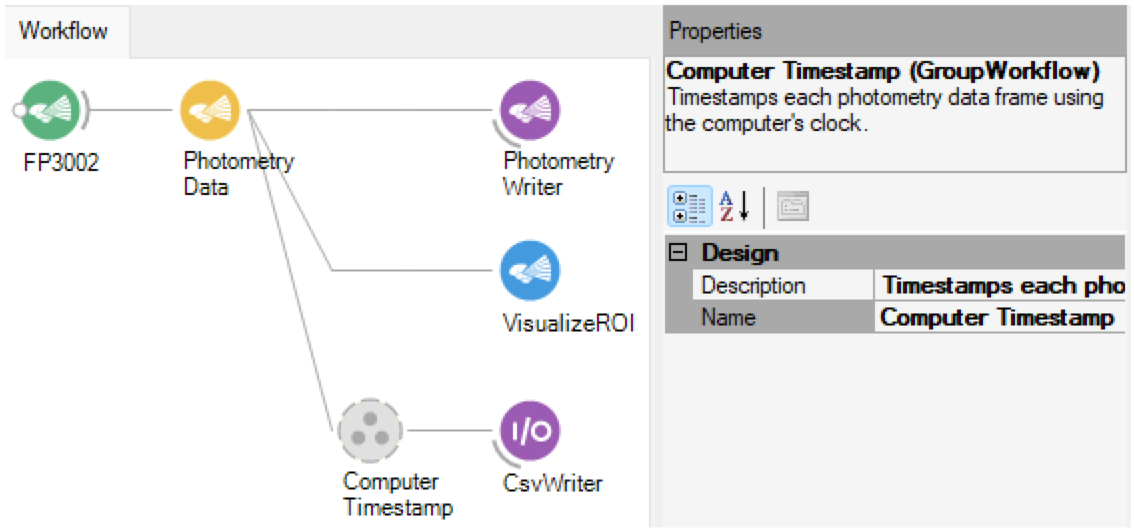
This will split the output of the “Timestamp” node into two data streams, one containing the current frame count of the photometry data, the other containing the computer’s timestamp of the current photometry data frame. Since we want to combine these two data streams back into one data stream such that they produce values at the same time, we can use the “Zip” node to combine them back together. This “Zip” node will combine the frame count and the computer’s timestamp into a Tuple that can be easily written to a *.csv* file using a “Csv Writer” node.



This expansion of the “Standard Photometry” workflow now saves two .csv files. One will contain the standard photometry data set, the other will contain the frame number timestamped using the computer’s clock. This makes the photometry data readily alignable to parallel data streams by using the same computer timestamping method for all other parallel data streams. Some minor cleanup can be done to make the workflow more readable. Simply select all of computer timestamp nodes, from the “Timestamp” to the “Zip” node, right click and select “Group → GroupWorkflow”.



This will hide all of the computer timestamp logic inside of a single grouped workflow that can be named and described from the properties panel.

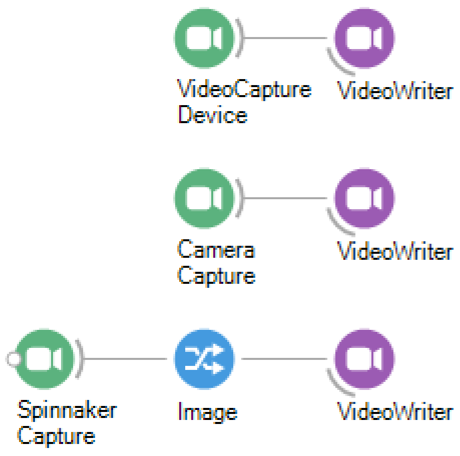


Computer Timestamp (GroupWorkflow)
Timestamps each photometry data frame using the computer's clock.

Description	Timestamps each pho
Name	Computer Timestamp

Now that the photometry data stream is constructed and organized, 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 "IplImage". However, the "SpinnakerDataFrame" consists of an "IplImage" 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 "IplImage" is selected from the "SpinnakerDataFrame".

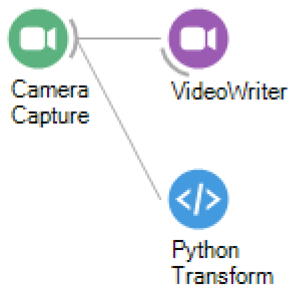
Create an external behavioral camera data stream by connecting the desired source node to a “Video Writer” node.



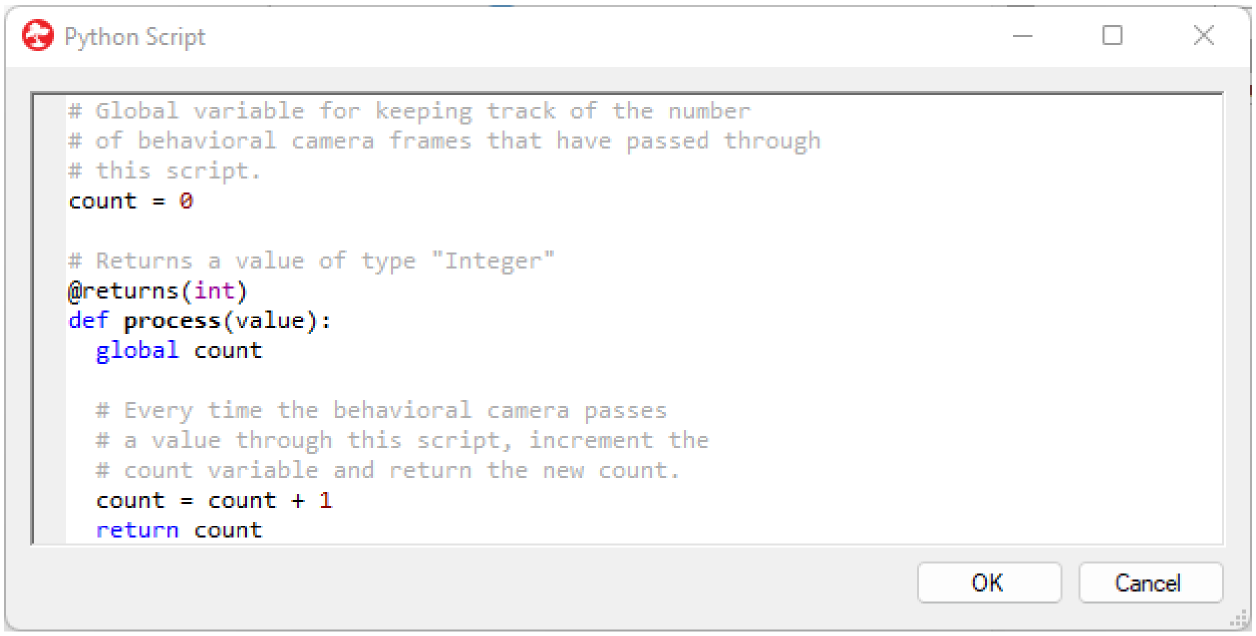
Some configuration is available for the capture nodes and the “Video Writer” node. All three capture nodes have the option to specify the camera’s index. For the “Camera Capture” and “Video Capture Device” nodes, the internal camera on the “FP3002” system will not be recognized, so if only one behavioral 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.

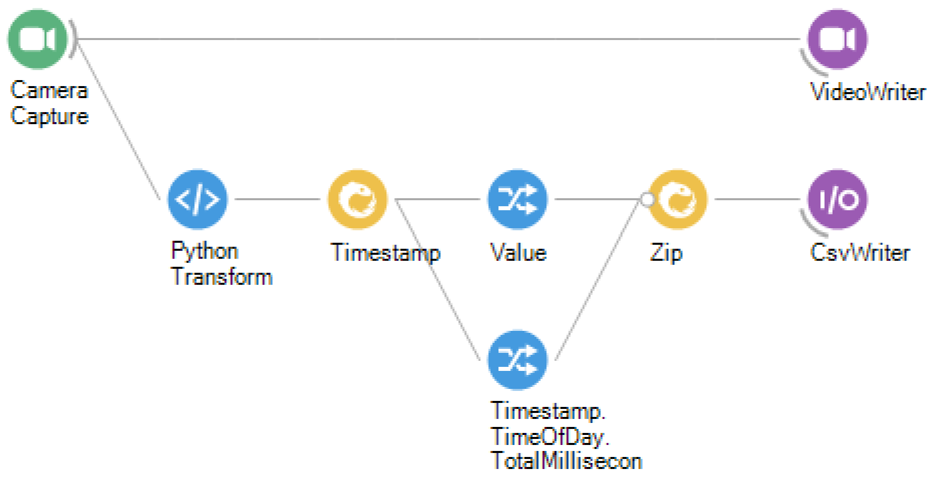


Edit the python script by double clicking the “Python Transform” node. Here we will insert a basic python counter script that will keep track of the number of behavioral camera frames that have passed through the “Python Transform” node.

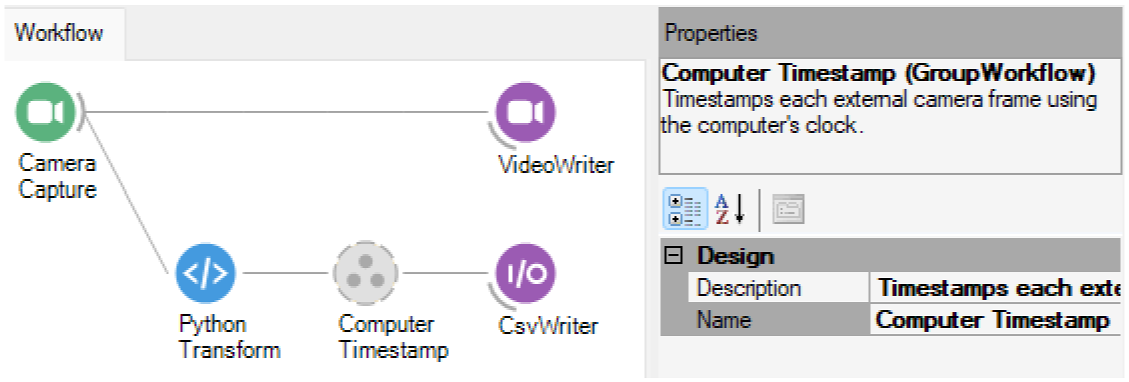


```
# Global variable for keeping track of the number
# of behavioral camera frames that have passed through
# this script.
count = 0

# 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
```

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.



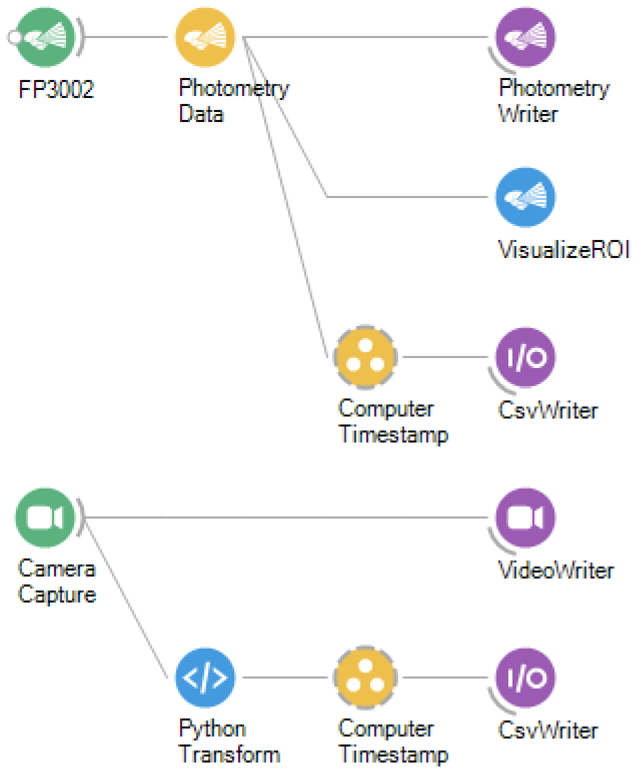
Again, we can group together all of the computer timestamp nodes into a single grouped workflow that can be named and described from the properties panel.



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

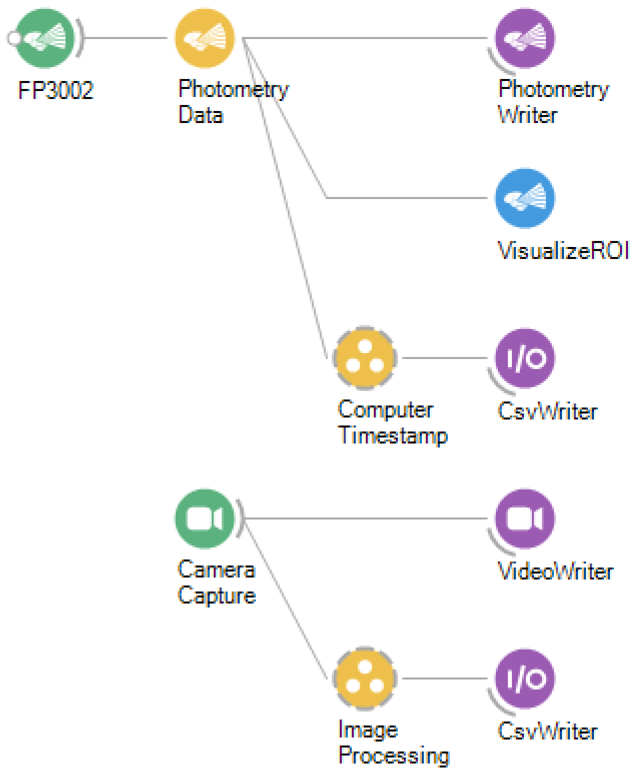
Description	Timestamps each exte
Name	Computer Timestamp

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.

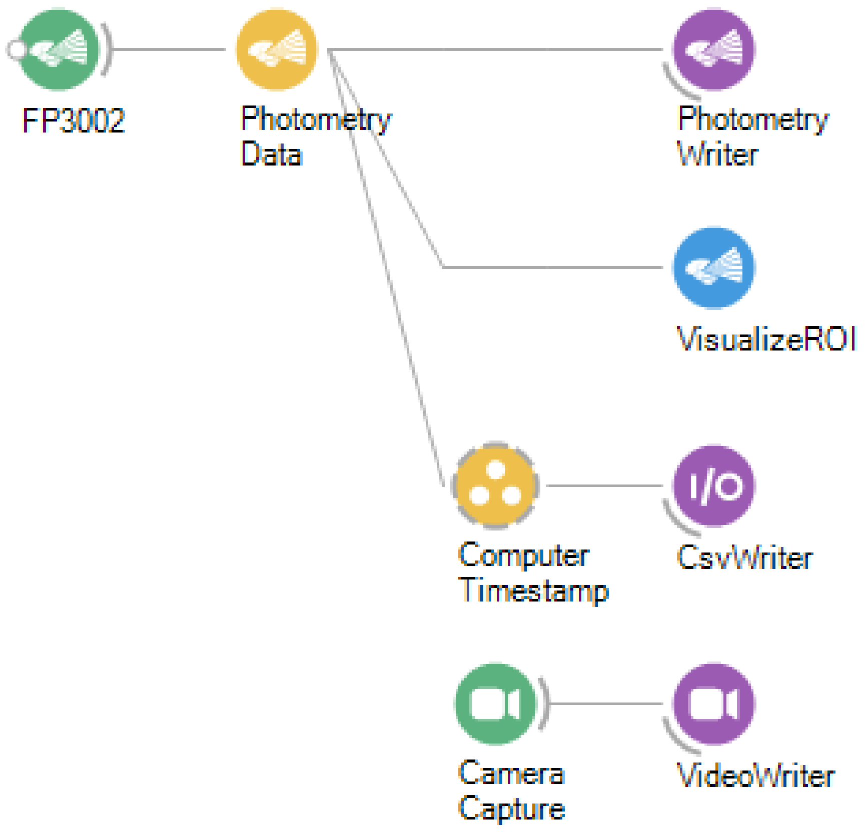


One Chamber Animal Tracking

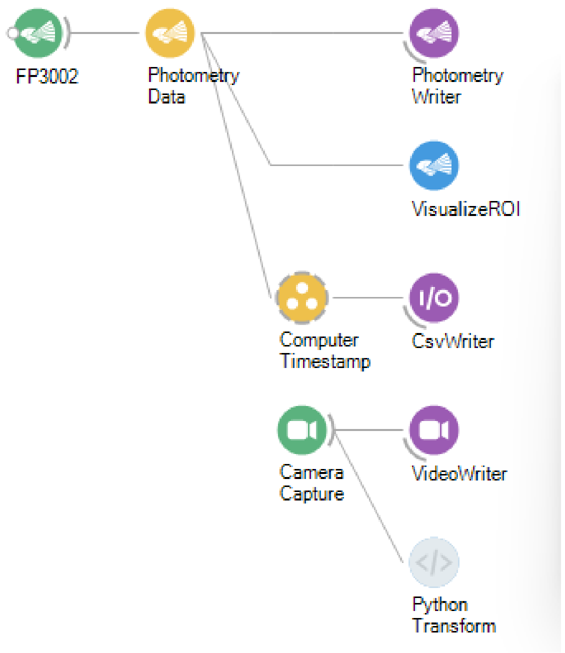
Previously we discussed how to synchronize a behavioral camera in a parallel data stream within Bonsai. This technique can be expanded to provide real-time processing of the behavioral camera's frame. In particular, this section will discuss tracking the position of an animal in a one chamber enclosure.

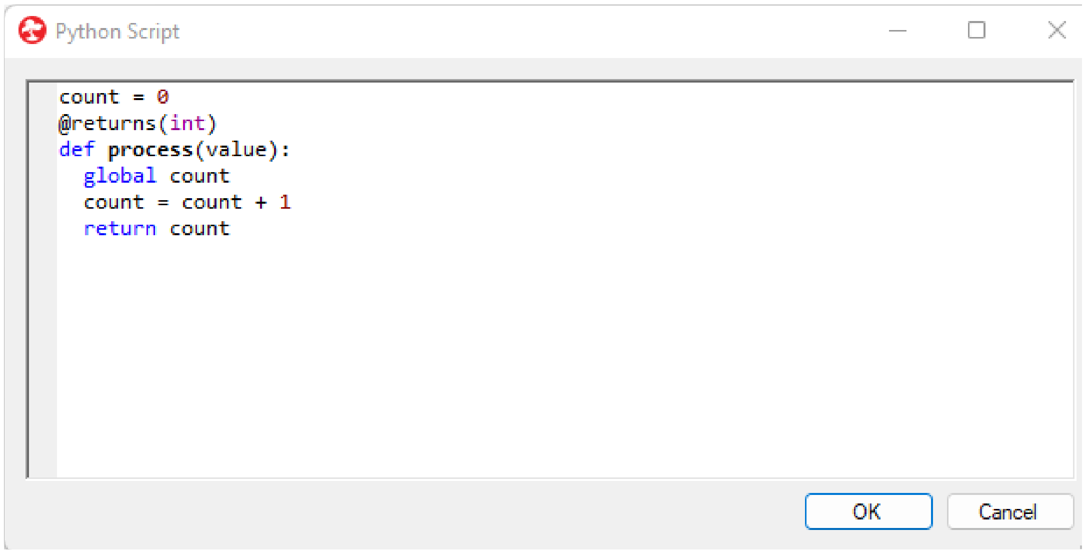


Begin construction of this workflow with the “Machine Vision: Synchronized Behavioral Camera” workflow discussed previously. Remove the “Python Transform” and timestamping logic from the behavioral camera’s data stream, this section of the workflow is where we will implement our image processing logic.



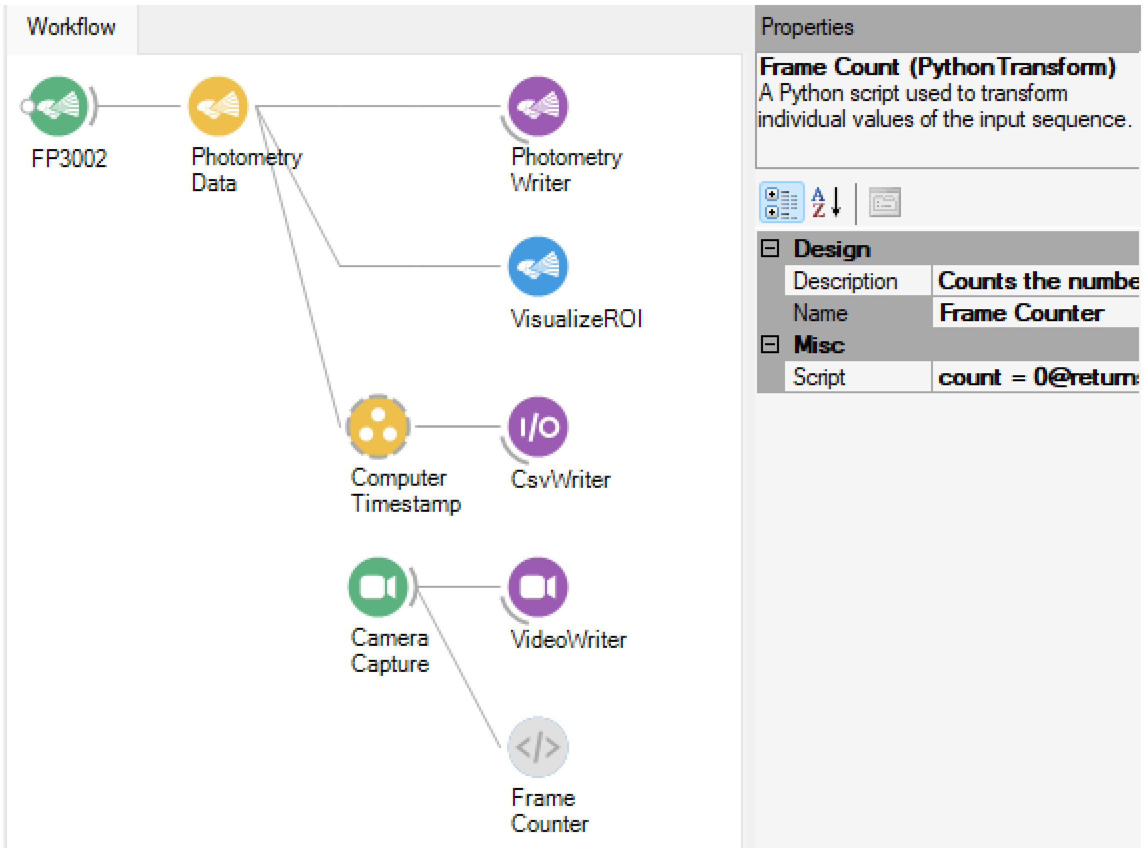
Our image processing logic needs to save four columns of data into a .csv file. We will configure our logic to save the frame number in column one, the computer's timestamp in column two, and the XY position in columns three and four. In parallel to the "Video Writer" node, connect a "Python Transform"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





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

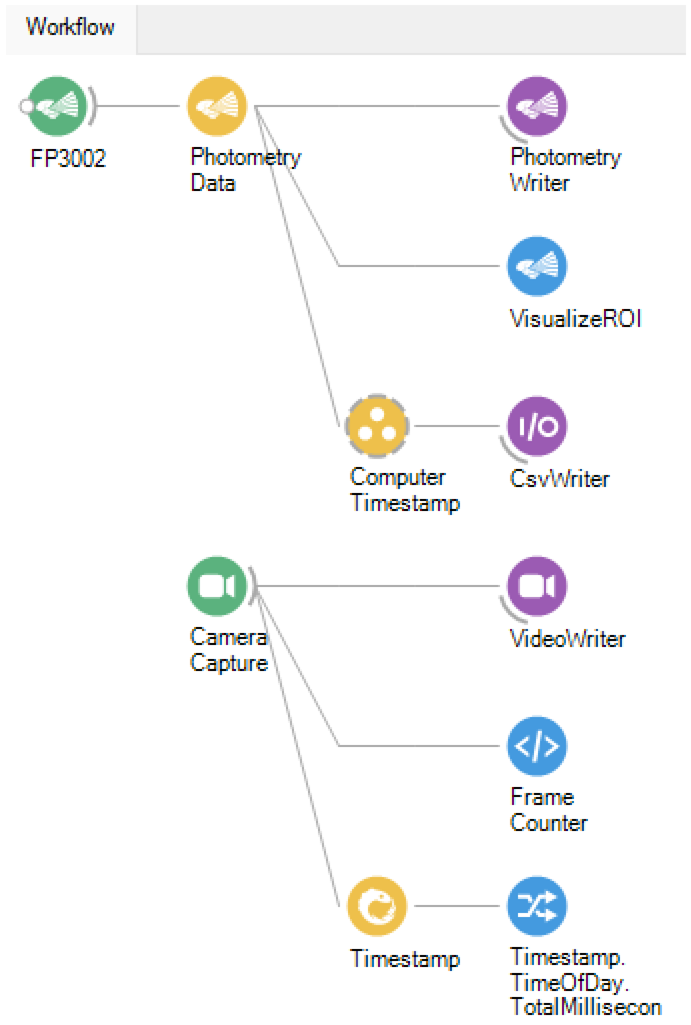
Be sure to name and describe the “Python Transform” node from its properties panel. This becomes important when constructing more complex workflows.



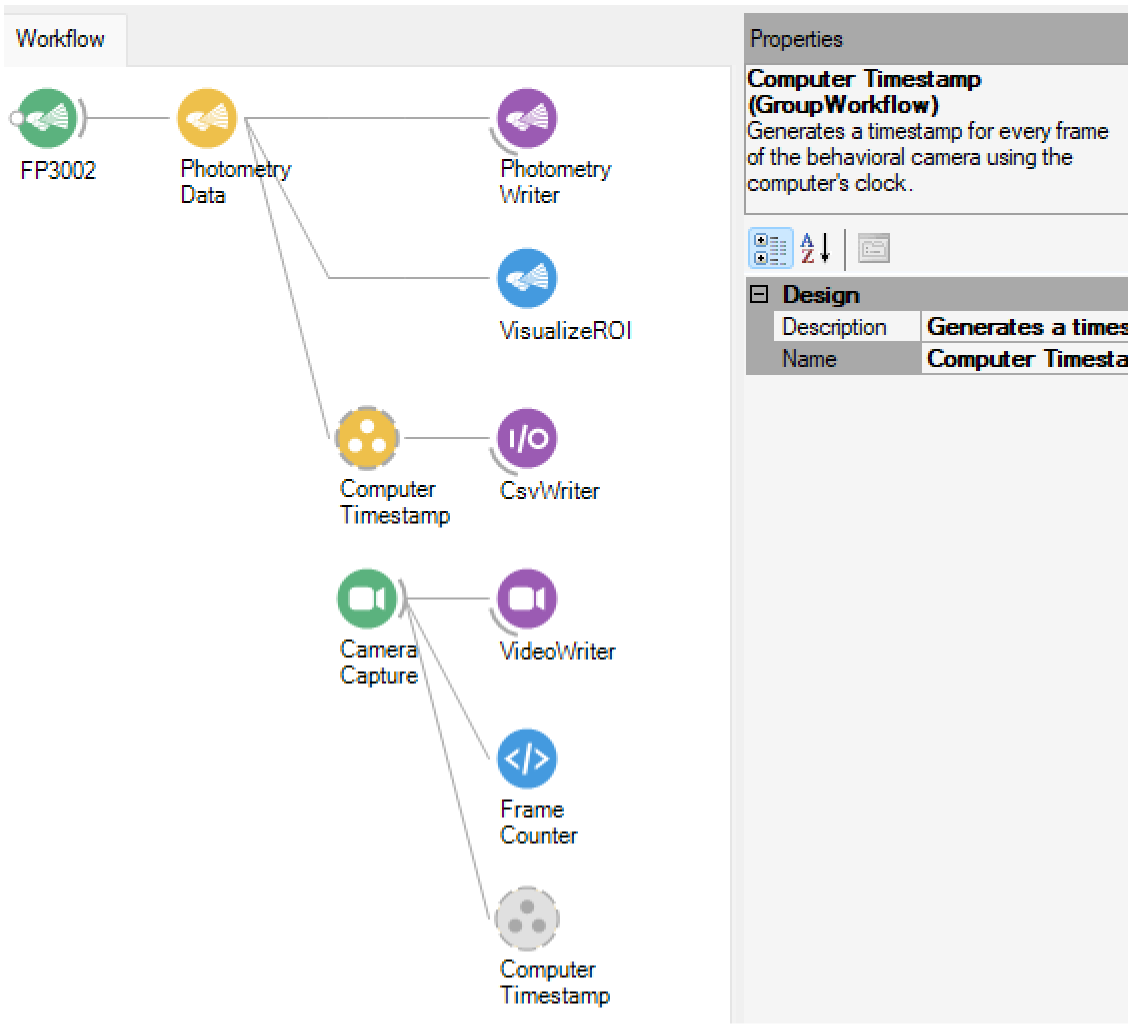
Frame Count (PythonTransform)
A Python script used to transform individual values of the input sequence.

Design	
Description	Counts the numbe
Name	Frame Counter
Misc	
Script	count = 0@return

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”.

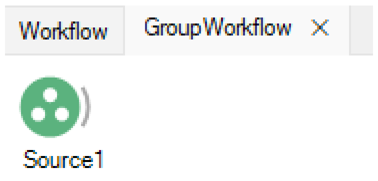


Computer Timestamp (GroupWorkflow)
Generates a timestamp for every frame of the behavioral camera using the computer's clock.

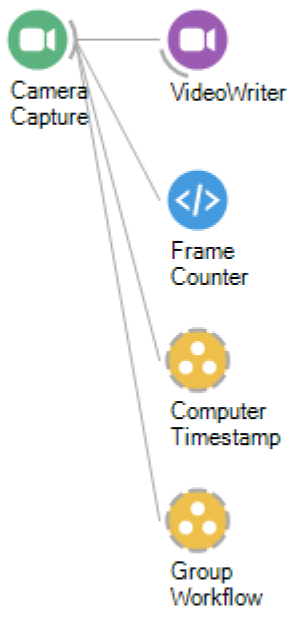
Design

Description	Generates a times
Name	Computer Timesta

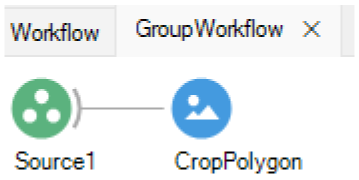
Now we need to create the logic for finding the centroid of an animal in a single chamber enclosure. We will put all of this logic inside a grouped workflow in order to keep everything organized. Begin by adding a “Group Workflow” from the toolbox into the workflow. Open the grouped workflow by double 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.



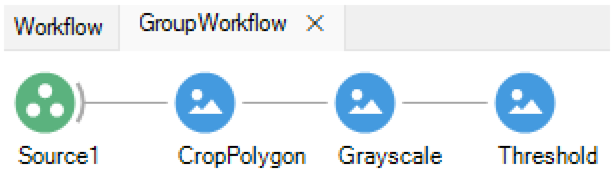
With this source node inside of the grouped workflow, we can now navigate back to the top-level of the workflow and connect the capture node to the grouped workflow.



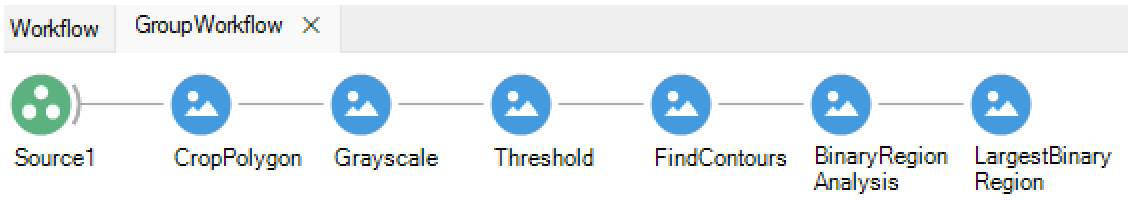
Returning to our grouped workflow, the first set of logic we want to implement in our “Find Centroid” algorithm will work to crop the image coming from the behavioral camera to the bounds of a single chamber enclosure. To do this connect a “Crop Polygon” node to the source node of the grouped workflow. 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.



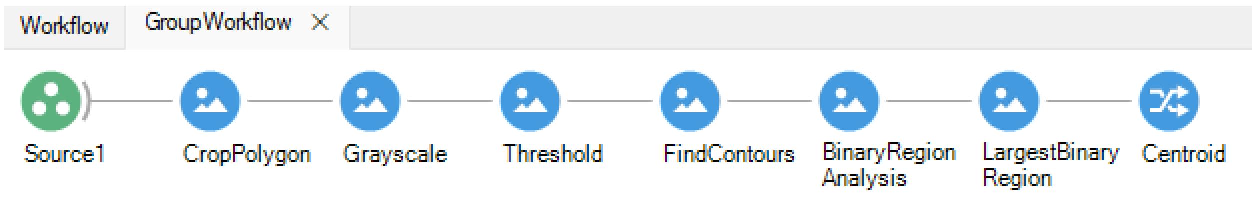
Next we need to apply an image segmentation technique to isolate the animal in the image. One way to do this is to convert the image to grayscale then threshold the grayscale image. Insert a “Grayscale” node followed by a “Threshold” node after the “Crop Polygon” node. The “Grayscale” node will convert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



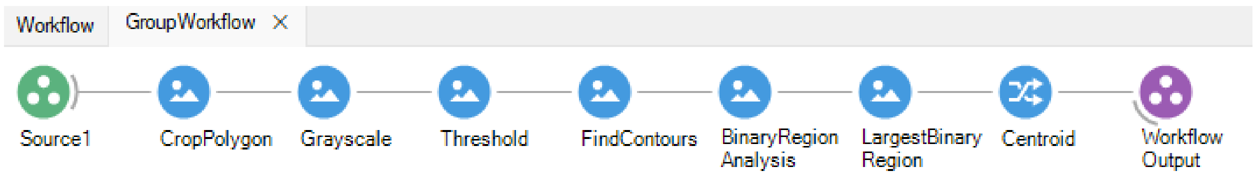
Now that we have applied image segmentation to isolate the animal, we can begin on our binary region analysis. This part of the algorithm begins with finding the contours of the animal, then we extract binary region properties from those contours and isolate the largest binary region (i.e. the animal 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”.



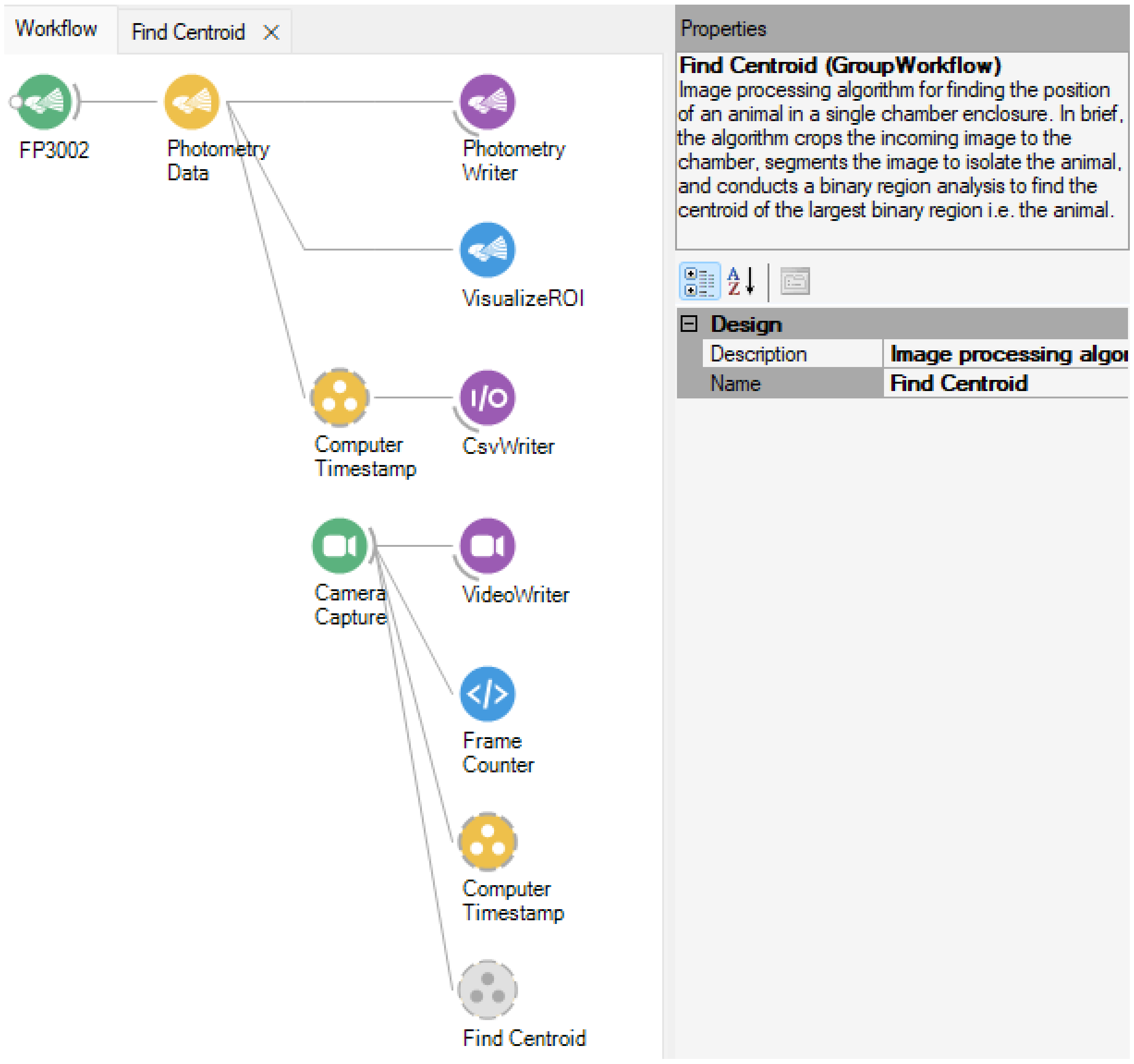
The bulk of this algorithm is complete, all we have to do is output the desired information. We can output the centroid of the largest binary region by right clicking the “Largest Binary Region” node and selecting “Output → Centroid”. Once all of our image processing nodes have been configured properly, 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.

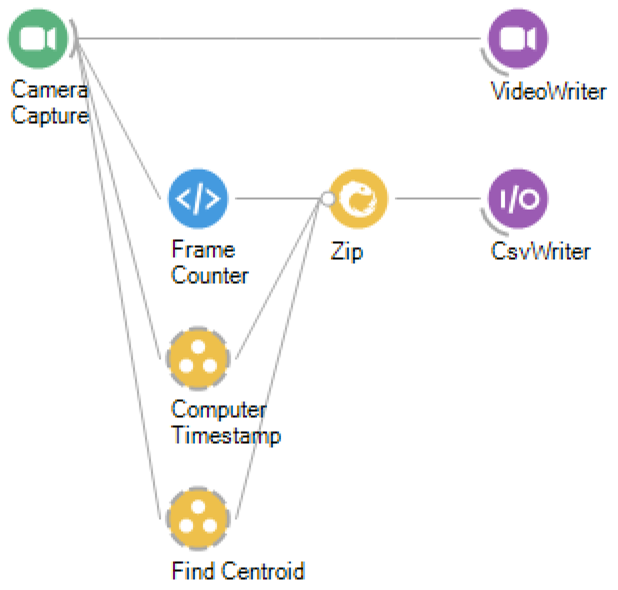


Find Centroid (GroupWorkflow)
Image processing algorithm for finding the position of an animal in a single chamber enclosure. In brief, the algorithm crops the incoming image to the chamber, segments the image to isolate the animal, and conducts a binary region analysis to find the centroid of the largest binary region i.e. the animal.

Design

Description	**Image processing algoı**
Name	**Find Centroid**

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.

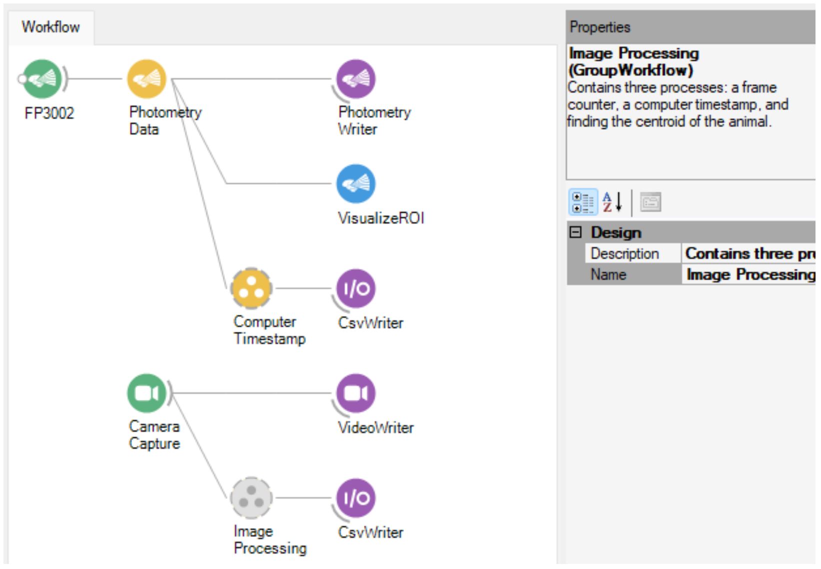
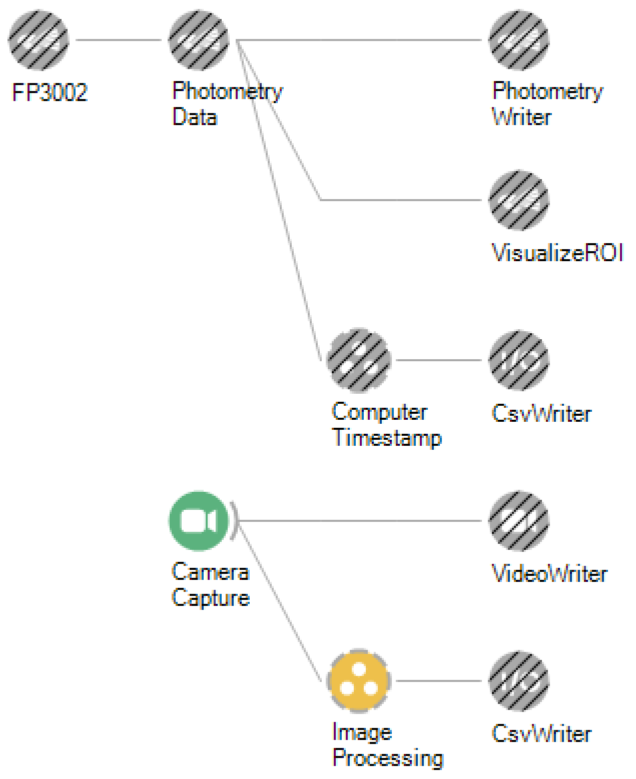


Image Processing (GroupWorkflow)
Contains three processes: a frame counter, a computer timestamp, and finding the centroid of the animal.

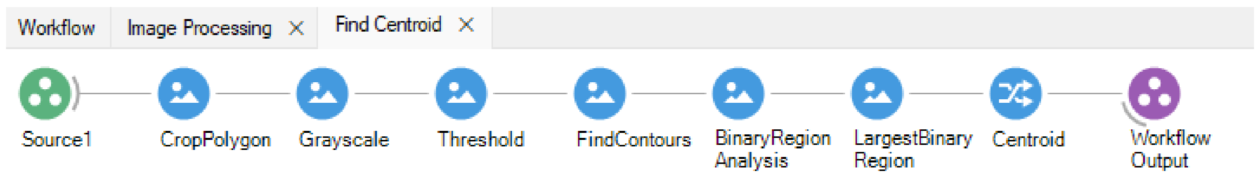
Design	
Description	Contains three pr
Name	Image Processing

Configuration:

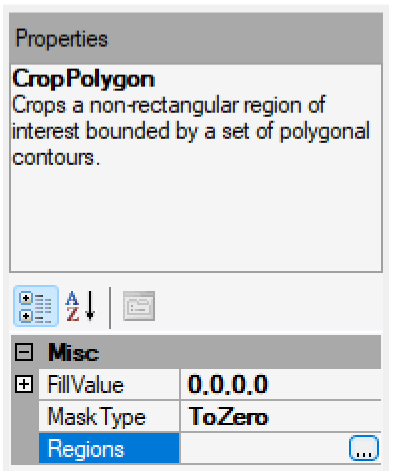
The nodes contained within our “Find Centroid” algorithm require configuration and should be reconfigured every experiment. We will configure these nodes while the workflow is running, so it is useful to temporarily disable all of the nodes contained within the photometry data stream as well as all of 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.

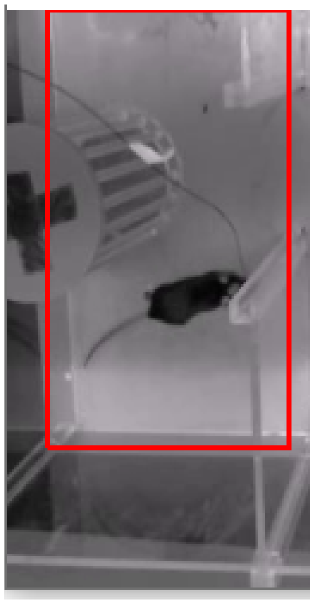

Properties

CropPolygon
Crops a non-rectangular region of interest bounded by a set of polygonal contours.

Misc

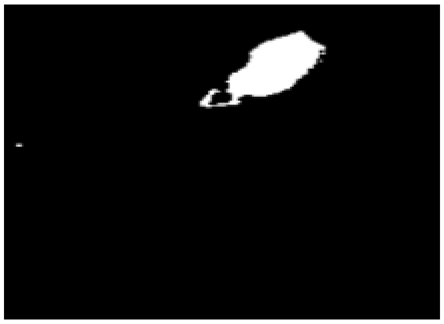
FillValue	0,0,0,0
MaskType	ToZero
Regions	...

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.



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

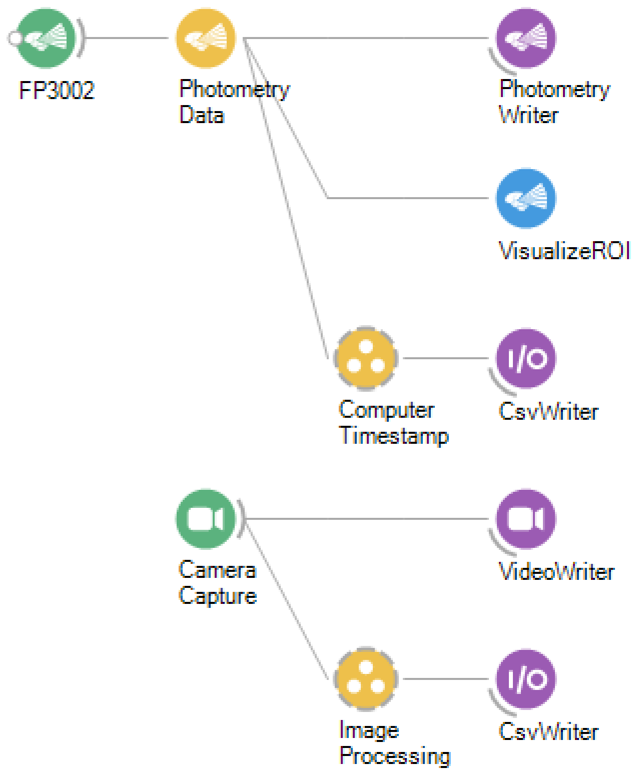
The next node that needs configuring is the “Threshold” node. First set the “Max Value” to the maximum possible pixel value. For most cases this will be 255, however when working with Mono16 images this value will be 65,535. Next, set the “Threshold Type” property to “Binary” or “BinaryInv”, depending 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

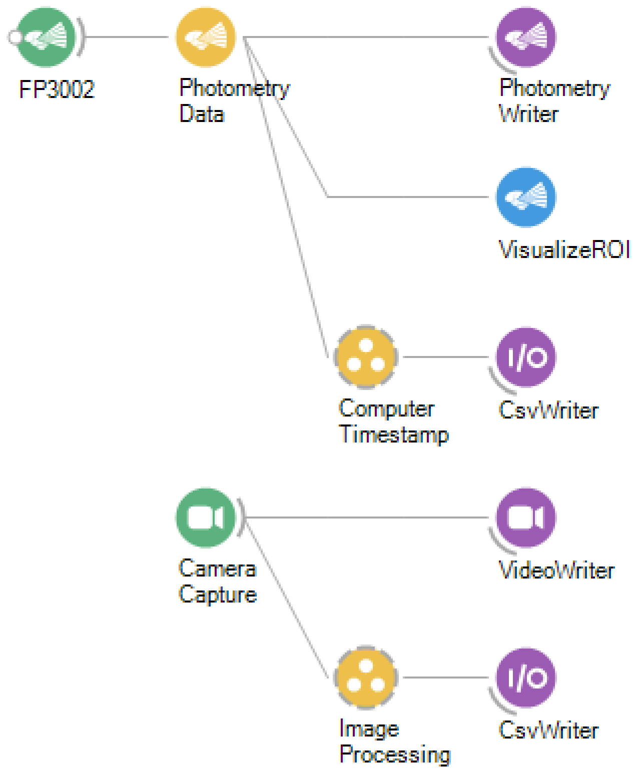
A useful feature of the visualizers for nodes in the “Bonsai.Vision” package is that when you right click the image in the visualizer, it will display the cursor position and the pixel values. You can use this feature to help configure the “Threshold” node by opening the “Grayscale” node’s visualizer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

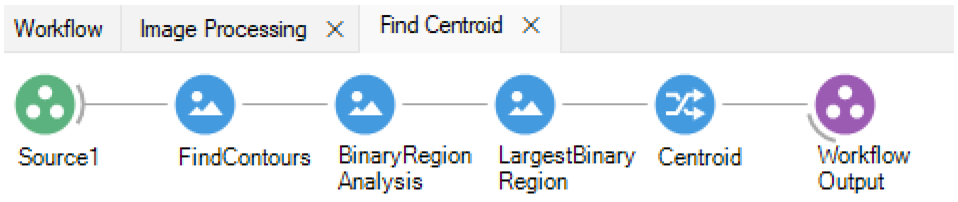


Three Chamber Animal Tracking

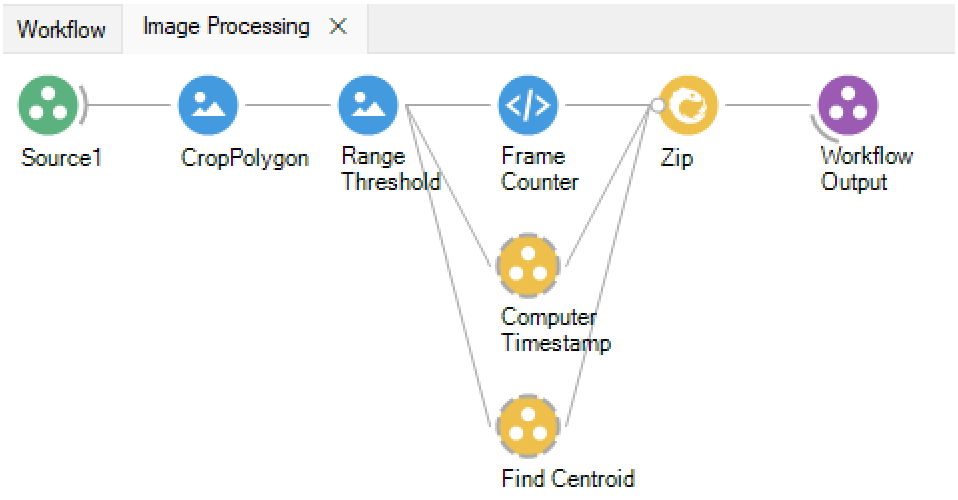
This section will expand the one chamber animal tracking workflow to allow for tracking an animal in a three chamber enclosure. Here we will save all of the same data as the previous workflow, except with an additional three columns in the behavior camera's *.csv* file to contain which chamber the animal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

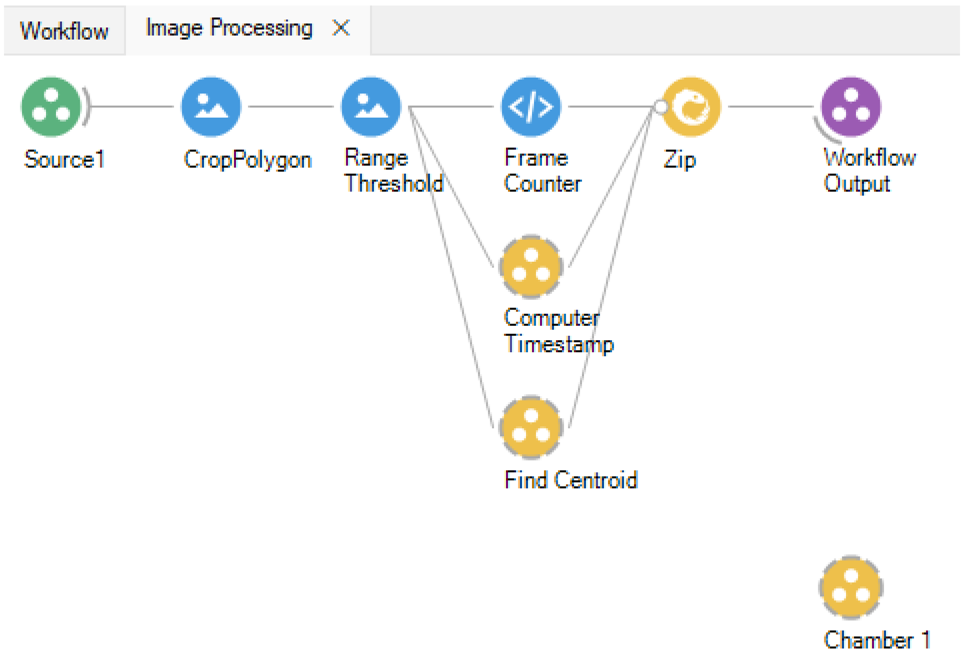


We will reimplement similar logic immediately after the “Source” node of the “Image Processing” grouped workflow. Instead of thresholding a grayscale image, let’s threshold a BGR image. This will allow you to have three dimensions to threshold the image instead of just one in the grayscale case. To do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

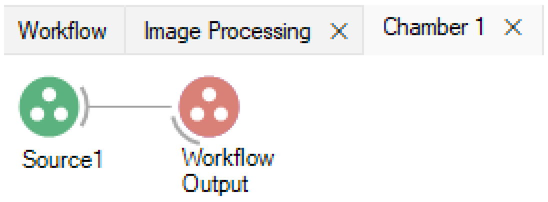


These image segmentation nodes will require configuration. We will cover some useful tricks for configuring these two nodes once all of our logic is set up. In particular, this section will cover cropping the image to non-rectangular enclosures and a way to visualize the BGR pixel values during thresholding.

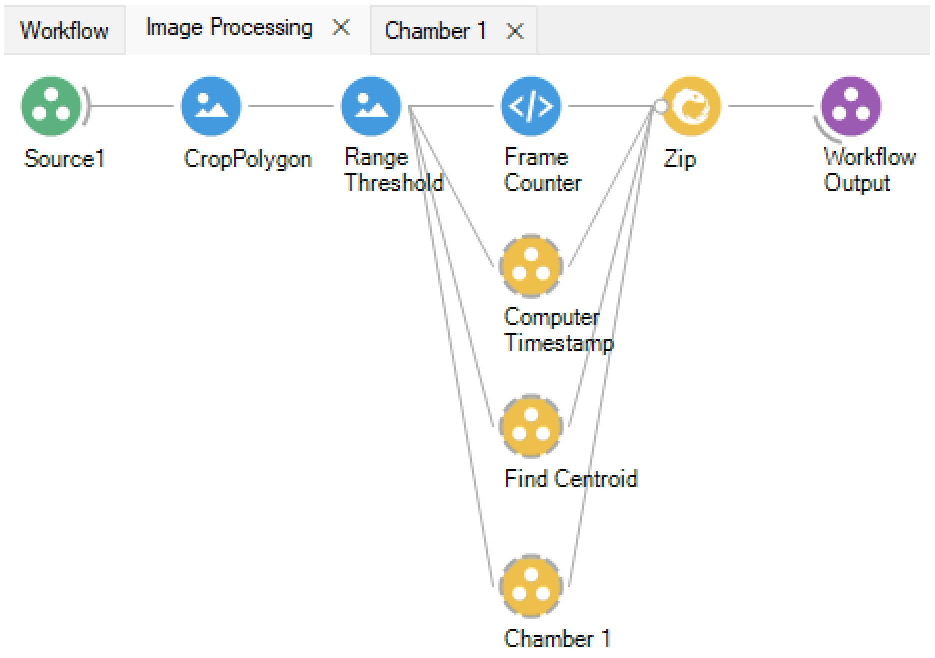
Next we will need to construct logic for determining whether or not the animal is located within a particular chamber of a multi-chamber enclosure. We will contain the logic for each chamber in separate grouped workflows. These chamber grouped workflows will accept the cropped, thresholded image and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



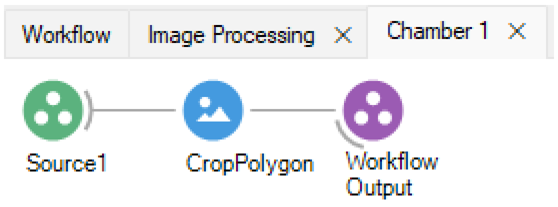
Inside of the “Chamber 1” grouped workflow, start by inserting a “Workflow Input” node followed by a “Workflow Output” node.



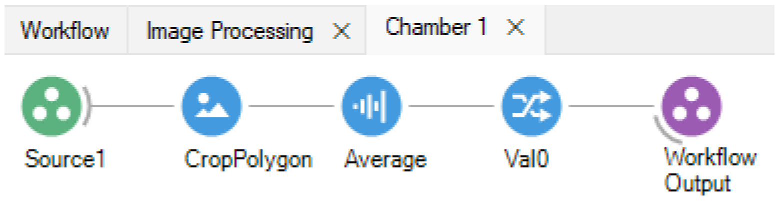
Now we can connect the “Range Threshold” node to the “Chamber 1” grouped workflow’s input and connect the output of the “Chamber 1” grouped workflow to the “Zip” node.



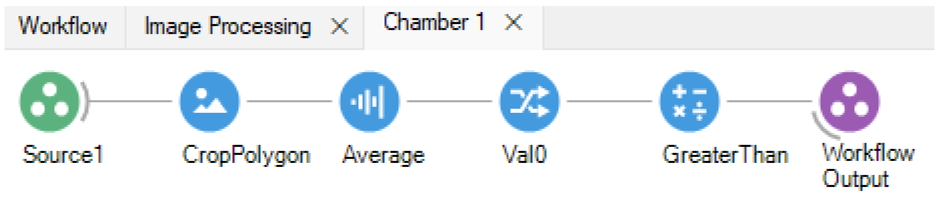
The image coming into the “Chamber 1” grouped workflow is already cropped to the full enclosure and thresholded. Within the “Chamber 1” node we need to crop the image again, this time to a particular chamber of the enclosure. Insert the “Crop Polygon” node immediately after the “Source” node of the “Chamber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



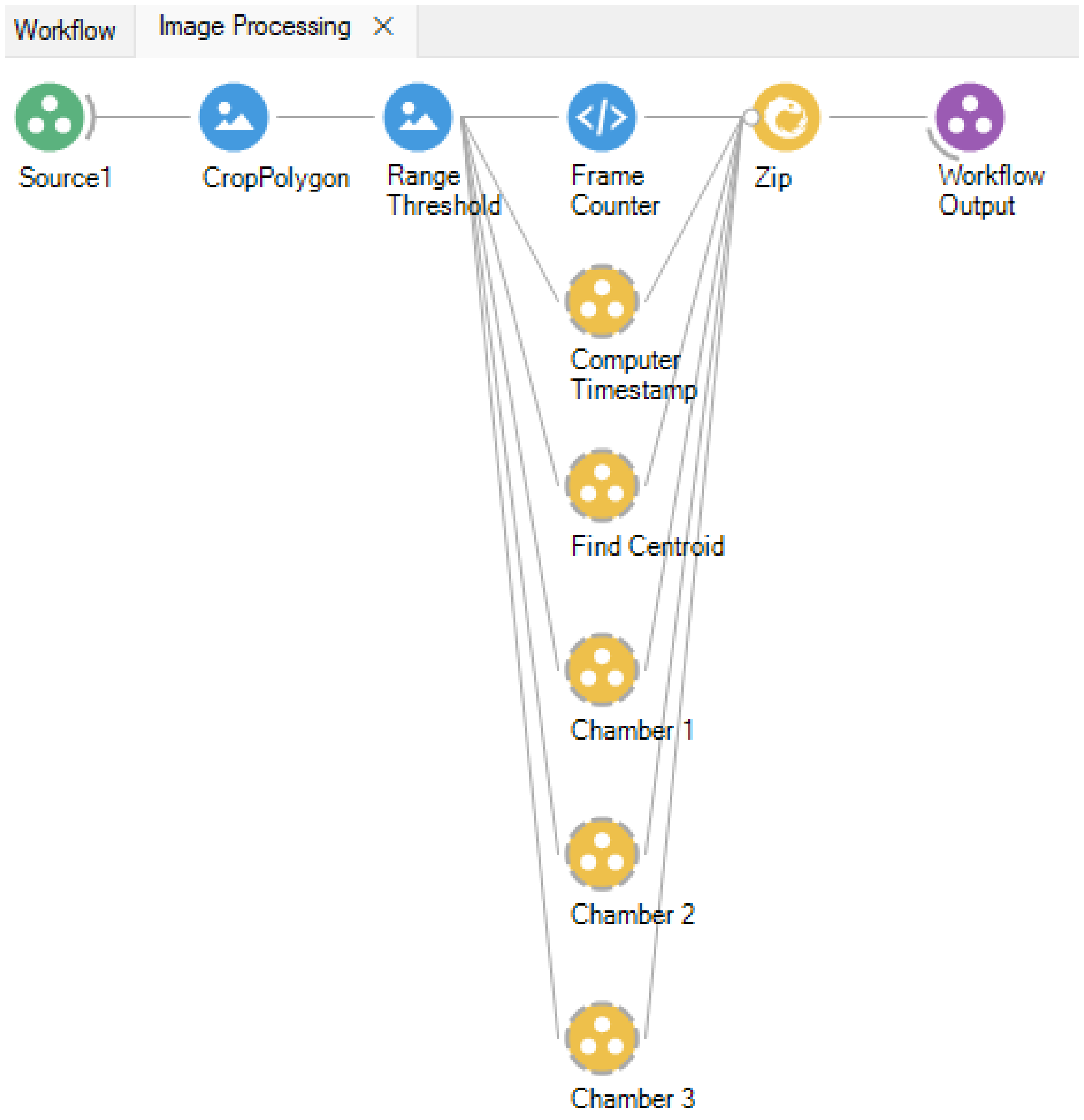
Since the image is already thresholded at this point of the workflow, after configuration, the image coming out of the “Crop Polygon” node should be mostly black. When the animal enters the chamber, the animal should appear white. This means that the average pixel value of the chamber when the animal 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.

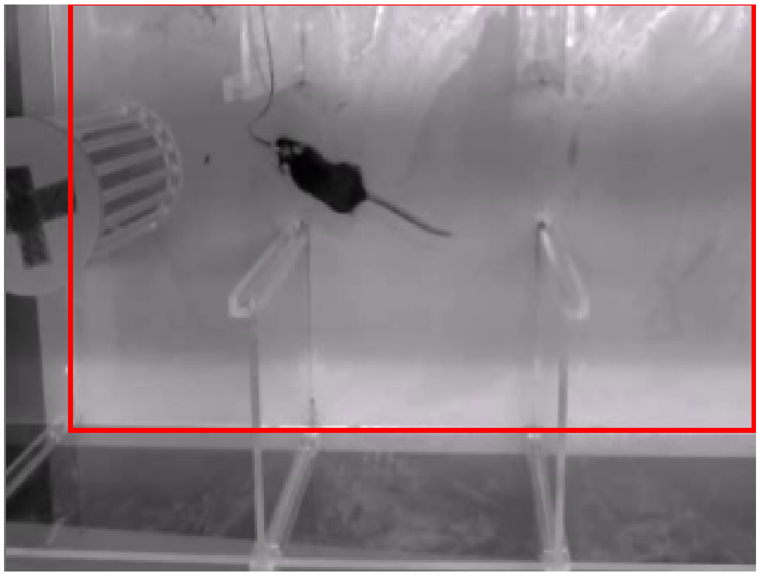


Our chamber analysis logic is now complete for “Chamber 1”. This exact same logic will be used for every other chamber in the enclosure. The only difference will be contained within the “Crop Polygon” node, configuring it to a different chamber. Duplicate the “Chamber 1” grouped workflow for each chamber 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.



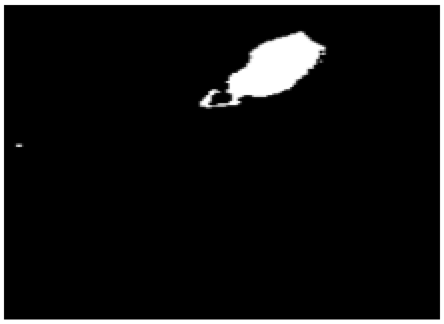
Configuration:

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.



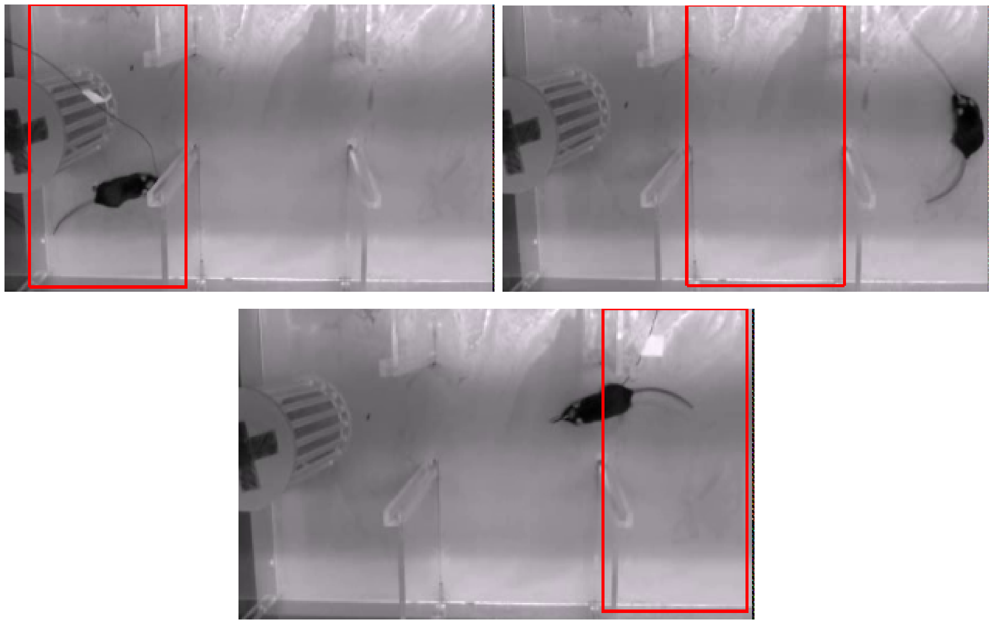
Dr Polter's lab - Institute for Neuroscience - The George Washington University

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.



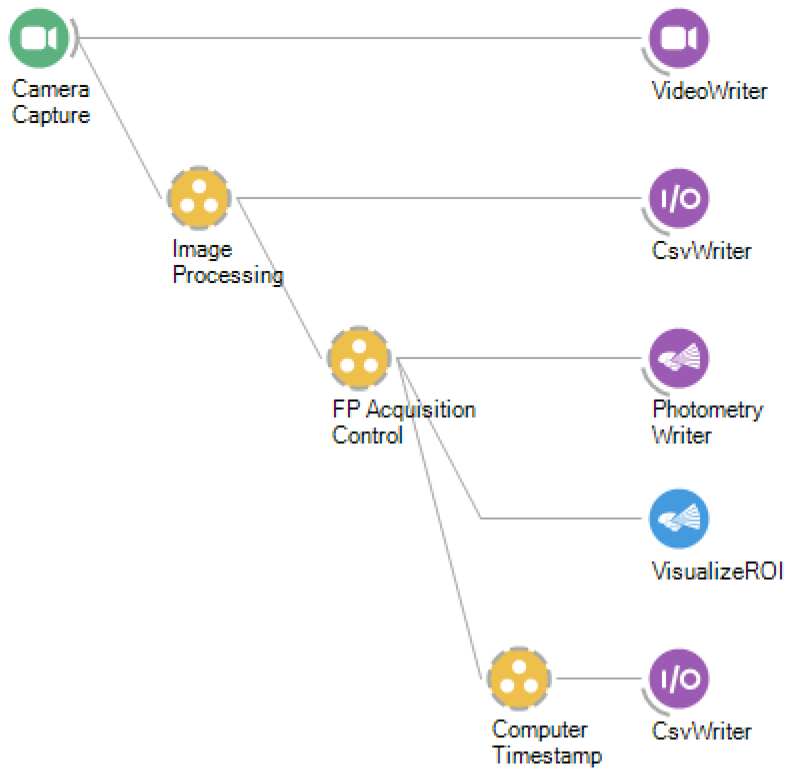
Dr Polter's lab - Institute for Neuroscience - The George Washington University

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.

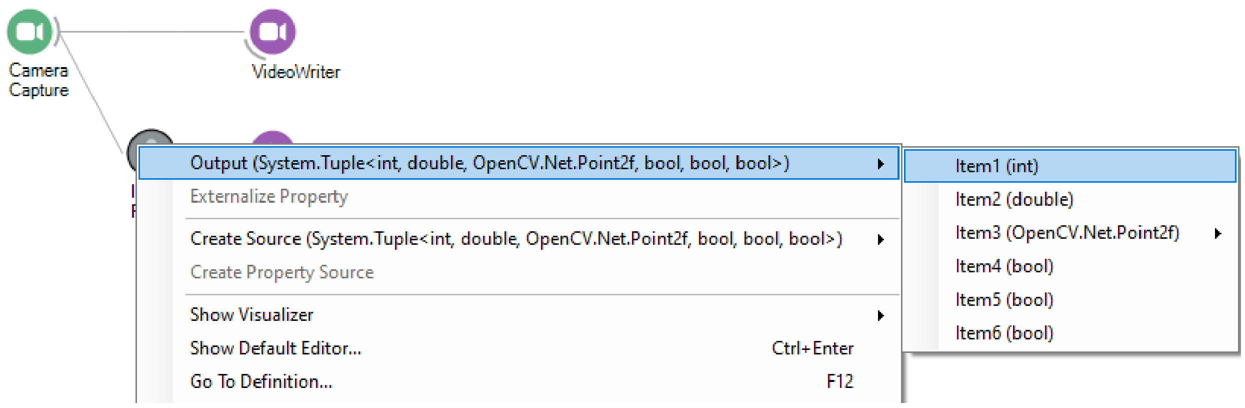
NEUROPHOTOMETRICS
The fiber photometry company

Chamber Dependent Data Acquisition

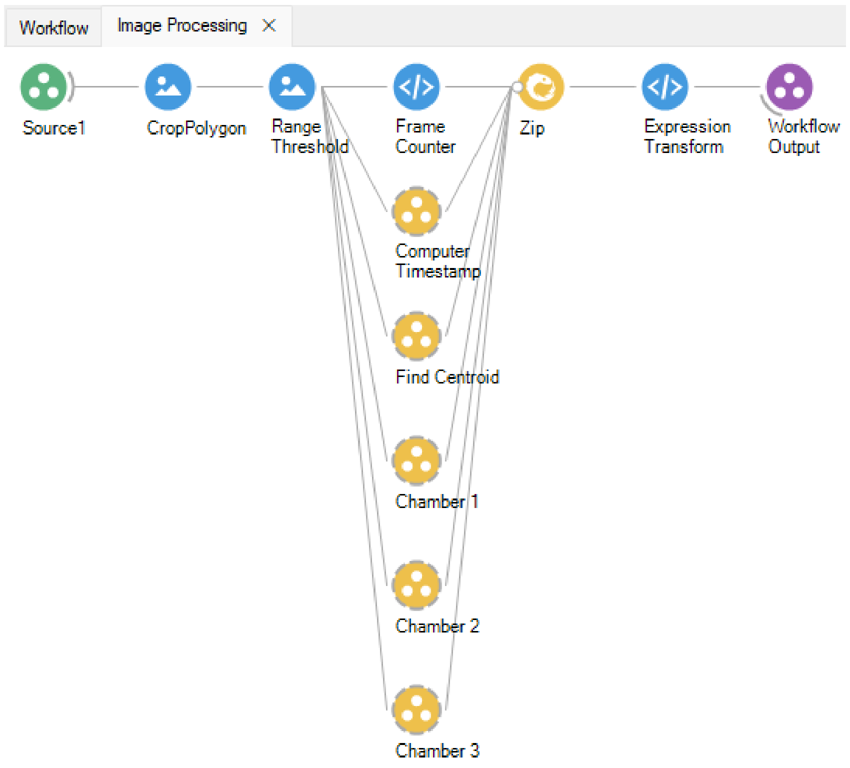
In the preceding sections, we have kept the photometry data and machine vision data streams parallel to each other so that they do not affect each other. However, in these last machine vision sections we will use the information that we acquire from our image processing algorithm to affect our photometry 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.



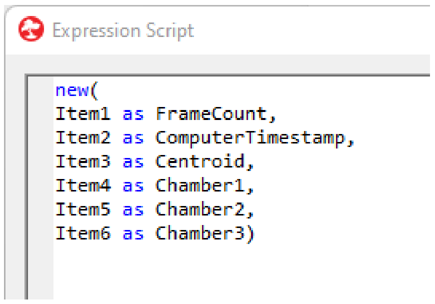
To begin constructing this workflow for chamber dependent data acquisition, begin with the “Three Chamber Animal Tracking” workflow. We will use the data coming from the chamber analysis grouped workflows to configure a software trigger that will control when data acquisition occurs. The first step is 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”.


Camera Capture
VideoWriter
Output (System.Tuple<int, double, OpenCV.Net.Point2f, bool, bool, bool>)
Externalize Property
Create Source (System.Tuple<int, double, OpenCV.Net.Point2f, bool, bool, bool>)
Create Property Source
Show Visualizer
Show Default Editor... Ctrl+Enter
Go To Definition... F12
Item1 (int)
Item2 (double)
Item3 (OpenCV.Net.Point2f)
Item4 (bool)
Item5 (bool)
Item6 (bool)

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.

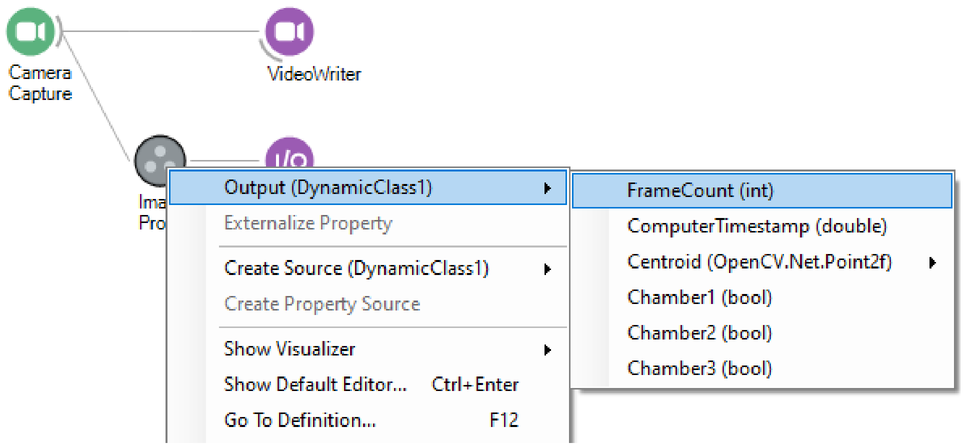


Open the editor for the "Expression Transform" by double clicking the node. Here we can create a dynamic class similar to a Tuple, where each element of the dynamic class has a user-specified name. In other words, you will be able to access each item using an informative name. Below is the syntax for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

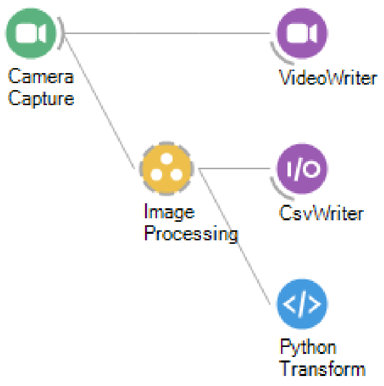


```
new(
Item1 as FrameCount,
Item2 as ComputerTimestamp,
Item3 as Centroid,
Item4 as Chamber1,
Item5 as Chamber2,
Item6 as Chamber3)
```

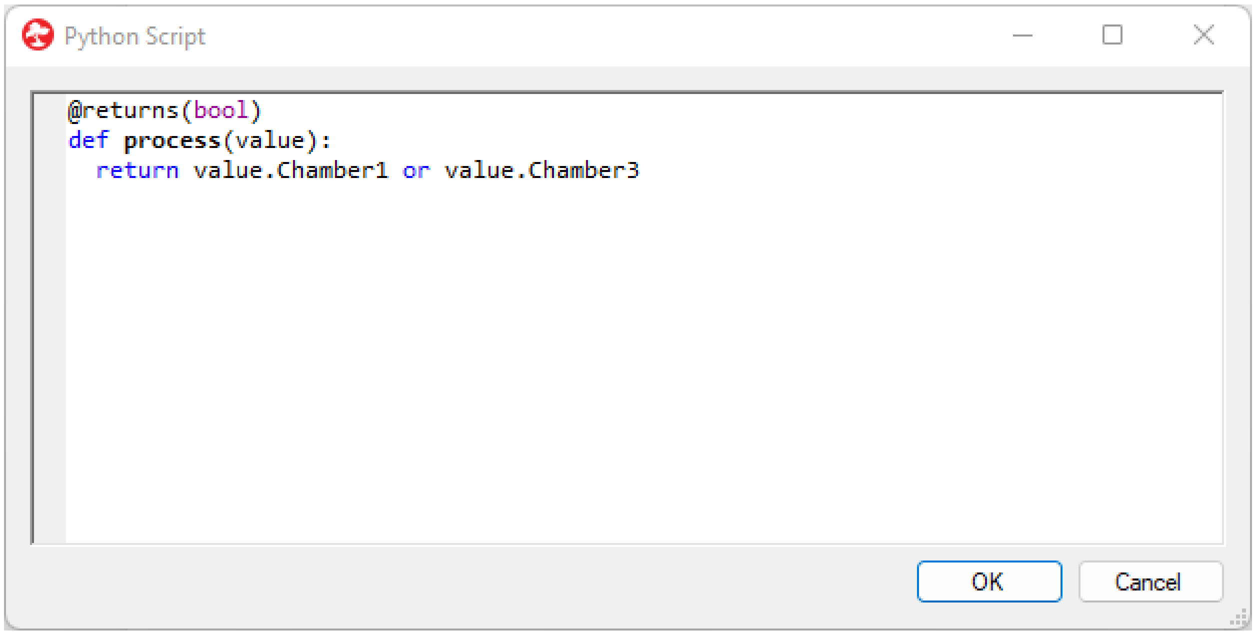
After implementing the above script within the "Expression Transform", you can see the effect of this script by rechecking the output of the "Image Processing" node. Instead of each element appearing as "Item1" through "Item6", we can actually see a descriptive name for each element being output.



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.

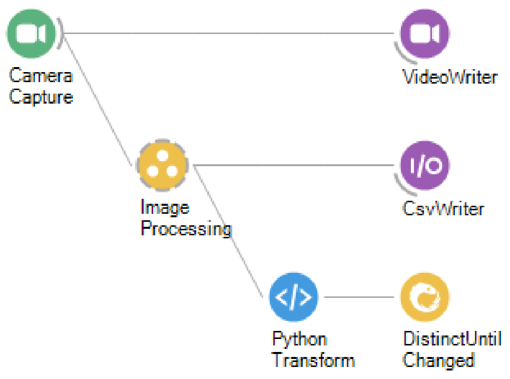


Inside the python script, we want to output the logical OR value of the "Chamber 1" and "Chamber 3" elements. This way, the "Python Transform" node will output True while the animal is in chambers one or three and False while the animal is in chamber two. Since we specified the names of items one through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

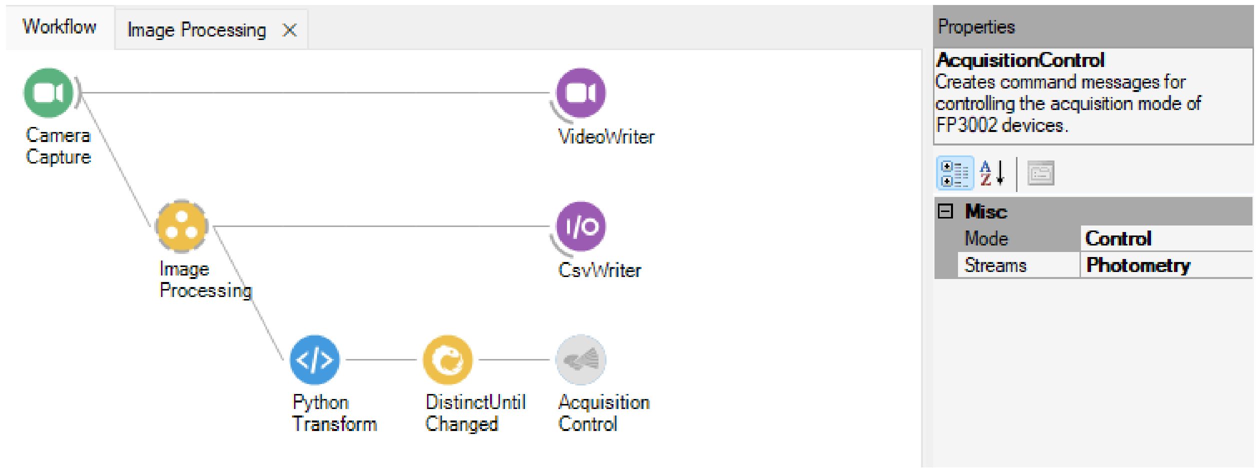


```
@returns(bool)
def process(value):
  return value.Chamber1 or value.Chamber3
```

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”.

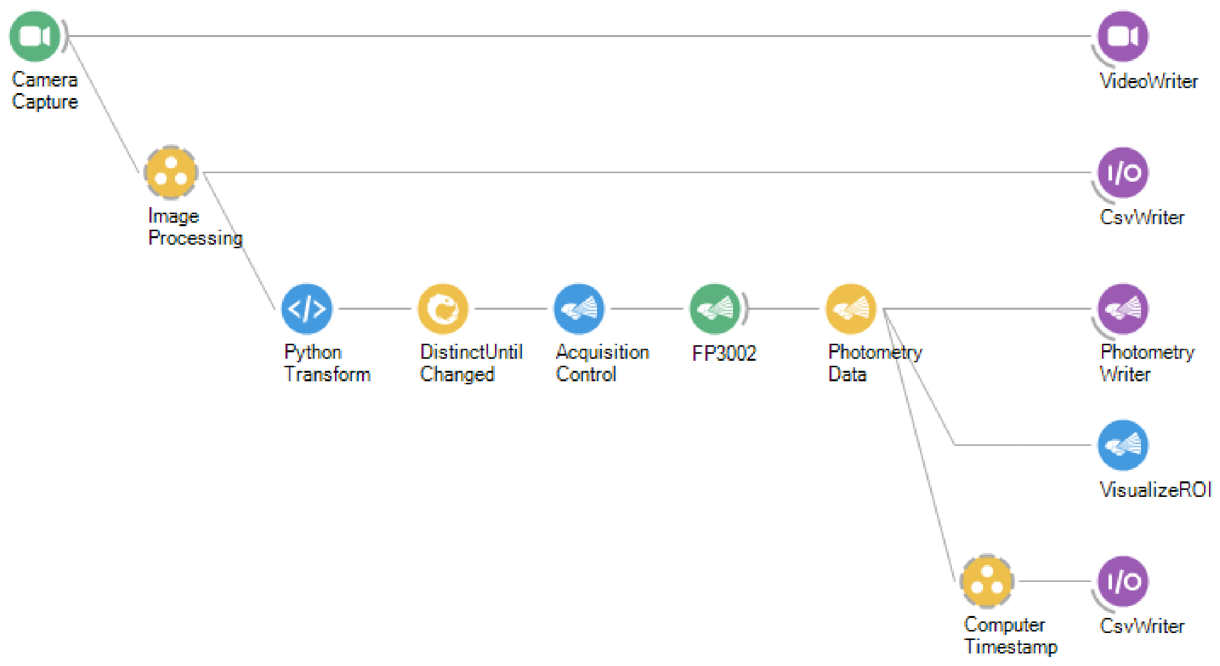


AcquisitionControl
Creates command messages for controlling the acquisition mode of FP3002 devices.

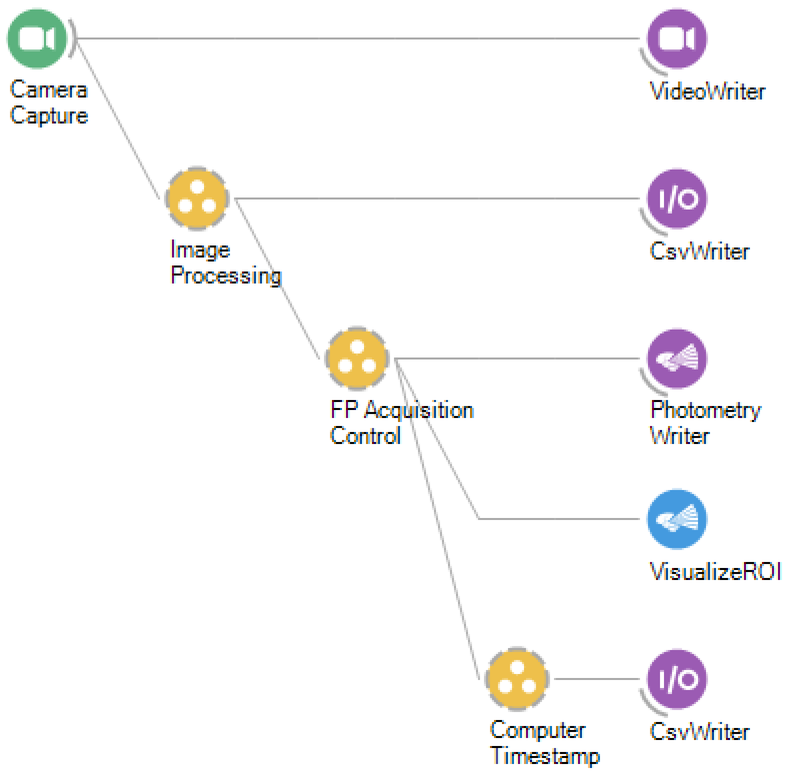
Misc

Mode	**Control**
Streams	**Photometry**

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



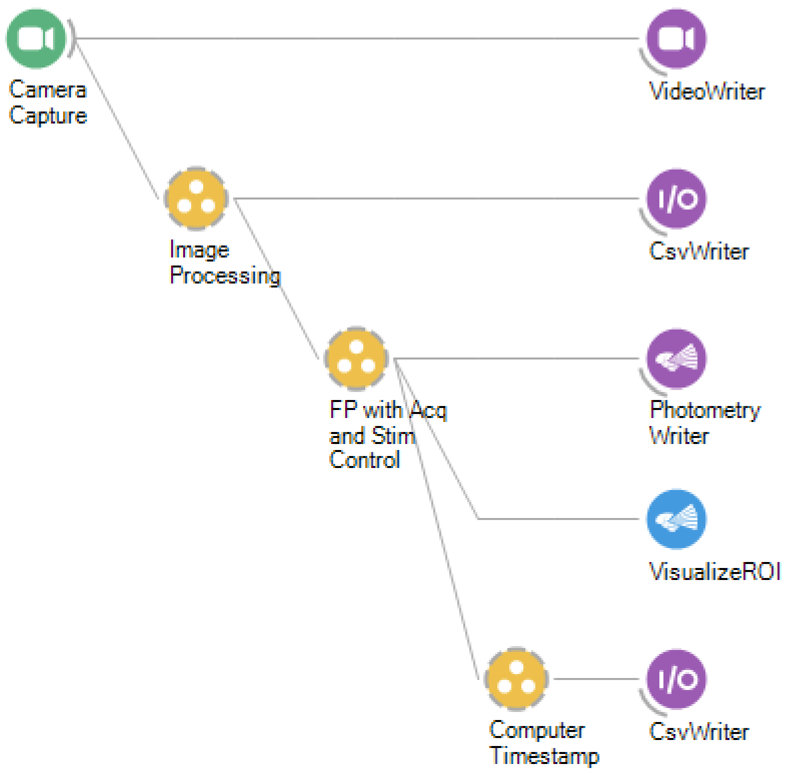
This workflow could use some more organization by grouping together the nodes used to create and control the flow of photometry data. In this example, we group all of the nodes from the “Python Transform” node to the “Photometry Data” node and name the grouped workflow “FP Acquisition Control”.



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.

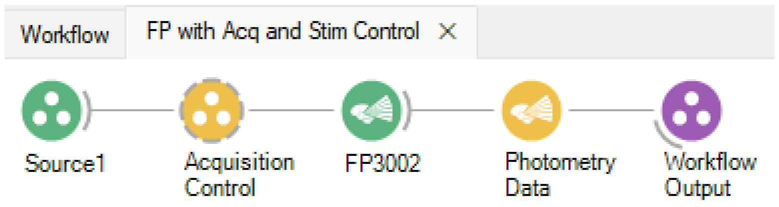
Chamber Dependent Stimulation

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.

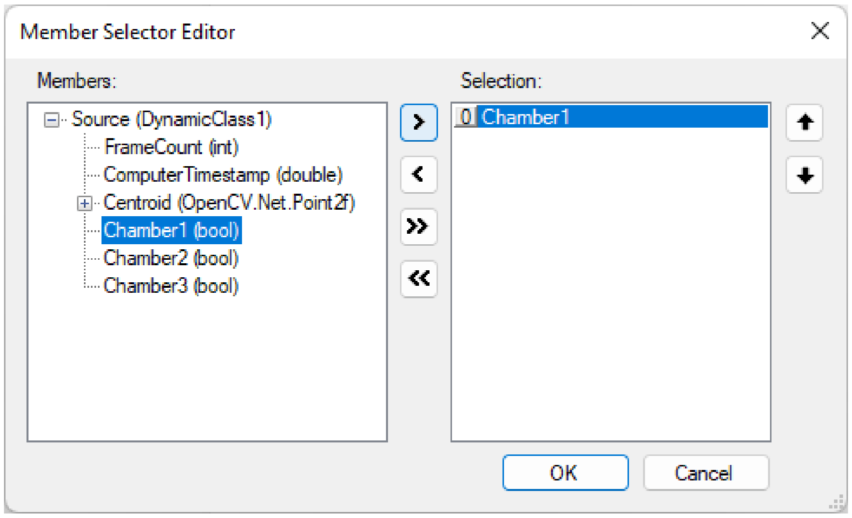


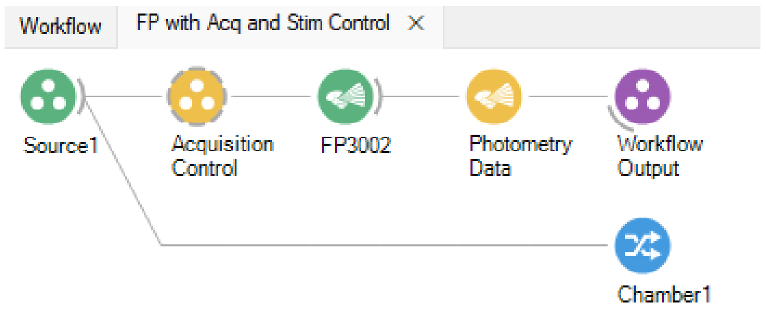
To construct this workflow, begin with the workflow created in the “Chamber Dependent Data Acquisition” section. Begin by renaming the grouped workflow containing the nodes used for creating and controlling the photometry data stream. In the previous example, this was named “FP Acquisition Control”. 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.

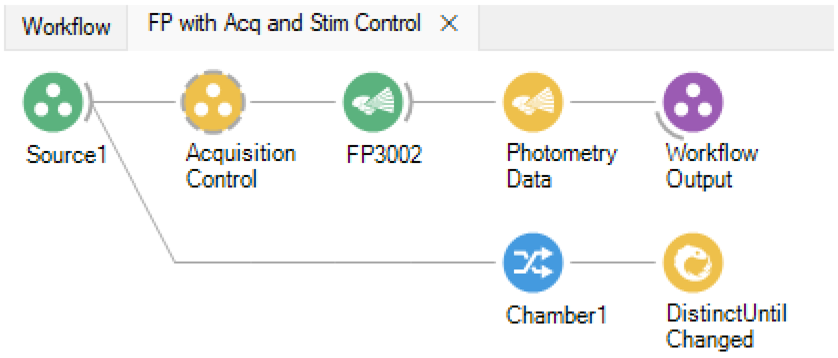


We will now connect our stimulation control logic to the output of the “Source1” node in parallel with the “Acquisition Control” grouped workflow. Begin by connecting a “Member Selector” node to the source node. Open the editor for this node, click the “Chamber1” member and select it by clicking the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

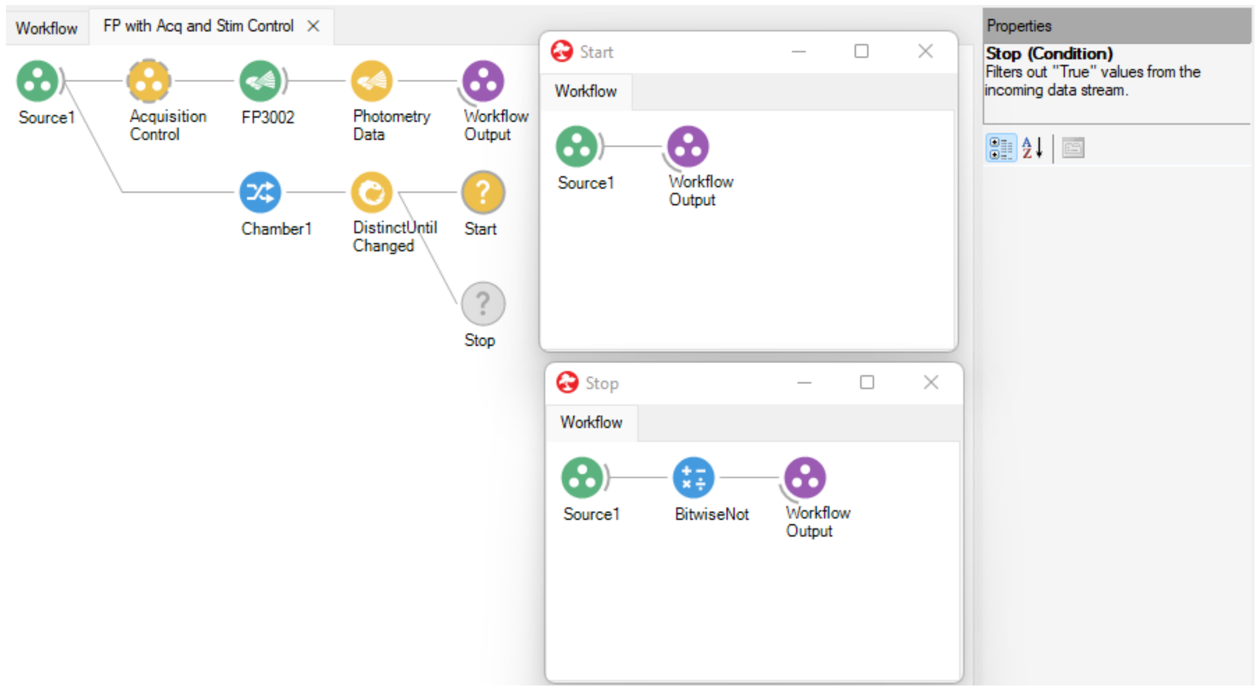




For every behavioral camera frame, this “Chamber1” node will output a “True” value while the animal is in chamber one and “False” otherwise. We only want to send a stimulation command when this value changes so be sure to include a “Distinct Until Changed” node immediately after.

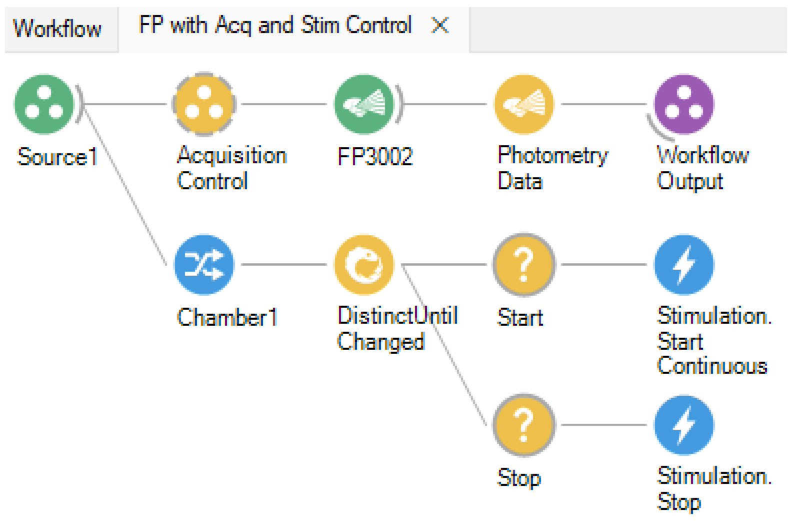


From the “Distinct Until Changed” node, connect two “Condition” nodes in parallel to deinterleave the boolean values into two data streams. One “Condition” node requires no changes to filter out the “False” values, we will name this one “Start”. The other “Condition” node requires a “Bitwise Not” node 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.

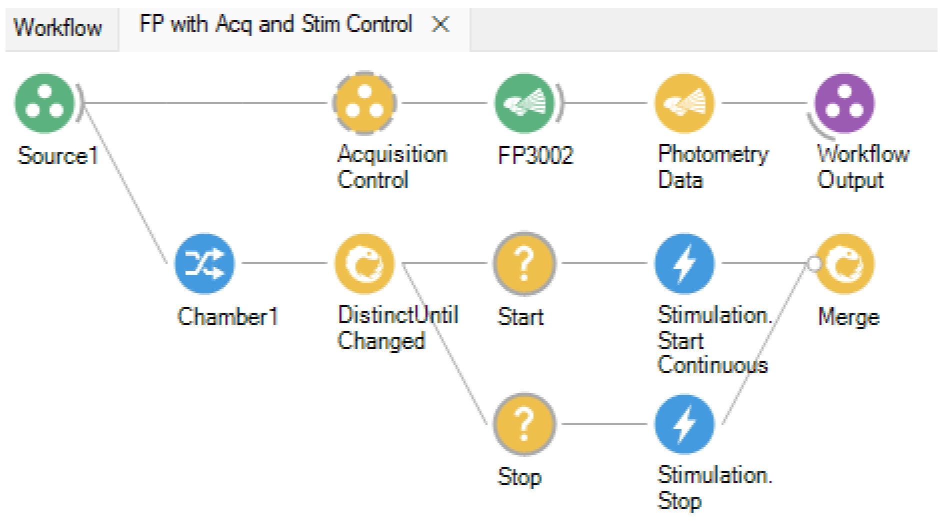


Stop (Condition)
Filters out "True" values from the incoming data stream.

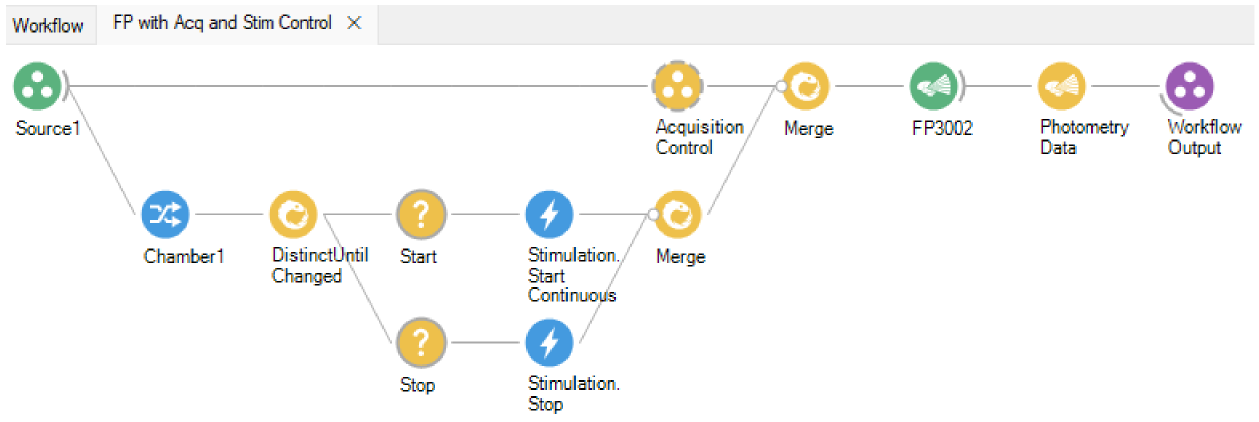
A “Stimulation” node needs to be connected to both of the “Condition” nodes. Configure the “Stimulation” node connected to the “Start” node to have a start stimulation command. Meanwhile, configure the “Stimulation” node connected to the “Stop” node to have a stop stimulation command.



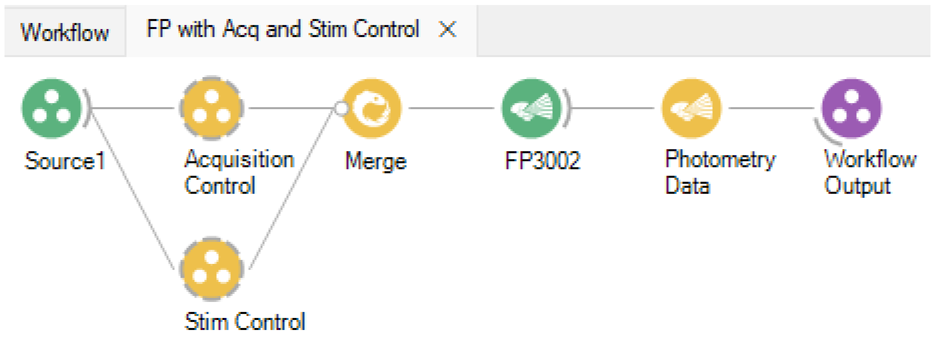
Next, merge together the two data streams containing stimulation commands.



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



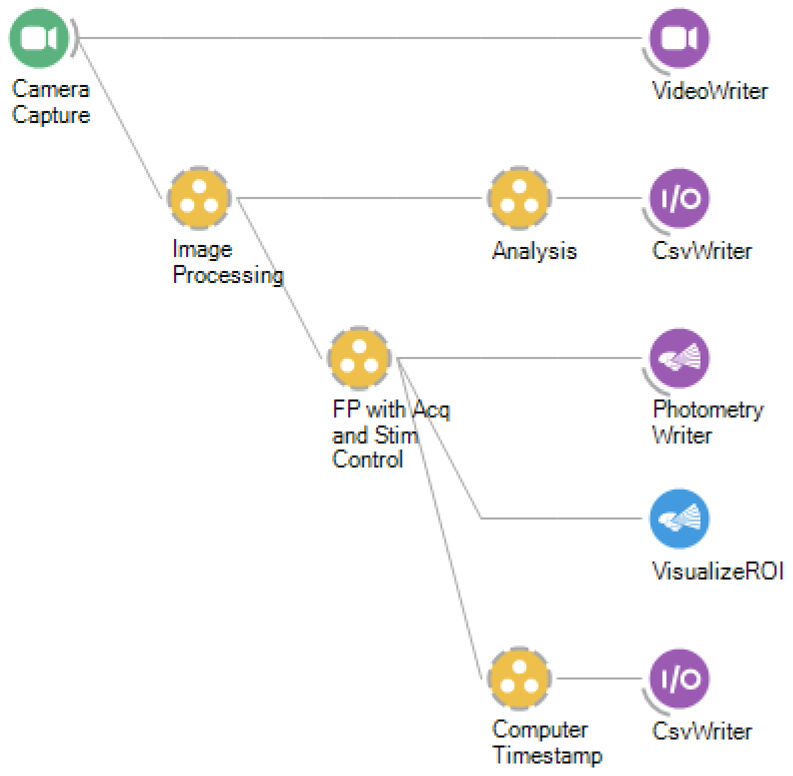
Finally, add some organization by grouping together all of the stimulation control logic. In this example, we group from the “Chamber 1” node to the first “Merge” node and name it “Stim Control”.



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.

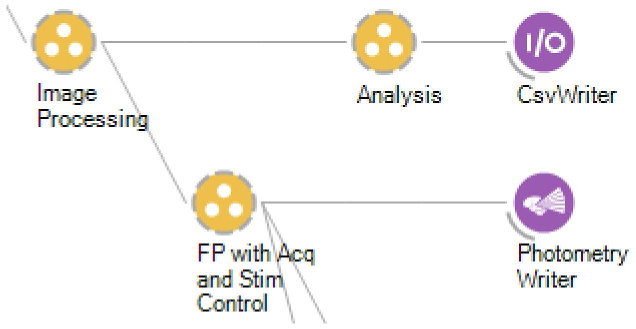
Real Time Analysis

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).

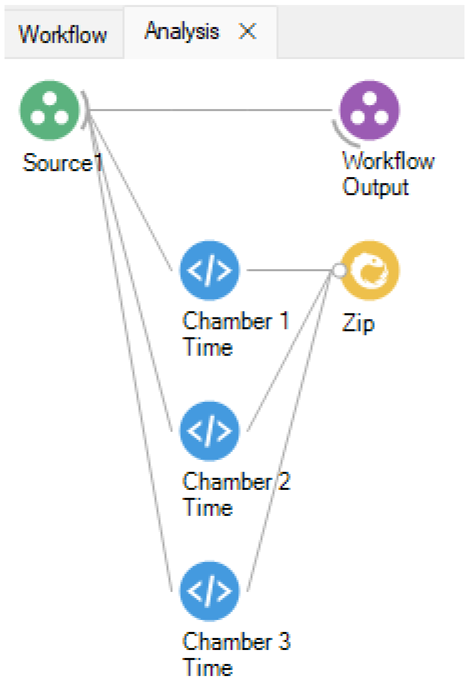


Organization:

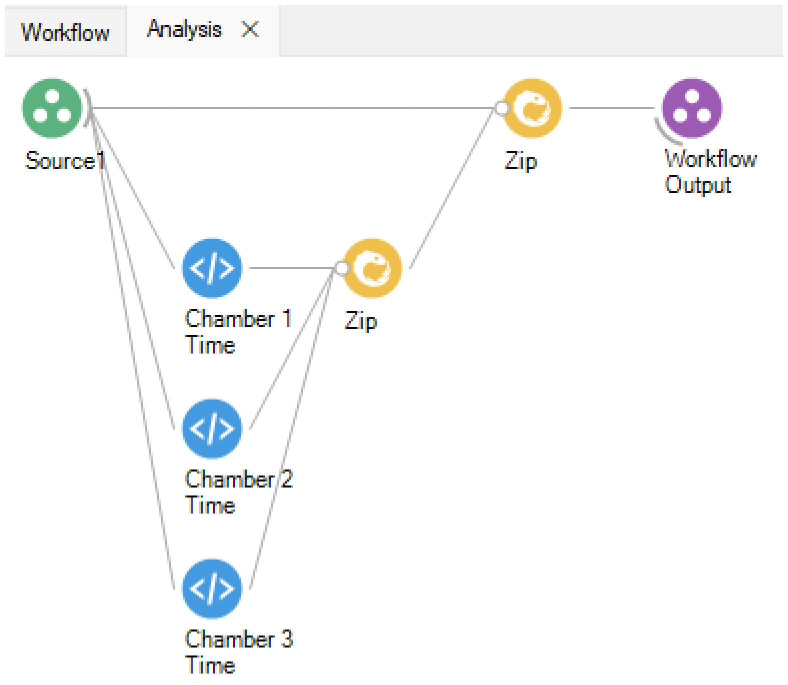
In this example, we will add these real time analysis algorithms to the workflow developed in the previous section. However, similar methods can be used to add analysis algorithms to other machine vision workflows. Our first task is to find a location within our workflow that we can insert these algorithms, 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".



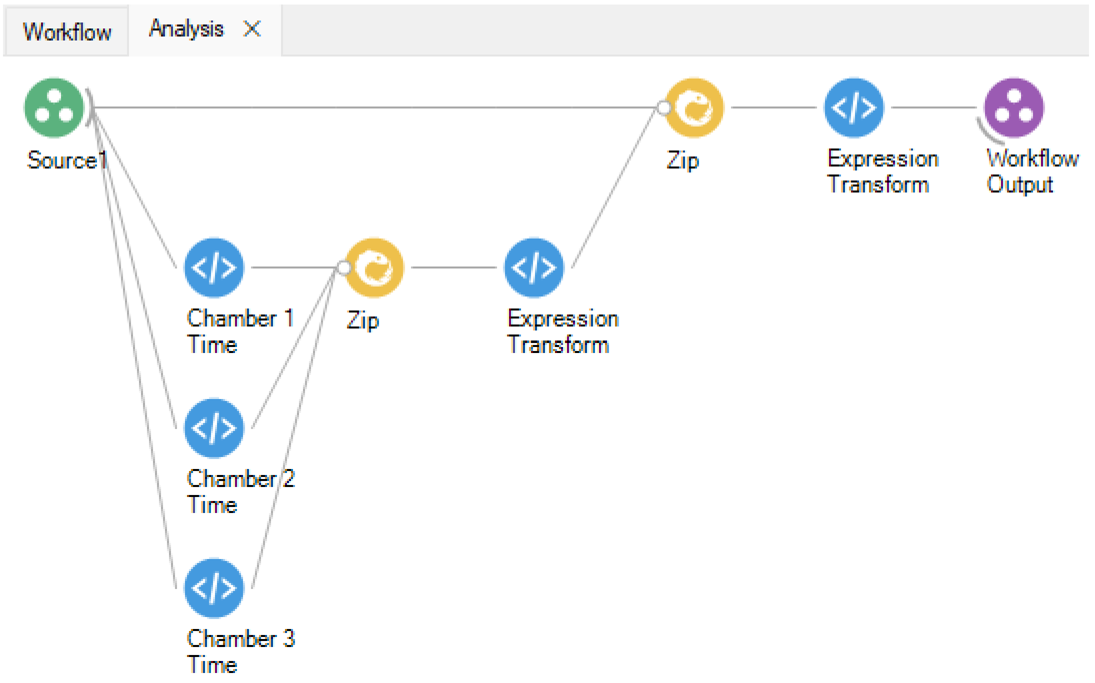
Inside of the "Analysis" grouped workflow, we will want to be sure to preserve the existing data produced by the "Image Processing" grouped workflow as well as analyze this incoming data. Here we will conduct all of our analyses in parallel data streams and combine them with our preserved incoming data. 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.



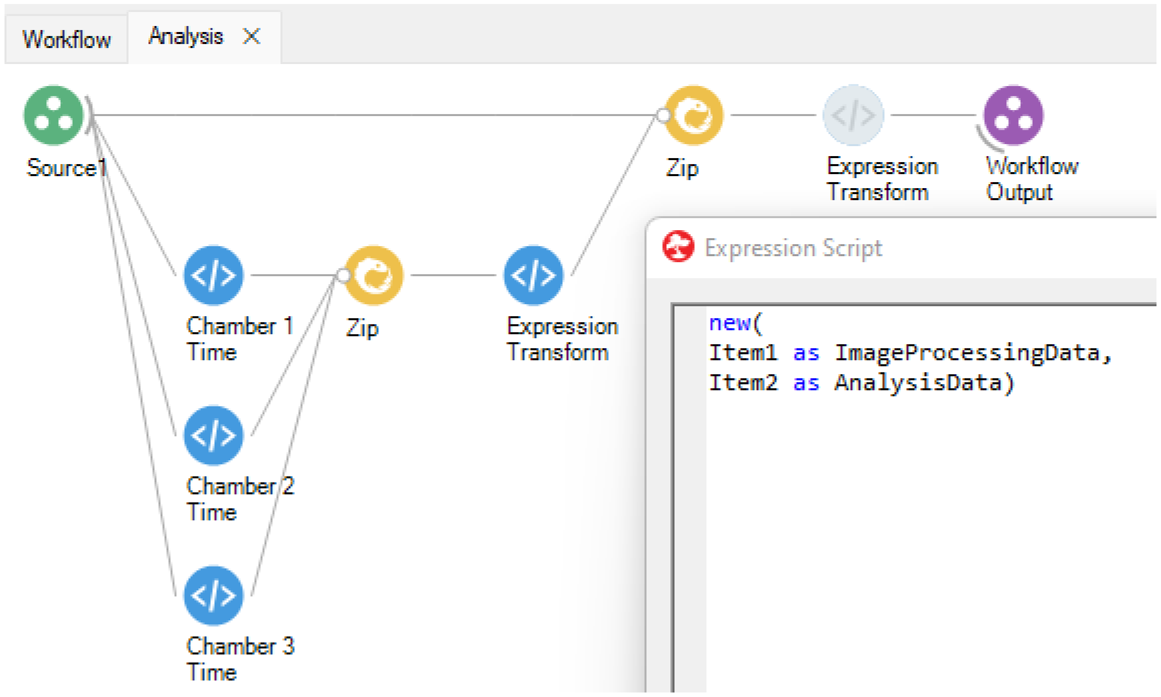
Before implementing our python scripts, let's finish the structure of the "Analysis" grouped workflow. This grouped workflow needs to output all of the image processing data as well as all of the analysis data. Since the analysis algorithms will produce data at the same rate as the image processing algorithms, 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.

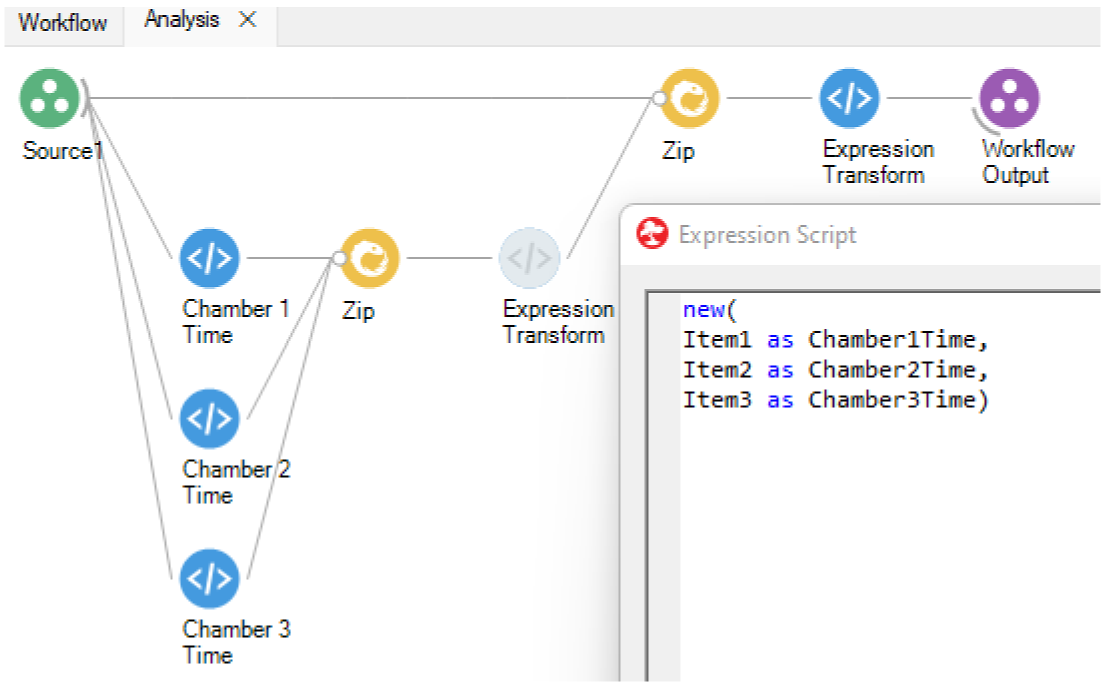


The “Expression Transform” connected to the “Workflow Output” will be used to specify between image processing data and analysis data. The below script will rename Item1 (the data coming from the “Source1” node) to “ImageProcessingData” and will rename Item2 (the data coming from the zipped analysis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



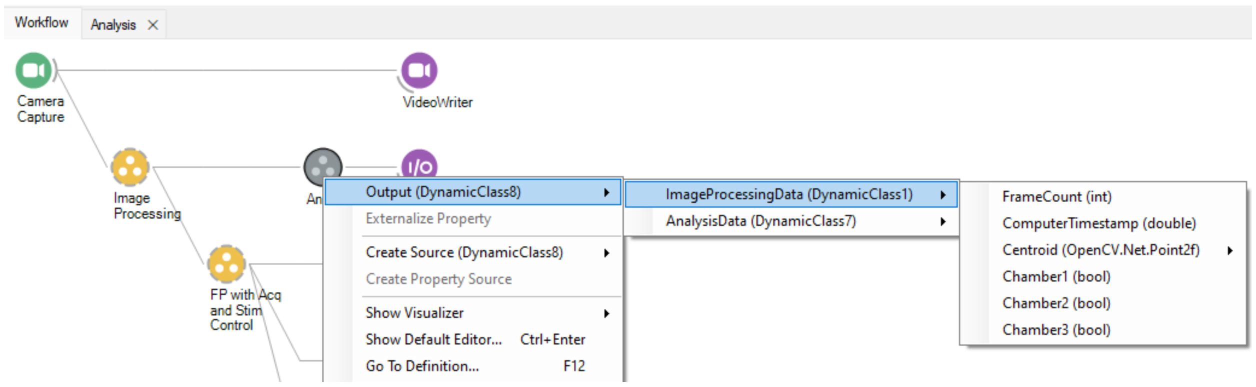
```
new(
Item1 as ImageProcessingData,
Item2 as AnalysisData)
```

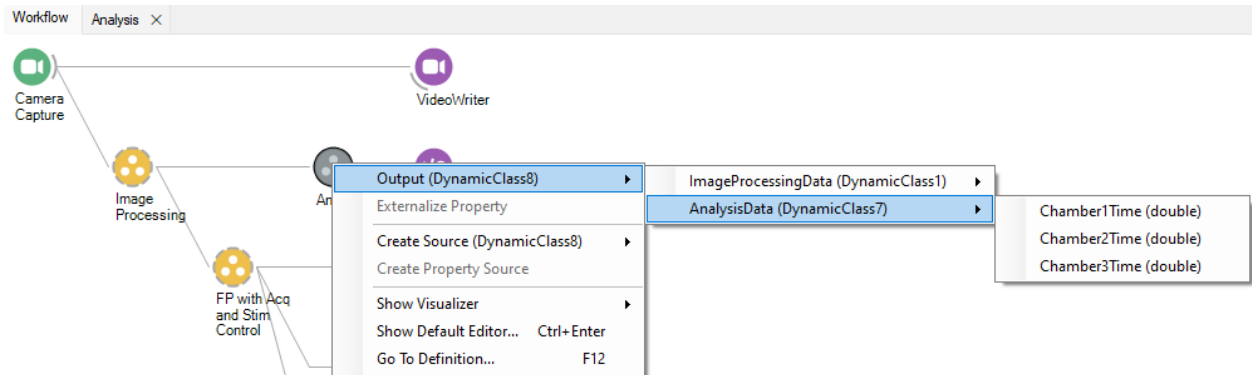
The other “Expression Transform” will rename the data coming from the zipped analyses. In this case, we will rename items one through three to “Chamber1Time” through “Chamber3Time”. We will update this “Expression Transform” node every time we add a new real time analysis algorithm.



```
new(
Item1 as Chamber1Time,
Item2 as Chamber2Time,
Item3 as Chamber3Time)
```

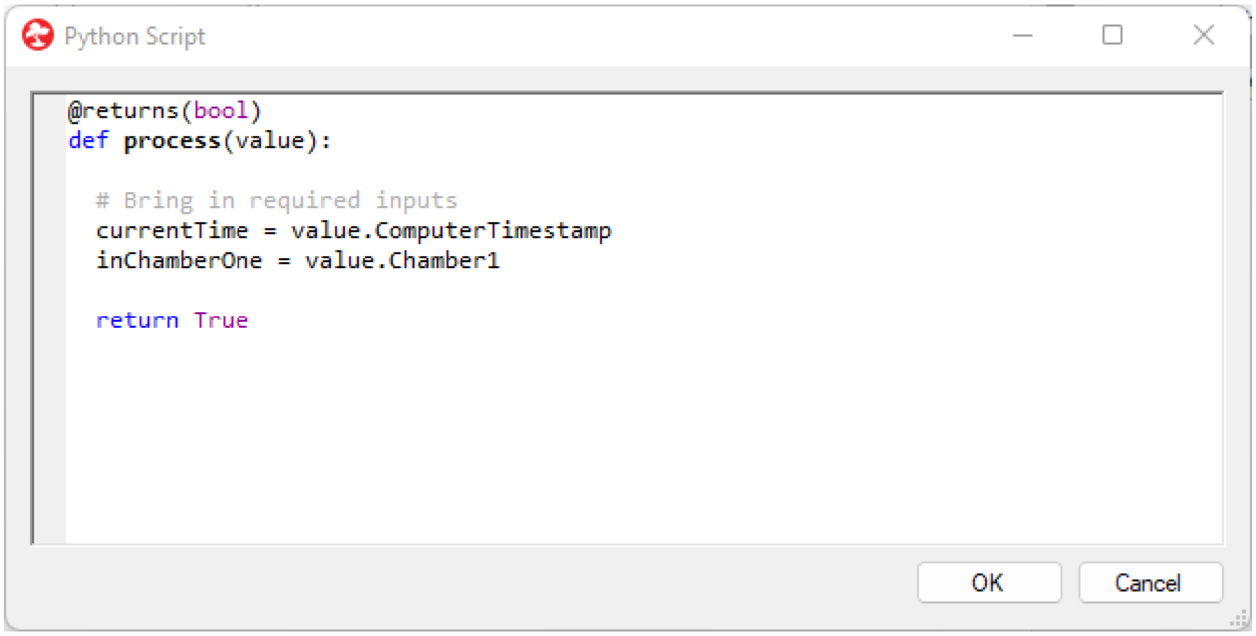
The immediate effect of this level of organization is that the element names are displayed in the output tree of the “Analysis” grouped workflow. Right click it and navigate the “Output” tree to see all of the elements contained in the output data of this grouped workflow. This organization is also useful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





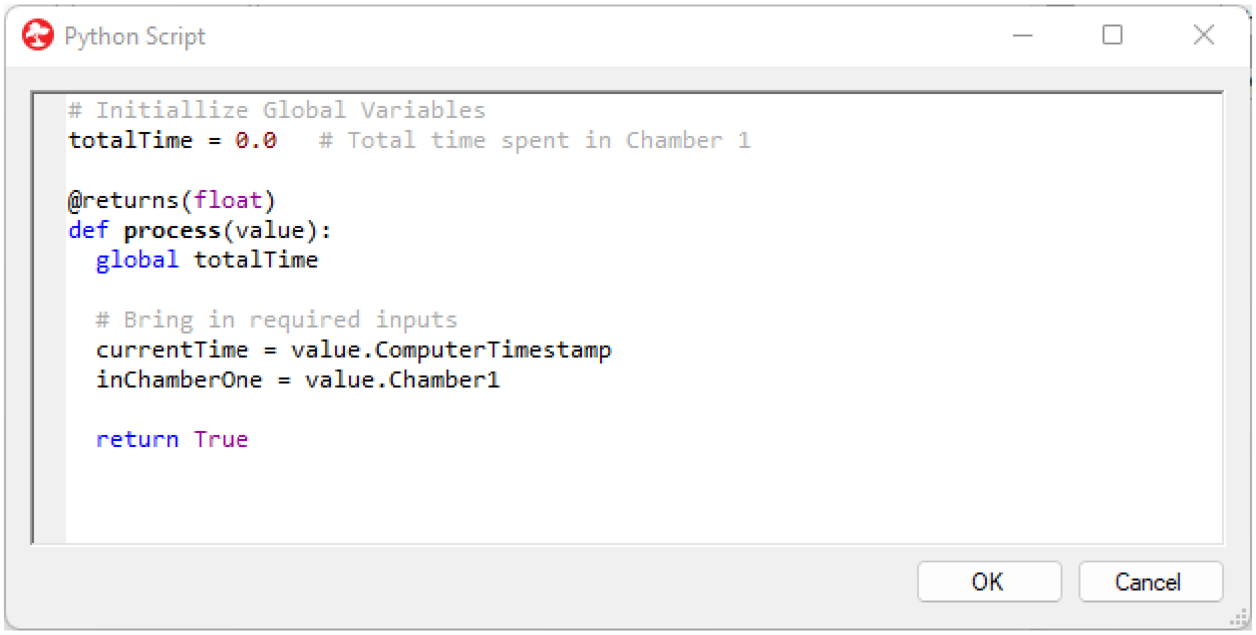
Chamber Duration:

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.



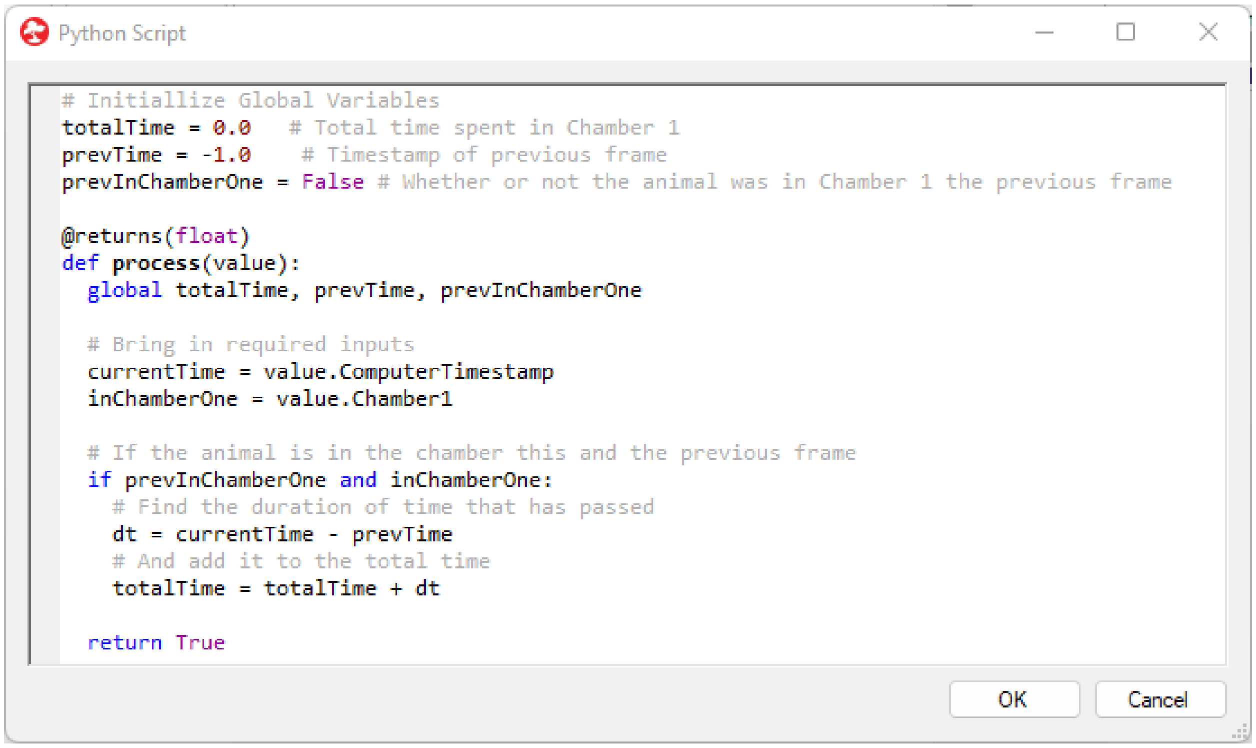
```python
@returns(bool)
def process(value):

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

  return True
```

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.



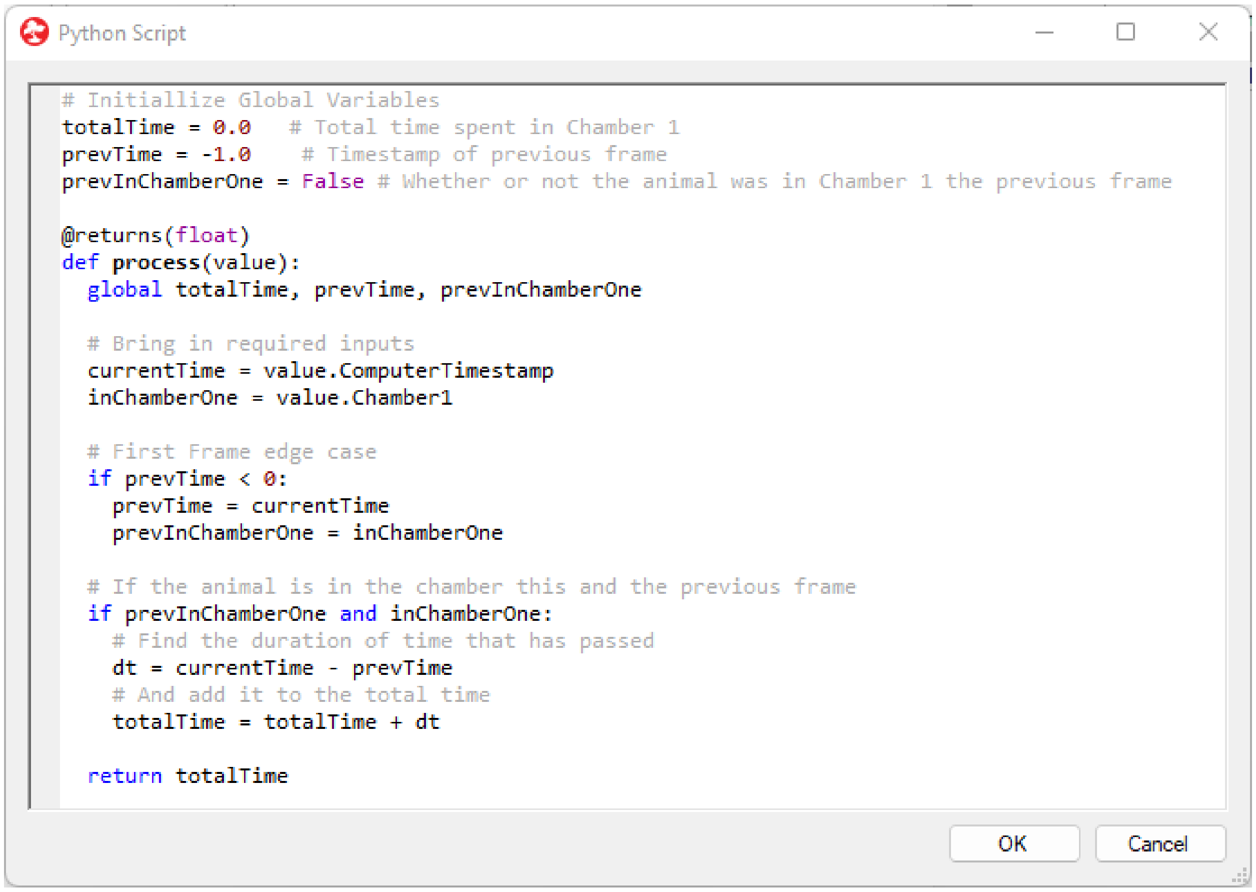
```
# Initiallize Global Variables
totalTime = 0.0    # Total time spent in Chamber 1

@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.



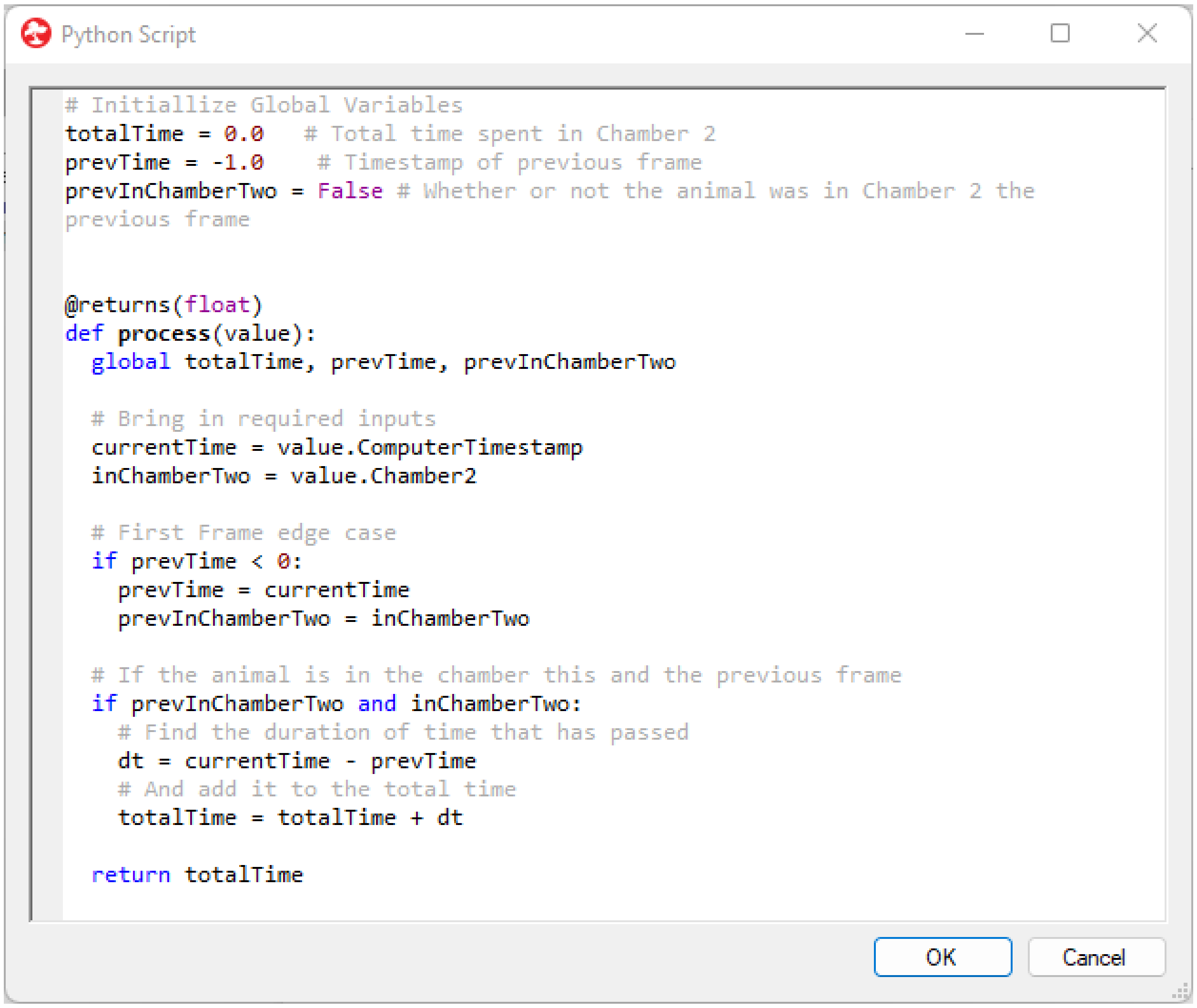
```python
# Initiallize Global Variables
totalTime = 0.0    # Total time spent in Chamber 1
prevTime = -1.0     # Timestamp of previous frame
prevInChamberOne = False # Whether or not the animal was in Chamber 1 the previous frame

@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
# 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

  # 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
```

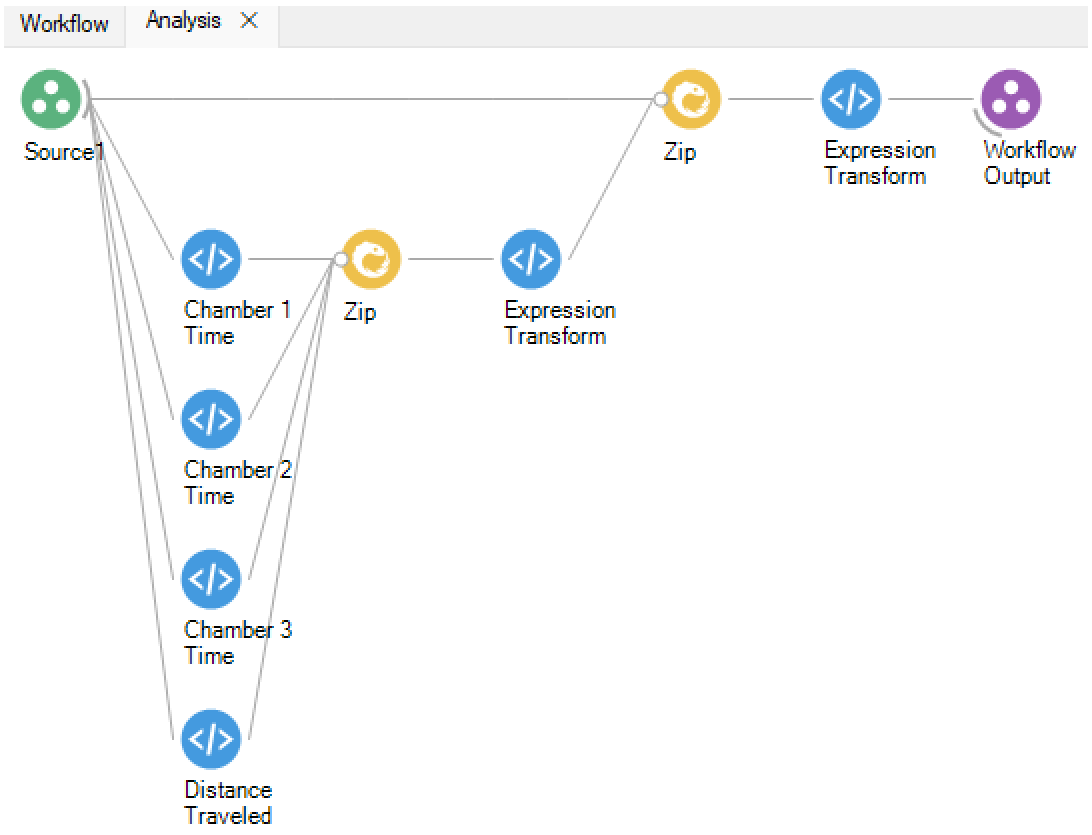
We have now completed the algorithm for tracking the duration of time the animal spends in chamber one. We can copy this script into the chamber two and three python scripts and make minor changes in order to complete those algorithms. Technically, all that has to be changed is reading in “value.Chamber1” 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.



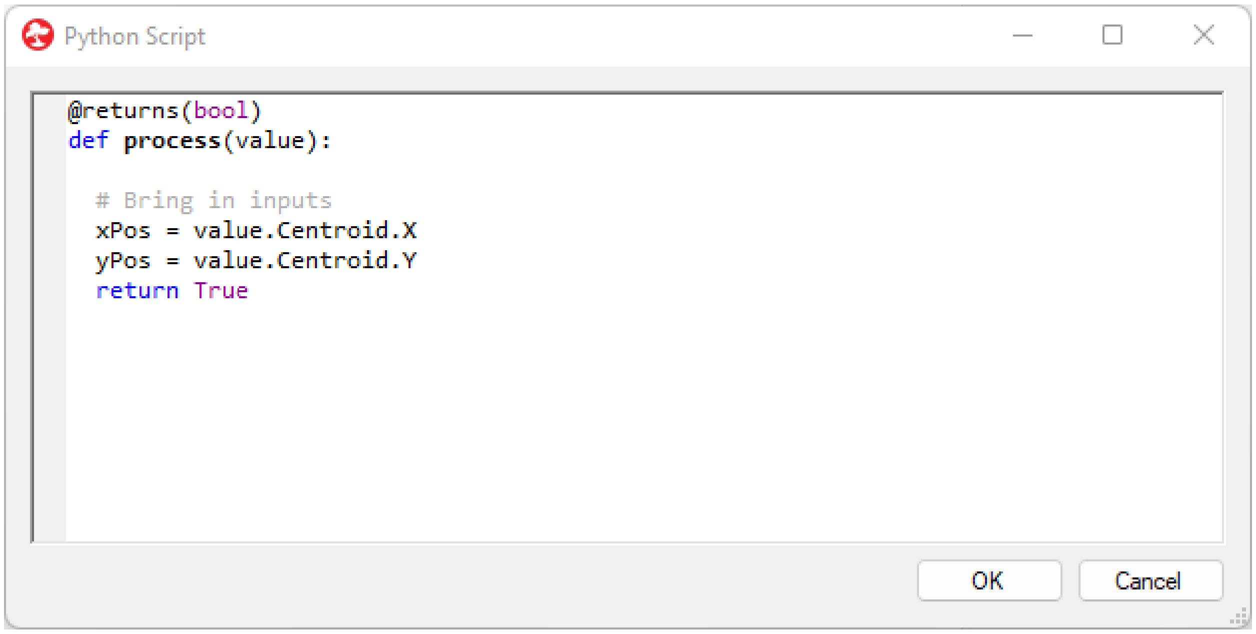
```
# Initiallize Global Variables
totalTime = 0.0   # Total time spent in Chamber 2
prevTime = -1.0    # Timestamp of previous frame
prevInChamberTwo = False # Whether or not the animal was in Chamber 2 the
previous frame


@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
```

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".

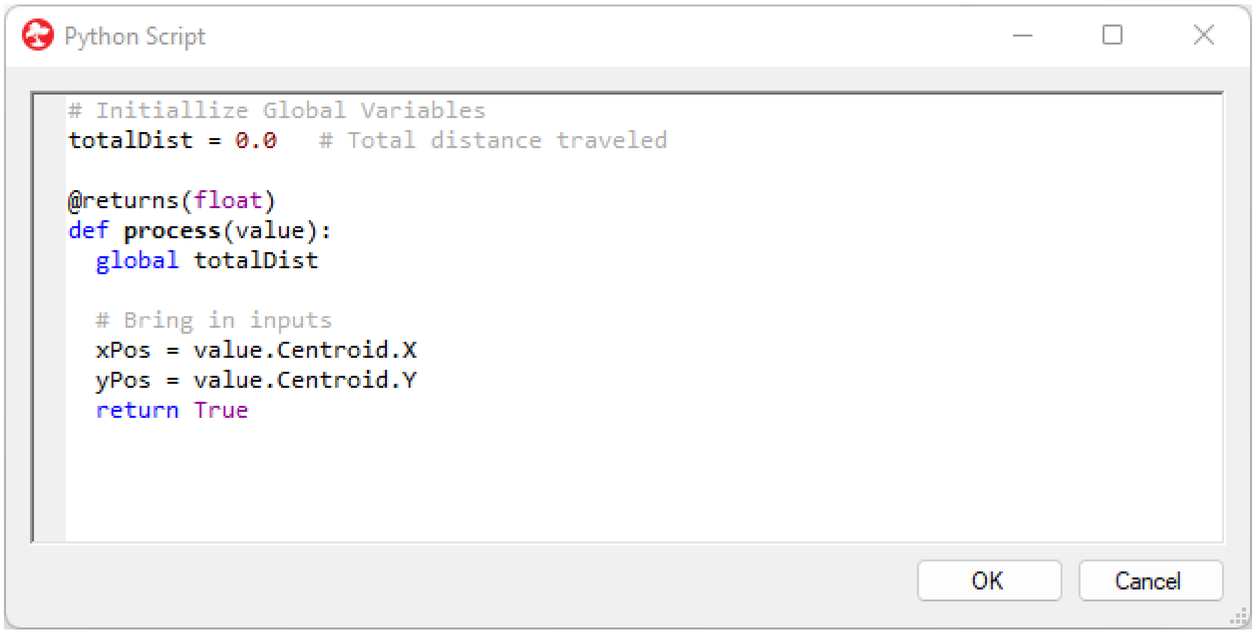


Begin the “Distance Traveled” script by bringing in the X and Y coordinate information from the Centroid input.



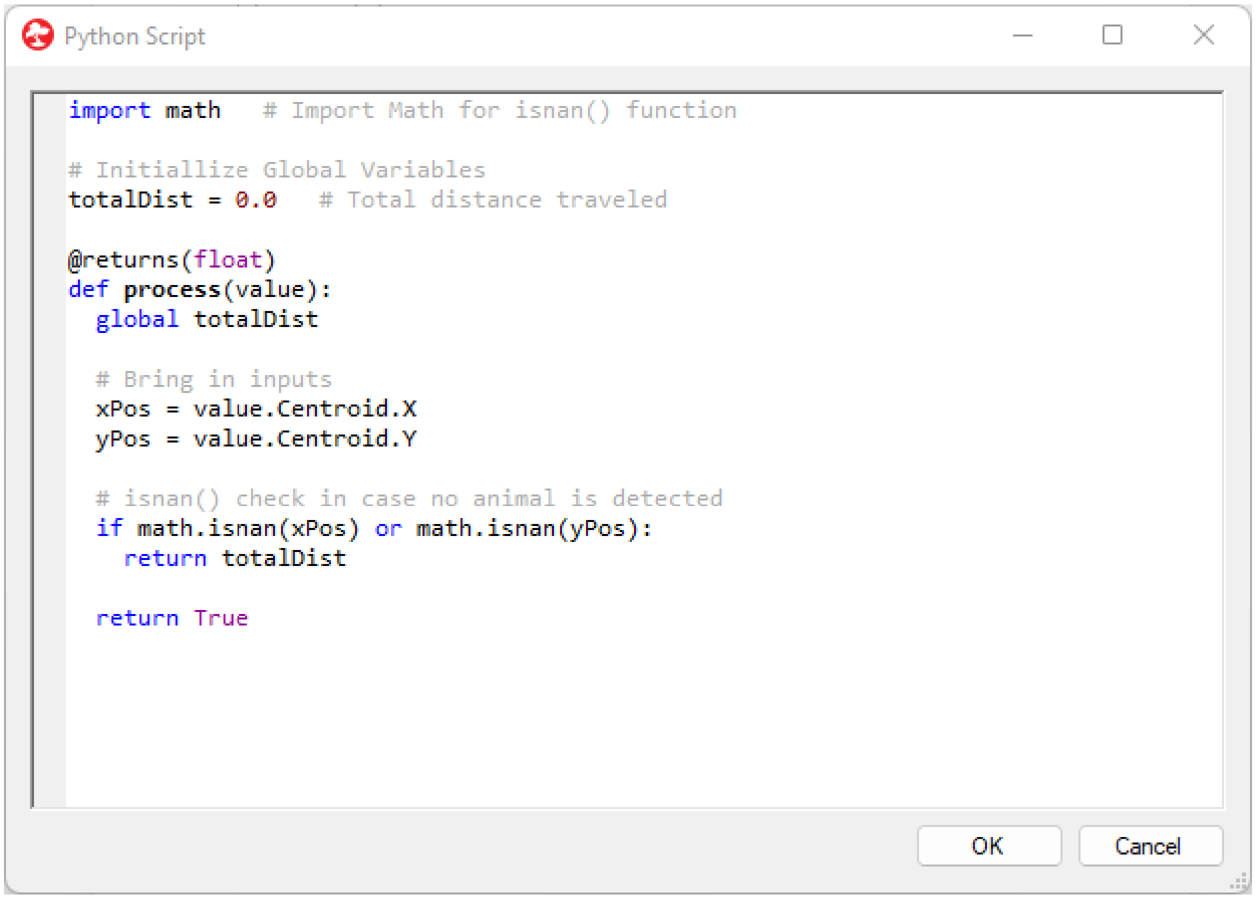
```
@returns(bool)
def process(value):

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

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.



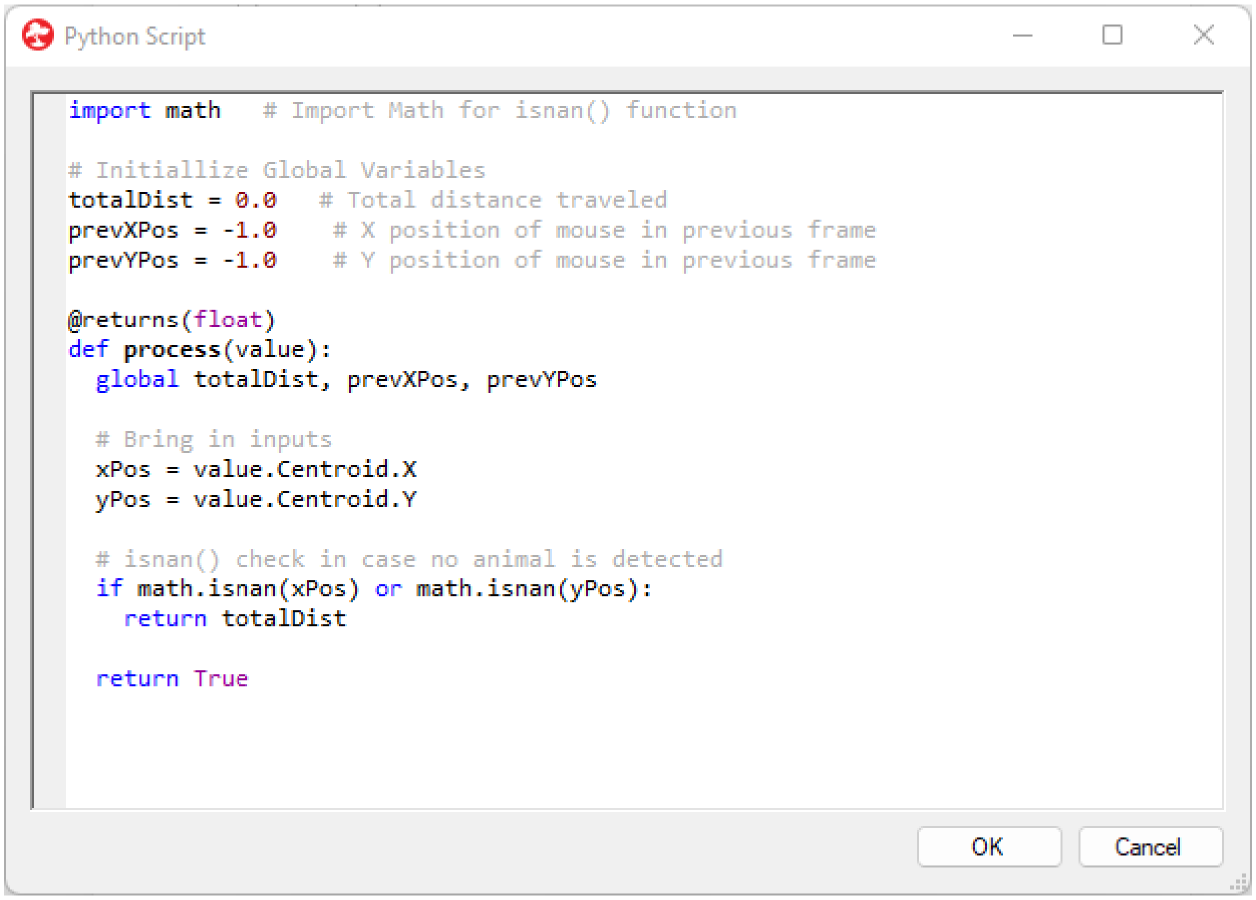
```
# Initiallize Global Variables
totalDist = 0.0    # Total distance traveled

@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.



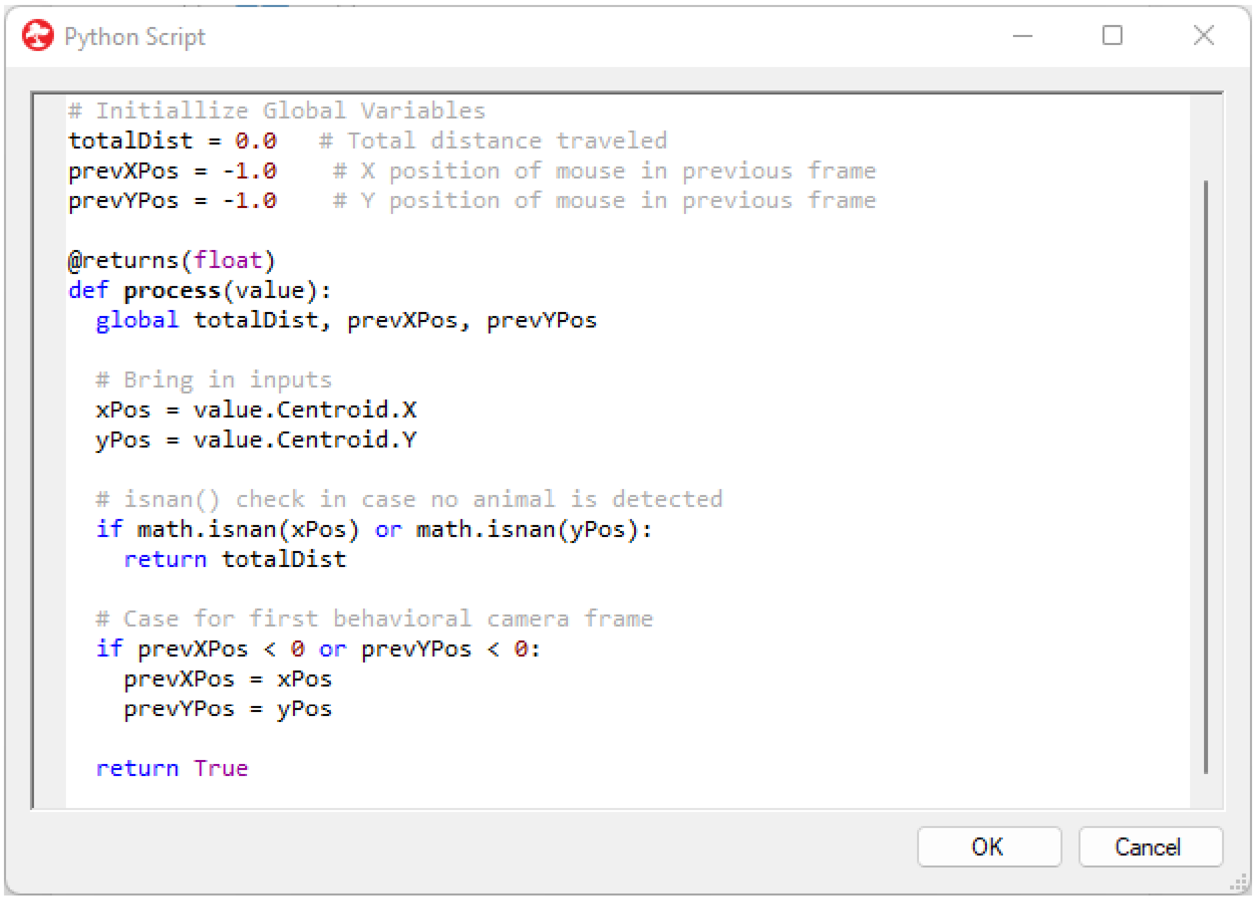
```
import math    # Import Math for isnan() function

# 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

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

  return True
```

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.



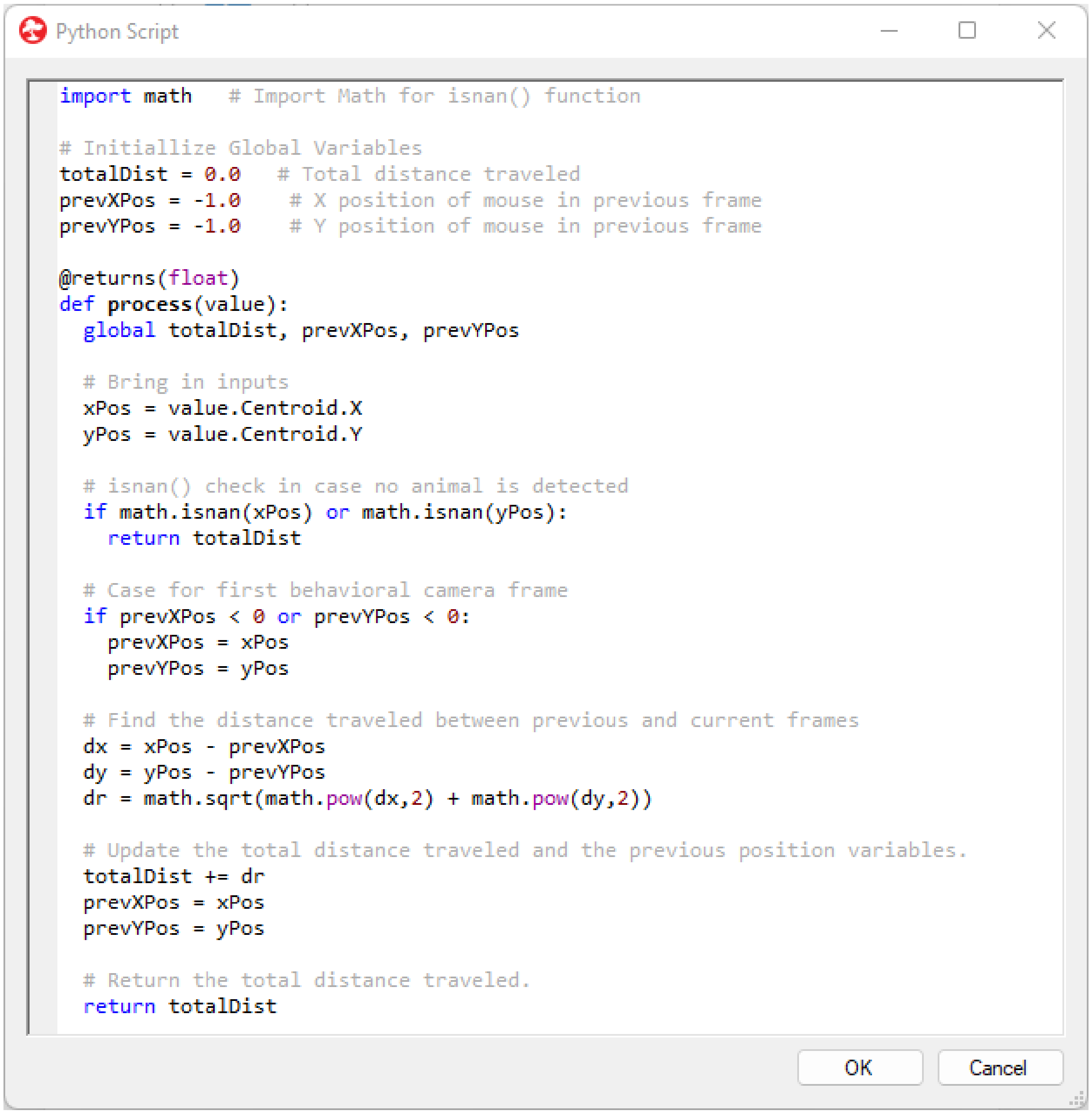
```
import math   # Import Math for isnan() function

# 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
```

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.



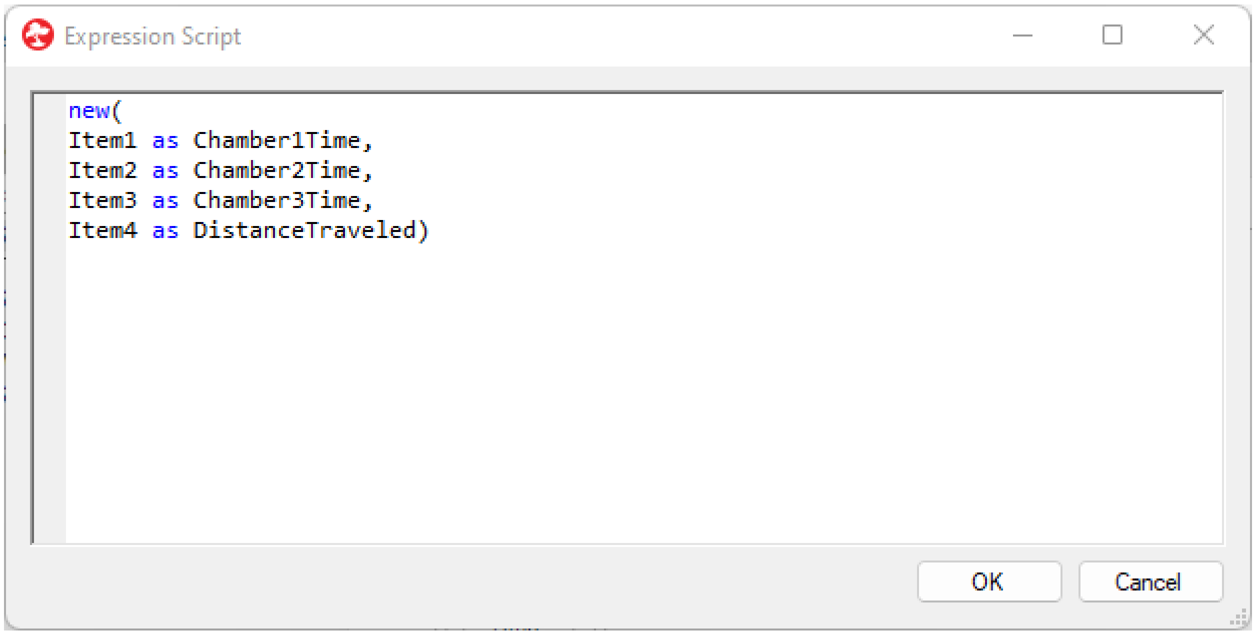
```python
# Initiallize Global Variables
totalDist = 0.0   # Total distance traveled
prevXPos = -1.0    # X position of mouse in previous frame
prevYPos = -1.0    # Y position of mouse in previous frame

@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
```

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.



```
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
```

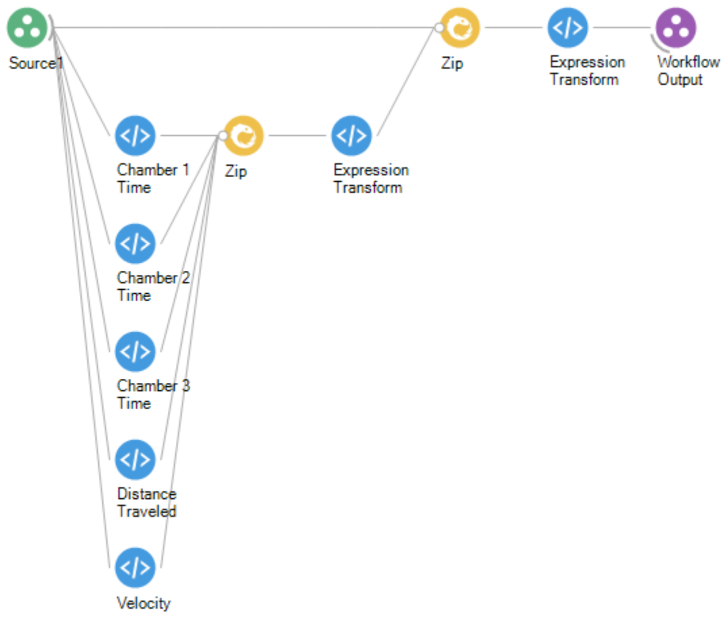
With that, the distance traveled algorithm is complete, be sure to update the “Expression Transform” that names the elements coming from the analysis algorithms.



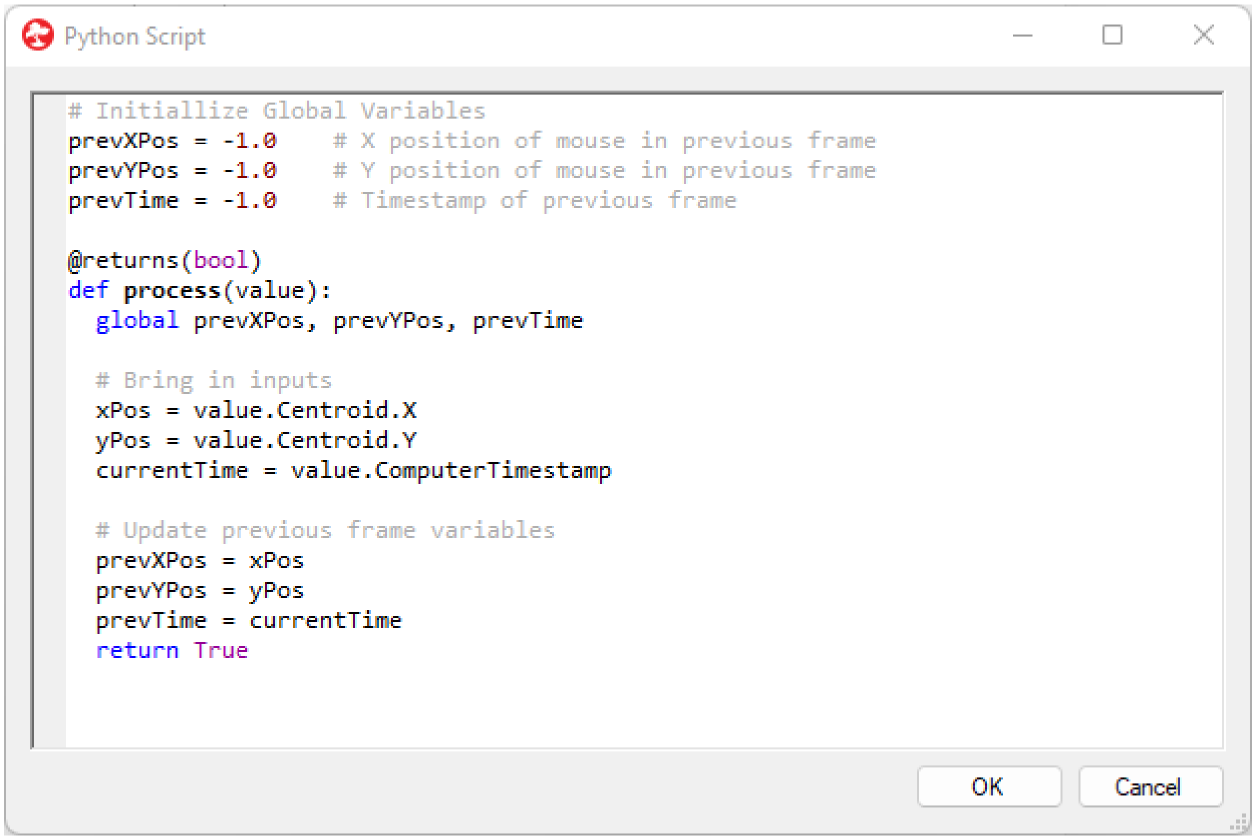
```
new(
Item1 as Chamber1Time,
Item2 as Chamber2Time,
Item3 as Chamber3Time,
Item4 as DistanceTraveled)
```

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”.

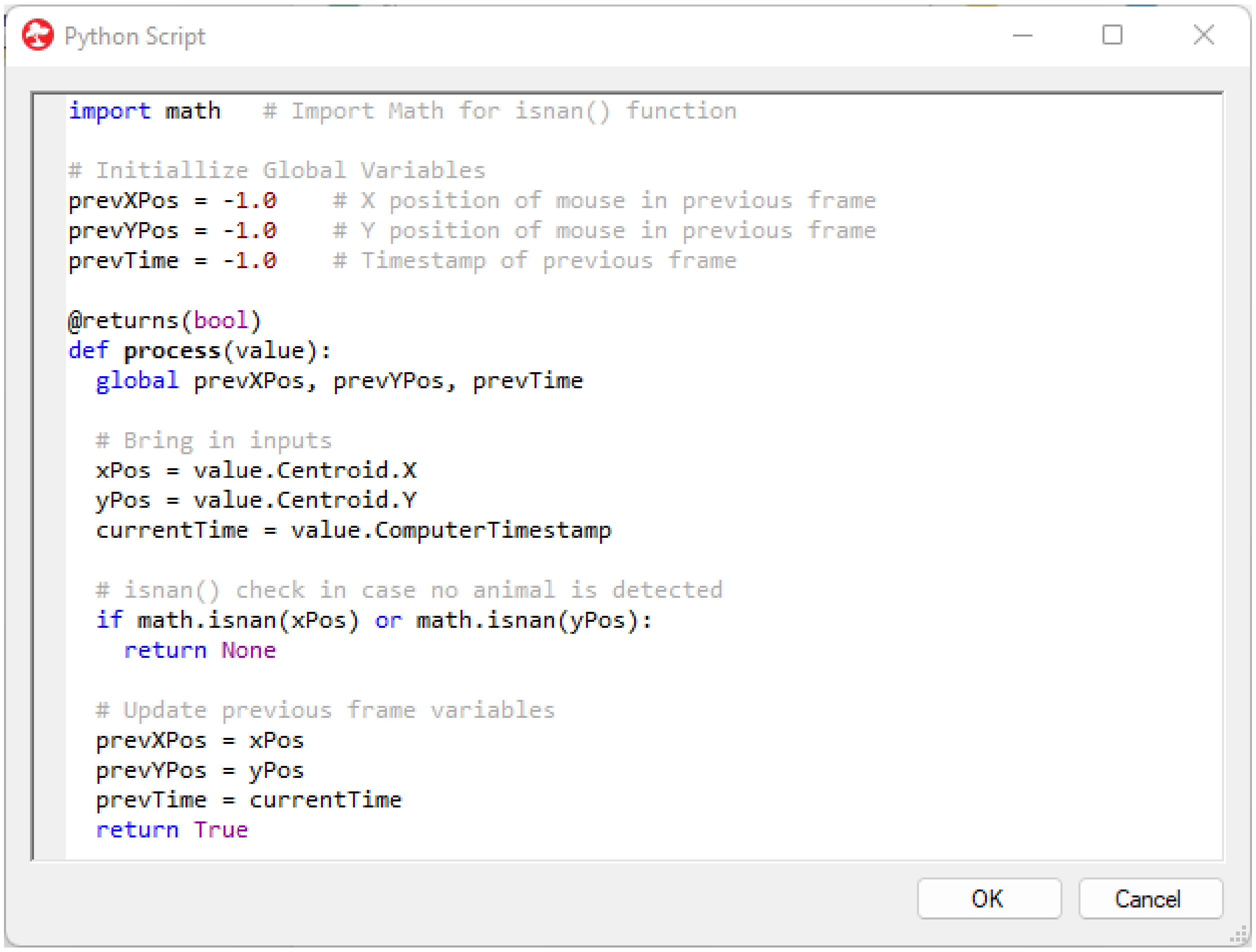


For the velocity algorithm, we will need to bring in the input values containing the X and Y position of the animal as well as the computer timestamp of the current frame. We will also use global variables to contain all of these values for the previous frame.



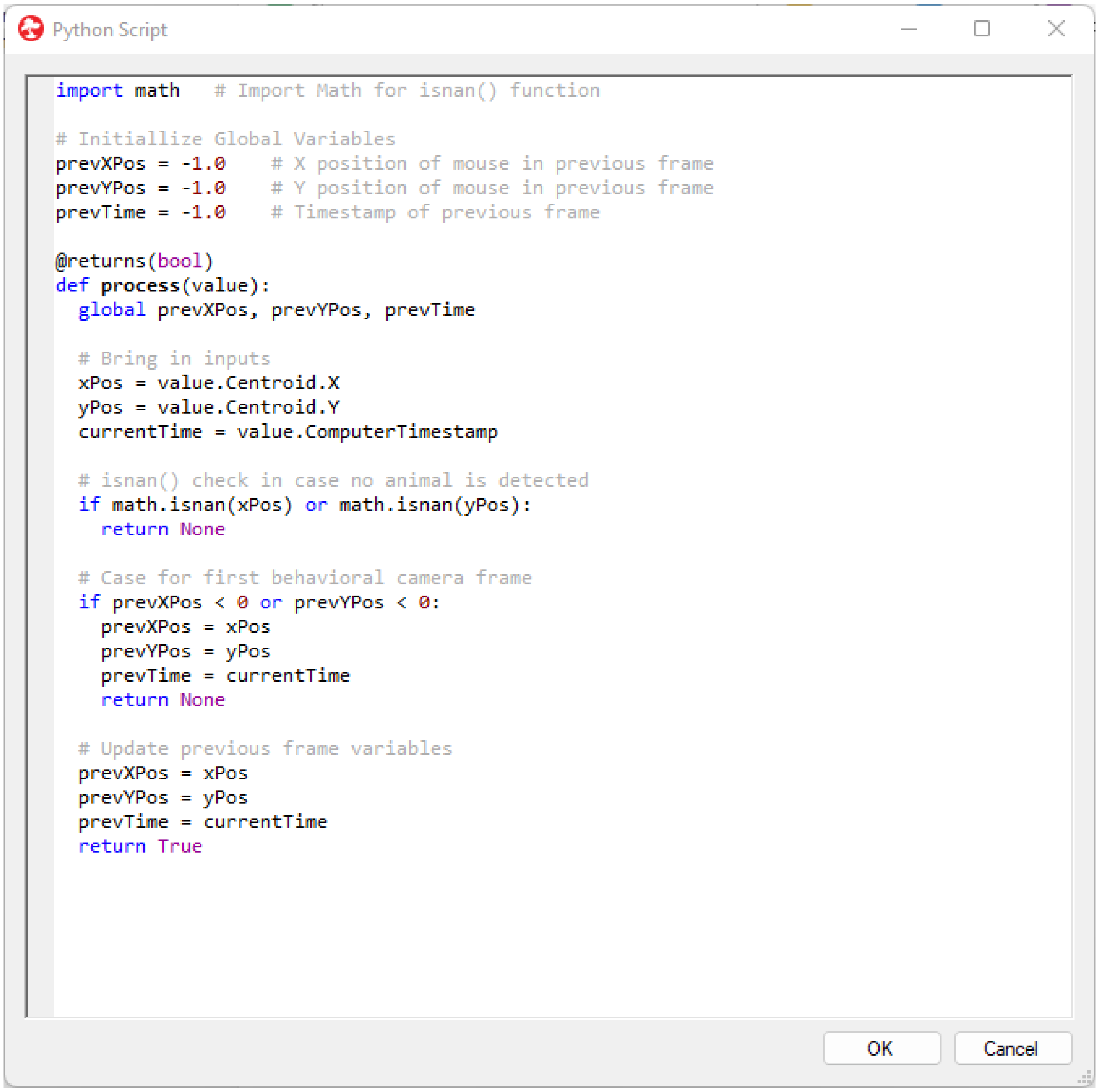
```
# Initiallize Global Variables
prevXPos = -1.0     # X position of mouse in previous frame
prevYPos = -1.0     # Y position of mouse in previous frame
prevTime = -1.0     # Timestamp of previous frame

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

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

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

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.



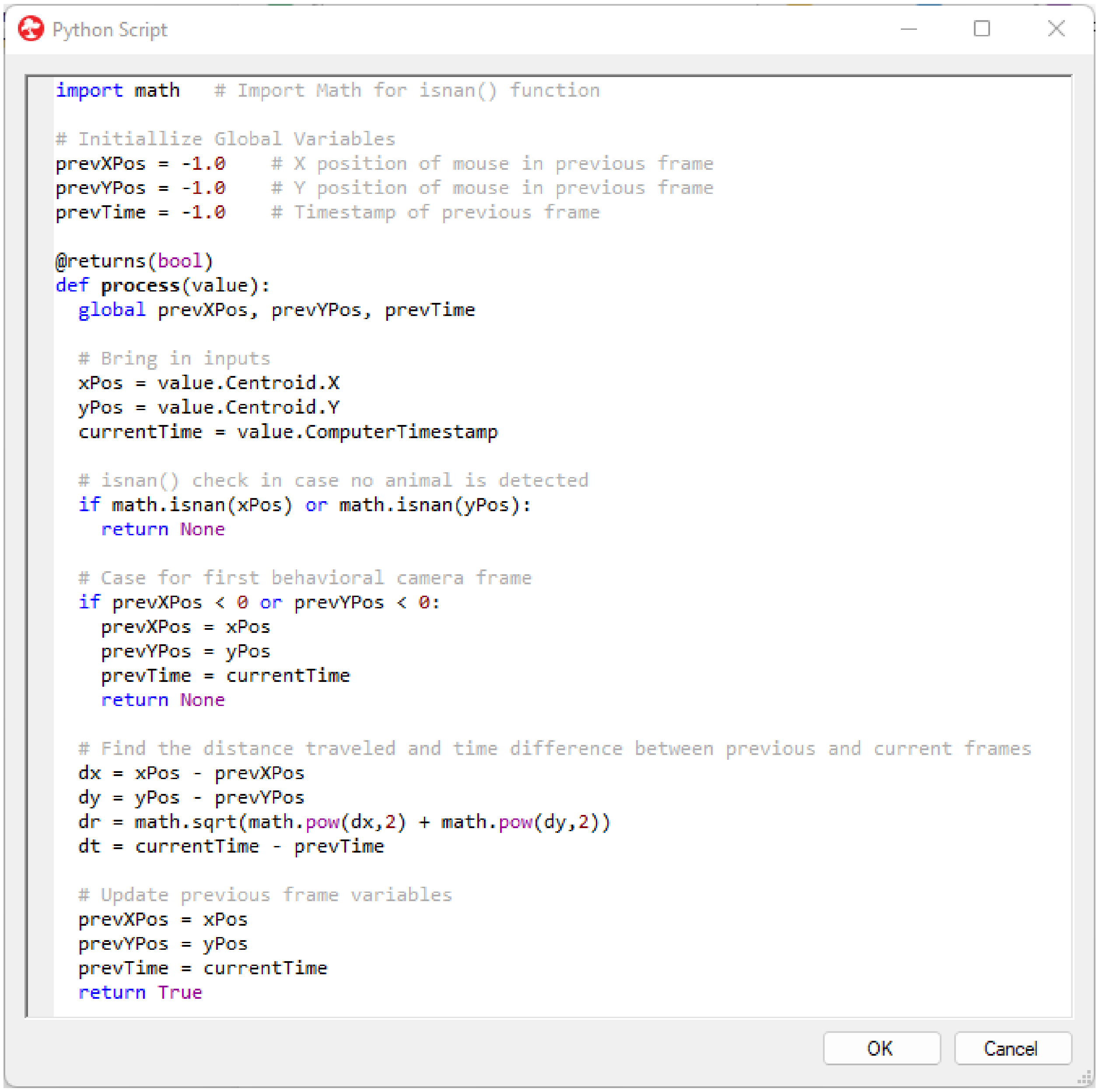
```
import math   # Import Math for isnan() function

# 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

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

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.



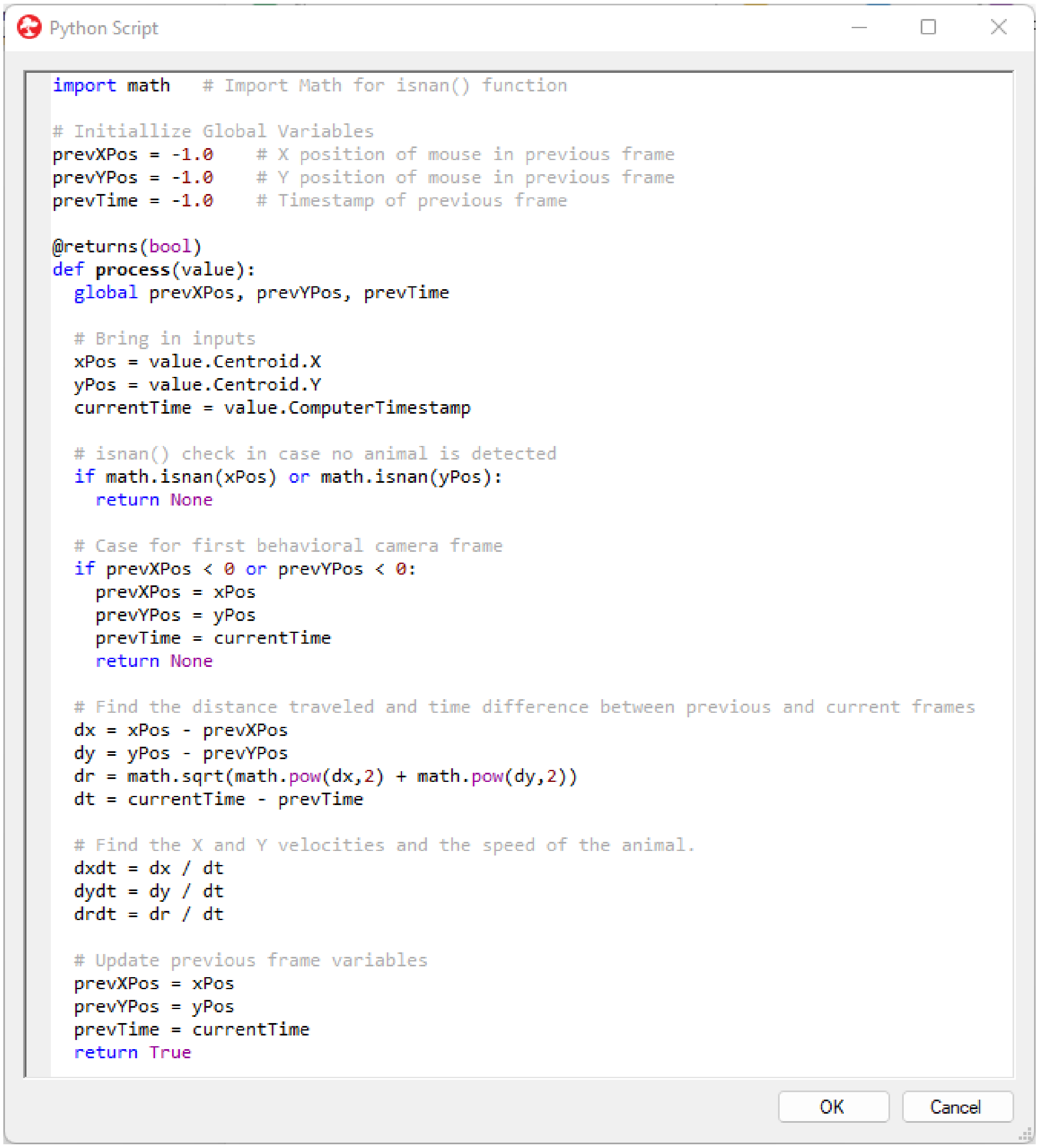
```python
import math   # Import Math for isnan() function

# 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.



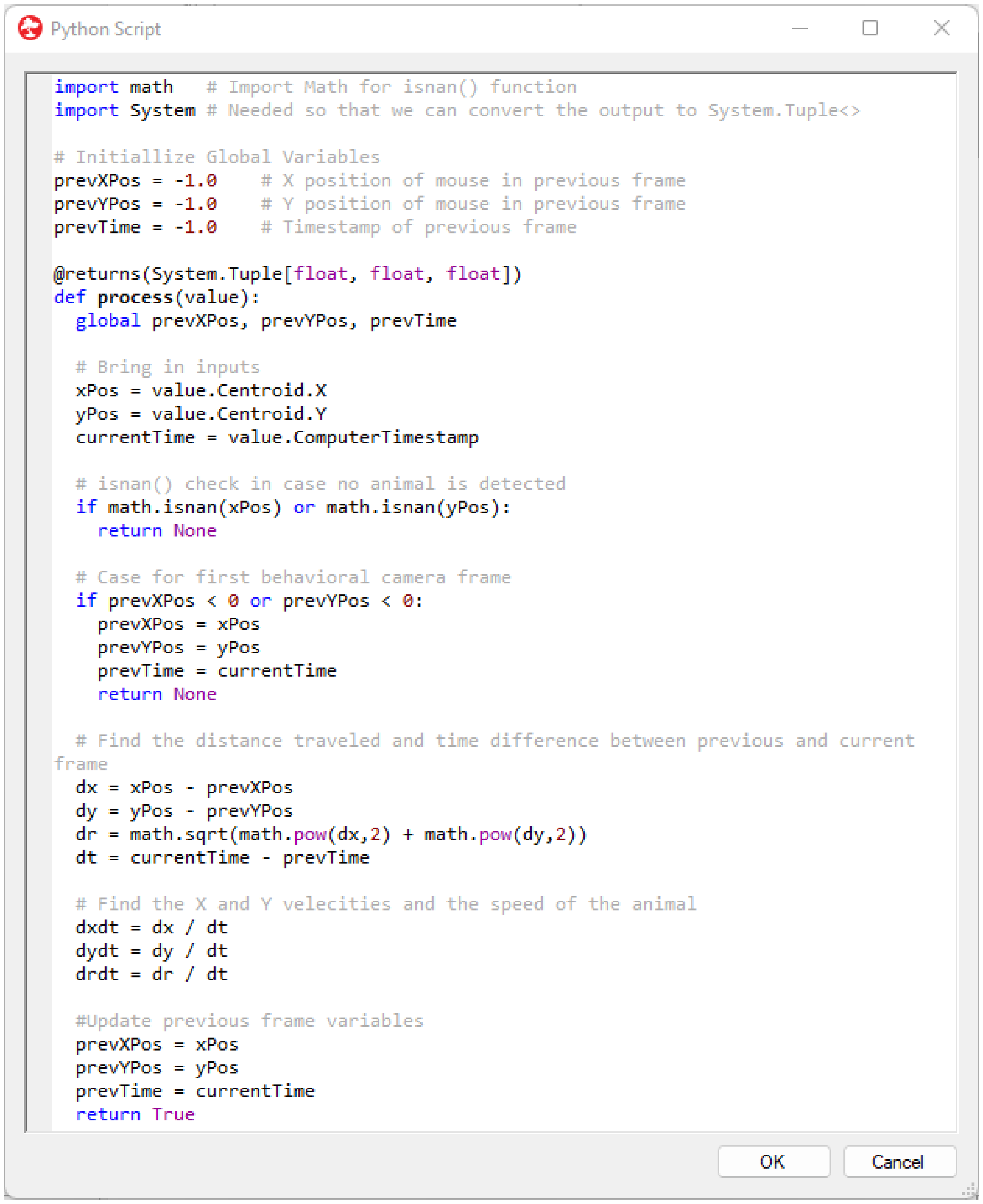
```
import math   # Import Math for isnan() function

# 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”.



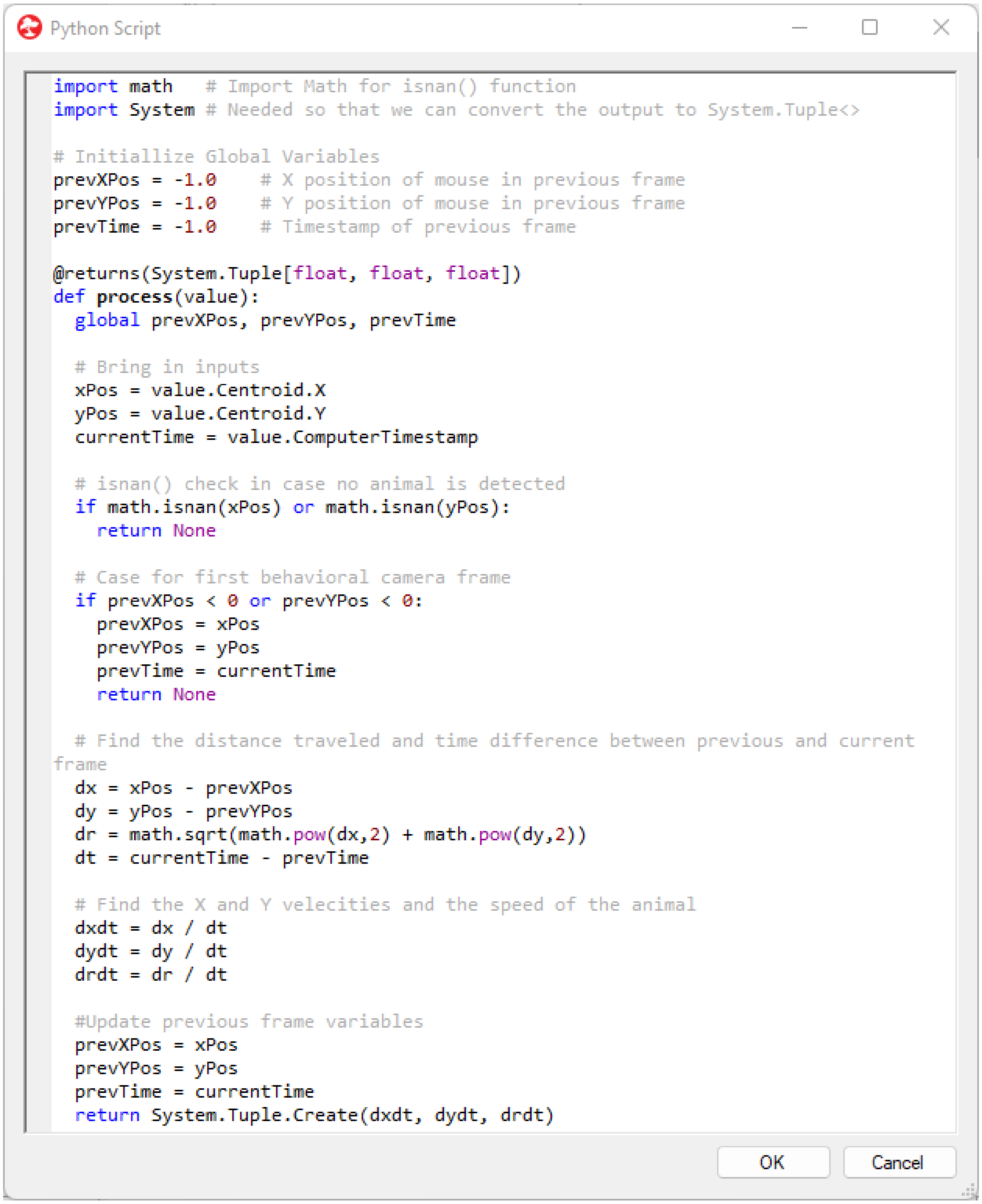
```python
import math   # Import Math for isnan() function

# 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

  # 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.



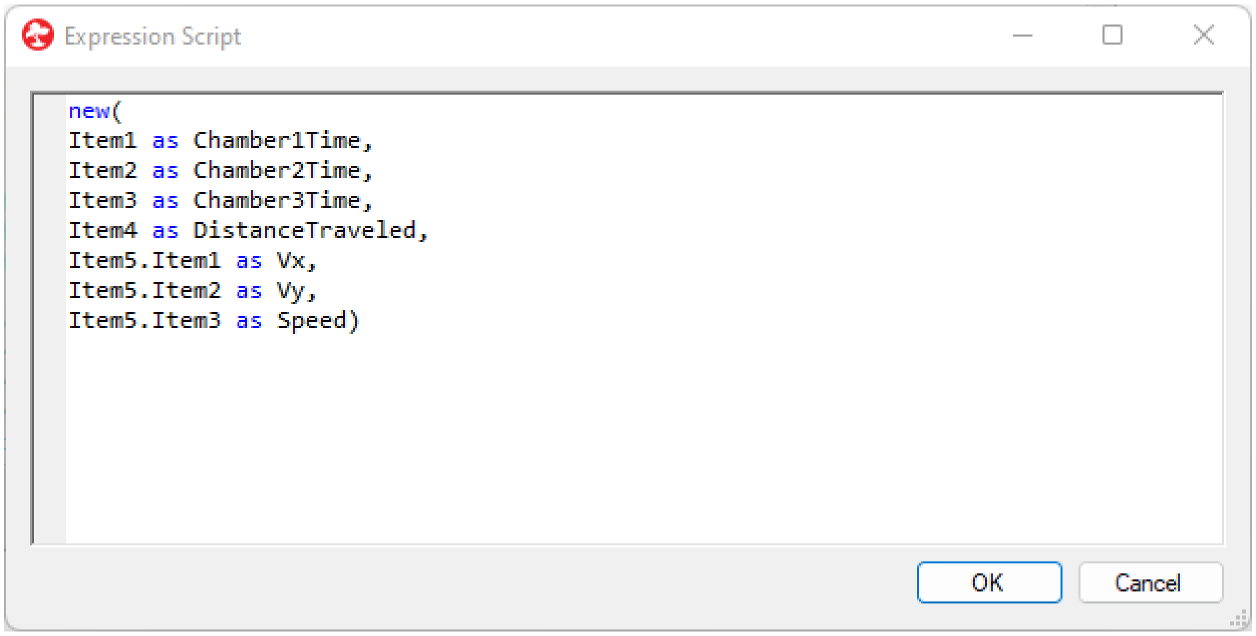
```
import math   # Import Math for isnan() function
import System # Needed so that we can convert the output to System.Tuple<>

# 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(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 velecities 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
```

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



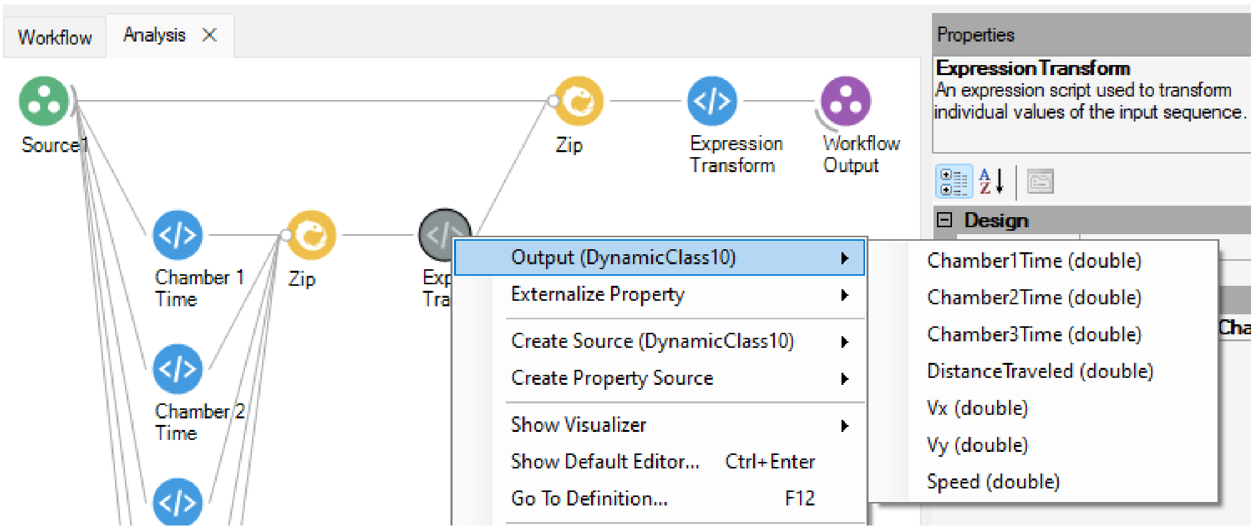
```
import math   # Import Math for isnan() function
import System # Needed so that we can convert the output to System.Tuple<>

# 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(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 velecities 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.


Expression Script

```
new(
Item1 as Chamber1Time,
Item2 as Chamber2Time,
Item3 as Chamber3Time,
Item4 as DistanceTraveled,
Item5.Item1 as Vx,
Item5.Item2 as Vy,
Item5.Item3 as Speed)
```

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.



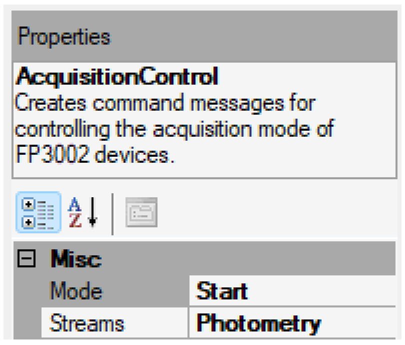
Expression Transform
An expression script used to transform individual values of the input sequence.

Chamber1Time (double)
Chamber2Time (double)
Chamber3Time (double)
DistanceTraveled (double)
Vx (double)
Vy (double)
Speed (double)

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.


Properties

AcquisitionControl
Creates command messages for controlling the acquisition mode of FP3002 devices.

Misc

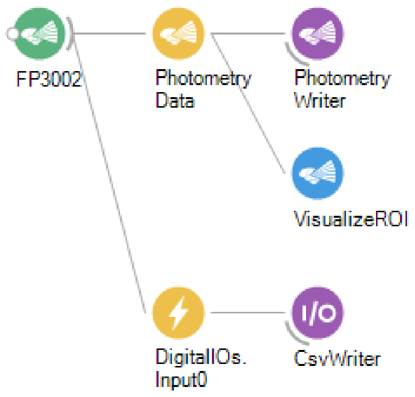
Mode	**Start**
Streams	**Photometry**

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.

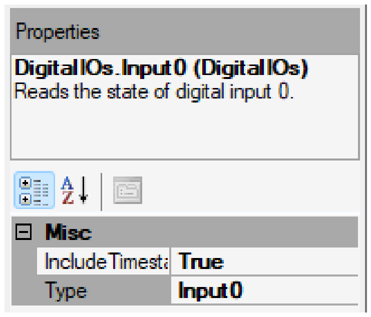
This node can be used in a variety of experimental designs and examples of its implementation can be found throughout the “Data Acquisition” chapter.

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 two properties that must be specified: “Include Timestamp” and “Type”. The “Include Timestamp” property should almost always be set to “True” and will cause the output data type to be of type “Bonsai.Harp.Timestamped<bool>”. This data type is readily writable to storage so a “Csv Writer” can be connected directly after the node, saving a .csv file with two columns containing the system timestamp and the state of the port. When the “Include Timestamp” node is set to “False”, the output will be of type “boolean”. This boolean value should not be saved to storage without a timestamp since it will not be alignable to other saved datasets.

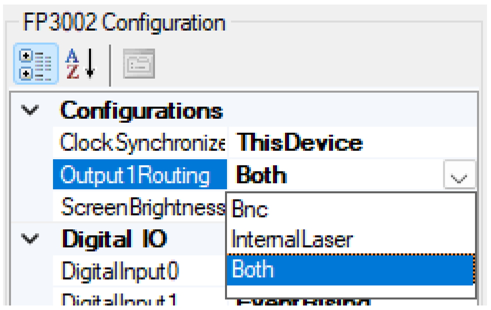

Properties

DigitalIOs.Input0 (DigitalIOs)
Reads the state of digital input 0.

Misc

IncludeTimest	**True**
Type	**Input0**

The “Type” property dictates which port the node will read the state of. You can select a specific port or you can select all the ports using the “State” value. This will either cause the output to be of type “byte” or “Bonsai.Harp.Timestamped<byte>” depending on if the “Include Timestamp” property is set to True or False. One common use for this node is to provide precise timestamps of the laser ON/OFF state. The data in the “Stimulation” column of the .csv file saved by the “Photometry Writer” only indicates if stimulation is occurring and does not indicate precisely when the laser changes state during stimulation. With some configuration, this node has this capability. First, be sure that the “Output 1 Routing” 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 “Include Timestamp” set to “True” and “Type” set to “Output 1”. 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.



FP3002 Configuration

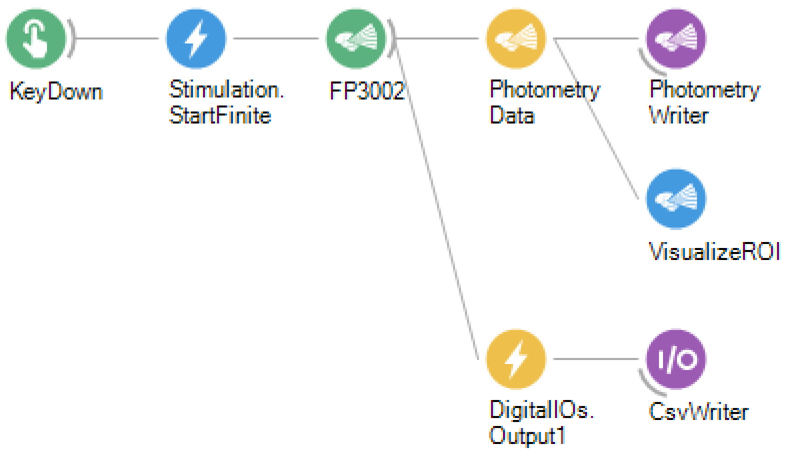
Configurations

ClockSynchronizer	ThisDevice
Output1Routing	Both
ScreenBrightness	

Bnc
InternalLaser
Both

Digital IO

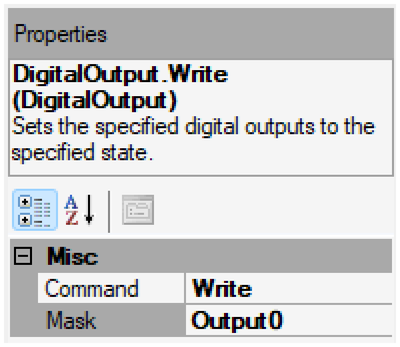
DigitalInput0	



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.


Properties

DigitalOutput.Write (DigitalOutput)
Sets the specified digital outputs to the specified state.

Misc

Command	Write
Mask	Output0

Below are descriptions of each “Command” and “Mask” property:

Commands:

<u>Set -</u> Sets the internal signal to HIGH
<u>Clears -</u> Clears the internal signal to LOW
<u>Toggle -</u> Toggles the internal signal to its opposite state.
<u>Write -</u> Sets the internal signal to HIGH when True is passed to the node, otherwise toggles the signal to LOW.

Commands:

<u>None -</u> Cause the command to be a null command, making no state changes.
<u>L415/L470/L560 -</u> Specifies the internal signal to be one of the LEDs.
<u>Output 0/1 -</u> Specifies the internal signal to be one of the digital output ports.
<u>Trigger -</u> Species the internal signal to be the trigger line of the internal camera, causing the camera to take another frame.
<u>Line 2/3 -</u> Species the internal signal to be the Camera GPIO lines 2 or 3.

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, frame flags, and activity data. The “FP3001” node possesses the following properties:

AutoCrop: A boolean value used to specify whether or not the camera will automatically crop the incoming image. When set to “True” the “FP3001” node will crop the image to the smallest rectangle bounding all of the drawn ROIs. This act of cropping the image allows the camera on the system to run at faster frequencies.

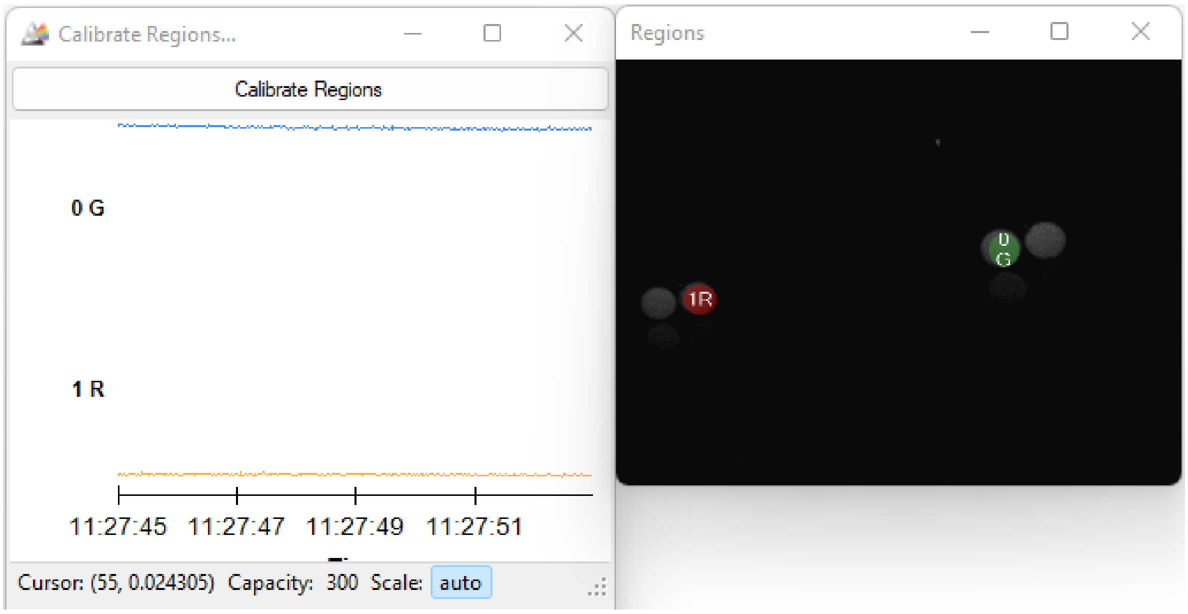
ExposureTime: An integer value used to specify the exposure time of the internal camera of the FP3001 system. This value must agree with the FPS set on the driver box such that the exposure time is at least one millisecond less than the period of data acquisition. For example, when the FPS is set to 40Hz, the period of data acquisition is equal to $Period = \frac{1}{40Hz} \star \frac{1000ms}{1sec} = 25ms$. In this 25ms duration, the camera must go through its exposure time and its dead time. The dead time must be at least 1ms which means in this 40Hz case, the exposure time is recommended to be 24ms.

Index: An integer value used to specify which connected spinnaker camera to connect. This property is used to ensure that the “FP3001” node correctly connects to the FP3001 system and not a behavioral camera.

Regions: A custom data type property used to store the dimensions and locations of the user-defined ROIs. This property is only used to store this data and not used to define the ROIs. In order to define the ROIs, double click the “FP3001” node while the workflow is stopped and the system is connected.

Serial Number: A string value used to specify the serial number of the camera to connect. Similar to the “Index” property, this property can be used to ensure that the “FP3001” node correctly connects to the FP3001 system and not to a behavioral camera.

Trigger Mode: A dropdown menu used to specify the trigger sequence of the FP3001 system. This must agree with the driver box in order for the "FP3001" to assign the correct frame flags to the photometry data frames. In order to configure the FP3001 node be sure to specify the properties mentioned above. Then, double click the FP3001 node to open a calibration window. Begin data acquisition on the driver box and 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. To draw ROIs, click the "Calibrate Regions" button to open a new window containing the camera image that you can draw regions by left clicking and dragging.

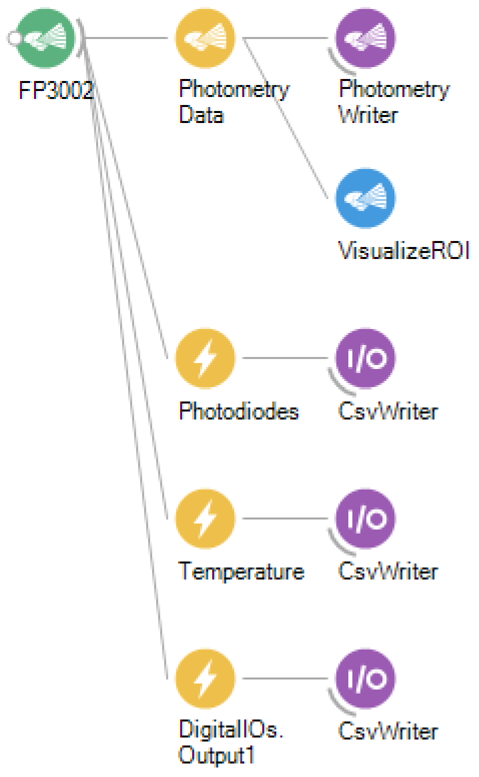


Cursor: (55, 0.024305) Capacity: 300 Scale: auto

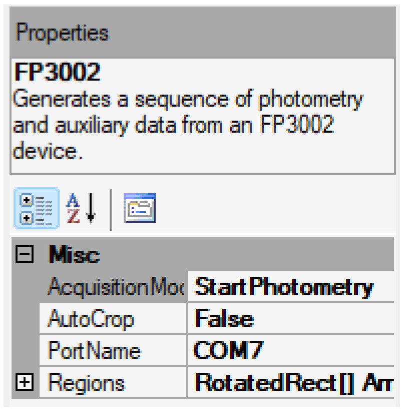
Within the "Regions" window you can move already drawn ROIs by left clicking them and dragging them to a new position. You can resize an existing ROI by right clicking it and dragging. While interacting with the "Regions" window be cautious about stray clicks within the window as they will draw ROI that are too small to see. Whenever drawing ROIs, double check that the number of signals shown in the "Calibrate Regions…" window matches the desired number of drawn ROIs in the "Regions" window. 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 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 a Harp Message. This message is 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:


Properties

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

Misc

AcquisitionMoc	StartPhotometry
AutoCrop	False
PortName	COM7
Regions	RotatedRect[] Arr

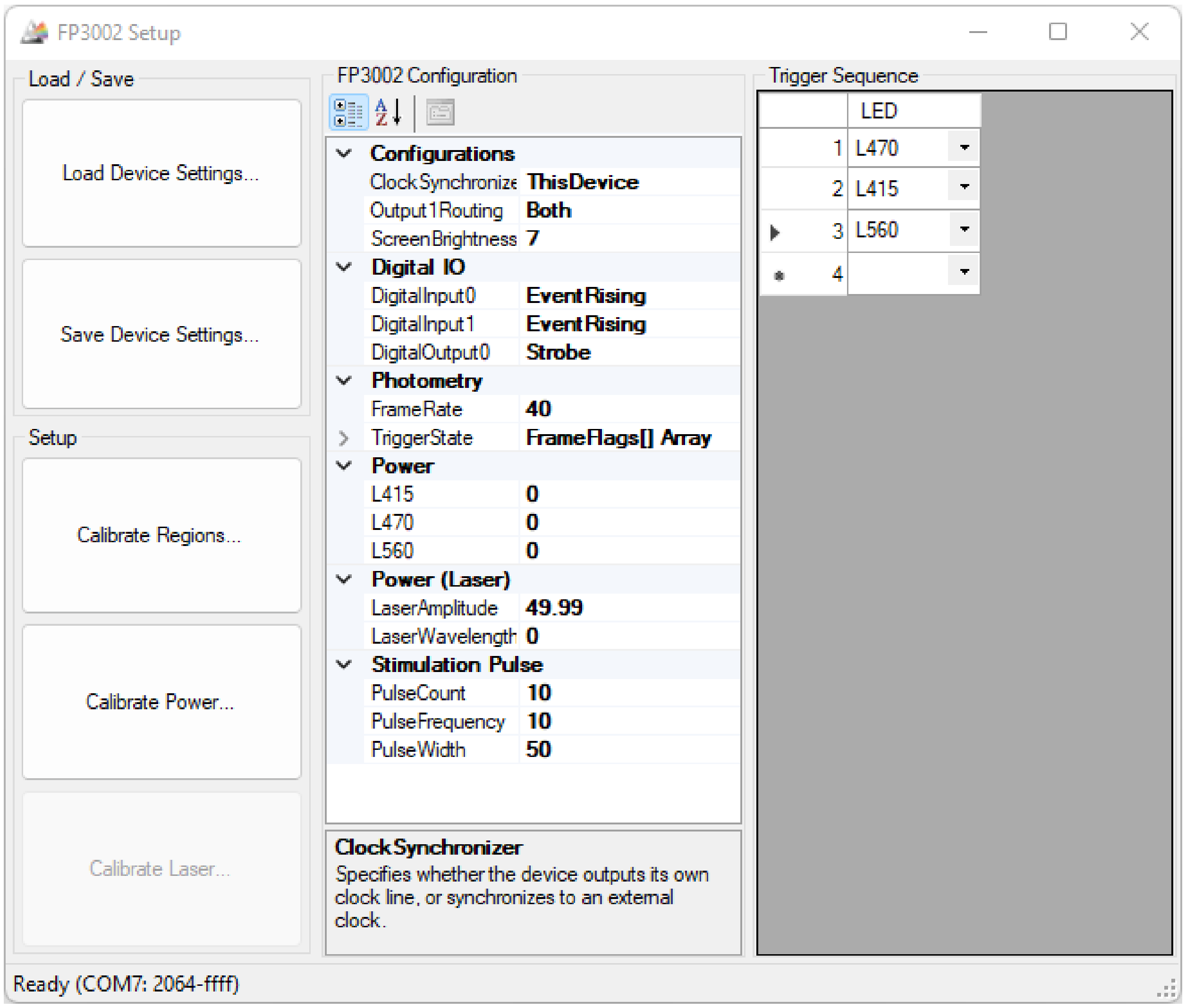
Acquisition Mode: A dropdown menu used to specify the initial state of data acquisition when the Bonsai workflow starts. When the workflow is started this is the first command that the “FP3002” node sends to the system. In most cases this command will be to either start or stop the photometry data acquisition. However, you can also use this to start or stop an external camera connected to the FP3002 system.

Auto Crop: A boolean value used to specify whether or not the camera will automatically crop the incoming image. When set to “True” the “FP3002” node will crop the image to the smallest rectangle bounding all of the drawn ROIs. This act of cropping the image allows the camera on the system to run at faster frequencies.

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.

Regions: A custom data type property used to store the dimensions and locations of the user-defined ROIs. This property is only used to store this data and not used to define the ROIs.

The “FP3002” node has a variety of configuration properties that are accessible through the “FP3002 Setup” window. To access these configuration properties, double click the “FP3002” node while the workflow is stopped and the system is connected and powered on.



Configurations	
ClockSynchronize	ThisDevice
Output1Routing	Both
ScreenBrightness	7
Digital IO	
DigitalInput0	EventRising
DigitalInput1	EventRising
DigitalOutput0	Strobe
Photometry	
FrameRate	40
TriggerState	FrameFlags[] Array
Power	
L415	0
L470	0
L560	0
Power (Laser)	
LaserAmplitude	49.99
LaserWavelength	0
Stimulation Pulse	
PulseCount	10
PulseFrequency	10
PulseWidth	50

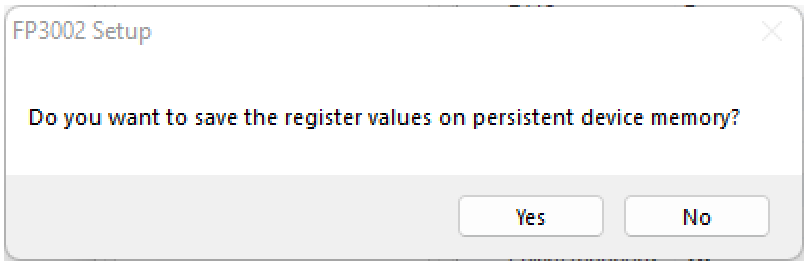
ClockSynchronizer
Specifies whether the device outputs its own clock line, or synchronizes to an external clock.

	LED
1	L470
2	L415
3	L560
4	

Ready (COM7: 2064-ffff)

This “FP3002 Setup” window contains all of the configurable system settings and a variety of tools used to calibrate, save and load these settings. Beginning with the organization of this window, the “FP3002 Setup” window consists of four primary panels: “Load / Save”, “Setup”, “FP3002 Configuration”, and “Trigger Sequence”.

Load / Save:
This panel is located in the top left corner of the window and is used to load and save FP3002 configuration settings. Once the FP3002 system is configured for a particular experiment, you can click the “Save Device Settings” button to save these settings to an XML file. When complete a new window will appear asking whether or not you want to “save the register values on persistent device memory”.

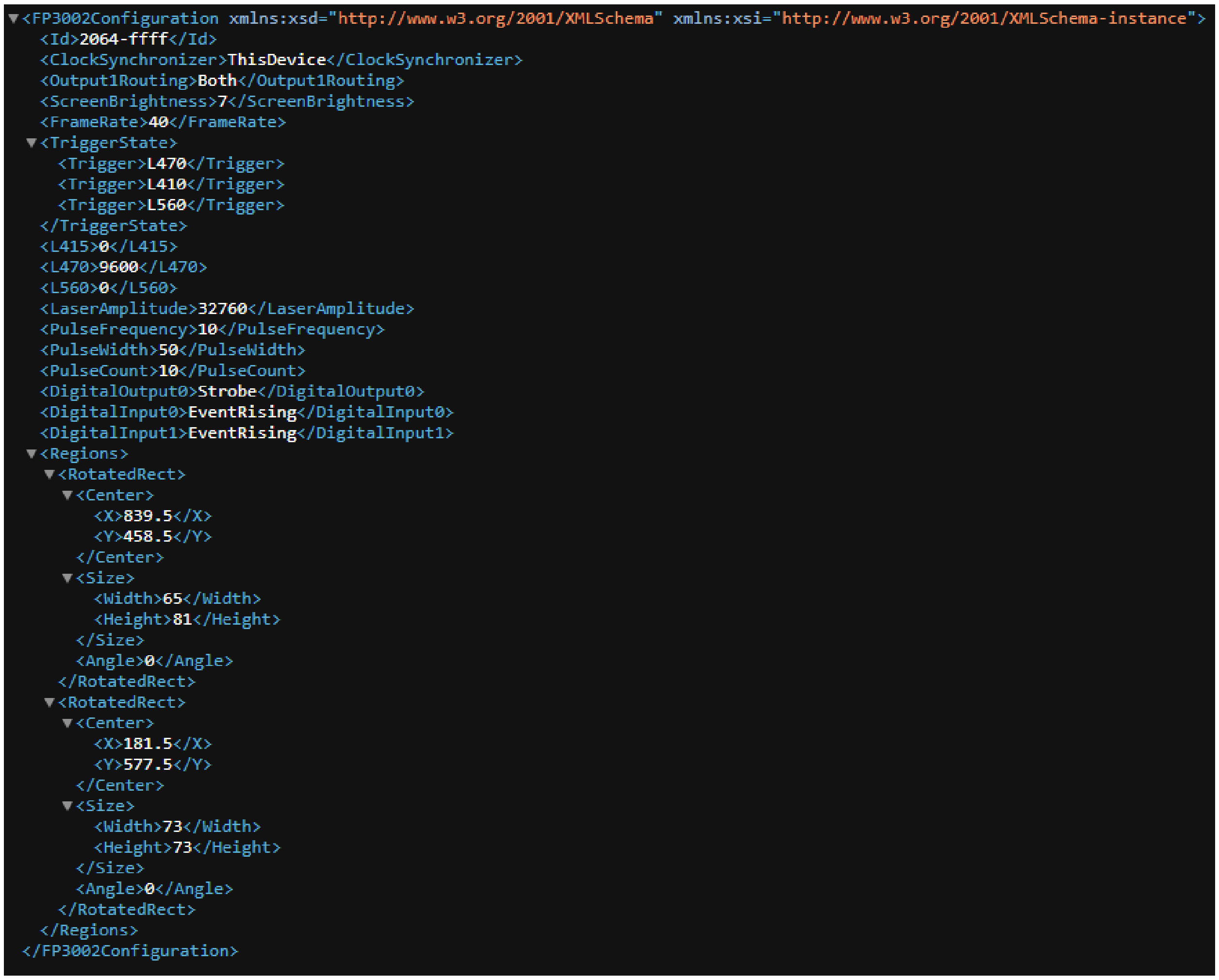

FP3002 Setup

Do you want to save the register values on persistent device memory?

If you select “Yes” here, the system settings will not only be saved to an XML file on your computer but will also be saved to the FP3002 system itself. This way, the system will continue to have these settings after disconnecting and power cycling the system. Selecting “No” will cause only the XML file to be saved to the computer and the system will continue to have its default settings after a disconnect and power cycle.

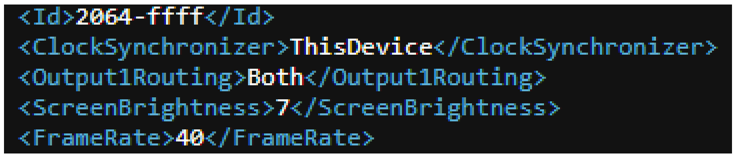
If an XML file containing the system settings for a particular experiment has already been previously saved, you can use the “Load Device Settings” button to load the settings into the “FP3002” node and onto the system itself.

The XML file itself can be referenced during post-hoc analysis to verify the system settings used during a particular experiment.



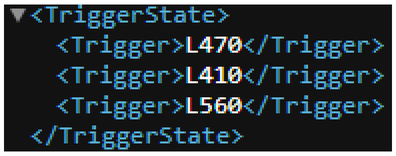
```
▼<FP3002Configuration xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <Id>2064-ffff</Id>
   <ClockSynchronizer>ThisDevice</ClockSynchronizer>
   <Output1Routing>Both</Output1Routing>
   <ScreenBrightness>7</ScreenBrightness>
   <FrameRate>40</FrameRate>
 ▼<TriggerState>
     <Trigger>L470</Trigger>
     <Trigger>L410</Trigger>
     <Trigger>L560</Trigger>
   </TriggerState>
   <L415>0</L415>
   <L470>9600</L470>
   <L560>0</L560>
   <LaserAmplitude>32760</LaserAmplitude>
   <PulseFrequency>10</PulseFrequency>
   <PulseWidth>50</PulseWidth>
   <PulseCount>10</PulseCount>
   <DigitalOutput0>Strobe</DigitalOutput0>
   <DigitalInput0>EventRising</DigitalInput0>
   <DigitalInput1>EventRising</DigitalInput1>
 ▼<Regions>
   ▼<RotatedRect>
     ▼<Center>
         <X>839.5</X>
         <Y>458.5</Y>
       </Center>
     ▼<Size>
         <Width>65</Width>
         <Height>81</Height>
       </Size>
       <Angle>0</Angle>
     </RotatedRect>
   ▼<RotatedRect>
     ▼<Center>
         <X>181.5</X>
         <Y>577.5</Y>
       </Center>
     ▼<Size>
         <Width>73</Width>
         <Height>73</Height>
       </Size>
       <Angle>0</Angle>
     </RotatedRect>
   </Regions>
 </FP3002Configuration>
```

This XML file contains all of the system settings, however some of them are stored in a way that is easier for the computer to understand but harder for the user to understand. The first few lines of this file contain the system's serial number labeled as "Id", "Clock Synchronizer", "Output 1 Routing", "Screen Brightness", and "Frame Rate" in Hertz. These are displayed the same way as in the "FP3002 Setup" window.



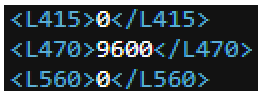
```
<Id>2064-ffff</Id>
<ClockSynchronizer>ThisDevice</ClockSynchronizer>
<Output1Routing>Both</Output1Routing>
<ScreenBrightness>7</ScreenBrightness>
<FrameRate>40</FrameRate>
```

Below that is the trigger sequence represented by a list called “Trigger State” where each item of the list is called a “Trigger” and informs which LED is to be triggered.



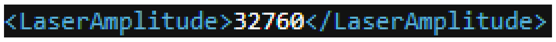
```
▼<TriggerState>
   <Trigger>L470</Trigger>
   <Trigger>L410</Trigger>
   <Trigger>L560</Trigger>
 </TriggerState>
```

Next is the power level of each LED. These values are stored as unsigned, 16-bit integers that range linearly from 9600 to 35200. The only deviation from this rule is that sometimes LED powers of 0% will appear as either 0 or 9600 here. In order to convert these values to a percent, like what is displayed in the “FP3002 Setup” window, use the following equation: $LED\ Power\ (\%)\ =\ \frac{LED\ Power\ (XML) - 9600}{35200 - 9600} \star\ 100\%.$



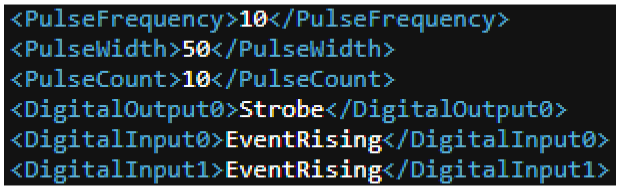
```
<L415>0</L415>
<L470>9600</L470>
<L560>0</L560>
```

Following the LED powers is the Laser amplitude. This value is stored as an unsigned, 16-bit integer that ranges linearly from 0 to 65535. In order to convert this value to a percent, like what is displayed in the “FP3002 Setup” window, use the following equation: $Laser\ Power\ (\%)\ =\ \frac{Laser\ Power\ (XML)}{65535} \star\ 100\%.$



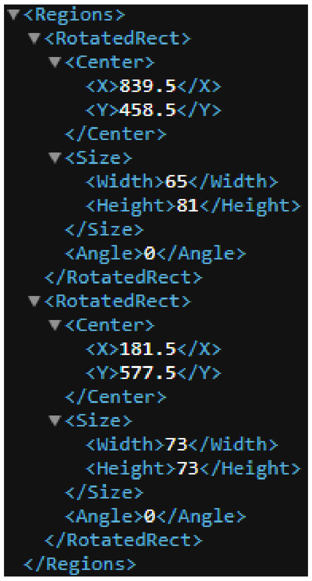
```
<LaserAmplitude>32760</LaserAmplitude>
```

From the Laser amplitude value, the next several values are stored the same way that they are displayed in the “FP3002 Setup” window. These values include the “Pulse Frequency” (in Hertz), “Pulse Width” (in milliseconds), “Pulse Count”, “Digital Output 0”, “Digital Input 0”, and “Digital Input 1”.



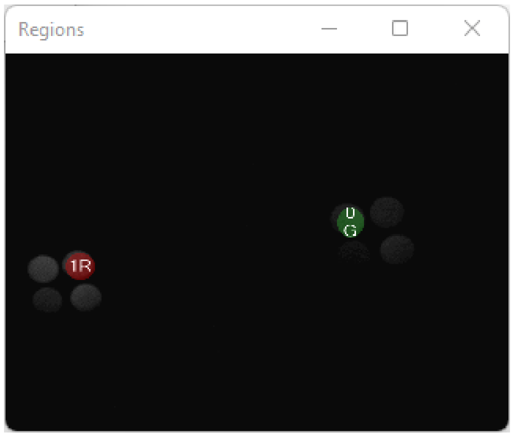
```
<PulseFrequency>10</PulseFrequency>
<PulseWidth>50</PulseWidth>
<PulseCount>10</PulseCount>
<DigitalOutput0>Strobe</DigitalOutput0>
<DigitalInput0>EventRising</DigitalInput0>
<DigitalInput1>EventRising</DigitalInput1>
```

The remainder of the information stored within the XML file relates to the ROIs used during the experiment. Each ROI will have its center point, width, height, and angle stored here. The units for the center point, width, and height are such that the x value describes the number of pixels from the left edge of the image and the y value describes the number of pixels from the top edge of the image. The angle value will always be zero since functionality for rotating ROIs does not exist.



```
▼<Regions>
  ▼<RotatedRect>
    ▼<Center>
        <X>839.5</X>
        <Y>458.5</Y>
      </Center>
    ▼<Size>
        <Width>65</Width>
        <Height>81</Height>
      </Size>
      <Angle>0</Angle>
    </RotatedRect>
  ▼<RotatedRect>
    ▼<Center>
        <X>181.5</X>
        <Y>577.5</Y>
      </Center>
    ▼<Size>
        <Width>73</Width>
        <Height>73</Height>
      </Size>
      <Angle>0</Angle>
    </RotatedRect>
  </Regions>
```

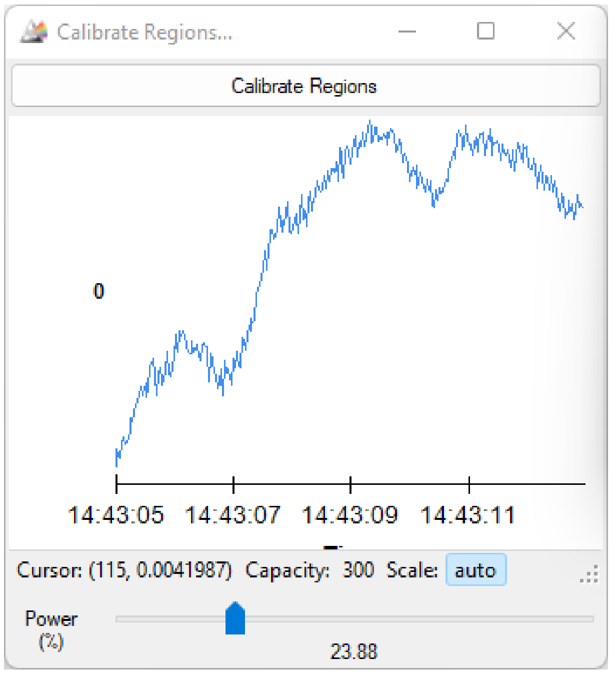
The order that the regions appear in the XML file matches the indices of the ROIs in the *.csv* file output by the “Photometry Writer” node and the indices displayed in the “Regions” window while drawing ROIs. So in the example above, since the first “RotatedRect” is on the right half of the image, it represents the ROI labeled “G0”. The second “RotatedRect” is on the left half of the image, and so it represents the ROI labeled “R1”.



Setup:
This panel is located in the bottom left corner of the window and is used to calibrate the ROIs, LED powers, and laser settings for stimulation. Below is information about how to use each of the tools found within the "Setup" panel:

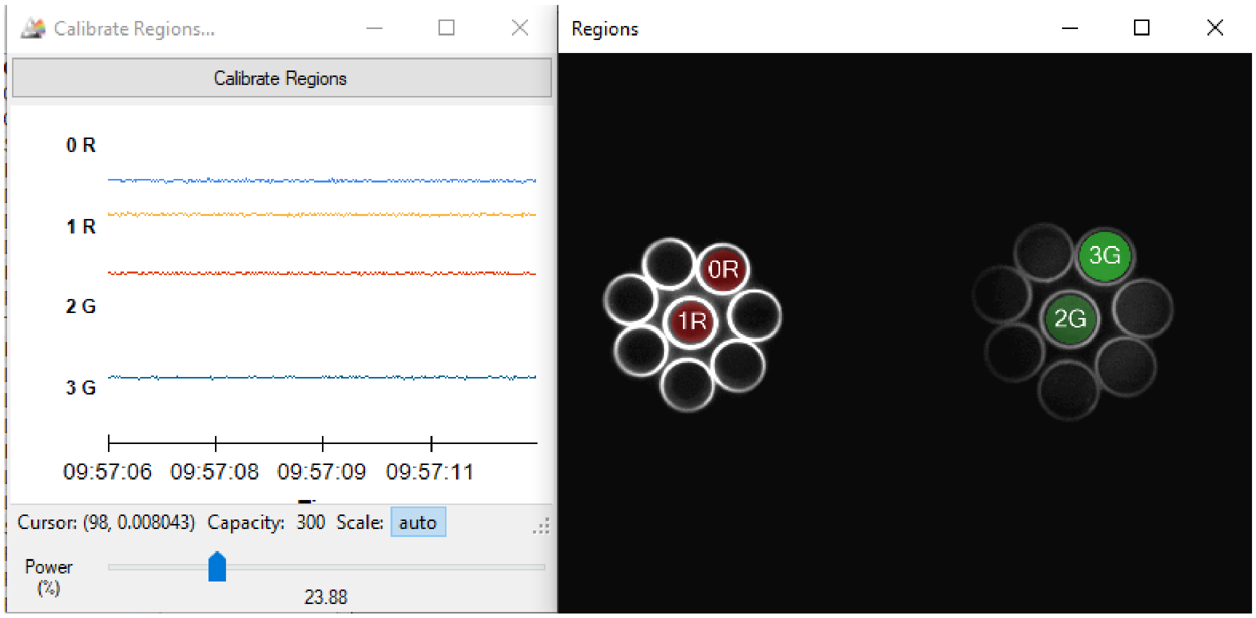
"Calibrate Regions" -

The "Calibrate Regions" tool is used to define the regions of interest (ROIs). These ROIs will be used in two ways. First, when the "Auto Crop" property of the "FP3002" node is set to "True", these regions will dictate the dimensions and location of the cropped image, allowing the FP3002 system to acquire data at faster frame rates. Second, these ROIs will indicate to the "FP3002" node which pixels are grouped together for averaging. Upon clicking the "Calibrate Regions" button, the system will begin to acquire at 40Hz with a trigger sequence consisting of only the 470nm LED. A "Calibrate Regions" window will open. This window consists of a button labeled "Calibrate Regions", a data visualizer, and a power scroll bar. Here, the "Calibrate Regions" button is used to open a new window called "Regions" that allows the user to draw ROIs onto the image. The data visualizer is a rolling plot containing the signal extracted from each ROI. If no ROIs are defined, then the signal will represent the average pixel value of the whole image. The power scroll bar is used to adjust the power of the L470 so that each fiber in the image is easily visible.



Cursor: (115, 0.0041987) Capacity: 300 Scale: auto

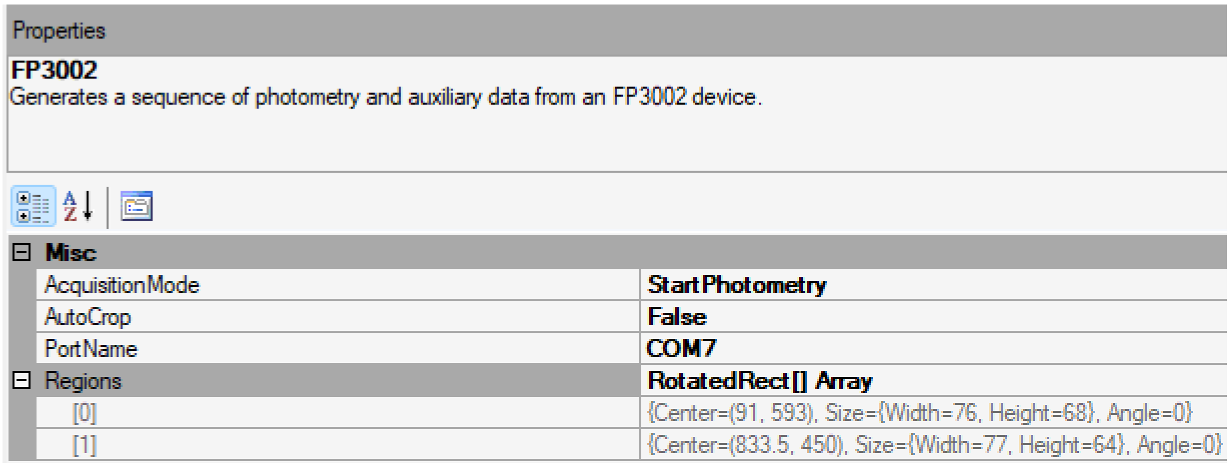
Once the “Regions” window is opened using the “Calibrate Regions” button found in the “Calibrate Regions” window, adjust the power scroll bar and focus the image using the translator on the system until the fibers are easily visible in the image. Then left click and drag to draw a new ROI. Move the new ROI by left clicking and dragging it to be centered on a fiber. Finally, resize the ROI by right clicking and dragging it to fit inside of the fiber. Whenever a new ROI is created, the data visualizer will update such that there is a signal for every existing ROI.



Cursor: (98, 0.008043) Capacity: 300 Scale: auto

During this process be cautious of stray clicks inside of the “Regions” window. Any click and drag on the window, no matter how small, will create a new ROI. That being said, it is possible to unknowingly create a new ROI that is barely visible and not easily selectable. Always verify that the number of signals found in the data visualizer matches the number of desired ROIs. If there are more signals than desired ROIs, then use the “Tab” key within the “Regions” window to cycle the selected ROI until the extra ROI is selected. Once selected use the “Del” key to delete it.

Once the ROIs are successfully drawn, close the “Regions” and “Calibrate Regions” window. The ROI information will be stored within the “Regions” property in the properties panel of the “FP3002” node. Note: This information will be displayed once the “FP3002 Setup” window is closed and the “FP3002” node is deselected and reselected.



Properties

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

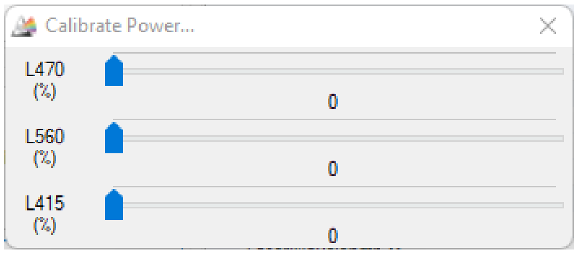
Misc	
AcquisitionMode	**StartPhotometry**
AutoCrop	**False**
PortName	**COM7**
Regions	**RotatedRect[] Array**
[0]	{Center=(91, 593), Size={Width=76, Height=68}, Angle=0}
[1]	{Center=(833.5, 450), Size={Width=77, Height=64}, Angle=0}

For more information on how to interact with the “Regions” window please see the “Calibrate Regions” section of the Appendix II: Hotkeys

“Calibrate Power” -

The “Calibrate Power” tool 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 50μW for 200μm fibers and approximately 120μW for 400μm 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.

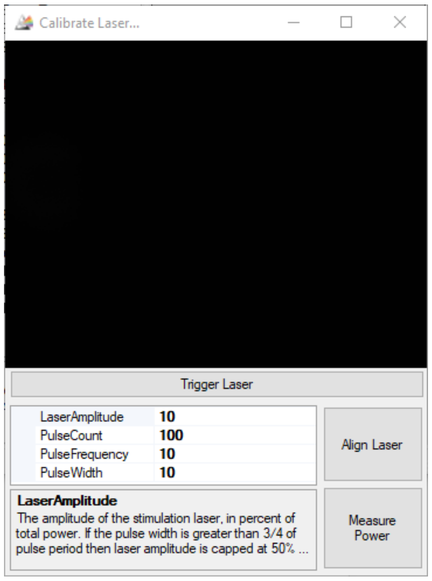
Upon clicking the “Calibrate Power” button in the “FP3002 Setup” window, a “Calibrate Power” window will appear. This window contains a power scroll bar for each LED that allows you to manually adjust each LED’s power while measuring the power of the light coming out of the ferrule.



Once the appropriate percentage of power is found for each LED, close the “Calibrate Power” window and enter these values into the “FP3002 Configuration” panel.

“Calibrate Laser” -

The “Calibrate Laser” tool is used in conjunction with a power meter to specify the appropriate laser power percentage and desired pulse train parameters. Once the “Laser Wavelength” parameter within the “FP3002 Configuration” panel is specified to “450” or “635”, the “Calibrate Laser” button will become accessible. After clicking the “Calibrate Laser” button the “Calibrate Laser” window will appear. Depending on the specified “Laser Wavelength”, the “Calibrate Laser” window will appear differently. Below is the “Calibrate Laser” window for the 635nm laser:



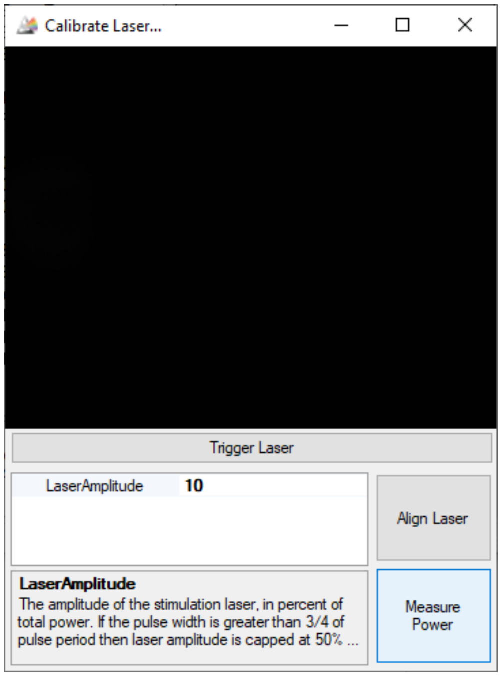
LaserAmplitude	10
PulseCount	100
PulseFrequency	10
PulseWidth	10

LaserAmplitude
The amplitude of the stimulation laser, in percent of total power. If the pulse width is greater than 3/4 of pulse period then laser amplitude is capped at 50% ...

The 635nm laser's "Calibrate Laser" window has two different modes, alignment mode and power measurement mode. When this window is initially opened, it will be in alignment mode. The user has access to all of the stimulation parameters to help with this alignment. Once the parameters are specified, click the "Trigger Laser" button to begin stimulation and start aligning the laser using the two-axis translator at the back of the optical housing. Once aligned to the desired fiber, click the "Stop Laser" button to stop the laser pulse train. While aligning the laser, the pulse train will continue until stopped. However, there is a safety measure in place such that the duration of the pulse train will be limited to 30 seconds if all of the conditions below are met:

1. The amplitude is over 50%
2. The duty cycle is over 75%
3. The total pulse train duration is over 30 seconds.

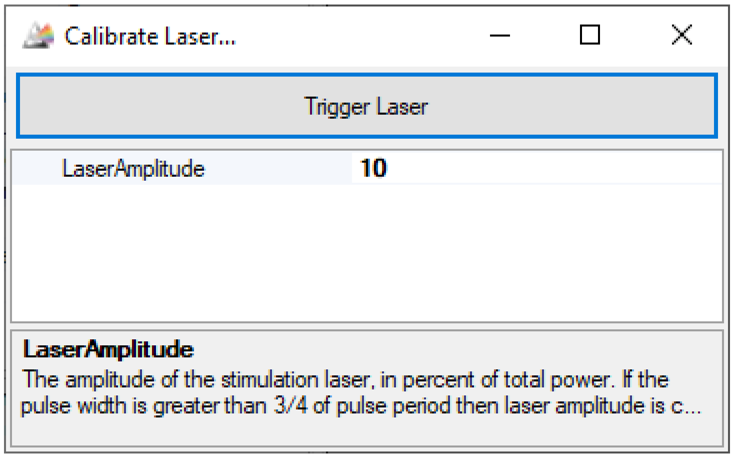
To measure the power of the laser, click the "Measure Power" button. This will configure the laser to be in constant mode for 30 seconds, allowing power measurements to be acquired. Here, the only stimulation property users have access to is the "LaserAmplitude" property.



LaserAmplitude 10

LaserAmplitude
The amplitude of the stimulation laser, in percent of total power. If the pulse width is greater than 3/4 of pulse period then laser amplitude is capped at 50% ...

When the “450” laser is specified, the “Calibrate Laser” window is more simplified since users do not have to align to a patch cord. In this case, the users can stimulate the laser in constant mode for power measurement purposes. Do this by specifying the “Laser Amplitude” value and clicking “TriggerLaser” to start stimulation.



LaserAmplitude	10

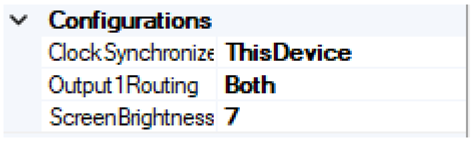
LaserAmplitude
The amplitude of the stimulation laser, in percent of total power. If the pulse width is greater than 3/4 of pulse period then laser amplitude is c...

Once the laser’s power and pulse train parameters are set, close the “Calibrate Laser” window and the parameters will be updated in the “FP3002 Configuration” panel. However, when working with the 450nm laser, you will still need to set the desired pulse train parameters within the “FP3002 Configuration” panel.

FP3002 Configuration: This panel is located in the center of the window and contains all of the configurable system settings. The system settings are grouped together into six sections: “Configurations”, “Digital IO”, “Photometry”, “Power”, “Power (Laser)”, and “Stimulation Pulse”.

“Configurations” -

The “Configurations” section allows the user to change the hardware configuration of the FP3002 system. Here the user has control over three system settings: “Clock Synchronizer”, “Output 1 Routing”, and “Screen Brightness”.



Configurations	
Clock Synchronize	ThisDevice
Output 1 Routing	Both
Screen Brightness	7

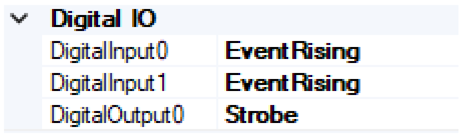
“Clock Synchronizer”: Specifies whether the FP3002 system outputs its own clock line, or synchronizes to an external clock. In most experiments this property is set to “ThisDevice”.

“Output 1 Routing”: Specifies whether the digital output pin 1 state is routed to the BNC port, internal laser, or both. In experiments involving 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.

Screen Brightness”: Specifies the brightness of the LCD screen on the FP3002 system.

“Digital IO” -

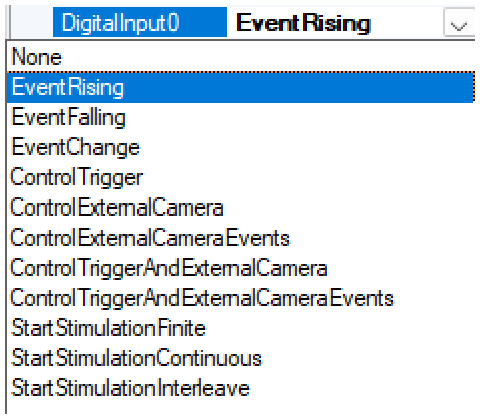
The “Digital IO” section allows users to configure the Digital Input ports and the Digital Output 0 port.



Digital IO

DigitalInput0	EventRising
DigitalInput1	EventRising
DigitalOutput0	Strobe

“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.



DigitalInput0 EventRising

- None
- EventRising
- EventFalling
- EventChange
- ControlTrigger
- ControlExternalCamera
- ControlExternalCameraEvents
- ControlTriggerAndExternalCamera
- ControlTriggerAndExternalCameraEvents
- StartStimulationFinite
- StartStimulationContinuous
- StartStimulationInterleave

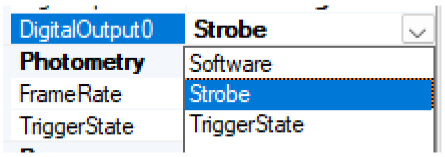
None: Nothing will occur when there is a TTL signal on the Digital Input port.
Event Rising: Sends a Harp Message to Bonsai indicating an event occurred whenever the +5V TTL signal changes from LOW to HIGH.

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.

Start Stimulation Interleave: A currently disabled mode of stimulation. When enabled, it will allow for stimulation to occur during the dead time of the camera.

“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.


DigitalOutput0 Strobe
Photometry
FrameRate
TriggerState

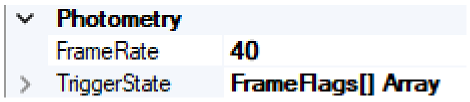
Software: Specifies that the digital output signal will be generated within Bonsai using a software trigger and the “Digital Output” node, then sent to the system through the “FP3002” node.

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.

“Photometry” -

The “Photometry” section contains general photometry settings for the system.

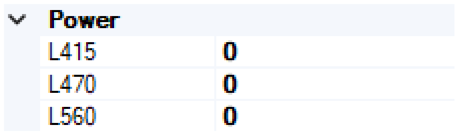

Photometry
FrameRate 40
TriggerState FrameFlags[] Array

“Frame Rate”: Dictates the frequency at which photometry data frames are generated. This value has units of “Hertz” and directly determines the frame rate of the internal camera. Indirectly, this value also determines the frequency of each LED since the current LED in the trigger sequence transitions every camera frame. This value ranges from 16-200Hz, however for higher frame rate, the “Auto Crop” property in the “FP3002” node’s property panel must be set to “True”.

“Trigger State”: Contains the trigger sequence information generated by the “Trigger Sequence” panel.

“Power” -

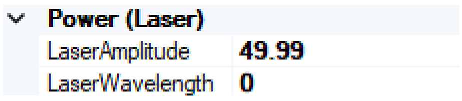
The “Power” section contains the power percentage of the LEDs. These powers should be configured using the “Calibrate Power” tool in the “Setup” panel. The units of these values are displayed as the percentage of the max LED power.



Power	
L415	0
L470	0
L560	0

“Power (Laser)” -

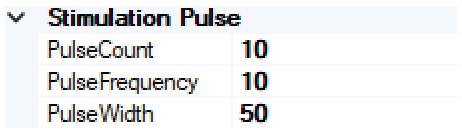
The “Power (Laser)” section contains the information pertaining to the internal laser’s amplitude and wavelength. When conducting an experiment with stimulation, the “Laser Wavelength” is the first laser property that needs to be set. When this is set to “450” or “635” the “Calibrate Laser” tool will become accessible within the “Setup” window. Then, the “Laser Amplitude” setting can be configured with the “Calibrate Laser” tool.



Power (Laser)	
LaserAmplitude	49.99
LaserWavelength	0

“Stimulation Pulse” -

The “Stimulation Pulse” section stores the laser pulse train parameters for experiments with stimulation. These stimulation properties possess a safety measure meant to prevent overuse of the internal laser. When the stimulation properties are configured with an amplitude over 50% and a duty cycle over 75%, then the “Pulse Count” setting will be limited such that the total duration of a pulse train is less than 30 seconds.



Stimulation Pulse	
PulseCount	10
PulseFrequency	10
PulseWidth	50

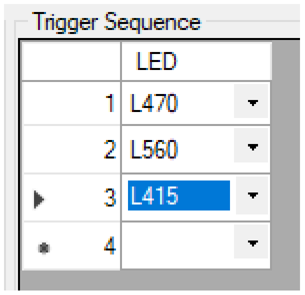
“Pulse Count”: Specifies the number of laser pulses in a finite pulse train. In continuous pulse trains, this parameter is ignored.

“Pulse Frequency”: Specifies the frequency parameter for laser pulse trains. This value has units of Hertz with a range of 1-1000Hz.

“Pulse Width”: Specifies the pulse width of laser pulses in a pulse train. This value has units of milliseconds and must be less than or equal to:

$$\frac{1}{Pulse\ Frequency\ (Hz)} * \frac{1000ms}{1sec}.$$

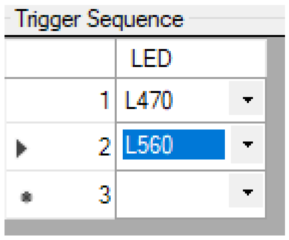
Trigger Sequence: This panel is located on the right side of the window and contains a tool for configuring the trigger sequence of the LEDs. This tool configures which LEDs are triggered and in what order. The last row in the panel does not affect the trigger sequence and is used to add new LEDs to the sequence.



Trigger Sequence

	LED
1	L470
2	L560
▶ 3	L415
* 4	

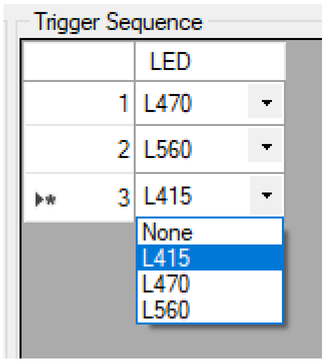
To delete an LED, select a row in the sequence and click the delete (Del) button on the keyboard.



Trigger Sequence

	LED
1	L470
▶ 2	L560
* 3	

To add an LED, click the dropdown menu on the bottom row and select which LED to use.



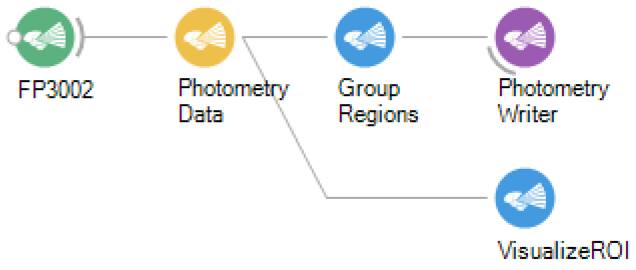
Trigger Sequence

	LED
1	L470
2	L560
▶* 3	L415

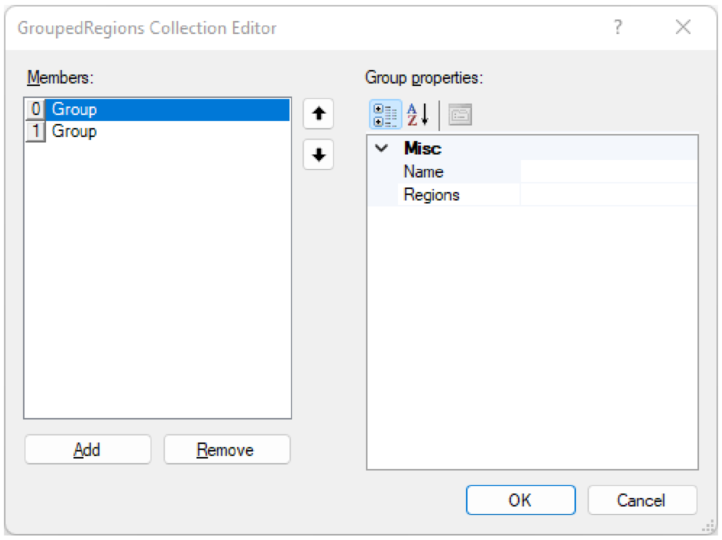
To configure the order, click the drop down menu on each row and select where each LED goes in the sequence. The duty cycle of the LEDs can also be configured by inserting “None” frames, where no LED will be ON during particular data frames.

Group Regions

The “Group Regions” node allows the user to organize the photometry data signals into local groups by adding custom naming conventions to the photometry data. This node accepts inputs of type “PhotometryDataFrame” and outputs data of type “GroupedPhotometryDataFrame”. This node can be connected directly to the “Photometry Writer” node and will cause the output .csv file to contain custom column names for the photometry data.



To configure the “Group Regions” node, double click it while the workflow is stopped to open its editor. Begin by using the “Add” button to add a new member for each local group.


GroupedRegions Collection Editor

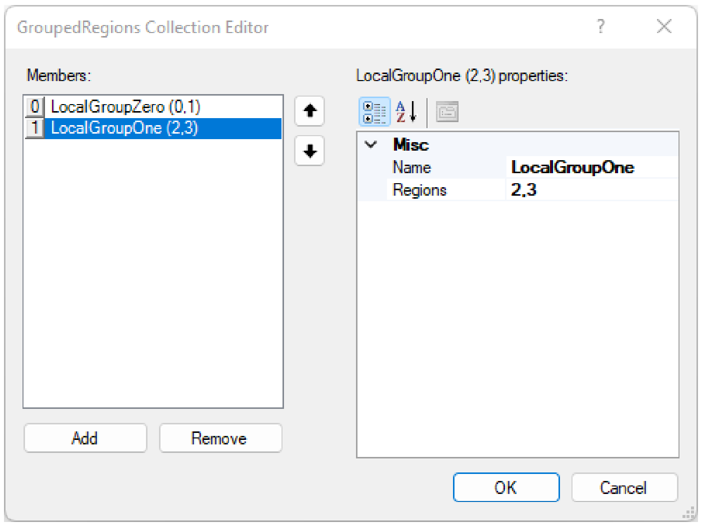
Members:

0 Group
1 Group

Group properties:

Misc
Name
Regions

Then for each local group, provide a custom name in the “Name” property and designate which ROIs belong to each group by listing the index of each region in the “Regions” property. The indices in the list of regions should be separated by commas.



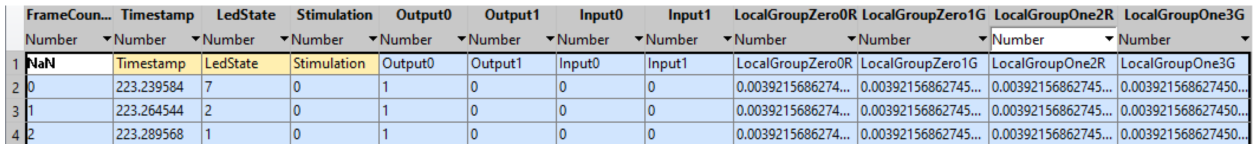
0 LocalGroupZero (0,1)
1 LocalGroupOne (2,3)

LocalGroupOne (2,3) properties:

Misc

Name	LocalGroupOne
Regions	2,3

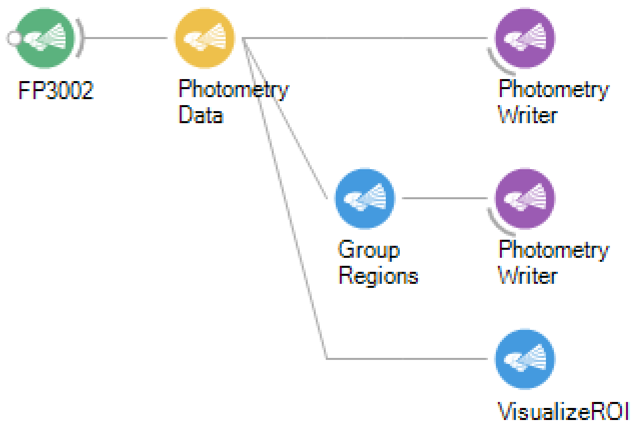
The effect of these local group names can be seen in the column names shown below:



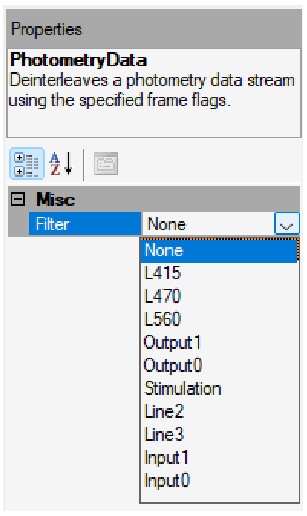
	FrameCoun...	Timestamp	LedState	Stimulation	Output0	Output1	Input0	Input1	LocalGroupZero0R	LocalGroupZero1G	LocalGroupOne2R	LocalGroupOne3G
	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number
1	NaN	Timestamp	LedState	Stimulation	Output0	Output1	Input0	Input1	LocalGroupZero0R	LocalGroupZero1G	LocalGroupOne2R	LocalGroupOne3G
2	0	223.239584	7	0	1	0	0	0	0.0039215686274...	0.00392156862745...	0.00392156862745...	0.003921568627450...
3	1	223.264544	2	0	1	0	0	0	0.0039215686274...	0.00392156862745...	0.00392156862745...	0.003921568627450...
4	2	223.289568	1	0	1	0	0	0	0.0039215686274...	0.00392156862745...	0.00392156862745...	0.003921568627450...

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 any of the following three nodes: "Group Regions", "Photometry Writer", "Visualize ROI".



This node possesses a "Filter" property that can be set to filter the outgoing data based on the specified frame flag. This is only used for purposes of real-time visualization of deinterleaved signals.

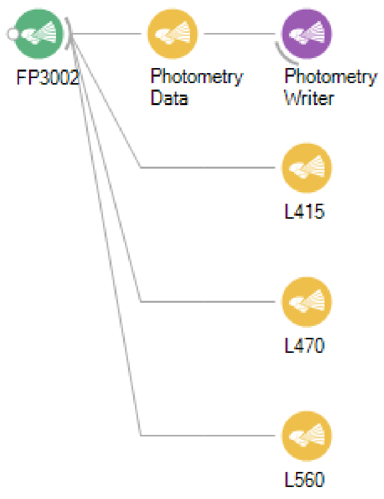

Properties

PhotometryData
Deinterleaves a photometry data stream using the specified frame flags.

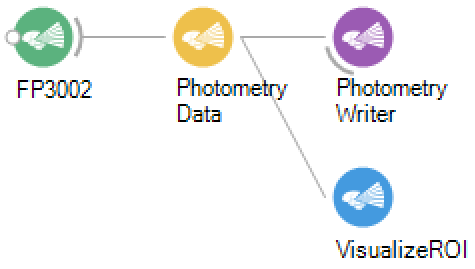
Misc
Filter None

None
L415
L470
L560
Output1
Output0
Stimulation
Line2
Line3
Input1
Input0

It is recommended to keep this value as the default “None” filter for data that is being saved. However, additional “Photometry Data” nodes with filters other than “None” can be inserted in parallel data streams that do not save data. In general, it is good practice to only save the raw, interleaved data set, and use deinterleaved data streams only for visualization during the experiment.



Although the above workflow is sufficient for saving the raw, interleaved data while also providing visualizations of the deinteleaved signals, there is a more compact and user-friendly way to visualize deinterleaved signals. If interested in visualizing deinterleaved signals, please look into the use of the “Visualize ROI” node.



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, Timestamp:

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 “1” indicates the L415, “2” indicates the “L470”, “4” indicates the “L560”, and “7” indicates a null frame (no LEDs).

Column 4, Stimulation:

A boolean value that represents whether stimulation is occurring during this frame. This is NOT used to determine the Laser state, please see the “Chapter 5: Stimulation” or the “Digital IOs” node’s entry in the “Appendix I: Node Glossary” for more information on how to record the laser’s state.

Column 5/6, Output 0/1:

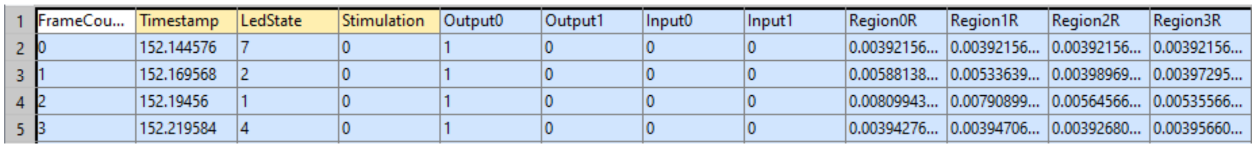
A boolean value that represents the state of the digital output ports during this frame. If you are sending digital outputs at a rate different from the photometry frame rate, please see the “Digital IOs” node’s entry in the “Appendix I: Node Glossary” for more information on how to record the digital output port state.

Column 7/8, Input 0/1:

A boolean value that represents the state of the digital input ports during this frame. For higher precision recording of the digital input ports, please see “Digital IOs” node’s entry in the “Appendix I: Node Glossary” for more information on how to record the digital input port state.

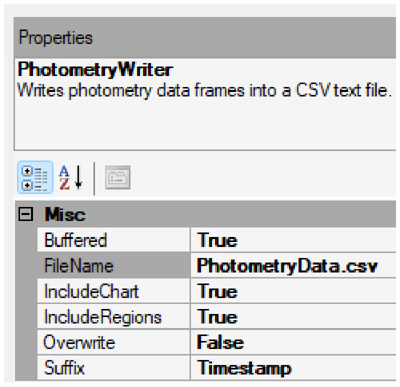
Column 9+, Region Data:

These are the columns where the relative fluorescence data will appear. Each pre-defined ROI will have its own column.



1	FrameCou...	Timestamp	LedState	Stimulation	Output0	Output1	Input0	Input1	Region0R	Region1R	Region2R	Region3R
2	0	152.144576	7	0	1	0	0	0	0.00392156...	0.00392156...	0.00392156...	0.00392156...
3	1	152.169568	2	0	1	0	0	0	0.00588138...	0.00533639...	0.00398969...	0.00397295...
4	2	152.19456	1	0	1	0	0	0	0.00809943...	0.00790899...	0.00564566...	0.00535566...
5	3	152.219584	4	0	1	0	0	0	0.00394276...	0.00394706...	0.00392680...	0.00395660...

The “Photometry Writer” node has a variety of configurable properties. Three of the properties it has in common with the “Csv Writer” node: “File Name”, “Overwrite”, “Suffix”. You can specify the “File Name” property by either double clicking the node while the workflow is stopped, or clicking the “...” in the “File Name” text box. Be sure to specify the file extension as “.csv” in the filename. The “Overwrite” property allows the software to overwrite any files of the same name as specified in the “File Name” property. The “Suffix” property allows you to keep the same file name for multiple experiments by appending a unique suffix to the file name. You can either specify this unique suffix to be an integer value or as a date-time value. The “Include Chart” and “Include Regions” options allow you to generate a chart of the photometry data collected during the experiment and an image showing the labeled regions.



Properties

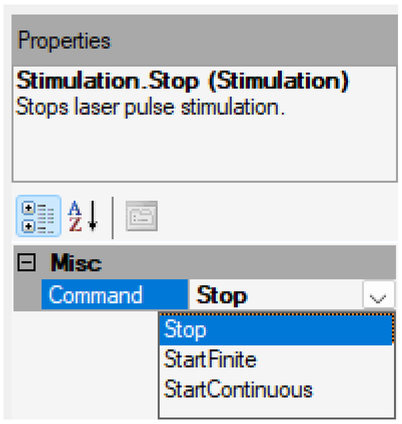
PhotometryWriter
Writes photometry data frames into a CSV text file.

Misc

Buffered	**True**
FileName	**PhotometryData.csv**
IncludeChart	**True**
IncludeRegions	**True**
Overwrite	**False**
Suffix	**Timestamp**

Stimulation

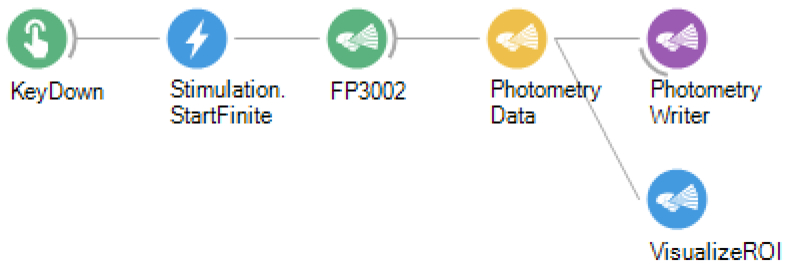
The “Stimulation” node generates Harp messages that can be sent to the FP3002 system through the “FP3002” node. These messages work to command the system to start and/or stop stimulation. This node has a configurable “Command” property used to specify the type of stimulation command it will generate. The “Command” property will dictate whether the node generates a “Stop” stimulation command, “Start Finite” stimulation command, or a “Start Continuous” stimulation command.


Properties

Stimulation.Stop (Stimulation)
Stops laser pulse stimulation.

Misc
Command: **Stop**

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.

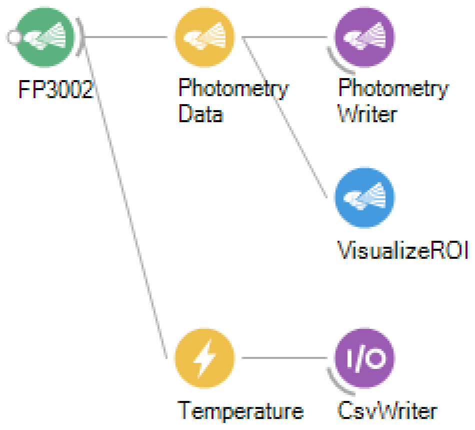


The “Stimulation” node differentiates between a “Finite” and “Continuous” stimulation. With both stimulation modes, a laser pulse train will be started using the “Amplitude”, “Wavelength”, “Pulse Frequency”, and “Pulse Width” parameters set in the “FP3002 Setup” window. However, the “Continuous” pulse 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.

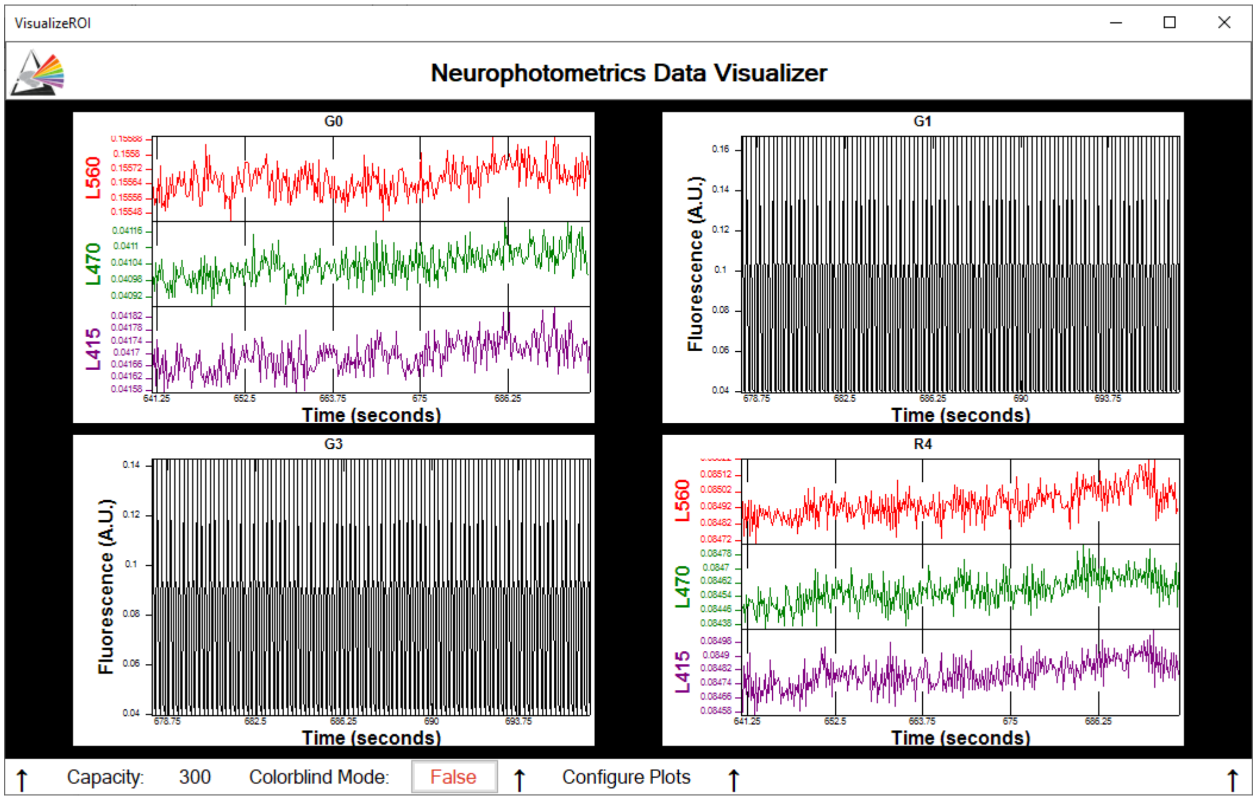
Temperature

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.



Visualize ROI

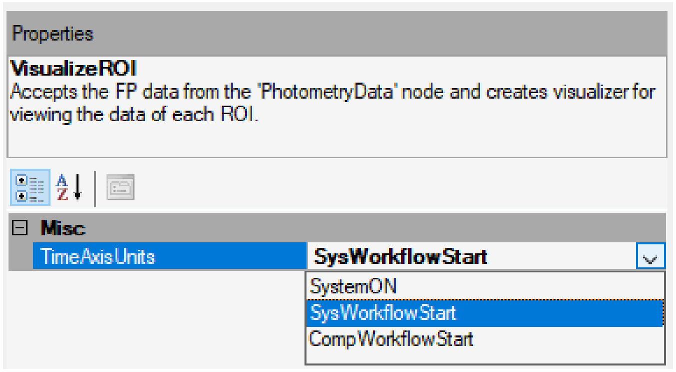
The “Visualize ROI” node allows for the data from the “Photometry Data” node to be displayed in the form of rolling plots. It has an automated layout that will maximize the size of each plot based on the number of visible plots and the dimensions of the visualizer window. This node has a user interface (UI) that allows for the user to adjust the configuration settings of each ROI’s plot. The user has control over which plots are visible, which are deinterleaved, and which are autoscaled. When an ROI’s plot is deinterleaved, the user also has control over which LED plots are visible and which are autoscaled. This new node also allows the user to toggle ON/OFF colorblind mode, creating a more colorblind friendly color palette for deinterleaved plots. The “Visualize ROI” node also gives the option to change the capacity of the plots, showing more or less data points per window.



This node can only accept data directly from the “Photometry Data” node. The “Photometry Data” node must have the “Filter” value set to “None” in order for this visualizer to work properly. You will be able to deinterleave the data within the “Visualize ROI” node’s visualizer so there is no need to do it beforehand.

The “Visualize ROI” node has one property that must be set before starting the workflow. In the properties section of the “Visualize ROI” node, be sure to set the “Time Axis Units” before starting an experiment. Here you can choose what clock/units you want to use for the x-axis of the plots. The options are “System ON”, “Sys Workflow Start”, and “Comp Workflow Start”. Below are the description of each TimeAxisUnits value:

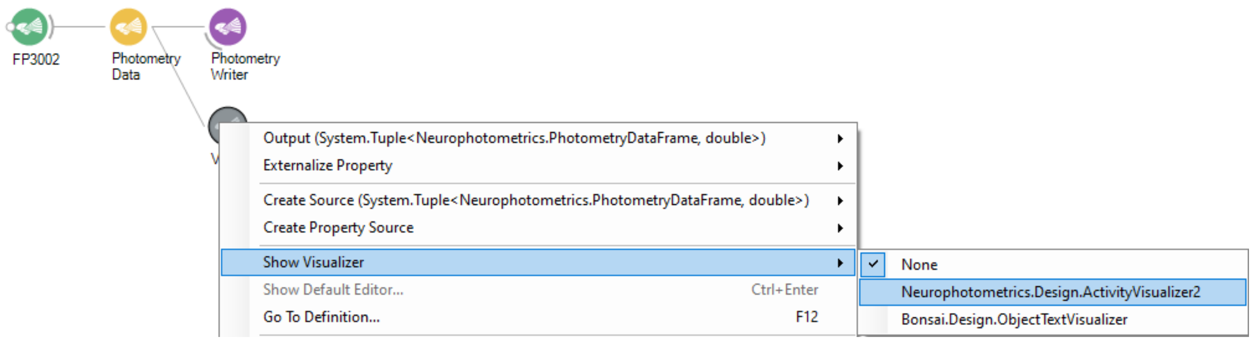
- “System ON”: This will dictate that the x-axis units will be in <u>seconds since the **system** turned on, using the clock on the system</u>. This is the native time units used for the data coming out of the “Photometry Data” node, and what is used when saving to a *.csv* file using the “Photometry Writer” node.
- “Sys Workflow Start”: This will dictate that the x-axis units will be in <u>seconds since the **workflow** is started, using the clock on the system</u>. This uses the same clock as the “SystemON”, but will zero the time to when the workflow is started.
- “Comp Workflow Start”: This will dictate that the x-axis units will be in <u>seconds since the workflow is started, using the **computer’s clock**</u>. This is using a different clock than the “System ON” and “Sys Workflow Start” parameters, but will use the same clock as other data streams that use the “Timestamp” node.


Properties
VisualizeROI
Accepts the FP data from the 'PhotometryData' node and creates visualizer for viewing the data of each ROI.

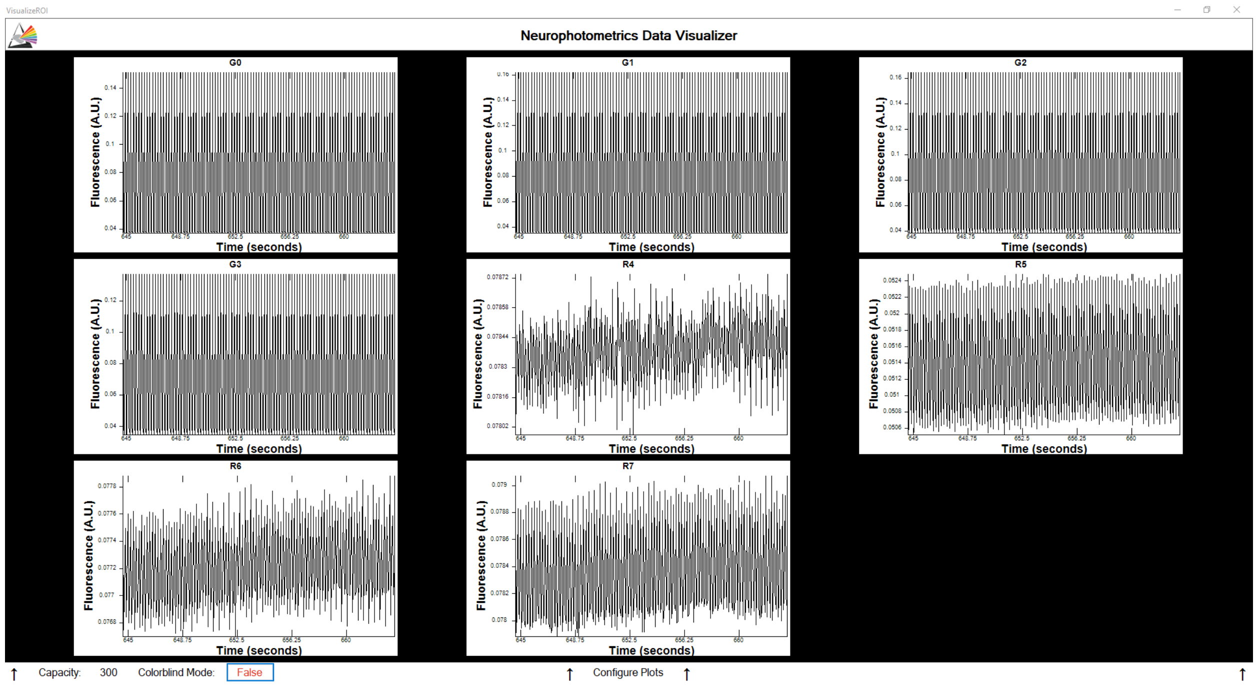
Misc
TimeAxisUnits **SysWorkflowStart**

<u>Be sure to select the appropriate “Time Axis Units” before starting the workflow, as it will not be able to change once the workflow is started.</u>

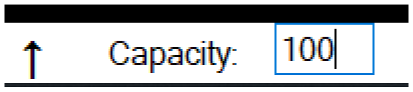
While the workflow is running, double check that the “Visualize ROI” node is using the correct visualizer. Right click on the “Visualize ROI” node, click “Show Visualizer”, and make sure that “Neurophotometrics.Design.ActivityVisualizer2” is selected for the visualizer.


FP3002
Photometry Data
Photometry Writer
Output (System.Tuple<Neurophotometrics.PhotometryDataFrame, double>)
Externalize Property
Create Source (System.Tuple<Neurophotometrics.PhotometryDataFrame, double>)
Create Property Source
Show Visualizer
Show Default Editor... Ctrl+Enter
Go To Definition... F12
None
Neurophotometrics.Design.ActivityVisualizer2
Bonsai.Design.ObjectTextVisualizer

Double click the “Visualize ROI” node to open the activity visualizer. Adjust the size of the window until the plots are easily visible, you can maximize the window to make the plots as large as possible. As you adjust the size of the window, the layout and size of the plots will automatically change to maximize the size and visibility of each plot.

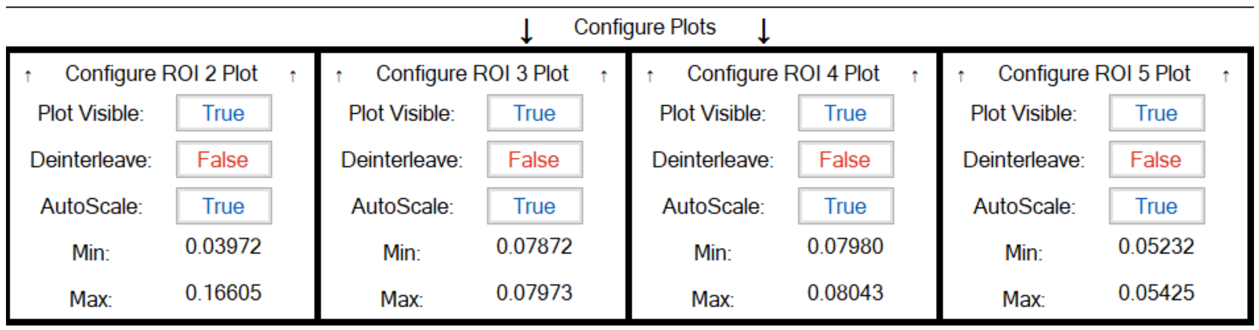


This visualizer has a built-in UI for easy manipulation of the plots. First, you can change the amount of data points each plot shows at a time by clicking on the capacity value, typing in the new number of data points, then pressing the return key or clicking elsewhere on the visualizer. Note that increasing the capacity will display more points starting from the time at which capacity was increased. Increasing or decreasing the capacity will not display data points collected before the capacity was altered.



Capacity: 100

To access the configuration settings of each plot click on the "Configure Plots" tab at the bottom of the visualizer. Then click the Configuration tab for the plot you wish to configure.



Configure Plots

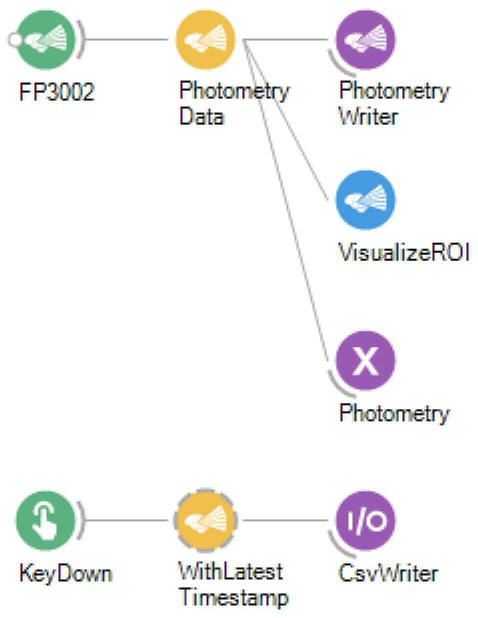
Configure ROI 2 Plot		Configure ROI 3 Plot		Configure ROI 4 Plot		Configure ROI 5 Plot	
Plot Visible:	True	Plot Visible:	True	Plot Visible:	True	Plot Visible:	True
Deinterleave:	False	Deinterleave:	False	Deinterleave:	False	Deinterleave:	False
AutoScale:	True	AutoScale:	True	AutoScale:	True	AutoScale:	True
Min:	0.03972	Min:	0.07872	Min:	0.07980	Min:	0.05232
Max:	0.16605	Max:	0.07973	Max:	0.08043	Max:	0.05425

Here you have the options to adjust the following settings:

- Plot Visible: Toggles whether the plot is visible. The less plots that are visible the more space the visible plots will have, making them larger and easier to see.
- Deinterleave: Toggles whether to deinterleave the incoming FP data. This will separate the plot into multiple axes, one for each LED.
- AutoScale: Toggles whether the plot's y-axis will be autoscaled or not. Autoscale will configure and update the y-axis minimum and maximum limits such that every data point will be visible.
- Min/Max: When AutoScale is True, this displays the current min and max values of the y-axis plot and they are uneditable. When AutoScale is False, these values can be manually set by clicking on the values, typing in the new value, and pressing the return key or clicking elsewhere.

With Latest Timestamp

The “With Latest Timestamp” node is used to generate timestamps for data streams parallel to the photometry data stream. This node will use the latest photometry data timestamp for elements of the parallel data stream. Generating timestamps in this method allows parallel data streams to be timestamped using the system’s clock such that their data sets will already be aligned to the photometry data set.



There is a significant limitation to this method of synchronizing data streams in that the accuracy of these parallel data streams’ timestamps will be dependent on the frame rate of the photometry data stream. Specifically, the timestamps for the parallel data streams will only be accurate to $\frac{1}{Frame\ Rate\ (Hz)}$. Due to this limitation, the “With Latest Timestamp” node should only be used for parallel data streams that generate data at a rate less than or equal to half of the photometry frame rate. For more information on synchronization, please visit the “Synchronization” chapter.

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
Shift + Control: Register camera serial number

Calibrate Regions:

Control + Left Click + Drag: Create uniformly scaled ROI
Control + Right Click + Drag: Uniformly scale existing ROI

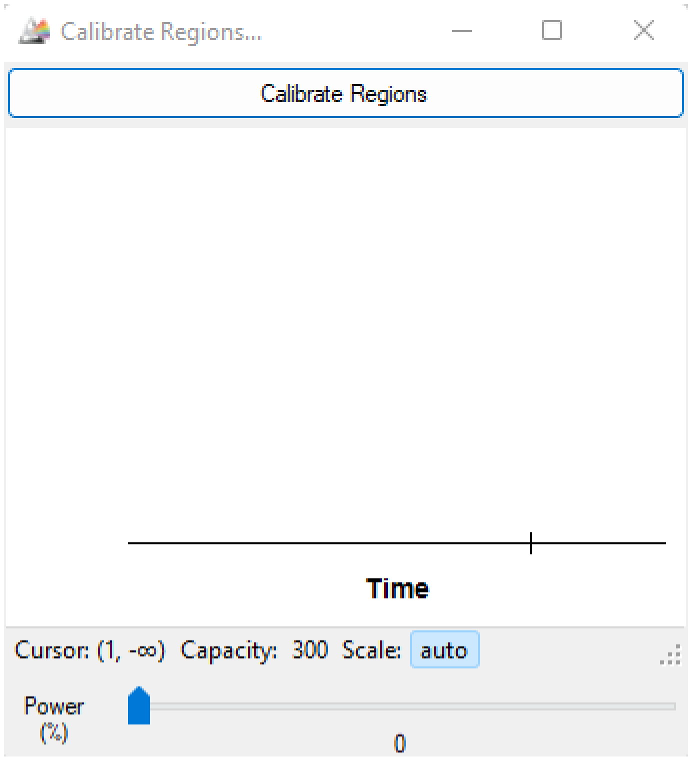
Right Click + Drag: Scale existing ROI
Left Click + Drag: Move existing ROI
Page Up: Increment Image Scale (Brightness)
Page Down: Decrement Image Scale (Brightness)
Control + Z: Undo
Control + Y: Redo
Control + C: Copy ROI
Control + V: Paste ROI
Delete: Delete ROI
Tab: Select next ROI

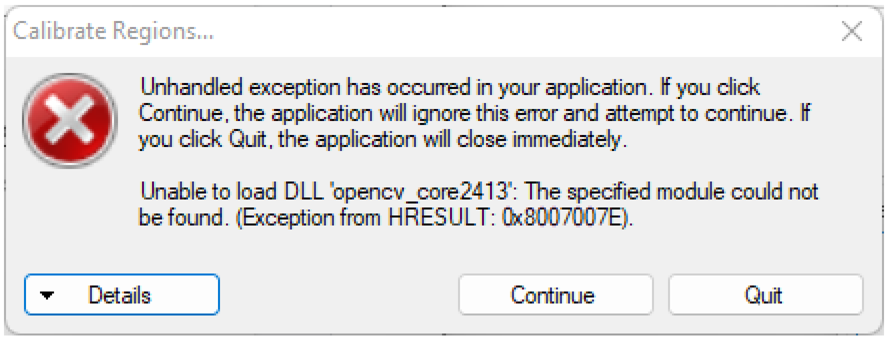
Photometry Data Plot:
Control + P: Print Plot
Control + S: Save Plot

Appendix III: Troubleshooting

OpenCV Errors

These errors appear when clicking the “Calibrate Regions” button from the “FP3002 Setup” window. This error will present itself by not populating the “Calibrate Regions” window with data and displaying an error message.





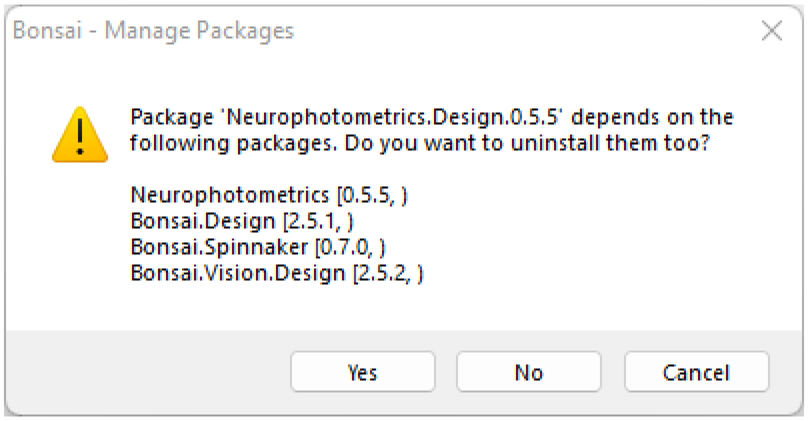
Unhandled exception has occurred in your application. If you click Continue, the application will ignore this error and attempt to continue. If you click Quit, the application will close immediately.

Unable to load DLL 'opencv_core2413': The specified module could not be found. (Exception from HRESULT: 0x8007007E).

Here, Bonsai could not load the “opencv_core2413” DLL file because the wrong version of the “OpenCV” package was installed. When this error occurs, **DO NOT** try and fix it through the “Bonsai - Manage Packages” window. The “OpenCV” package is unique in that many packages directly or indirectly depend on it. So in the process of trying to uninstall the “OpenCV” package, you would uninstall almost all Bonsai packages and eventually run into an error that can only be fixed by uninstalling and reinstalling Bonsai.

There are two ways to properly fix this OpenCV error. The first option is to fully uninstall and reinstall Bonsai. To do this, close out of all open Bonsai applications. Then search “Bonsai” in the Windows Search Bar. Right click on the Bonsai application icon and click uninstall. This should open the “Programs and Features” window. Find and click the “Bonsai” program, then click “Uninstall”. Once uninstalled, start a fresh install of Bonsai. When a fresh version of Bonsai is installed, install the “Bonsai - Starter Pack” and “Neurophotometrics.Design” packages. These two packages will automatically install the needed packages for standard operation of fiber photometry workflows.

The second option is a little more complex of a process, but will allow you to fix the error offline, without having to uninstall and reinstall Bonsai. Start by uninstalling the "Neurophotometrics.Design" package and when the warning message shown below pops up, click "No". Then uninstall the "Neurophotometrics" package, also clicking "No" to the popup warning.

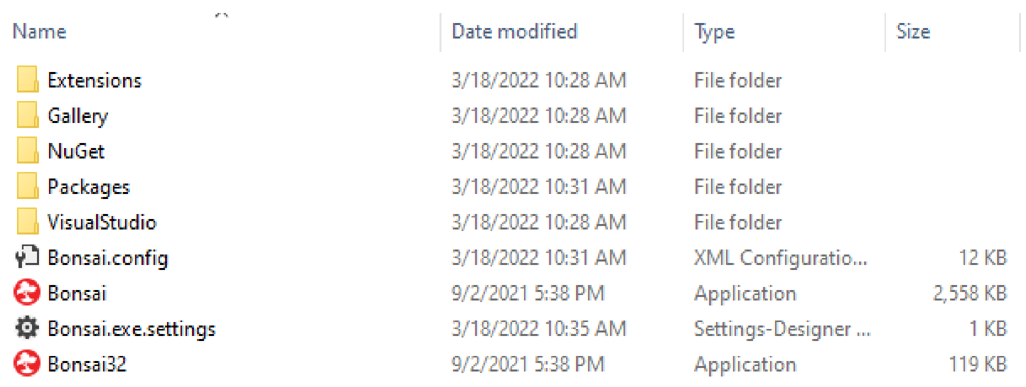

Bonsai - Manage Packages

Package 'Neurophotometrics.Design.0.5.5' depends on the following packages. Do you want to uninstall them too?

Neurophotometrics [0.5.5,)
Bonsai.Design [2.5.1,)
Bonsai.Spinnaker [0.7.0,)
Bonsai.Vision.Design [2.5.2,)

It is important to uninstall the "Neurophotometrics.Design" package before the "Neurophotometrics" package because "Neurophotometrics.Design" depends on "Neurophotorics", throwing an error when trying to uninstall "Neurophotometrics" first. It is also important to click "No" to the warning messages that pop up because clicking "Yes" will cause Bonsai to try and fail to uninstall all of the dependencies that these two packages have.

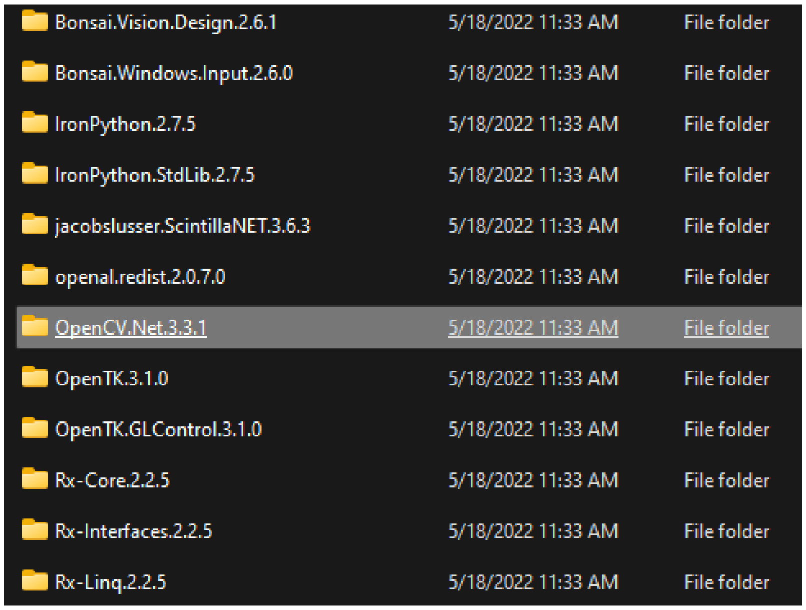
Once the "Neurophotometrics" and "Neurophotometrics.Design" packages are uninstalled, close out of all open Bonsai application windows. Open up the Windows "File Explorer" and navigate to "C:\Users\***UserName***" where ***UserName*** is the user logged in to the computer. Then navigate to "C:\Users\***UserName***\AppData\Local\Bonsai". Note, on most computers the "AppData" folder does not show up in the "File Explorer" so you will have to manually type it in the path at the top of the window. Bonsai's directory should look like this:



Name	Date modified	Type	Size
Extensions	3/18/2022 10:28 AM	File folder	
Gallery	3/18/2022 10:28 AM	File folder	
NuGet	3/18/2022 10:28 AM	File folder	
Packages	3/18/2022 10:31 AM	File folder	
VisualStudio	3/18/2022 10:28 AM	File folder	
Bonsai.config	3/18/2022 10:31 AM	XML Configuratio...	12 KB
Bonsai	9/2/2021 5:38 PM	Application	2,558 KB
Bonsai.exe.settings	3/18/2022 10:35 AM	Settings-Designer ...	1 KB
Bonsai32	9/2/2021 5:38 PM	Application	119 KB

Navigate into the "Packages" folder and delete the folder that's name starts with "OpenCV".

Caution: Be especially careful and do not delete any folders that start in "OpenTK", they look very similar to the "OpenCV" folder. If this happens you might need to uninstall and reinstall Bonsai to fix the issue.

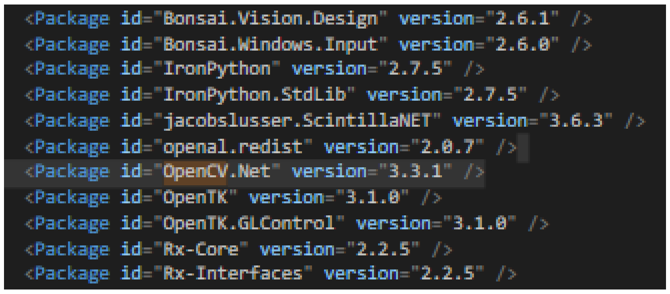


Bonsai.Vision.Design.2.6.1	5/18/2022 11:33 AM	File folder
Bonsai.Windows.Input.2.6.0	5/18/2022 11:33 AM	File folder
IronPython.2.7.5	5/18/2022 11:33 AM	File folder
IronPython.StdLib.2.7.5	5/18/2022 11:33 AM	File folder
jacobslusser.ScintillaNET.3.6.3	5/18/2022 11:33 AM	File folder
openal.redist.2.0.7.0	5/18/2022 11:33 AM	File folder
OpenCV.Net.3.3.1	5/18/2022 11:33 AM	File folder
OpenTK.3.1.0	5/18/2022 11:33 AM	File folder
OpenTK.GLControl.3.1.0	5/18/2022 11:33 AM	File folder
Rx-Core.2.2.5	5/18/2022 11:33 AM	File folder
Rx-Interfaces.2.2.5	5/18/2022 11:33 AM	File folder
Rx-Linq.2.2.5	5/18/2022 11:33 AM	File folder

Now that the "OpenCV" packages are deleted, leave the "Packages" folder to return to the "Bonsai" folder. Currently, these packages are still listed in the "Bonsai.config" file so when Bonsai is opened again, it will automatically reinstall all of the deleted folders to correct itself. To prevent this, the "Bonsai.config" file needs to be carefully edited.

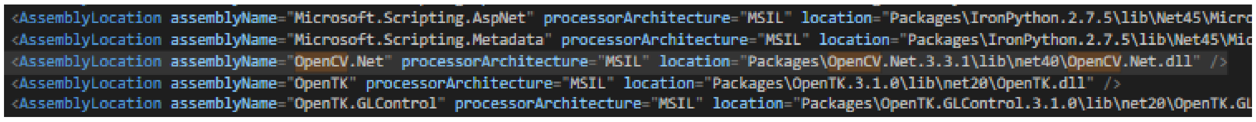
Open the “Bonsai.config” in “Visual Studio” or in “Notepad”. This file has four main sections: “Packages”, “AssemblyReferences”, “AssemblyLocations”, and “LibraryFolders”. In the “Packages” section, delete all references to any “OpenCV” packages. To help find all references to “OpenCV” press “CTRL” + “F” to search for “OpenCV”.

Caution: When deleting all references to any “OpenCV” packages and assembly locations, be very careful not to delete references to “OpenTK” packages and assembly locations. This might require a full uninstall and reinstall of Bonsai.



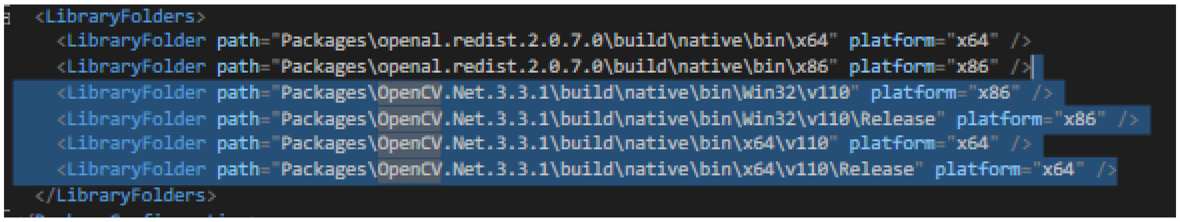
```
<Package id="Bonsai.Vision.Design" version="2.6.1" />
<Package id="Bonsai.Windows.Input" version="2.6.0" />
<Package id="IronPython" version="2.7.5" />
<Package id="IronPython.StdLib" version="2.7.5" />
<Package id="jacobslusser.ScintillaNET" version="3.6.3" />
<Package id="openal.redist" version="2.0.7" />
<Package id="OpenCV.Net" version="3.3.1" />
<Package id="OpenTK" version="3.1.0" />
<Package id="OpenTK.GLControl" version="3.1.0" />
<Package id="Rx-Core" version="2.2.5" />
<Package id="Rx-Interfaces" version="2.2.5" />
```

In the “AssemblyLocations” section, delete all references to any “OpenCV” assembly locations:



```
<AssemblyLocation assemblyName="Microsoft.Scripting.AspNet" processorArchitecture="MSIL" location="Packages\IronPython.2.7.5\lib\Net45\Micro
<AssemblyLocation assemblyName="Microsoft.Scripting.Metadata" processorArchitecture="MSIL" location="Packages\IronPython.2.7.5\lib\Net45\Mic
<AssemblyLocation assemblyName="OpenCV.Net" processorArchitecture="MSIL" location="Packages\OpenCV.Net.3.3.1\lib\net40\OpenCV.Net.dll" />
<AssemblyLocation assemblyName="OpenTK" processorArchitecture="MSIL" location="Packages\OpenTK.3.1.0\lib\net20\OpenTK.dll" />
<AssemblyLocation assemblyName="OpenTK.GLControl" processorArchitecture="MSIL" location="Packages\OpenTK.GLControl.3.1.0\lib\net20\OpenTK.GL
```

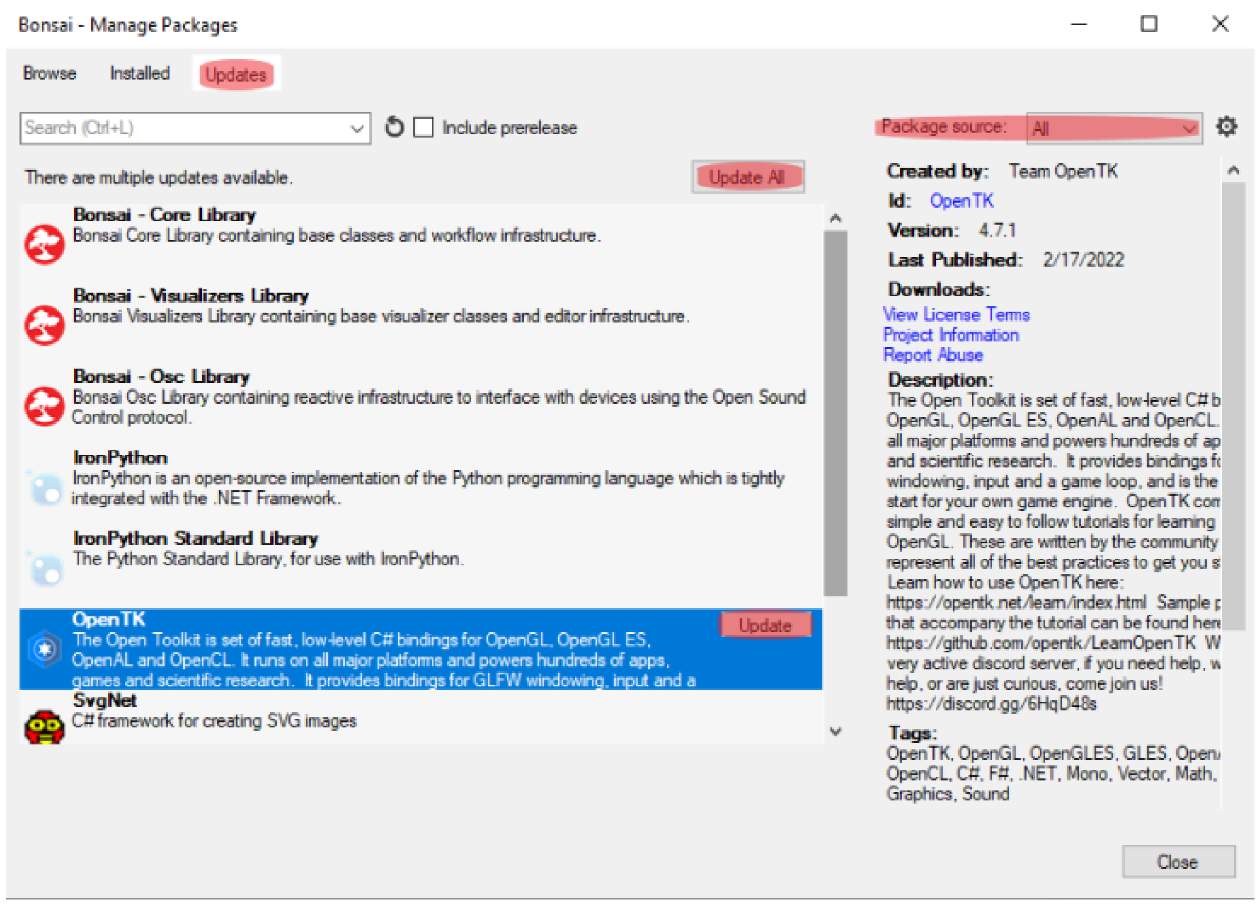
In the “LibraryFolder” section, delete all references to any “OpenCV” library locations:



```
<LibraryFolders>
  <LibraryFolder path="Packages\openal.redist.2.0.7.0\build\native\bin\x64" platform="x64" />
  <LibraryFolder path="Packages\openal.redist.2.0.7.0\build\native\bin\x86" platform="x86" />
  <LibraryFolder path="Packages\OpenCV.Net.3.3.1\build\native\bin\Win32\v110" platform="x86" />
  <LibraryFolder path="Packages\OpenCV.Net.3.3.1\build\native\bin\Win32\v110\Release" platform="x86" />
  <LibraryFolder path="Packages\OpenCV.Net.3.3.1\build\native\bin\x64\v110" platform="x64" />
  <LibraryFolder path="Packages\OpenCV.Net.3.3.1\build\native\bin\x64\v110\Release" platform="x64" />
</LibraryFolders>
```

Once all references to any “OpenCV” package or assembly location have been deleted, save the “Bonsai.config” file and close it. Reopen the Bonsai application and reinstall the “Neurophotometrics.Design” package. This will automatically install the “Neurophotometrics” and “OpenCV” packages with the correct version. This process should have fixed the error. Check by inserting the “FP3002” node into a workflow and opening the “Calibrate Regions” window. This window will now populate with data if the fix was successful. However, if there is also a versioning issue for the “OpenTK” process, then the “Calibrate Regions” button in the “Calibrate Regions” window will also cause an error. Please see the troubleshooting guide for “OpenTK” errors for help with this.

These errors are caused by the installed OpenCV package being a different version than what the “Bonsai.Vision.Design” package is looking for. This versioning issue usually occurs when users manually update the OpenCV package instead of allowing the “Neurophotometrics” or “Bonsai.Vision.Design” packages automatically installing the correct version. An example of what **NOT** to do within the “Bonsai - Manage Packages” window is to enter the “Update” tab, select “All” for the “Package Source”, and click “Update All” or click “Update” on the “OpenCV” package.


Bonsai - Manage Packages

Package source: All

There are multiple updates available.

Bonsai - Core Library
Bonsai Core Library containing base classes and workflow infrastructure.

Bonsai - Visualizers Library
Bonsai Visualizers Library containing base visualizer classes and editor infrastructure.

Bonsai - Osc Library
Bonsai Osc Library containing reactive infrastructure to interface with devices using the Open Sound Control protocol.

IronPython
IronPython is an open-source implementation of the Python programming language which is tightly integrated with the .NET Framework.

IronPython Standard Library
The Python Standard Library, for use with IronPython.

OpenTK
The Open Toolkit is set of fast, low-level C# bindings for OpenGL, OpenGL ES, OpenAL and OpenCL. It runs on all major platforms and powers hundreds of apps, games and scientific research. It provides bindings for GLFW windowing, input and a

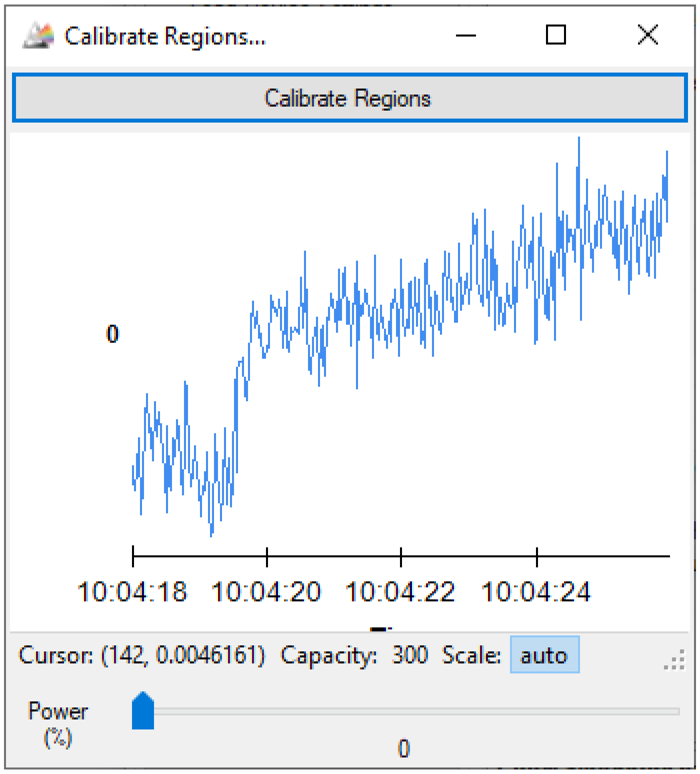
SvgNet
C# framework for creating SVG images

Created by: Team OpenTK
Id: OpenTK
Version: 4.7.1
Last Published: 2/17/2022
Downloads:
View License Terms
Project Information
Report Abuse
Description:
The Open Toolkit is set of fast, low-level C# b
OpenGL, OpenGL ES, OpenAL and OpenCL.
all major platforms and powers hundreds of ap
and scientific research. It provides bindings fo
windowing, input and a game loop, and is the
start for your own game engine. OpenTK com
simple and easy to follow tutorials for learning
OpenGL. These are written by the community
represent all of the best practices to get you s
Learn how to use OpenTK here:
https://opentk.net/learn/index.html Sample p
that accompany the tutorial can be found here
https://github.com/opentk/LearnOpenTK W
very active discord server, if you need help, w
help, or are just curious, come join us!
https://discord.gg/6HqD48s
Tags:
OpenTK, OpenGL, OpenGLES, GLES, Open
OpenCL, C#, F#, .NET, Mono, Vector, Math,
Graphics, Sound

This process for updating packages will update the “OpenCV” package to a newer version than the version used within the “Bonsai - Vision” package, causing an error when trying to use the cameras on Neurophotometrics Fiber Photometry systems.

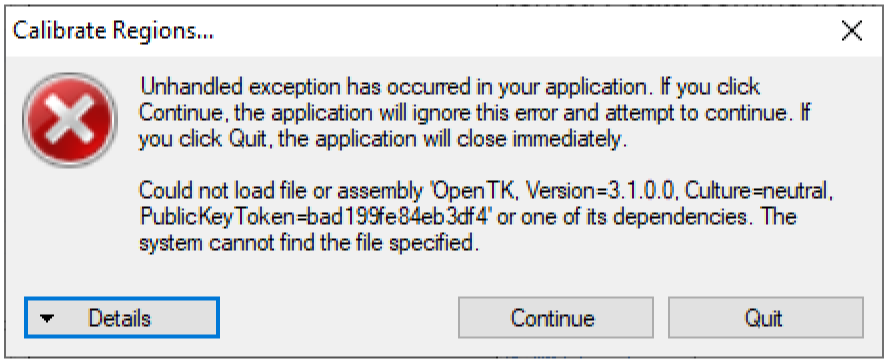
OpenTK Errors

After clicking the “Calibrate Regions” button within the “FP3002 Setup” window, a “Calibrate Regions” window appears, showing the photometry data coming from the camera and giving the user control over the L470 LED.



Cursor: (142, 0.0046161) Capacity: 300 Scale: auto

The OpenTK error occurs when pressing the “Calibrate Regions” button within the “Calibrate Regions” window. The error is shown below for reference.


Calibrate Regions...

Unhandled exception has occurred in your application. If you click Continue, the application will ignore this error and attempt to continue. If you click Quit, the application will close immediately.

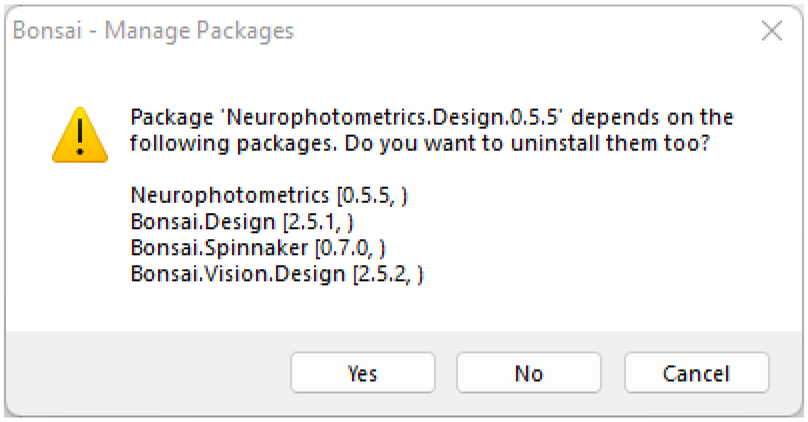
Could not load file or assembly 'OpenTK, Version=3.1.0.0, Culture=neutral, PublicKeyToken=bad199fe84eb3df4' or one of its dependencies. The system cannot find the file specified.

Here, Bonsai could not load the “OpenTK” package with version “3.1.0.0”, which is the required version for “Bonsai.Vision.Design.2.6.1” and any package that depends on it (i.e. “Neurophotometrics.0.5.1”).

When this error occurs, **DO NOT** try and fix it through the “Bonsai - Manage Packages” window. The “OpenTK” package is unique in that many packages directly or indirectly depend on it. So in the process of trying to uninstall the “OpenTK” package, you would uninstall almost all Bonsai packages and eventually run into an error trying to uninstall “Bonsai - Spinnaker” that can only be fixed by uninstalling and reinstalling Bonsai.

There are two ways to properly fix this OpenTK error. The first option is to fully uninstall and reinstall Bonsai. To do this, close out of all open Bonsai applications. Then search “Bonsai” in the Windows Search Bar. Right click on the Bonsai application icon and click uninstall. This should open the “Programs and Features” window. Find and click the “Bonsai” program, then click “Uninstall”. Once uninstalled, start a fresh install of Bonsai. When a fresh version of Bonsai is installed, install the “Bonsai - Starter Pack” and “Neurophotometrics.Design” packages. These two packages will automatically install the needed packages for standard operation of fiber photometry workflows.

The second option is a little more complex of a process, but will allow you to fix the error offline, without having to uninstall and reinstall Bonsai. Start by uninstalling the “Neurophotometrics.Design” package and when the warning message shown below pops up, click “No”. Then uninstall the “Neurophotometrics” package, also clicking “No” to the popup warning.

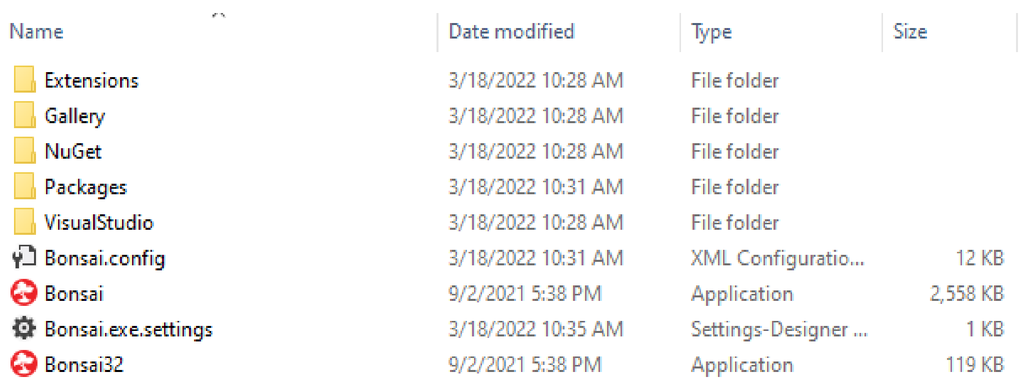

Bonsai - Manage Packages

Package 'Neurophotometrics.Design.0.5.5' depends on the following packages. Do you want to uninstall them too?

Neurophotometrics [0.5.5,)
Bonsai.Design [2.5.1,)
Bonsai.Spinnaker [0.7.0,)
Bonsai.Vision.Design [2.5.2,)

It is important to uninstall the “Neurophotometrics.Design” package before the “Neurophotometrics” package because “Neurophotometrics.Design” depends on “Neurophotorics”, throwing an error when trying to uninstall “Neurophotometrics” first.It is also important to click “No” to the warning messages that pop up because clicking “Yes” will cause Bonsai to try and fail to uninstall all of the dependencies that these two packages have.

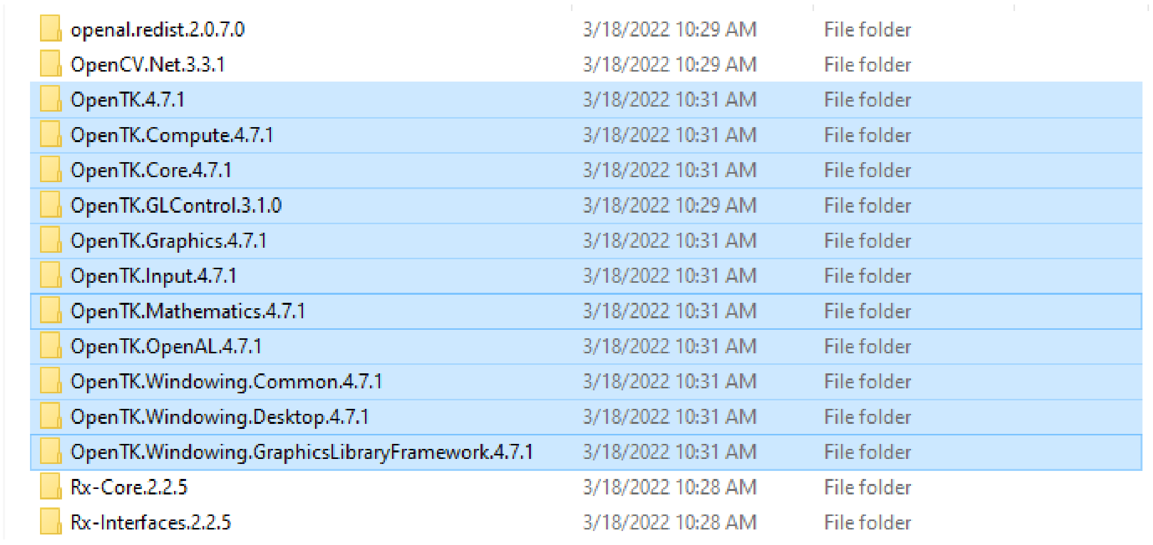
Once the “Neurophotometrics” and “Neurophotometrics.Design” packages are uninstalled, close out of all open Bonsai application windows. Open up the Windows “File Explorer” and navigate to “C:\Users\***UserName***” where ***UserName*** is the user logged in to the computer. Then navigate to “C:\Users\***UserName***\AppData\Local\Bonsai”. Note, on most computers the “AppData” folder does not show up in the “File Explorer” so you will have to manually type it in the path at the top of the window. Bonsai’s directory should look like this:



Name	Date modified	Type	Size
Extensions	3/18/2022 10:28 AM	File folder	
Gallery	3/18/2022 10:28 AM	File folder	
NuGet	3/18/2022 10:28 AM	File folder	
Packages	3/18/2022 10:31 AM	File folder	
VisualStudio	3/18/2022 10:28 AM	File folder	
Bonsai.config	3/18/2022 10:31 AM	XML Configuratio...	12 KB
Bonsai	9/2/2021 5:38 PM	Application	2,558 KB
Bonsai.exe.settings	3/18/2022 10:35 AM	Settings-Designer ...	1 KB
Bonsai32	9/2/2021 5:38 PM	Application	119 KB

Navigate into the “Packages” folder and delete all folders with names starting in “OpenTK”. This manually deletes all of the “OpenTK” packages.

Caution: Be especially careful and do not delete any folders that start in “OpenCV”, they look very similar to the “OpenTK” folders. If this happens you might need to uninstall and reinstall Bonsai to fix the issue.

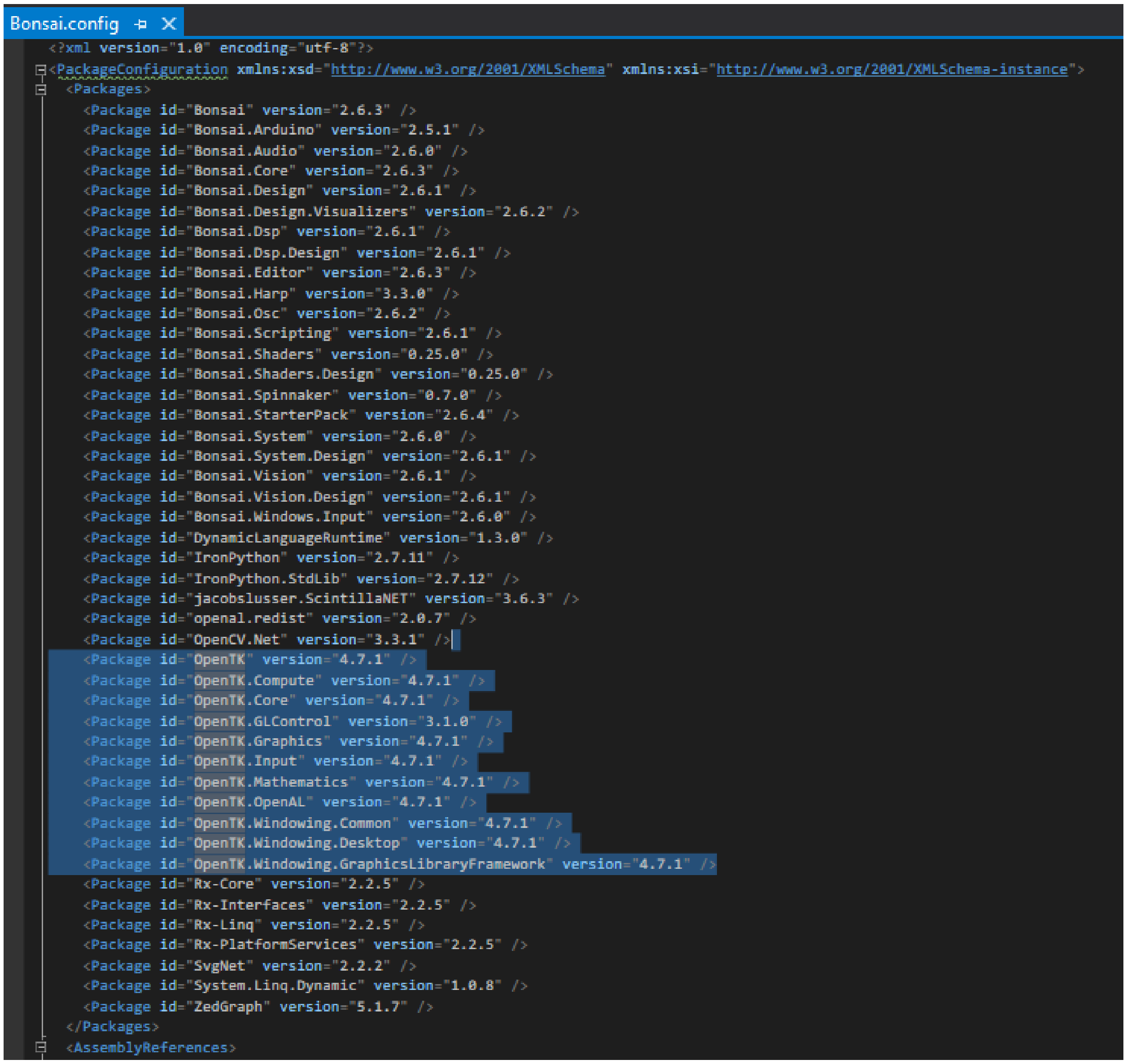


openal.redist.2.0.7.0	3/18/2022 10:29 AM	File folder
OpenCV.Net.3.3.1	3/18/2022 10:29 AM	File folder
OpenTK.4.7.1	3/18/2022 10:31 AM	File folder
OpenTK.Compute.4.7.1	3/18/2022 10:31 AM	File folder
OpenTK.Core.4.7.1	3/18/2022 10:31 AM	File folder
OpenTK.GLControl.3.1.0	3/18/2022 10:29 AM	File folder
OpenTK.Graphics.4.7.1	3/18/2022 10:31 AM	File folder
OpenTK.Input.4.7.1	3/18/2022 10:31 AM	File folder
OpenTK.Mathematics.4.7.1	3/18/2022 10:31 AM	File folder
OpenTK.OpenAL.4.7.1	3/18/2022 10:31 AM	File folder
OpenTK.Windowing.Common.4.7.1	3/18/2022 10:31 AM	File folder
OpenTK.Windowing.Desktop.4.7.1	3/18/2022 10:31 AM	File folder
OpenTK.Windowing.GraphicsLibraryFramework.4.7.1	3/18/2022 10:31 AM	File folder
Rx-Core.2.2.5	3/18/2022 10:28 AM	File folder
Rx-Interfaces.2.2.5	3/18/2022 10:28 AM	File folder

Now that the “OpenTK” packages are deleted, leave the “Packages” folder to return to the “Bonsai”. Currently, these packages are still listed in the “Bonsai.config” file so when Bonsai is opened again, it will automatically reinstall all of the deleted folders to correct itself. To prevent this, the “Bonsai.config” file needs to be carefully edited.

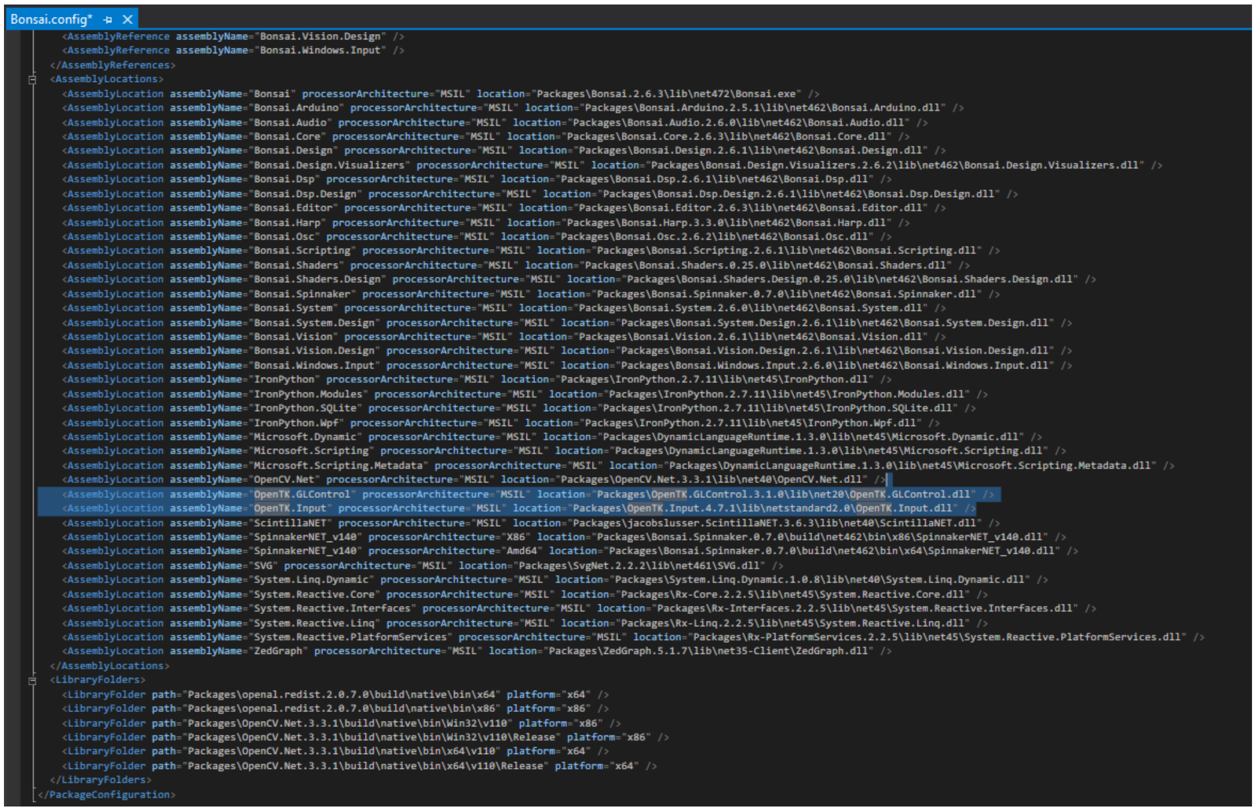
Open the “Bonsai.config” in “Visual Studio” or in “Notepad”. This file has four main sections: “Packages”, “AssemblyReferences”, “AssemblyLocations”, and “LibraryFolders”. In the “Packages” section, delete all references to any “OpenTK” packages. To help find all references to “OpenTK” press “CTRL” + “F” to search for “OpenTK”.

Caution: When deleting all references to any “OpenTK” packages and assembly locations, be very careful not to delete references to “OpenCV” packages and assembly locations. This might require a full uninstall and reinstall of Bonsai.



```
<?xml version="1.0" encoding="utf-8"?>
<PackageConfiguration xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Packages>
    <Package id="Bonsai" version="2.6.3" />
    <Package id="Bonsai.Arduino" version="2.5.1" />
    <Package id="Bonsai.Audio" version="2.6.0" />
    <Package id="Bonsai.Core" version="2.6.3" />
    <Package id="Bonsai.Design" version="2.6.1" />
    <Package id="Bonsai.Design.Visualizers" version="2.6.2" />
    <Package id="Bonsai.Dsp" version="2.6.1" />
    <Package id="Bonsai.Dsp.Design" version="2.6.1" />
    <Package id="Bonsai.Editor" version="2.6.3" />
    <Package id="Bonsai.Harp" version="3.3.0" />
    <Package id="Bonsai.Osc" version="2.6.2" />
    <Package id="Bonsai.Scripting" version="2.6.1" />
    <Package id="Bonsai.Shaders" version="0.25.0" />
    <Package id="Bonsai.Shaders.Design" version="0.25.0" />
    <Package id="Bonsai.Spinnaker" version="0.7.0" />
    <Package id="Bonsai.StarterPack" version="2.6.4" />
    <Package id="Bonsai.System" version="2.6.0" />
    <Package id="Bonsai.System.Design" version="2.6.1" />
    <Package id="Bonsai.Vision" version="2.6.1" />
    <Package id="Bonsai.Vision.Design" version="2.6.1" />
    <Package id="Bonsai.Windows.Input" version="2.6.0" />
    <Package id="DynamicLanguageRuntime" version="1.3.0" />
    <Package id="IronPython" version="2.7.11" />
    <Package id="IronPython.StdLib" version="2.7.12" />
    <Package id="jacobslusser.ScintillaNET" version="3.6.3" />
    <Package id="openal.redist" version="2.0.7" />
    <Package id="OpenCV.Net" version="3.3.1" />
    <Package id="OpenTK" version="4.7.1" />
    <Package id="OpenTK.Compute" version="4.7.1" />
    <Package id="OpenTK.Core" version="4.7.1" />
    <Package id="OpenTK.GLControl" version="3.1.0" />
    <Package id="OpenTK.Graphics" version="4.7.1" />
    <Package id="OpenTK.Input" version="4.7.1" />
    <Package id="OpenTK.Mathematics" version="4.7.1" />
    <Package id="OpenTK.OpenAL" version="4.7.1" />
    <Package id="OpenTK.Windowing.Common" version="4.7.1" />
    <Package id="OpenTK.Windowing.Desktop" version="4.7.1" />
    <Package id="OpenTK.Windowing.GraphicsLibraryFramework" version="4.7.1" />
    <Package id="Rx-Core" version="2.2.5" />
    <Package id="Rx-Interfaces" version="2.2.5" />
    <Package id="Rx-Linq" version="2.2.5" />
    <Package id="Rx-PlatformServices" version="2.2.5" />
    <Package id="SvgNet" version="2.2.2" />
    <Package id="System.Linq.Dynamic" version="1.0.8" />
    <Package id="ZedGraph" version="5.1.7" />
  </Packages>
  <AssemblyReferences>
```

In the “AssemblyLocations” sections, delete all references to any “OpenTK” assembly locations:



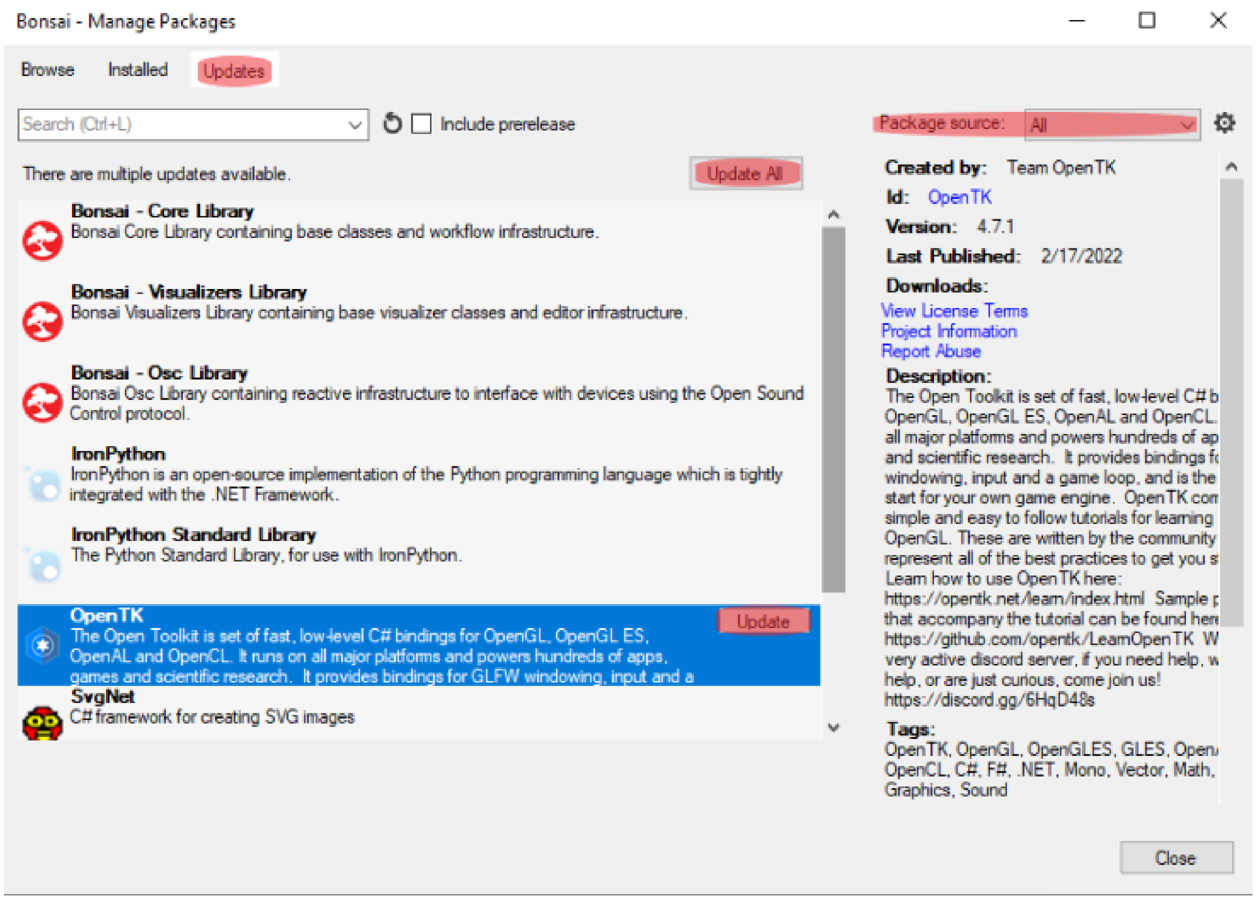
Bonsai.config*

```
    <AssemblyReference assemblyName="Bonsai.Vision.Design" />
    <AssemblyReference assemblyName="Bonsai.Windows.Input" />
  </AssemblyReferences>
  <AssemblyLocations>
    <AssemblyLocation assemblyName="Bonsai" processorArchitecture="MSIL" location="Packages\Bonsai.2.6.3\lib\net472\Bonsai.exe" />
    <AssemblyLocation assemblyName="Bonsai.Arduino" processorArchitecture="MSIL" location="Packages\Bonsai.Arduino.2.5.1\lib\net462\Bonsai.Arduino.dll" />
    <AssemblyLocation assemblyName="Bonsai.Audio" processorArchitecture="MSIL" location="Packages\Bonsai.Audio.2.6.0\lib\net462\Bonsai.Audio.dll" />
    <AssemblyLocation assemblyName="Bonsai.Core" processorArchitecture="MSIL" location="Packages\Bonsai.Core.2.6.3\lib\net462\Bonsai.Core.dll" />
    <AssemblyLocation assemblyName="Bonsai.Design" processorArchitecture="MSIL" location="Packages\Bonsai.Design.2.6.1\lib\net462\Bonsai.Design.dll" />
    <AssemblyLocation assemblyName="Bonsai.Design.Visualizers" processorArchitecture="MSIL" location="Packages\Bonsai.Design.Visualizers.2.6.2\lib\net462\Bonsai.Design.Visualizers.dll" />
    <AssemblyLocation assemblyName="Bonsai.Dsp" processorArchitecture="MSIL" location="Packages\Bonsai.Dsp.2.6.1\lib\net462\Bonsai.Dsp.dll" />
    <AssemblyLocation assemblyName="Bonsai.Dsp.Design" processorArchitecture="MSIL" location="Packages\Bonsai.Dsp.Design.2.6.1\lib\net462\Bonsai.Dsp.Design.dll" />
    <AssemblyLocation assemblyName="Bonsai.Editor" processorArchitecture="MSIL" location="Packages\Bonsai.Editor.2.6.3\lib\net462\Bonsai.Editor.dll" />
    <AssemblyLocation assemblyName="Bonsai.Harp" processorArchitecture="MSIL" location="Packages\Bonsai.Harp.3.3.0\lib\net462\Bonsai.Harp.dll" />
    <AssemblyLocation assemblyName="Bonsai.Osc" processorArchitecture="MSIL" location="Packages\Bonsai.Osc.2.6.2\lib\net462\Bonsai.Osc.dll" />
    <AssemblyLocation assemblyName="Bonsai.Scripting" processorArchitecture="MSIL" location="Packages\Bonsai.Scripting.2.6.1\lib\net462\Bonsai.Scripting.dll" />
    <AssemblyLocation assemblyName="Bonsai.Shaders" processorArchitecture="MSIL" location="Packages\Bonsai.Shaders.0.25.0\lib\net462\Bonsai.Shaders.dll" />
    <AssemblyLocation assemblyName="Bonsai.Shaders.Design" processorArchitecture="MSIL" location="Packages\Bonsai.Shaders.Design.0.25.0\lib\net462\Bonsai.Shaders.Design.dll" />
    <AssemblyLocation assemblyName="Bonsai.Spinnaker" processorArchitecture="MSIL" location="Packages\Bonsai.Spinnaker.0.7.0\lib\net462\Bonsai.Spinnaker.dll" />
    <AssemblyLocation assemblyName="Bonsai.System" processorArchitecture="MSIL" location="Packages\Bonsai.System.2.6.0\lib\net462\Bonsai.System.dll" />
    <AssemblyLocation assemblyName="Bonsai.System.Design" processorArchitecture="MSIL" location="Packages\Bonsai.System.Design.2.6.1\lib\net462\Bonsai.System.Design.dll" />
    <AssemblyLocation assemblyName="Bonsai.Vision" processorArchitecture="MSIL" location="Packages\Bonsai.Vision.2.6.1\lib\net462\Bonsai.Vision.dll" />
    <AssemblyLocation assemblyName="Bonsai.Vision.Design" processorArchitecture="MSIL" location="Packages\Bonsai.Vision.Design.2.6.1\lib\net462\Bonsai.Vision.Design.dll" />
    <AssemblyLocation assemblyName="Bonsai.Windows.Input" processorArchitecture="MSIL" location="Packages\Bonsai.Windows.Input.2.6.0\lib\net462\Bonsai.Windows.Input.dll" />
    <AssemblyLocation assemblyName="IronPython" processorArchitecture="MSIL" location="Packages\IronPython.2.7.11\lib\net45\IronPython.dll" />
    <AssemblyLocation assemblyName="IronPython.Modules" processorArchitecture="MSIL" location="Packages\IronPython.2.7.11\lib\net45\IronPython.Modules.dll" />
    <AssemblyLocation assemblyName="IronPython.SQLite" processorArchitecture="MSIL" location="Packages\IronPython.2.7.11\lib\net45\IronPython.SQLite.dll" />
    <AssemblyLocation assemblyName="IronPython.Wpf" processorArchitecture="MSIL" location="Packages\IronPython.2.7.11\lib\net45\IronPython.Wpf.dll" />
    <AssemblyLocation assemblyName="Microsoft.Dynamic" processorArchitecture="MSIL" location="Packages\DynamicLanguageRuntime.1.3.0\lib\net45\Microsoft.Dynamic.dll" />
    <AssemblyLocation assemblyName="Microsoft.Scripting" processorArchitecture="MSIL" location="Packages\DynamicLanguageRuntime.1.3.0\lib\net45\Microsoft.Scripting.dll" />
    <AssemblyLocation assemblyName="Microsoft.Scripting.Metadata" processorArchitecture="MSIL" location="Packages\DynamicLanguageRuntime.1.3.0\lib\net45\Microsoft.Scripting.Metadata.dll" />
    <AssemblyLocation assemblyName="OpenCV.Net" processorArchitecture="MSIL" location="Packages\OpenCV.Net.3.3.1\lib\net40\OpenCV.Net.dll" />
    <AssemblyLocation assemblyName="OpenTK.GLControl" processorArchitecture="MSIL" location="Packages\OpenTK.GLControl.3.1.0\lib\net20\OpenTK.GLControl.dll" />
    <AssemblyLocation assemblyName="OpenTK.Input" processorArchitecture="MSIL" location="Packages\OpenTK.Input.4.7.1\lib\netstandard2.0\OpenTK.Input.dll" />
    <AssemblyLocation assemblyName="ScintillaNET" processorArchitecture="MSIL" location="Packages\jacobslusser.ScintillaNET.3.6.3\lib\net40\ScintillaNET.dll" />
    <AssemblyLocation assemblyName="SpinnakerNET_v140" processorArchitecture="X86" location="Packages\Bonsai.Spinnaker.0.7.0\build\net462\bin\x86\SpinnakerNET_v140.dll" />
    <AssemblyLocation assemblyName="SpinnakerNET_v140" processorArchitecture="Amd64" location="Packages\Bonsai.Spinnaker.0.7.0\build\net462\bin\x64\SpinnakerNET_v140.dll" />
    <AssemblyLocation assemblyName="SVG" processorArchitecture="MSIL" location="Packages\SvgNet.2.2.2\lib\net461\SVG.dll" />
    <AssemblyLocation assemblyName="System.Linq.Dynamic" processorArchitecture="MSIL" location="Packages\System.Linq.Dynamic.1.0.8\lib\net40\System.Linq.Dynamic.dll" />
    <AssemblyLocation assemblyName="System.Reactive.Core" processorArchitecture="MSIL" location="Packages\Rx-Core.2.2.5\lib\net45\System.Reactive.Core.dll" />
    <AssemblyLocation assemblyName="System.Reactive.Interfaces" processorArchitecture="MSIL" location="Packages\Rx-Interfaces.2.2.5\lib\net45\System.Reactive.Interfaces.dll" />
    <AssemblyLocation assemblyName="System.Reactive.Linq" processorArchitecture="MSIL" location="Packages\Rx-Linq.2.2.5\lib\net45\System.Reactive.Linq.dll" />
    <AssemblyLocation assemblyName="System.Reactive.PlatformServices" processorArchitecture="MSIL" location="Packages\Rx-PlatformServices.2.2.5\lib\net45\System.Reactive.PlatformServices.dll" />
    <AssemblyLocation assemblyName="ZedGraph" processorArchitecture="MSIL" location="Packages\ZedGraph.5.1.7\lib\net35-Client\ZedGraph.dll" />
  </AssemblyLocations>
  <LibraryFolders>
    <LibraryFolder path="Packages\openal.redist.2.0.7.0\build\native\bin\x64" platform="x64" />
    <LibraryFolder path="Packages\openal.redist.2.0.7.0\build\native\bin\x86" platform="x86" />
    <LibraryFolder path="Packages\OpenCV.Net.3.3.1\build\native\bin\Win32\v110" platform="x86" />
    <LibraryFolder path="Packages\OpenCV.Net.3.3.1\build\native\bin\Win32\v110\Release" platform="x86" />
    <LibraryFolder path="Packages\OpenCV.Net.3.3.1\build\native\bin\x64\v110" platform="x64" />
    <LibraryFolder path="Packages\OpenCV.Net.3.3.1\build\native\bin\x64\v110\Release" platform="x64" />
  </LibraryFolders>
</PackageConfiguration>
```

Once all references to any “OpenTK” package or assembly location have been deleted, save the “Bonsai.config” file and close it. Reopen the Bonsai application and reinstall the “Neurophotometrics.Design” package. This will automatically install the “Neurophotometrics” and “OpenTK” packages with the correct version. This process should have fixed the error. Check by inserting the “FP3002” node into a workflow, opening the “Calibrate Regions” window and clicking the “Calibrate Regions” button. Now instead of the error popping up, the “Regions” window opens, allowing you to align the patch cord and draw ROIs.

These errors are caused by the installed OpenTK packages being a different version than what the “Bonsai.Vision.Design” package is looking for. This versioning issue usually occurs when users manually update the OpenTK package instead of allowing the “Neurophotometrics” or “Bonsai.Vision.Design” packages automatically installing the correct version. An example of what **NOT** to do within the “Bonsai - Manage

Packages" window is to enter the "Update" tab, select "All" for the "Package Source", and click "Update All" or click "Update" on the "OpenTK" package.


Bonsai - Manage Packages

Browse Installed Updates

Search (Ctrl+L) Include prerelease

Package source: All

There are multiple updates available. Update All

Bonsai - Core Library
Bonsai Core Library containing base classes and workflow infrastructure.

Bonsai - Visualizers Library
Bonsai Visualizers Library containing base visualizer classes and editor infrastructure.

Bonsai - Osc Library
Bonsai Osc Library containing reactive infrastructure to interface with devices using the Open Sound Control protocol.

IronPython
IronPython is an open-source implementation of the Python programming language which is tightly integrated with the .NET Framework.

IronPython Standard Library
The Python Standard Library, for use with IronPython.

OpenTK Update
The Open Toolkit is set of fast, low-level C# bindings for OpenGL, OpenGL ES, OpenAL and OpenCL. It runs on all major platforms and powers hundreds of apps, games and scientific research. It provides bindings for GLFW windowing, input and a

SvgNet
C# framework for creating SVG images

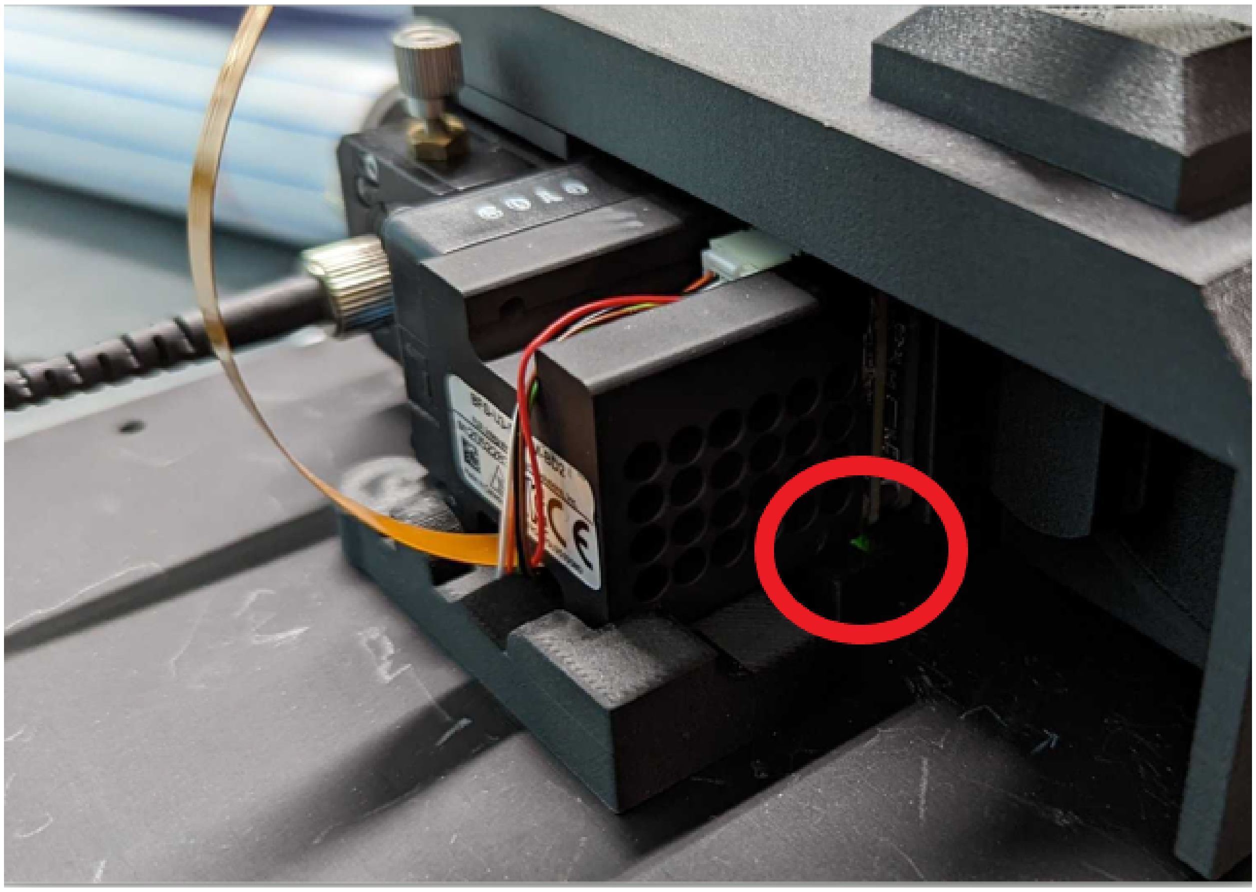
Created by: Team OpenTK
Id: OpenTK
Version: 4.7.1
Last Published: 2/17/2022
Downloads:
View License Terms
Project Information
Report Abuse
Description:
The Open Toolkit is set of fast, low-level C# b
OpenGL, OpenGL ES, OpenAL and OpenCL.
all major platforms and powers hundreds of ap
and scientific research. It provides bindings fo
windowing, input and a game loop, and is the
start for your own game engine. OpenTK com
simple and easy to follow tutorials for learning
OpenGL. These are written by the community
represent all of the best practices to get you s
Learn how to use OpenTK here:
https://opentk.net/learn/index.html Sample p
that accompany the tutorial can be found here
https://github.com/opentk/LearnOpenTK W
very active discord server, if you need help, w
help, or are just curious, come join us!
https://discord.gg/6HqD48s
Tags:
OpenTK, OpenGL, OpenGLES, GLES, Open
OpenCL, C#, F#, .NET, Mono, Vector, Math,
Graphics, Sound

Close

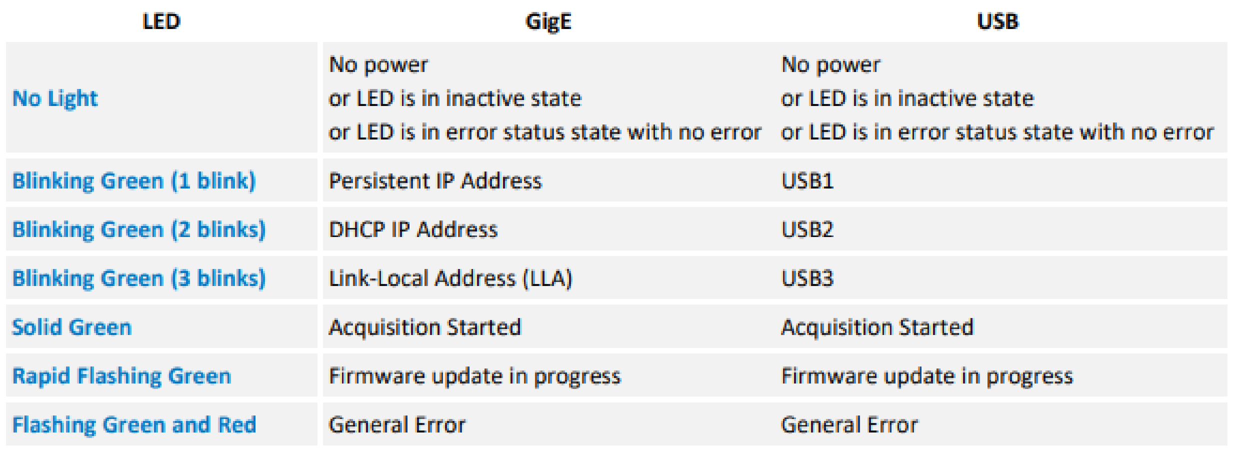
This process for updating packages will update the "OpenTK" package to a newer version than the version used within the "Bonsai - Vision" package, causing an error when trying to use the cameras on Neurophotometrics Fiber Photometry systems.

Camera Connection

Begin by turning on the FP3002 system and connecting it to a USB 3 port on the computer. Open the lid of the FP3002 system and locate the “Status Indicator LED” of the camera. This LED is visible when looking at the bottom of the camera on the side opposite of the camera’s fan.



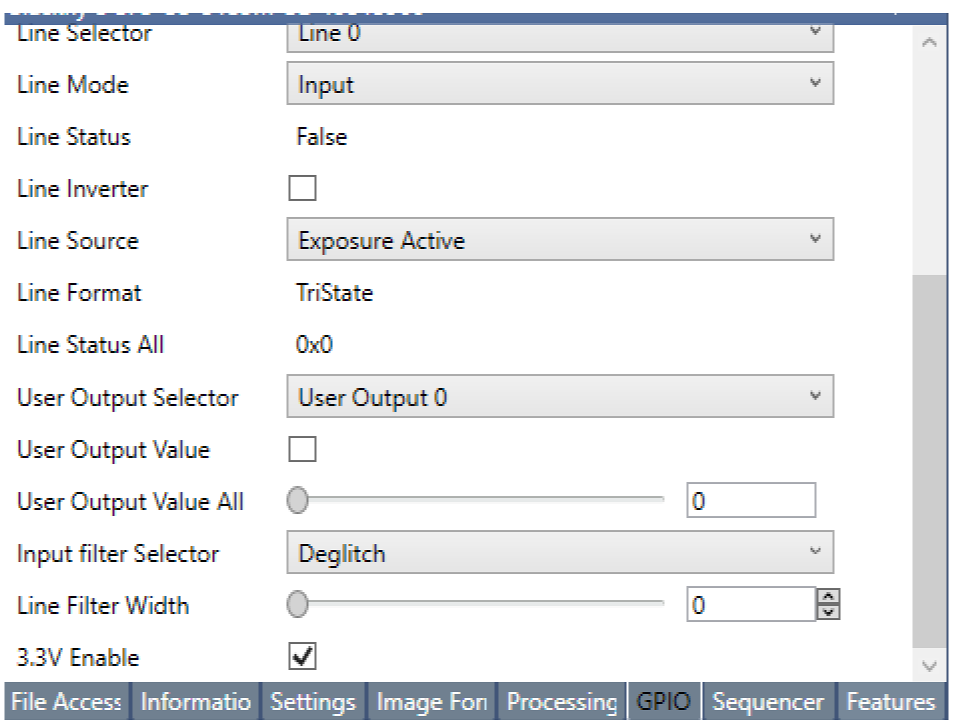
When properly connected, this LED should periodically flash green three times. See the table below for details on the “Status Indicator LED”.



LED	GigE	USB
No Light	No power or LED is in inactive state or LED is in error status state with no error	No power or LED is in inactive state or LED is in error status state with no error
Blinking Green (1 blink)	Persistent IP Address	USB1
Blinking Green (2 blinks)	DHCP IP Address	USB2
Blinking Green (3 blinks)	Link-Local Address (LLA)	USB3
Solid Green	Acquisition Started	Acquisition Started
Rapid Flashing Green	Firmware update in progress	Firmware update in progress
Flashing Green and Red	General Error	General Error

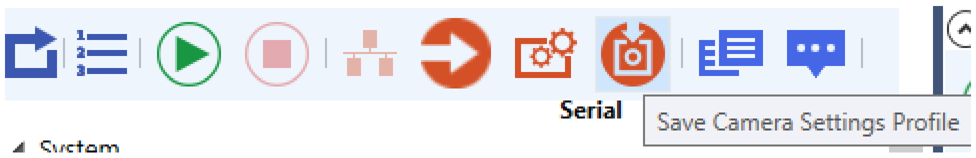
If no light is present, then the camera is operating as if no connection has been made. This can be indicative of a port failure, a failed USB3 to Micro-B cable, or a broken Micro-B port on the system. Disconnect from the current USB3 port and reconnect to any and all other USB3 or USB2 ports. Check this “Status Indicator LED” for every port.

If the “Status Indicator LED” displays the USB3 code, then open SpinView and double check the camera settings. In particular verify that the “3.3V Enable” property in the “GPIO” tab is checked.



Line Selector	Line 0
Line Mode	Input
Line Status	False
Line Inverter	☐
Line Source	Exposure Active
Line Format	TriState
Line Status All	0x0
User Output Selector	User Output 0
User Output Value	☐
User Output Value All	0
Input filter Selector	Deglitch
Line Filter Width	0
3.3V Enable	✓

If this is unchecked, then there might be communication issues between the internal camera and Bonsai. In this case, enable the “3.3V Enable” property and click the “Save Camera Settings Profile” button, agreeing to make this user profile the default camera profile.


Serial
Save Camera Settings Profile
System