STRÏMM User Guide

Author(s): Jacob Francis

Document version: 1.0.0

Last updated: Monday, 31 July 2023

This user guide applies to STRÏMM v1.0.81

Contents

[Installation 2](#_Toc141710417)

[Prerequisites 2](#_Toc141710418)

[Change Log 3](#_Toc141710419)

[Concepts And Main Features 4](#_Toc141710420)

[Akka Framework 4](#_Toc141710421)

[Configuration 5](#_Toc141710422)

[Logging 5](#_Toc141710423)

[Hardware timing and NIDAQ interface 5](#_Toc141710424)

[Saving Data 6](#_Toc141710425)

[Loading previously acquired data 7](#_Toc141710426)

[Running acquisitions 9](#_Toc141710427)

[Acquisition methods 9](#_Toc141710428)

[Source Methods 9](#_Toc141710429)

[Flow Methods 14](#_Toc141710430)

[Sink Methods 19](#_Toc141710431)

[Troubleshooting and Known Issues 23](#_Toc141710432)

[Transitioning between preview and live configurations 23](#_Toc141710433)

[Error message popup when stopping acquisition with NIDAQ 23](#_Toc141710434)

[Image stacks don't initially display in load previous experiment functionality 23](#_Toc141710435)

# Installation

STRÏMM is intended to be run from a stand-alone executable. This executable in turn calls the main jar file “mainModule”.

## Prerequisites

* JDK (Java Development Kit) v20 64bit (<https://www.oracle.com/uk/java/technologies/downloads/#jdk20-windows>)
* Microsoft Visual C++ redistributables 2015-202X. These may be automatically installed on the PC (view installed programs). If not download and install x64 versions here: <https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170>
* NI-DAQmx software suite: <https://www.ni.com/en-gb/support/downloads/drivers/download.ni-daq-mx.html#484356>

A screenshot of a computer

Description automatically generatedOnce the prerequisites above have been installed, download the latest release of STRÏMM <https://github.com/strimmcode/strimm/releases/tag/latest>. Unzip the files into a folder and run STRÏMM by double clicking on “Launch\_STRIMM.exe”. You should see the STRÏMM logo appear, and the program appear shortly after.

A logo of a camera

Description automatically generated

# Change Log

STRÏMM is currently at version v1.0.81. A full, detailed change log can be found by looking at the source repository commit history <https://github.com/strimmcode/strimm/commits/main> but below is a condensed and more readable version of the most significant changes.

1.0.81

* Implementation of histogram flow and window
* Implementation of feature to change exposure on image feed during preview.
* Fixed issue #17 (<https://github.com/strimmcode/strimm/issues/17>)

1.0.7

* Allowed flows to accept multiple inputs.

1.0.6

* Upgraded version of Kotlin from v1.2.61 to v1.8.22. Upgraded the version of the JVM from v1.8 to v17. Note – JDK v20 should be downloaded for running STRÏMM from this version onward.
* Recompiled C++ projects (“Test” and “Launch\_STRIMM”) with Visual Studio 2022

1.0.51

* Added functionality to plot event markers with associated trace data when loading previous experiment.
* Fixed issue #13 (<https://github.com/strimmcode/strimm/issues/13>)
* Fixed issue #8 (<https://github.com/strimmcode/strimm/issues/8>)

1.0.4

* Added image splitting capabilities (ImageSplitterFlow)
* Fixed issue #7 (<https://github.com/strimmcode/strimm/issues/7>)

1.0.3

* Added load existing experiment functionality (review data from a previously acquired acquisition).

1.0.0-1.0.2

* Initial commit
* Various non-functional code changes
* Fixed issue with bRepeat flag not working if set to “false”

# Concepts And Main Features

## Akka Framework

STRÏMM is designed to take data from multiple heterogeneous data sources, transform that data, and display and store it. To do this, acquisitions use a framework called Akka. The Akka framework (<https://doc.akka.io/docs/akka/current/stream/index.html>) has many features but abstracts data streams into concepts of sources, flows, and sinks. These are collectively called nodes. A source represents a device (physical or virtual) that generates data. A flow represents a transformation to the data. A sink is where the data terminates. When data from a source is connected to a flow or sink, a connection is made with the source having an outlet, and the flow or sink having an inlet. The combination of sources, flows, and sinks, and how they connect is called a graph (sometimes also called a runnable graph). The below figure gives an example.

A diagram of a software source

Description automatically generated

There are some rules that any graph must follow:

* Sources can only ever have outlets to either flows or sinks. They cannot have inlets.
* Flows can have inlets from sources or other flows, and outlets to other flows or sinks
* Sinks can only ever have inlets from either sources or flows.
* Any outlet or inlet can be broadcasted/merged. If an outlet is connected to two or more other nodes, it is called a broadcast. If a node has two or more inlets, it is called a merge.
* All nodes, inlets, and outlets must be connected to form a closed graph (see <https://doc.akka.io/docs/akka/current/stream/stream-composition.html> for further information)

## Configuration

To facilitate the implementation of the Akka framework detailed above, STRÏMM uses a series of configuration files. These files are then loaded into STRÏMM where it will read them and run the necessary code. The main file will be a JSON file. JSON (JavaScript Object Notation) files can be read and edited like text files and provide a helpful formatting for specifying things. There will be several associated .cfg and .csv files, depending on the acquisition being run. Both these file types can be edited like text files also. With CFG files there are general CFG files, and MicroManager generated CFG files. The figure below describes how the files are arranged. Note that only the JSON file needs to be selected in STRÏMM when running an acquisition.

A diagram of a computer system

Description automatically generated

## Logging

STRÏMM will produce logs as different parts of code execute. These can be found in the /log/ folder in the main STRÏMM folder as a text file. Logs can be useful for seeing if there are any errors being thrown, however it often requires technical expertise to interpret what an error will mean.

## Hardware timing and NIDAQ interface

Hardware timing and the triggering of devices from hardware can be achieved by interfacing with a National Instruments Data Acquisition (NIDAQ) board. Currently this works with any NIDAQ board that has an internal clock. To use NIDAQ board in STRÏMM a source of type NIDAQSourceMethod must be used (see section on acquisition methods).

A series of stimulations in a CSV file must be specified in the configuration, which the NIDAQ board will use. This is called a protocol. Currently protocols are run in an episodic mode with the NIDAQ board meaning all stimulation values must be provided prior to running. It will then run the episode and stop. If a repeat flag is true then it will repeat the episode again. See NIDAQSourceMethod description for how the protocol CSV file should be structured.

## Saving Data

STRÏMM saves all data to a HDF5 (Hierarchical Data Format 5) file with a .h5 extension. The HDF5 format is intended to be flexible to heterogeneous data and is ideally suited to STRÏMM’s operating. The structure of a h5 file is very much like a folder structure, with parent and child nodes representing groupings, before the data itself which is termed a “dataset”.

STRÏMM separates data into either image data or trace data, depending on the data source. This will determine the exact contents of the node groupings in the h5 file. Trace data will have a single dataset called “data” in a node called “traceData”. Image data will have a single dataset called “data” in a node called “traceData” that contains timing information, and image data will have one dataset per frame in a node called “imageData”. Note that many of the group nodes will have attributes containing various important pieces of metadata.

The easiest way to view data in a h5 file is to use download and use HDFView (<https://www.hdfgroup.org/downloads/hdfview/#download>) which will make it easy to view the file contents in general (it has limited data plotting/viewing functionality).

The structure of the h5 file STRÏMM creates follows this schema:

File name

-> Dataset group

-> Instance (a number)

-> Image Data group

-> Image Datasets (one frame=one dataset)

-> Trace Data group

-> Trace Dataset

The figure below shows an example of this schema:

A screenshot of a computer

Description automatically generated

## Loading previously acquired data

You can view previously acquired data in STRÏMM using the load previous experiment feature. This is accessed via the button  in the main GUI. You can select and h5 file that is present in the main STRÏMM folder. Note – there is a known issue where image datasets initially do not appear when loaded, this feature is detailed at <https://github.com/strimmcode/strimm/issues/18> and a work around is to click each image display once.

A screenshot of a computer

Description automatically generated

# Running acquisitions

## Acquisition methods

An acquisition method is code that will be executed with a specified source, flow, or sink.

Note – all CFG files that are not MicroManager generated CFG files follow a simple “property,value” format.

### Source Methods

#### MMCameraSource

The MMCameraSource is the method that should be used for any cameras. An additional CFG file is also necessary for camera settings.

Output can go to: ImageSplitterFlow, ROIFlowMethod, EventMarkerFlow, HistogramFlow, SinkImageJDisplayMethod, SinkSaveMethod

JSON source fields

|  |  |  |  |
| --- | --- | --- | --- |
| Config Field Name | Description | Example Value | Necessary or optional? |
| sourceName | The name of the source, must be unique to all other source, flow, and sink names | “lowResCamera” | Necessary |
| sourceType | The name of the source method associated with this source | “MMCameraSource” | Necessary |
| sourceCfg | The CFG file containing additional config settings | “lowResCameraConfig.cfg” | Necessary |
| isTimeLapse | Determines if the source method will tick (repeat) at an interval (set by intervalMs) (True), or as fast as possible (False). True by default. | True | Optional but default used if not specified. |
| intervalMs | The interval, in milliseconds, by which the source method should tick (repeat). Used only when isTimeLapse is True. Note that often the device associated with the source e.g. a camera will dictate that actual interval. | 10.0 | Optional but default of 0.0 (as fast as possible) used if not specified |

CFG file fields

|  |  |  |  |
| --- | --- | --- | --- |
| Config Field Name | Description | Example Value | Necessary or optional? |
| isImageSnapped | Tells STRÏMM to use the snap() method of the MMCore. Can be True or False. If this is True, “isTriggered” must be False and vice versa. Note – the snap() method is slower than a triggered acquisition and is not recommended for high frame rates | True | Necessary |
| isTriggered | Tells STRÏMM to run an acquisition using the MMCore. Can be True or False. If this is True, “isImageSnapped” must be False and vice versa. Requires an additional CFG file (see below) | True | Necessary |
| exposureMs | The exposure for the camera in milliseconds | 100.0 | Necessary |
| framesInCircularBuffer | How many frames should be in the MMCore’s circular buffer. Most use cases in STRÏMM will use -1 which translates to “as fast as the frames come” | -1 | Necessary |
| MMDeviceConfig | The name of the MicroManager generated CFG file | PVCam.cfg | Necessary |
| pixelType | The pixel type to describe the bytes per pixel. Can either be “Byte” (bit depth=8), “Short” (bit depth=16), or “Float” (bit depth=32) | Short | Necessary |
| numChannels | The number of channels you want to use for this camera | 1 | Necessary |
| X | The start x pixel to take a subset of the total pixels | 0 | Necessary |
| y | The start y pixel to take a subset of the total pixels | 0 | Necessary |
| w | The width in pixels to take a subset of the total pixels | 512 | Necessary |
| h | The height in pixels to take a subset of the total pixels | 512 | Necessary |

Additional CFG file if isTriggered=True and isImageSnapped=False

This additional CFG file must be named the same as the CFG file specified for MMDeviceConfig and must be placed in /DeviceAdapters/CameraMMConfigsTrigger/

|  |  |  |  |
| --- | --- | --- | --- |
| Config Field Name | Description | Example Value | Necessary or optional? |
| TriggerMode | Specify the type of trigger mode for the camera to use. Trigger mode options will vary from camera to camera. These properties must use the same wording as they would if they were specified in MicroManager. | EdgeTrigger | Necessary |

#### NIDAQSourceMethod

The NIDAQSourceMethod is the method that should be used when you want to use a NIDAQ board.

Output can go to: NIDAQBuffer\_to\_SignalBufferFlow, SinkTraceMethod, SinkSaveMethod

Corequisite(s): Must connect to NIDAQBuffer\_to\_SignalBufferFlow flow before any sink methods are used.

JSON source fields

|  |  |  |  |
| --- | --- | --- | --- |
| Config Field Name | Description | Example Value | Necessary or optional? |
| sourceName | The name of the source, must be unique to all other source, flow, and sink names | “NIDAQ6343” | Necessary |
| sourceType | The name of the source method associated with this source | “NIDAQSourceMethod” | Necessary |
| sourceCfg | The CFG file containing additional config settings | “NIDAQSource.cfg” | Necessary |

CFG file fields

|  |  |  |  |
| --- | --- | --- | --- |
| Config Field Name | Description | Example Value | Necessary or optional? |
| deviceID | The number of the NIDAQ device (use NI Device Monitor to find this, it is the number in the device name like “Dev 1”) | 1 | Necessary |
| deviceName | The name of the device as it appears in NI Device Monitor, usually this is always “Dev” | Dev | Necessary |
| szCsv | The full path location of the CSV file containing the stimulations | C:/MyFolder/episodicProtocol.csv | Necessary |
| bCompound | If the specified protocols are compound i.e. multiple protocols run one after another. If a protocol is not a compound protocol it is called a simple protocol. Can be true or false. | false | Necessary |
| bRepeat | Flag to set if the protocol should repeat. Can be true or false | false | Necessary |
| bStartTrigger | Specifies whether a protocol uses a start trigger. Can be true or false. Is used in conjunction with bRisingEdge. | false | Necessary |
| bRisingEdge | Specifies whether the start trigger uses a rising edge or falling edge. True for rising edge, false for falling edge | false | Necessary |
| pFIx | The port and pin on the NIDAQ board to be used as the trigger pin | 0 | Necessary |
| timeoutSec | The timeout for running protocols on the NIDAQ board. -1 indicates infinite timeout. | -1.0 | Necessary |
| minV | The minimum allowed voltage the NIDAQ board will use | -10.0 | Necessary |
| maxV | The maximum allowed voltage the NIDAQ board will use | 10.0 | Necessary |
| timingMethod | Different timing methods for protocols. 0=simple protocols are played consecutively, 1=simple protocols are triggered by the previously defined trigger signal.  2=PC timing – the program will wait until it arrives at the stated time from the beginning of the program. Typically timingMethod 0 is mostly used. | 0 | Necessary |

Stimulation CSV file

The stimulation CSV file will be read by STRÏMM and the values within it used as the direct pulses the NIDAQ board will use. The structure should be as follows.

IMPORTANT: DI0 is a reserved channel so the first useable DI channel is DI1.

IMPORTANT: At least one AI, AO, DI, and DO channel must be specified, event if not all are used.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Column name | Column number | Description | Example value | Necessary or optional? |
| numSamples | 1 | The exact number of samples each channel will use. All channels must use the same number of samples. | 100 | Necessary |
| sampFreq | 2 | The frequency in Hz the NIDAQ board will use the samples. | 50 | Necessary |
| DIPort | 3 | The digital in port number | 0 | Necessary |
| DOPort | 4 | The digital out port number | 0 | Necessary |
| Any AO, AI, DO, DI channels | 5..n | A column representing the samples that will be used for that particular channel. Column must start with the channel name like AO, AI, DO, DI followed by a number starting from 0 e.g. AI0, DO2. If the channel is an input channel (AI or DI) all values must be zero. All values are the pure voltages that will be used. | A column of numbers. Number of rows=numSamples | Necessary |

An example spreadsheet providing 100 samples at 50Hz:

A screenshot of a graph

Description automatically generated

### Flow Methods

#### HistogramFlow

The HistogramFlow will take 1 or more image feeds and calculate a histogram based on pixel intensities. The histogram flow is designed to be agnostic to the incoming images and the images can come from any source or flow that produces an image. Image data will be binned into 256 bins regardless of bit depth, and then the counts normalised between 0 and 1.

Input can come from: MMCameraSource, ImageSplitterFlow

Output can go to: HistogramSink

JSON source fields

|  |  |  |  |
| --- | --- | --- | --- |
| Config Field Name | Description | Example Value | Necessary or optional? |
| flowName | The name of the flow, must be unique to all other source, flow, and sink names | “histogramFlow” | Necessary |
| flowType | The name of the flow method associated with this source | “HistogramFlow” | Necessary |
| inputNames | The names of sources or flows that connect to this flow | “lowResCamera”,  “highResCamera” | Necessary |

#### ImageSplitterFlow

The ImageSplitterFlow will take 1 image feed and take a subset of the images based on specified coordinates. These coordinates will be a start position – *x*,*y* and a width *w* and height *h*.

Note – the x,y,w,h coordinates are *relative* to the incoming image. For example if the maximum image size on the camera is 3200px x 3200px, but the source provides an image 512px x 512px, then the *w*,*h* coordinates in the ImageSplitterFlow cannot exceed 512 and 512.

Input can come from: MMCameraSource, ImageSplitterFlow

Output can go to: ImageSplitterFlow, ROIFlowMethod, EventMarkerFlow, HistogramFlow, SinkImageJDisplayMethod, SinkSaveMethod

JSON source fields

|  |  |  |  |
| --- | --- | --- | --- |
| Config Field Name | Description | Example Value | Necessary or optional? |
| flowName | The name of the flow, must be unique to all other source, flow, and sink names | “lowResCamSplitLeft” | Necessary |
| flowType | The name of the flow method associated with this source | “ImageSplitterFlow” | Necessary |
| inputNames | The name of the source or flow that connect to this flow | “lowResCamera” | Necessary |
| splitCoordinates | The *relative* coordinates of the subset image | [{  “x” : 0,  “y” : 0,  “w” : 100,  “h” : 200  }] | Necessary |

#### ROIFlowMethod

The ROIFlowMethod will take an image and calculate the values or one or more ROIs on the image. The ROIs will be specified on the associated SinkImageJDisplayMethod. The associated SinkImageJDisplayMethod must be specified in an additional config file for ROIs to be calculated and shown. The ROI calculation is currently to calculate the average pixel intensity for pixels within each ROI.

Input can come from: MMCameraSource, ImageSplitterFlow

Output can go to: EventMarkerFlow, SinkTraceMethod, SinkSaveMethod

JSON source fields

|  |  |  |  |
| --- | --- | --- | --- |
| Config Field Name | Description | Example Value | Necessary or optional? |
| flowName | The name of the flow, must be unique to all other source, flow, and sink names | “lowResCamROIFlow” | Necessary |
| flowType | The name of the flow method associated with this source | “ROIFlowMethod” | Necessary |
| flowCfg | The CFG file containing additional config settings | “lowResROIFlow.cfg” | Necessary |
| inputNames | The name of the source or flow that connect to this flow | “lowResCamera” | Necessary |

CFG file fields

|  |  |  |  |
| --- | --- | --- | --- |
| Config Field Name | Description | Example Value | Necessary or optional? |
| ImageJSinkName | The name of the *image* display sink that shows the images from the image feed the ROIs were calculated | “lowResCamDisplay” | Necessary |

#### NIDAQBuffer\_to\_SignalBufferFlow

This method will allow the input and output data of the NIDAQ board to flow through the Akka stream so it can be displayed and/or stored. Must be used before displaying or storing data from NIDAQ board.

Input can come from: NIDAQSourceMethod

Output can go to: SinkTraceMethod, SinkSaveMethod

JSON source fields

|  |  |  |  |
| --- | --- | --- | --- |
| Config Field Name | Description | Example Value | Necessary or optional? |
| flowName | The name of the flow, must be unique to all other source, flow, and sink names | “NIDAQFlow” | Necessary |
| flowType | The name of the flow method associated with this source | “NIDAQBuffer\_to\_SignalBufferFlow” | Necessary |
| flowCfg | The CFG file containing additional config settings | “NIDAQFlowConfig.cfg” | Optional |
| inputNames | The name of the NIDAQSourceMethod source that connect to this flow | “NIDAQ6343” | Necessary |

#### EventMarkerFlow

This method will allow the marking of events on another image or trace. Based on specified event keys, event marker data will be stored. Currently event markers can only be single digit numbers. EventMarkerFlows will not have a display component within the Akka stream, but feedback text in the UI will appear when an event is logged from a key press.

Input can come from: MMCameraSource, ImageSplitterFlow, ROIFlowMethod

Output can go to: SinkSaveMethod

JSON source fields

|  |  |  |  |
| --- | --- | --- | --- |
| Config Field Name | Description | Example Value | Necessary or optional? |
| flowName | The name of the flow, must be unique to all other source, flow, and sink names | “lowResCamMarkerFlow” | Necessary |
| flowType | The name of the flow method associated with this source | “EventMarkerFlow” | Necessary |
| inputNames | The name of the NIDAQSourceMethod source that connect to this flow | “NIDAQ6343” | Necessary |
| eventKeys | The event keys associated with this flow that will be detected as markers when pressed during acquisition | [“1”,  “2”,  “3”] | Necessary |
| associatedSaveSink | Associates the event markers being generated with the sink of the associated data feed. This is so that it can be shown correctly when loading a previous experiment. The assocatedSaveSink must be a sink with a SinkSaveMethod sinkType. | “lowResImageSave” | Optional but necessary for load previous experiment feature |

### Sink Methods

##### HistogramSink

The HistogramSink will plot and show the histogram(s) calculated in the HistogramFlow. The HistogramSink will plot one histogram for each different image feed processed in the HistogramFlow.

Input can come from: HistogramFlow

JSON source fields

|  |  |  |  |
| --- | --- | --- | --- |
| Config Field Name | Description | Example Value | Necessary or optional? |
| sinkName | The name of the sink, must be unique to all other source, flow, and sink names | “histogramSink” | Necessary |
| sinkType | The name of the sink method associated with this sink | “HistogramSink” | Necessary |
| inputNames | The name of the HistogramFlow flow that connects to this sink | “histogramFlow” | Necessary |

##### SinkSaveMethod

The SinkSaveMethod is the main type of sink method used to save data to the h5 file. It will accept data from only 1 source or flow.

Input can come from: MMCameraSource, ImageSplitterFlow, ROIFlowMethod, EventMarkerFlow, NIDAQBuffer\_to\_SignalBufferFlow

JSON source fields

|  |  |  |  |
| --- | --- | --- | --- |
| Config Field Name | Description | Example Value | Necessary or optional? |
| sinkName | The name of the sink, must be unique to all other source, flow, and sink names | “lowResCamSave” | Necessary |
| sinkType | The name of the sink method associated with this sink | “SinkSaveMethod” | Necessary |
| inputNames | The name of the source or flow that connects to this sink | “lowResCam” | Necessary |

##### SinkImageJDisplayMethod

The SinkImageJDisplayMethod is the main type of sink method used to display image data. It will accept data from only 1 source or flow that produces image data. The SinkImageJDisplayMethod will create an ImageJ dataset with the incoming image data and then create an ImageJ image display with it. The image display will always be a stack with only 1 image that is constantly updated.

Input can come from: MMCameraSource, ImageSplitterFlow

JSON source fields

|  |  |  |  |
| --- | --- | --- | --- |
| Config Field Name | Description | Example Value | Necessary or optional? |
| sinkName | The name of the sink, must be unique to all other source, flow, and sink names | “lowResCamDisplay” | Necessary |
| sinkType | The name of the sink method associated with this sink | “SinkSaveMethod” | Necessary |
| sinkCfg | The CFG file containing additional config settings | “lowResCamSink.cfg” | Necessary |
| inputNames | The name of the source or flow that connects to this sink | “lowResCam” | Necessary |

CFG file fields

|  |  |  |  |
| --- | --- | --- | --- |
| Config Field Name | Description | Example Value | Necessary or optional? |
| w | The width of the image being displayed. | 512 | Necessary |
| h | The height of the image being displayed. | 512 | Necessary |
| pixelType | The pixel type to describe the bytes per pixel. Must be the same as the pixelType in MMCameraSource. Can either be “Byte” (bit depth=8), “Short” (bit depth=16), or “Float” (bit depth=32) | Short | Necessary |
| numChannels | The number of channels you want to use for this camera | 1 | Necessary |
| previewInterval | The interval, in milliseconds the image display is updated. | 100 | Necessary |
| lut | The name of the look up table. Must be present in /luts/ folder in main STRÏMM folder. Can be empty if no look up table needed. | glow.lut | Optional (property must be present but can be empty) |
| roiSz | The name of the CSV or zip file containing ROI data. File must be present in the main STRÏMM folder | cameraROIs.csv | Optional (property must be present but can be empty) |
| normalise | True or False flag if you want the image to be normalised. Must be true to use “autostretch” function on HistogramSink | True | Necessary |

ROI CSV file

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Column name | Column number | Description | Example value | Necessary or optional? |
| name | 1 | The name of the ROI. Can be anything. Not used by the STRÏMM program. | Left ROI | Necessary |
| type | 2 | The type of ROI. 1=rectangular ROI, 2=elliptical ROI | 1 | Necessary |
| x | 3 | The x coordinate of the ROI | 0 | Necessary |
| y | 4 | The y coordinate of the ROI | 0 | Necessary |
| w | 5 | The width of the ROI | 50 | Necessary |
| h | 6 | The height of the ROI | 100 | Necessary |

Example of ROI CSV file:

A screenshot of a table

Description automatically generated

# Troubleshooting and Known Issues

Current bugs and known issues are logged in the “Issues” section of the GitHub repository: <https://github.com/strimmcode/strimm/issues>

### Transitioning between preview and live configurations

Currently when transitioning between preview and live configurations an error may occur that prevents the acquisition from running. The work around for now is to close and restart STRÏMM and start the configuration in one mode.

This issue is detailed here: <https://github.com/strimmcode/strimm/issues/10>

### Error message popup when stopping acquisition with NIDAQ

When running an acquisition with a NIDAQSourceMethod source, when stopping the acquisition, the following error message appears:

A screenshot of a computer

Description automatically generated

This error message can be ignored as it does not prevent the data from being saved correctly.

This issue is detailed here: <https://github.com/strimmcode/strimm/issues/20>

### Image stacks don't initially display in load previous experiment functionality

This is a known bug with ImageJ. Work around is to click on the image feed once and it should appear. Alternatively, one can press plus (+) or minus (-) buttons.

This issue is detailed here: <https://github.com/strimmcode/strimm/issues/18>