Inlinino Documentation

Release 1.0.1

Nils Haentjens

Sep 30, 2020

Contents

Inlinino is a simple data logger. It was developed to log measurments from scientific instruments during extended periods of time (months) but can be used or adapted to your need. It supports any instrument with serial or analog interface as long as there is an API for it or you can make one.

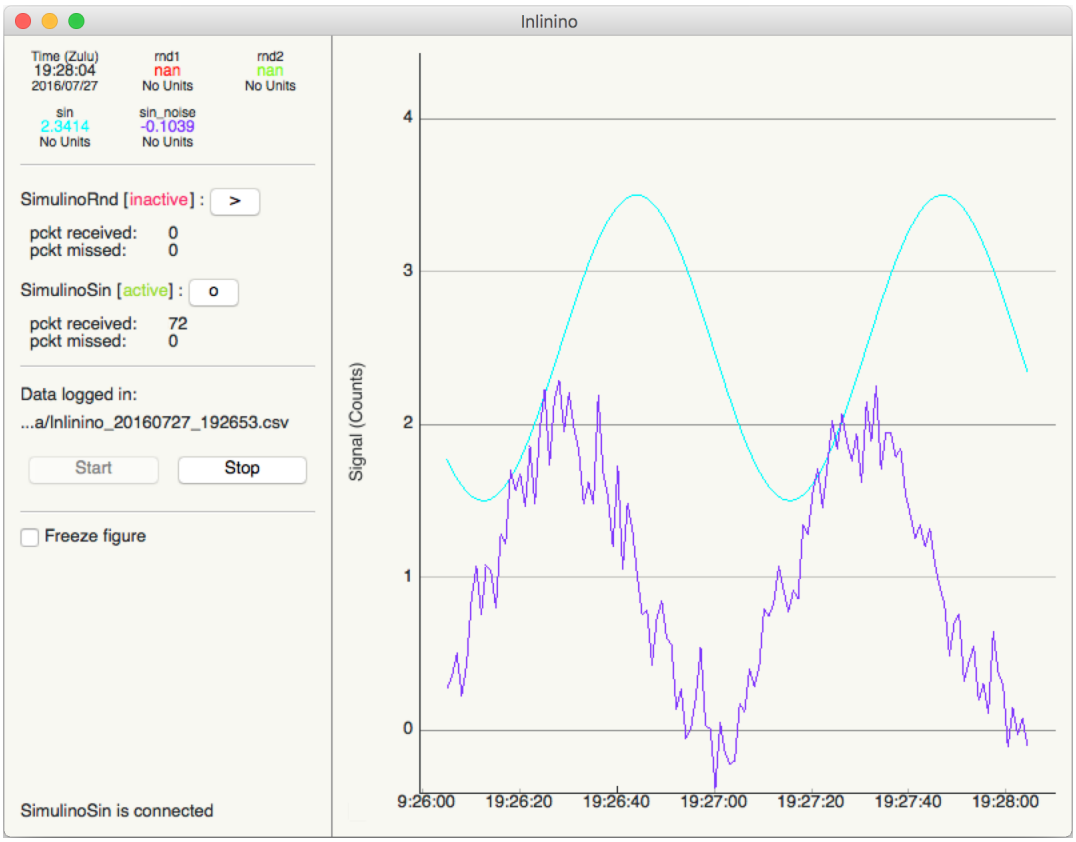
Few instruments from WETLabs are implemented and have been extensively tested during cruises: ECO-BB3, ECO-BB9, and WSCD. To log signal from analog instruments an Arduino, or an ADS-1X15 interface are available.

The configuration of the software is mostly done with json files but some instruments will require adding some python class (just basic coding notions are needed, we keep it simple).

Data recorded by Inlinino is saved hourly (by default) in csv files. Post processing can be done through the software it-self (after updating the code) or with any data analysis tools such as Python, R, or Matlab.

There are two interfaces available to date: a friendly Graphical User Interface also known as a GUI (default) and a Command Line Interface (CLI) for use on servers or power use.

The software is written in python 3+, on top of pySerial and numpy. The GUI is based on PyQt. The software was tested on both Windows 7 and OSX.



CHAPTER 1

Index

1.1 Quick Start

This document will show you how to get up and running with Inlinino. You will have inlinino installed in 10 minutes and start logging data in 15 !

If you are already using python 3.x, go ahead to those setup instructions.

Table of Contents

1.1.1 Easy installation

The quick and easy installation is available on Windows only for now. If you are on OSX or Linux please go ahead to *the advance installation section*.

Download

Download the setup file for your operating system:

- Windows 32-bits
- Windows 64-bits

If you are logging data from analog instruments and therefor use an Arduino make sure the appropriate driver are installed. Instructions are available on Arduino's website.

Install

Double click on the setup file you just downloaded and follow the instructions.

1.1.2 Quick configuration

Let's just set up the instrument(s) you want to log data from.

1. Create an empty text file in `C:\Program Files\Inlinino\cfg\` named "user_cfg.json".
2. Copy the following empty instrument configuration in the file:

```
{
  "instruments":{

  }
}
```

3. Pick one or mutiple instruments bellow and add there configuration to the configuration file you just created.

- *WET Labs* ECO-Triplet
- *WET Labs* ECO-BB9
- *SBE TSG*
- *Simulator* (for testing)

4. Your configuration file will look like this if you choose to log data from a TSG and a BBFL2.

```
{
  "instruments":{
    "TSG":{
      "module":"SBE",
      "name":"TSG",
      "variables":["T", "C", "S"],
      "units":["deg_C", "S/m", "no units"]
    },
    "BBFL2_200":{
      "module":"WETLabs",
      "name":"Triplet",
      "lambda":[660, 695, 460],
      "varname_header":["beta", "chl", "cdom"],
      "units":"counts"
    }
  }
}
```

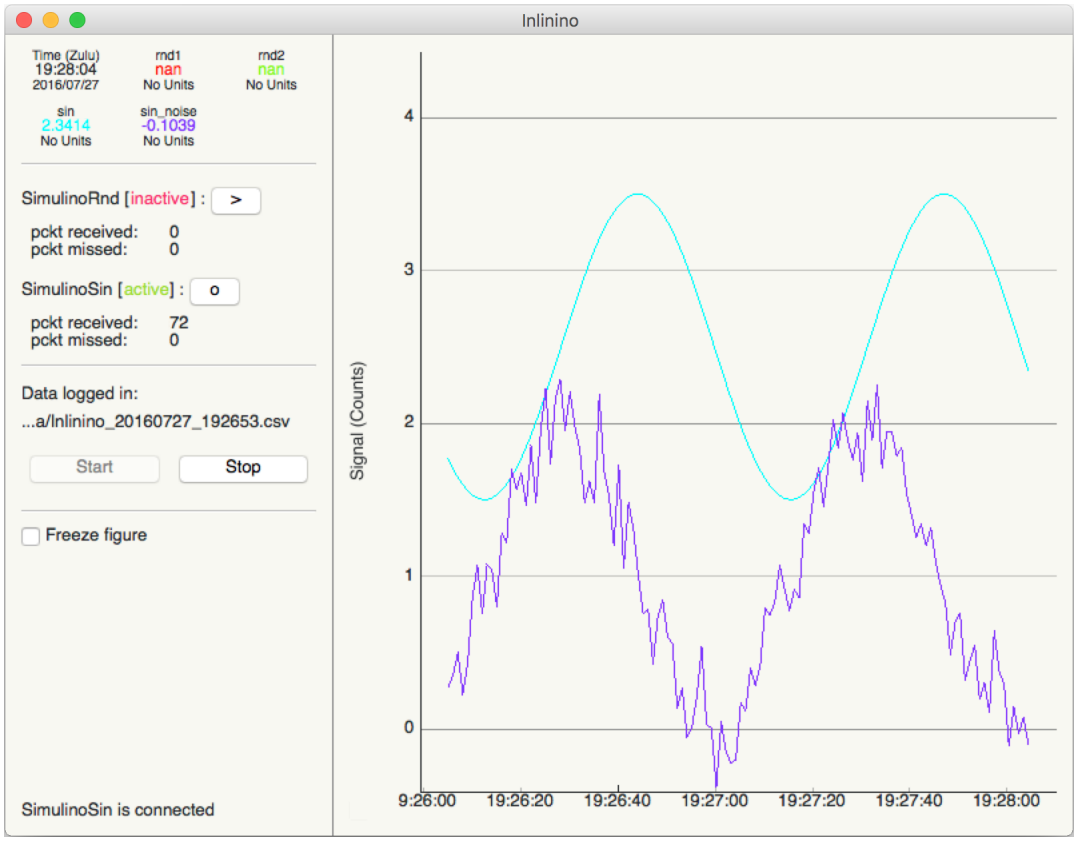
5. Edit path to the user configuration file you just created by editing line 16 of `"C:\Program Files\Inlinino\__main__.py`.

```
# Original line
Inlinino(os.path.join(sys.path[0], 'cfg', 'simulino_cfg.json'))
# User specific configuration
Inlinino(os.path.join(sys.path[0], 'cfg', 'user_cfg.json'))
```

Note: The user configuration file can have any name and can be located anywhere. By convention all Inlinino's configuration files are finishing by `_cfg.json` and are located in `"C:\Program Files\Inlinino\cfg\"`.

1.1.3 Log your first data

1. Start Inlinino by double clicking on one of the Icon shortcut (on the Desktop or in the Start menu > Programs > Inlinino). If everything went well you will see a window similar to this:



It can take up to 30 seconds for the interface to show up on slow computers.

Note: If nothing is showing up try to troubleshoot with indications available in *Common errors* section

2. Connect instruments:
    a. Click on > button on the left of each instrument.
    b. The status of the instrument will switch to "active".
    c. Data will be plotted on the figure on right and displayed on the top left.

Note: More details about the instruments configuration and interface can be found in the *Configuration* and *GUI* sections.

3. Start logging data
   a. Click on `Start` button at the bottom left to start recording the data.
   b. By default data is recorded at 1 Hz and new log file are made hourly.

Note: More details about the logger configuration and interface can be found in the *Configuration* and *GUI* sections.

1.1.4 Next step

The GUI is intuitive and very simple fill free to explore all the possibility by yourself, you cannot break anything. If you're not sure about a command, check the documentation.

Want to do more ? Look at the configuration files, few parameters can be adapted to your need there.

The instrument you would like to log data from is not available ? Add it ! There is an example on how to add the code required in the *Adding a custom instrument* section.

Note: Any difficulties ? Ideas of improvements ? Let me know, I will be happy to discuss them with you. Nils

1.2 Install

The easiest way to install Inlinino is to follow the procedure descibed in the *Quick Start* section of this documentation.

For more advanced deployment instructions are available directly on the GitHub repository of the project.

1.3 Configuration

The purpose of this section is to introduce users to the various settings available to tune Inlinino. It also explains how to add a variety of instruments to the interface in addition to plugin custom instruments.

All Inlinino's configuration is located in two json files:

- inlinino/src/cfg/default_cfg.json
- inlinino/src/cfg/<user_cfg>.json (user specific)

default_cfg.json should not be edited and is the first thing loaded when Inlinino starts. The path to the user specific configuration file should be specified as an argument when starting the application (More details in the *User configuration* section).

The configurations files contains 3 sections:

- *Application*
- *Log*
- *Instruments*

The two first are generally setup in default_cfg.json and are less susceptible to be ajusted. The *instruments* section is often the only section in <user_cfg.json>, in case any parameter from default_cfg.json needs to be modified in <user_cfg.json>, the entire section need to copied otherwise parameters will be missing and error will showup when starting Inlinino.

Table of Contents

1.3.1 Application

The main configuration of the application is done in the section `app_cfg`.

`interface: "< gui | cli >"` The two options are:

- gui: *Graphical User Interface*
- cli: *Command Line Interface*

Most of the functionnality are available on both interface. The GUI is recommended for people that want to visualize data in real-time and do not have any ressources limitations (Inlinino usually run using 1-5 % of the CPU). The CLI is meant for use on servers without X or on computer with limited ressources.

Note: A web interface might come out in further realise in order to monitor data from ship network.

`theme: "< inside | outside >"` *Available only for the gui interface.* The two options are:

- inside: the gui will be dark for a confortable use in lab or by night
- outside: the gui will be bright for a confortable use outside in a sunny environment

`ui_update_frequency: <float>` *Available only for the gui interface.* Frequency (in Hertz) at which the user interface is refreshing. This value is only for updating the figure and the digital display on top left. It's not taken into account for logging purposes.

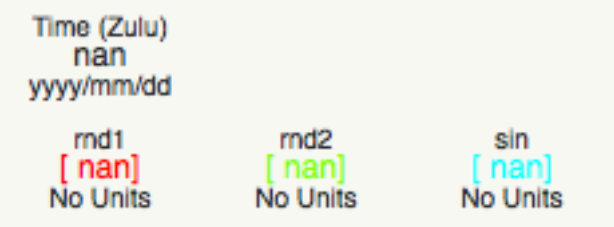
A recommended value is one 1 which correspond to updating the gui every second.

Note: If your computer is going very slowly when Inlinino is running trying to decrease the value should make it more fluid.

ui_disp_pos_shift: **<int>** *Available only for the gui interface.* This parameter allow to shift the digital display of the instrument on the top of the side bar.

- 1 would set the digital display just after the clock (see image bellow).

- 3 would set the digital display at the line below.



Note: This feature is usefull especially when using instruments with 3. 6 or 9 channels such as the ECO-Triplet or the ECO-BB9 which allow to have one of them per line.

verbosity: **< int >** This parameters take an integer between 0 and 5 0 corresponding to no informations relative to the application will be displayed 5 corresponding to a high verbosisty of the application

Note: Deprecated parameter, this parameter will be removed in futur release of Inlinino. Consider using the argument `-O` while starting Inlinino instead.

Example of configuration for the *Graphical User Interface*:

```
"app_cfg":{
  "verbosity":2,
  "interface":"gui",
  "theme":"outside",
  "ui_update_frequency":1,
  "ui_disp_pos_shift":1
}
```

Example of configuration for the *Command Line Interface*:

```
"app_cfg":{
  "verbosity":2,
  "interface":"cli"
}
```

1.3.2 Log

The log section of the configuration file concerns all the parameters of the logger, the core of Inlinino.

frequency: < float > Frequency (in Hertz) at which data is read from all the instruments and saved in the log file.

If multiple instruments are running at different frequency the logger frequency should be set to the maximum frequency in order to record all the data.

Note: If an instrument did not update its cache between two reading of the logger a *NaN* value will be kept at that time.

---

interval_write: < float > Interval at which data is written on the hard drive, a small interval will be hard for the hard drive whereas a too big interval might drive to a lost of more data in case of sudent power off of the computer.

The interval units are `1/frequency` second(s). For example:

```
frequency = 1
interval_write = 60
--> buffer is written every 60 seconds on the hard drive
```

Note: OSX seems to keep in a buffer the data up to when the file is closed, this result in lost of data in case of sudent power off of the computer. The maximum lost of data in case of unpredicted behaviours on OSX is determine by the length of the file. Windows is not affected by this issue.

---

buffer_size: < float > Size of the buffer (in seconds) to keep data in memory. This determine how much data will be plotted on the figure of the interface.

The `buffer_size` needs to be stricly greater than `interval_write`

The `buffer_size` units are `1/frequency` second(s). For example:

```
frequency = 1
buffer_size = 120
--> the figure of the GUI will display the last 2 minutes of data collected
```

Tip: If using Inlinino with the *Command Line Interface* the best size for the ring buffer is: `buffer_size = interval_write + 1`

Note: using a big ring buffer will make the application very slow as there will be more points to plot on the figure.

Note: Inlinino use a ring buffer in order to be able to run for weeks in a row.

---

length: < float > The length parameter caracterize how long the log files will be. The units are minutes.

The length parameter should be : `length >> interval_write`

header: "< string >" The header parameter indicate what should be the prefic of the log file name. This parameter can be modified (in the *GUI* or *CLI*) when the application is running.

Note: Log file name follow this syntax `<header>_YYYYMMDD_HHMMSS.csv`. For example a file with the

header Inlinino created July 9, 2016 at 16:01:00 UTC would be named: `Inlinino_20160709_160100.csv`.

`path: "< string >"` Path to where the log files will be saved. This parameter can be changed in the *GUI* when the application is running.

Note: On Windows, the path need to include two backslashes as they are special characters in JSON. For example: `C:\\Data\\Inlinino"`.

Example of log configuration:

```
"log":{
    "frequency":1,
    "interval_write":60,
    "buffer_size":120,
    "length":60,
    "header":"Inlinino",
    "path":"data"
  }
```

1.3.3 Instruments

This section explains how to setup instrument in Inlinino configuration files. Multiple instruments can be connected at once.

The following instruments are already implemented in Inlinino:

- *Analog* connection:
  - WET Labs WSCD: CDOM FLuorometer
- *Serial* connection:
  - SBE:
    - 45 Micro TSG: Thermosalinograph
  - WET Labs:
    - ECO-Triplet: 3 channels for backscatterers and/or fluorometers
    - ECO-BB9: 9 wavelength backscatterer
- *Simulators* (for testing purposes):
  - Random Gaussian Generator
  - Sinusoid with noise

Inlinino is not limited to those and is meant to log data from any kind of instruments for which a python API can be made. The *Adding a custom instrument* section is here to get started with that.

All the instruments should include the following parameters:

`instruments:{ < string > : {} }` The name of the instrument is in the instruments array and correspond to the name of a name-value set. The value of an instrument contains all its parameters.

Important: All the instrument in one instance of Inlinino must have unique names.

Note: It's a good practice to include the serial number of the instrument in his name as it helps to know where the data come from during post-processing. Example of instrument names:

```
BB9_007
BB3_001
TSG_254
```

module: **"<string>"** The module parameter refers to which parent-class needs to be loaded to communicate with an instrument. It usually correspond to the brand of the instrument. Module available are:

- Arduino
- SBE
- Simulino
- WETLabs

name: **"<string>"** The name parameter refers to which child-class needs to be loaded to be able to communicate with the instrument. Usually it correspond to the name of the instrument. Name are specific to each module:

- Arduino + ADS1015 + Board
- SBE + TSG
- Simulino + Gauss + Sinus
- WETLabs + BB9 + Triplet

Other fields are present for most of the instruments:

units: **"< string >"** Units to display in the log file

variables: **{}** Variables are used to specify one or more inputs from the instrument. Each variable/input can contain multiple settings.

frequency: **<int>** Frequency (in Hertz) at which the instrument is expected to run.

Example of instruments configuration:

```
{
  "instruments":{
    "SimulinoRnd":{
      "module":"Simulino",
      "name":"Gauss",
      "...":"..."
    }
  }
}
```

Availalble instruments

This section will help you to set the instruments that need to be logged by Inlinino, it also cover the parameters available for each instruments.

Note: Configuration files for most of the instruments implemented are available in `inlinino/src/cfg/` or `C:\Program Files\Inlinino\cfg\` if you installed the application through the Windows Installer (see *Quick*

Start).

Analog

In order to log data from instruments communicating through analog channels, a data acquisition (DAQ) device with a voltage range and resolution sufficient must be used. To date, only the *PASC* data acquisition system is available with Inlinino.

The parameters required for an analog instrument are:

- module
- name
- frequency
- gain (for ADS1X15 only)
- variables
  - pin
  - units

`frequency: < int >` Frequency (in Hertz) at which the Arduino will be reading and reporting voltage.

Note: Theorical maximum frequency are:

Uno	ADS-1015	ADS-1115
9600	3300	860

Inlinino maximum frequency (taking into account conversion delay):

Number	Uno	ADS-1015		ADS-1115	
of PIN	*SE*	*SE*	*DIF*	*SE*	*DIF*
1	50	1000	500	125	62
2	25	500	250	62	31
3	16	333	•	41	•
4	12	250	•	31	•
5	10	•	•	•	•

`gain: < int >` *Available only for the ADS1X15 interface.*

Set gain of ADS-1x15.

The ADC input range (or gain) can be changed via this parameter.

Available options are:

Gain	VDD	Resolution (1 bit = x mV)		
x	(+/- V)	Uno	ADS-1015	ADS-1115
2/3	6.144	•	3	0.1875
•	5.0	4.88	•	•
1	4.096	•	2	0.125
2	2.048	•	1	0.0625
4	1.024	•	0.5	0.03125
8	0.512	•	0.25	0.015625
16	0.256	•	0.125	0.0078125

Note: A gain of two third is set with `"gain":23`.

Warning: Never exceed the VDD +0.3V ! Exceeding the upper or lower limits may damage a channel of your ADC or destroy it ! Be carefull with this setting, be conservative and start with a gain of 2/3 (`"gain":23`) for an input of +/- 6.144 V

Note: Gain is displayed on the digital display on the top left of the GUI. Gain setting is recorded in the output log file with the units.

`variables: {}` Each pin connected to the board need to be declared in this section. Each variable has a name, a pin name and units.

`pin: "< string >"` Set which pin to read measurments from.

pin single ended options are:

- SIN_A0
- SIN_A1
- SIN_A2
- SIN_A3
- SIN_A4
- SIN_A5 (available only on Arduino Uno)

pin differential connections options are (available only on ADS-1X15):

- DIF_A01
- DIF_A23

Example of configuration for logging data of an analog fluorometer, the WSCD from WET Labs. The instrument output is 12 bit 0-5 Volts, as we are taking measurements in very clear water, signal should never go above 3 Volts. In order to match the resolution of the instrument, an ADS-1015 is used with a gain setting of 1x and a frequency of 1 Hz (as the instrument operates at 1 Hz). The <user_cfg.json> file look like:

```
"instruments":{
  "WSCD_859":{
    "module":"Arduino",
    "name":"ADS1015",
    "frequency":1,
    "gain":1,
    "variables":{
      "fdom":{
        "pin":"SIN_A0",
        "units":"counts"
      }
    }
  }
}
```

Serial

Instrument with serial connection should be plug to the computer serial port or with a serial adaptator.

Inlinino is able to communicate with few specific instruments but more can be added.

Sea-Bird Electronic (SBE)

ThermoSalinoGraph (TSG) To enable the SBE 45 MicroTSG the module to use is `SBE` and the name is `TSG`. An example of TSG configuration below:

```
"TSG_001":{
  "module":"SBE",
  "name":"TSG",
  "variables":["T", "C", "S"],
  "units":["deg_C", "S/m", "no units"]
}
```

instrument name ("TSG_001") The name of the instrument can be changed to anything.

variable names The TSG must have three variables and they must be in the same order as the output of the instruments.

units The units of the three variables should be specified in the same order as the variable names.

Note: Serial connection parameters of SBE instruments

Parameter	Value
Baud rate	19200
Byte size	8
Parity	None
Stop bits	1
Timeout	1 sec

WET Labs

ECO-BB9 To enable the ECO-BB9 from WET Labs, the module should be set to `WETLabs` and the name to `BB9`. An example of configuration below:

```
"BB9_279":{
  "module":"WETLabs",
  "name":"BB9",
  "lambda":[412, 440, 488, 510, 532, 595, 660, 676, 715],
  "varname_header":"beta",
  "units":"counts"
}
```

lambda A BB9 instruments has nine wavelength, they should be specified in the same order as the data show up (reading output from the instrument on TeraTerm will help).

varname_header Prefix of the variable name. It will be used to build the name of the column in the output log file. For example the name of the first variable will be `beta412` in the case above.

ECO-Triplet To enable an ECO-Triplet (MCOMS, FLBBCD, BB3...) from WET Labs, the module should be set to `WETLabs` and the name to `Triplet`. An example of configuration below:

```
"BB3_349":{
  "module":"WETLabs",
  "name":"Triplet",
  "lambda":[470, 532, 660],
  "varname_header":"beta",
  "units":"counts"
},

"BBFL2_200":{
  "module":"WETLabs",
  "name":"Triplet",
  "lambda":[660, 695, 460],
  "varname_header":["beta", "chl", "cdom"],
  "units":"counts"
}
```

lambda An ECO-Triplet instruments has 3 channels operating at 3 wavelengths, they should be specified in the same order as the data show up (reading output from the instrument on TeraTerm will help).

varname_header Prefix of the variable name. It will be used to build the name of the column in the output log file. For example the name of the first variable will be `beta470` in the first case above.

If only one prefix is given then all the channels will have the same prefix, this is intended for the ECO-BB3 or ECO-VSF instruments.

If three prefix are given, one per channel, the the prefix will be associated to each channel. Note: the order matters and should be the same as lambda.

Note: Serial connection parameters of WETLabs instruments:

Parameter	Value
Baud rate	19200
Byte size	8
Parity	None
Stop bits	1
Timeout	1 sec

Simulators

For developping and testing purposes Inlinino embeds two kind of instrument simulators, one directly in the python code as an instrument API (*Embedded Simulino*) and another one that run's on the Arduino enable to simulate serial connection from any instrument (*Arduino Simulino*)(more information on StackOverflow).

Embedded Simulino (only software) The simulator embedded in Inlinino is available in two version a constant signal with a Gaussian noise and a Sinusoidale signal with a Gaussian noise. An example of instrument configuration bellow:

```
{
  "instruments":{
    "SimulinoRnd":{
      "module":"Simulino",
      "name":"Gauss",
      "frequency":1,
      "variables":{
        "rnd1":{
          "mu":2.5,
          "sigma":1,
          "seed":1,
          "units":"No Units"
        },
        "rnd2":{
          "mu":0.5,
          "sigma":0.2,
          "seed":2,
          "units":"No Units"
        }
      }
    },
    "SimulinoSin":{
      "module":"Simulino",
      "name":"Sinus",
      "frequency":1,
      "variables":{
        "sin":{
          "mu":2.5,
          "sigma":0,
          "seed":1,
          "start":0,
          "step":0.1,
          "units":"No Units"
        },
        "sin_noise":{
          "mu":1.0,
          "sigma":0.2,
          "seed":2,
          "start":1.57,
          "step":0.1,
          "units":"No Units"
        }
      }
    }
  }
}
```

Arduino Simulino (require an Arduino) The idea is to flash your Arduino with the code that simulates

the behaviour of other instruments. An example of code is provided in `inlinino/arduino/Simulino.cpp`. Set the instrument you would like to emulate and the mode of emulation desired commenting/uncommenting the header of the Simulino.cpp. Compile and flash the Arduino.

Add the instrument simulated by the Arduino to the user configuration file of Inlinino. Start Inlinino and you will be able to connect to the emulated instrument.

Adding a custom instrument

- This section is intended for users familiar with python (if you have some coding experience that should be enough).*

Instrument are loaded when Inlinino starts following the two parameters: module and name indicated for each instrument in the user_configuration file. The module is the class contained in the `__init__.py` file a folder in `inlinino/src/instruments/`. The name of the folder should be in small letters, whereas the name of the class is sensitive to capital letters and should be exactly matching the value of module in the configuration file. The name value correspond to a child class of the module.

In the commented code below we will see how to add a new WET Labs instument. Add a file named `custom.py` in `inlinino/src/instruments/` containing:

```
from instruments.wetlabs import WETLabs


class Custom(WETLabs):

    def __init__(self, _name, _cfg):
        WETLabs.__init__(self, _name, _cfg)

        # Add parameters specific to the instrument
        # Dark parameter
        if 'dark' in _cfg.keys():
            self.m_dark = _cfg['dark']
        else:
            print(_name + ': Missing dark')
            exit() # Exit Inlinino
        # Scale factor parameter
        if 'scale' in _cfg.keys():
            self.m_scale = _cfg['scale']
        else:
            print(_name + ': Missing scale')
            exit() # Exit Inlinino


    def UpdateCache(self):
        # readline wait for \EOT or timeout and
        data = self.m_serial.readline()
        if data:
            # data is a byte array
            data = data.split(b'\t')
            # Check size of data received
            if len(data) == 9:
                # Loop through each value to read
                for i in range(3, 8, 2):
                    # Get the name of the variable
                    j = (i - 3) // 2 # j = 1, 2, 3 when i = 3, 5, 7
                    varname = self.m_varnames[j]
```

(continues on next page)

(continued from previous page)

```
                    # Update cache of instruments
                    self.m_cache[varname] = self.m_scale[j] * (int(data[i]) - self.m_
↪dark[j])
                    self.m_cacheIsNew[varname] = True
                # Update count with number of data read
                self.m_n += 1
            else:
                # Incomplete data transmission
                self.CommunicationError('Incomplete data transmission.\n' +
                                        'This might happen on few first ' +
                                        'bytes received.\nIf it keeps going ' +
                                        'try disconnecting and reconnecting ' +
                                        'the instrument.')
        else:
            # No data
            self.CommunicationError('No data after updating cache.\n' +
                                    'Suggestions:\n' +
                                    '\t- Serial cable might be unplug.\n' +
                                    '\t- Sensor power is off.\n')
```

The three core variables used above are:

`self.m_varnames` An array of string containing the unique name of each variable

`self.m_cache[varname]` An array of floats or integers containing the value that the instrument should report (displayed and logged).

`self.m_cacheIsNew[varname]` An array of logicals specifying if the cache of the instrument was updated. It should be set to true every time `self.m_cache` is updated.

This custom instrument is loaded with the following configuration:

```
"BBFL2_201":{
  "module":"WETLabs",
  "name":"Custom",
  "lambda":[660, 695, 460],
  "varname_header":["beta", "chl", "cdom"],
  "units":"counts",
  "dark":[40, 41, 42],
  "scale":[2.5580e-07, 0.00194, 0.005216]
}
```

Note: The value recorded in for the instrument of this example will already have some processing applied. More advance modification of the code would be require to display processed data and log raw data.

Note: When developping a class for a new instrument for Inlinino, it is recommended to start Inlinino in *debug mode*.

Note: Use `print()` to display information in the terminal to help for debug, as Inlinino is written in python 3+.

1.3.4 Arguments

Few arguments passed in the python command line to start Inlinino are interpreted. They are the only configuration parameters that are not set in the configuration file.

debug mode

Inlinino use the default `__debug__` variable of python in order to switch between optimized and debug mode. The debug mode allow to display more information when the program is running in order to help while developping new features or debugging in case of unexcpected behaviour.

The global constant `__debug__` is true if Python was not started with an `-O` option (from Python documentation).

Note: It's recommend to start the application with `python -O __main__.py` as the code will be optimized and therefor use less ressources.

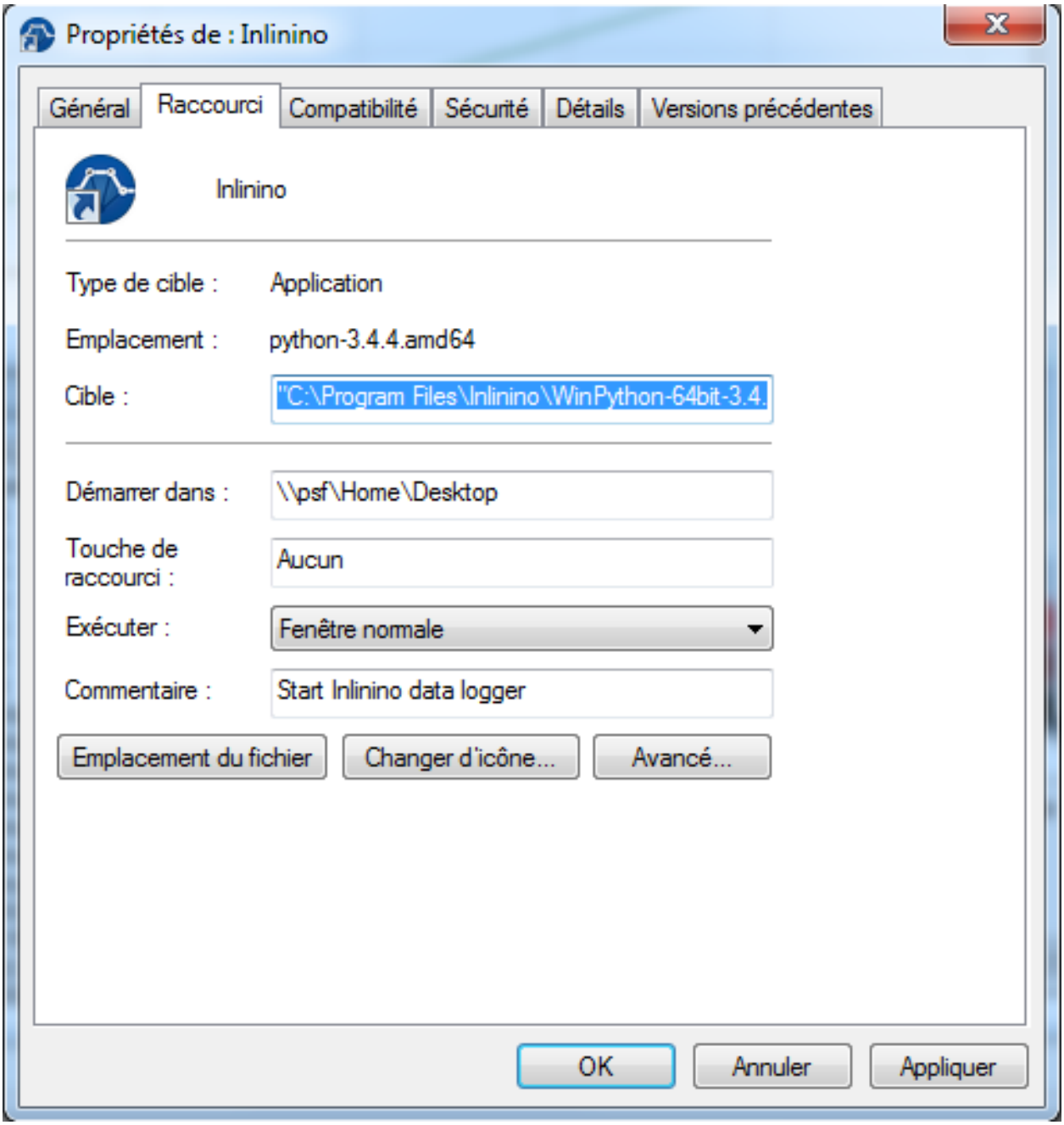
Note: If you installed the application through the Windows Installer (see *Quick Start*). *Inlinino* shortcut start the application with `pythonw -O __main__.py` whereas *Inlinino Debug*' start the application with `python __main__.py` which display a terminal window in addition to Inlinino where informations are displayed.

User configuration

The path to the user specific configuration file (<user_cfg.json>) should be passed in argument, for example:

```
python -O __main__.py user_cfg.json
```

Note: If you installed the application through the Windows Installer (see *Quick Start*). *Inlinino* shortcut start the application with `pythonw -O __main__.py` you can edit the shortcut properties (right click on the icon, choose properties) in order to setup your configuration file there, it would look like `"C:\Program Files\Inlinino\WinPython-64bit-3.4.4.2\python-3.4.4.amd64\pythonw.exe" -O "C:\Program Files\Inlinino\__main__.py" "C:\Program Files\Inlinino\cfg\user_cfg.json"`



Another way to modify path to the user configuration file is by editing line 16 of inlinino/src/__main__.py.

```
# Original line
Inlinino(os.path.join(sys.path[0], 'cfg', 'simulino_cfg.json'))
# User specific line
Inlinino(os.path.join(sys.path[0], 'cfg', 'user_cfg.json'))
```

Note: If you installed the application through the Windows Installer (see *Quick Start*). The `__main__.py` file is located in `C:\Program Files\Inlinino\__main__.py` by default.

1.3.5 Common errors

Inlinino is very sensitive, it will not like any typo in the configuration files. Those will often lead to an application not starting or undesired effect.

Application does not start Start inlinino in debug mode and look at the error messages displayed, it will tell you what part of the configuration file it does not understand.

If the error message is not helpfull, there is probably missing or extra: `{}`, `""`, or `,`.

1.4 Graphical User Interface

The purpose of this section is to go through the main functionnality of the graphical user interface also known as GUI.

The GUI is the default interface of Inlinino, if for some reason no windows are openning when you launch inlinino check that the *configuration* is set to use the gui interface, or that there is no errors in your *configuration files*.

Table of Contents

1.4.1 Instruments

This section explains how to connect, disconnect and list the instruments available in Inlinino running with the graphical user interface (GUI).

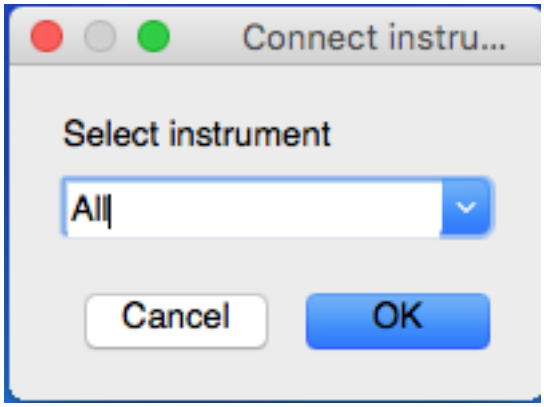
Connect

To connect an instrument to the current instance two methods are available: the menu bar (on top) or the side bar (on left). Both are doing exactly the same thing.

Note: Make sure that the instrument is properly connected to the computer and you know which COM Port it is connected to.

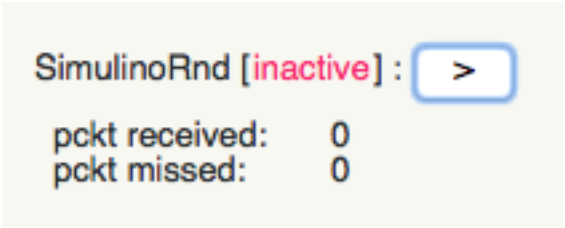
Menu Bar Connection

Click on: MenuBar > Instrument > Connect

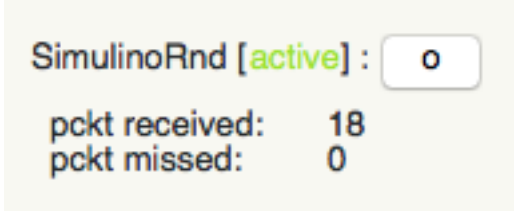


A pop-up will ask for which instrument you want to connect select the appropriate instrument and click on `[Ok]`. Note that you can connect all the instruments at once selecting the option: "All". For instruments using a serial port, another pop-up will show up and ask you for the COM port number. Select the correct one and validate your choice clicking on `[Ok]`.

Side Bar Connection



Click on: button `[>]` on the right of the instrument you want to connect. For instruments using a serial port a pop-up window will show up and ask for the COM port number. Select the correct one and validate your choice clicking on `[Ok]`.



Once the instrument is connected the status of the instrument (on right of the instrument name on the side bar) will switch from [inactive] to [active] on the side bar. The number of packets received (pckt received) will start increasing at the frequency selected in the configuration file.

The status bar will display `<instrument_name> is connected` if everything went well. In case of an error during the connection procedure, those would be indicated on the status bar too. The most common mistake is choosing the wrong COM port or the wrong instrument class.

Once the instrument is connected, the figure will start plotting data received and the digital display (top left) will start updating the values (see images below).

Note: You can cancel the procedure at anytime by clicking on the `[Cancel]` button, the connection process will be aborted.

Warning: Data is only displayed, no data is recorded. To record data look at the section *Start Logger*

Tip:

Keyboard shortcut

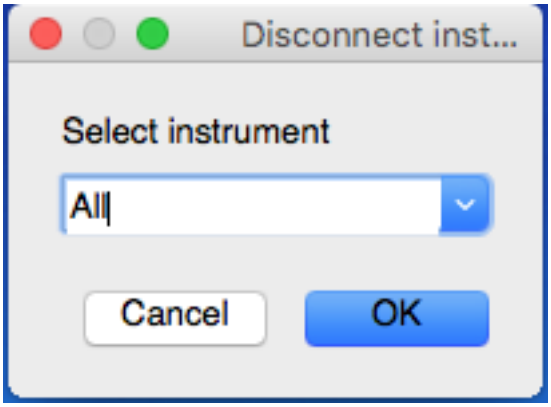
- on Windows: <Ctrl> + <S>
- on OSX: <Cmd> + <S>

Disconnect

To disconnect an instrument from the current instance two methods are available as for the connection procedure.

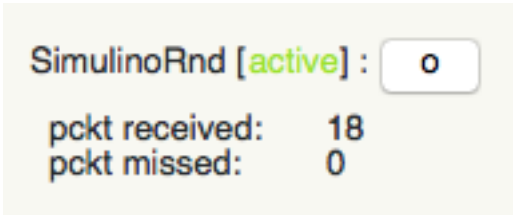
Menu Bar Disconnection

Click on: MenuBar > Instrument > Disconnect

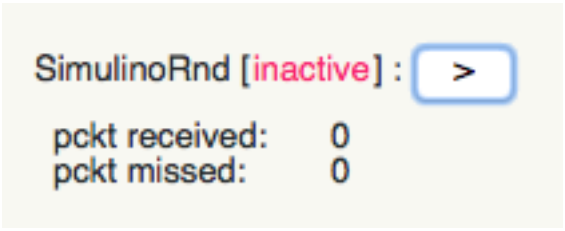


A pop-up will ask for which instrument you want to disconnect select the appropriate instrument and click on [Ok]. Note that you can disconnect all the instruments at once selecting the option: "All".

Side Bar Disconnection



Click on: button `[o]` on the right of the instrument you want to disconnect.



Once the instrument is disconnected the status of the instrument (on right of the instrument name on the side bar) will switch from `[active]` to `[inactive]` on the side bar. The number of packets received (pckt received) will stay constant.

Note: You can cancel the procedure at anytime by clicking on the `[Cancel]` button, the disconnection process will be aborted.

Warning: Data from this instrument stop displaying, but the logger is still running and recording *NaN* values for the instrument stopped. To stop recording data look at the section *Stop Logger*

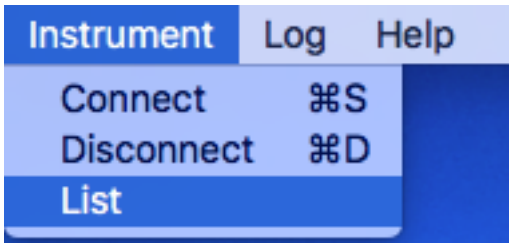
Tip:

Keyboard shortcut

- on Windows: <Ctrl> + <D>
- on OSX: <Cmd> + <D>

List

To list all instruments available in the current instance and their connection status.



Click on: MenuBar > Instrument > List

Note: This feature is coming from the CLI but is not really useful as all the instruments are already showing up on the sidebar of the GUI.

Adding/Removing

Instruments are added/removed through the configuration file. Please look at the configuration section.

Note: Instruments need to be added/removed before starting an instance of Inlinino. The list of instruments available cannot be modified once the instance of Inlinino is started.

1.4.2 Logger

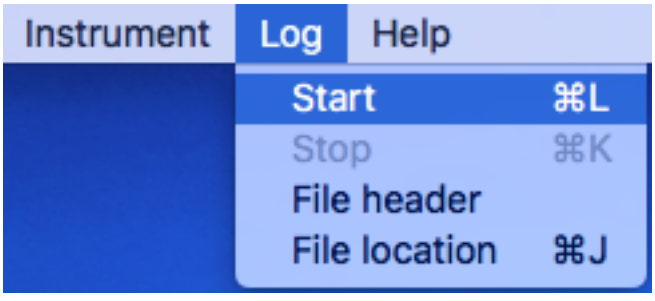
The data logger is the part of Inlinino that collects data from all the instruments connected, timestamp it and save it in a log file. The time stamp is based on the time of the computer and made in the UTC (Zulu time) time zone for easy post processing of data while crossing longitudes during the cruise. Make sure that the clock of your computer is properly set. Any change in time of the computer while logging data may produce unpredicted behaviours as most of the synchronisation processes of Inlinino are based on the computer clock. If you have any suggestion to improve this, we would be very happy to hear them.

This section explains how to log data with Inlinino. Basic functions such as starting the logger, stopping it, choosing the location of logs as well as log files header will be viewed here. For more advance configuration of the data logger please refer to the *configuration section <cfg.html>*.

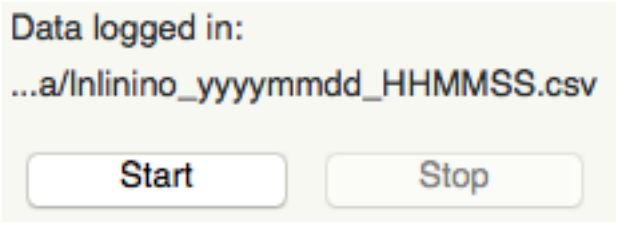
Start

To start logging data two methods are available: by the menu bar or by the side bar.

Menu Bar Logging



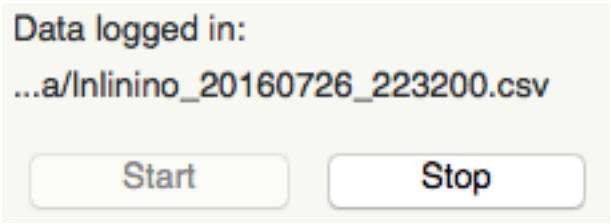
Click on: MenuBar > Log > Start



Side Bar Logging

Click on the [Start] button on the Sidebar

Once the logging starts, the `[Start]` button will be unclickable and the [Stop] will be clickable. The name of the log file change to the date and time log has started (ex: `Inlinino_20160709_091020.csv`).



Note:

If all the instruments are inactive (disconnected), a warning will pop-up to ask if you really want to start logging data. *NaN* values will be logged as all instruments are inactive. This feature can be useful if you want to log as soon as an instrument is connected.

If select `[No]`, the logger will not start, whereas selecting `[Yes]` will start the logger as usual.

Note: Data is logged in the folder specified via *Set Location*, by default it's logged in the folder specified in the configuration file.

Tip:

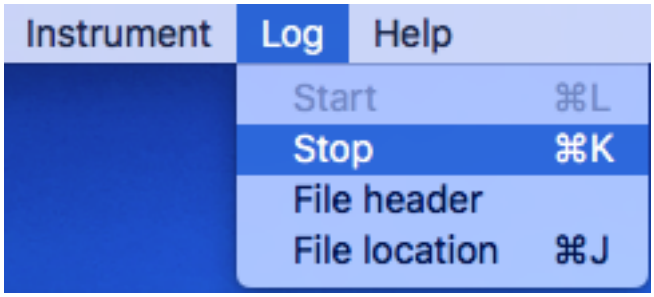
Keyboard shortcut

- on Windows: <Ctrl> + <L>
- on OSX: <Cmd> + <L>

Stop

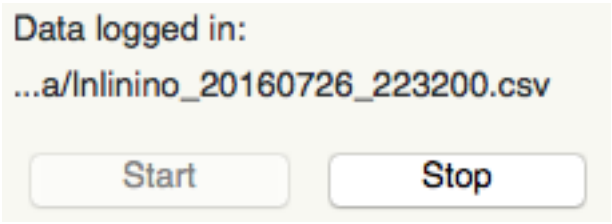
To stop logging data two methods are available: by the menu bar or by the side bar.

Menu Bar Logging



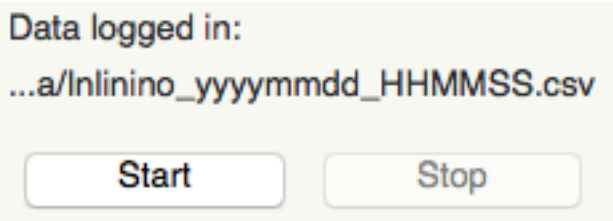
Click on: MenuBar > Log > Stop

Side Bar Logging



Click on the `[Stop]` button on the sidebar

Once the logging stops, the `[Stop]` button will be unclickable and the `[Start]` button is clickable. The name of the log file changes to `<header>_YYYYMMDD_HHMMSS.csv`



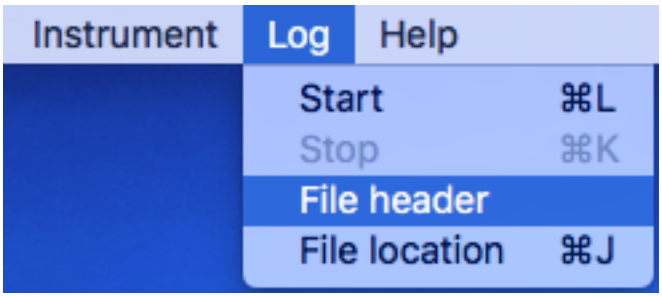
Tip:

Keyboard shortcut

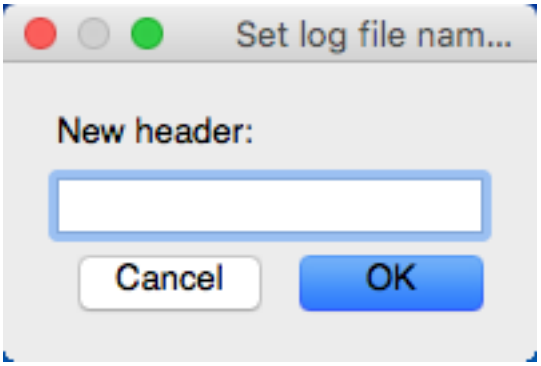
- on Windows: <Ctrl> + <K>
- on OSX: <Cmd> + <K>

Set Header

The prefix of the filename in which data is logged can be changed. The default value being the one set in the configuration file.



To change the header value click on: MenuBar > Log > File Header



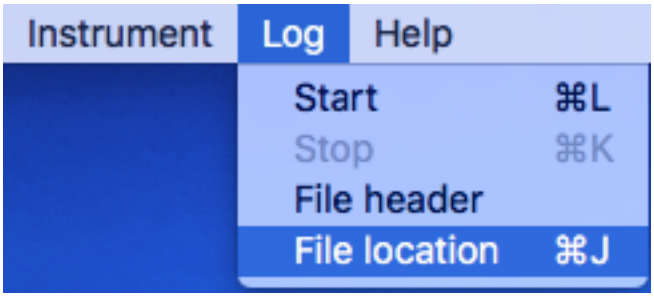
Set the appropriate prefix for the filename, use only alphanumeric characters [a-z,A-Z,0-9]. Apply change by clicking on the `[Ok]` button.

Note: You can cancel the procedure at anytime by clicking on the `[Cancel]` button, the prefix of the filename will not change.

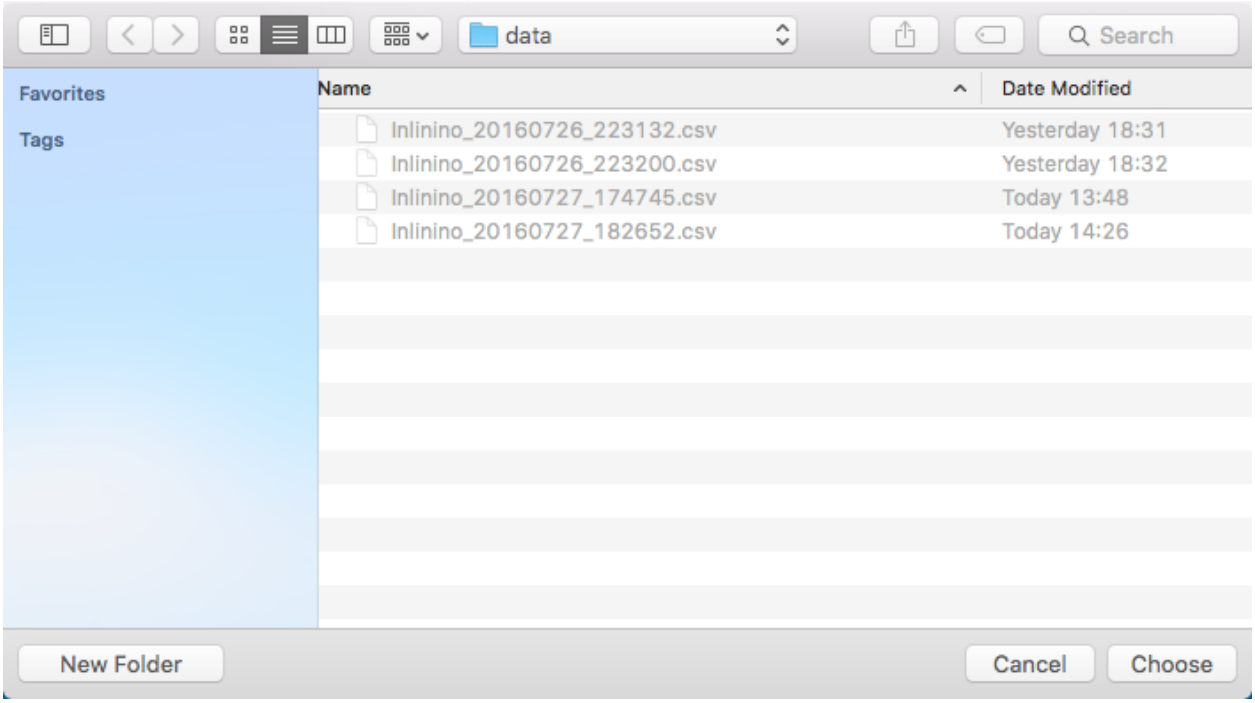
Note: If the modification is done while data is logging, it will be taken into account when a new log file is created.

Set Location

The location at which data is stored can be changed. The default value being the one set in the configuration file.



To change the header value click on: MenuBar > Log > File Location



Set the folder in which you would like to record the data and click on `[Choose]` to apply the modification.

Note: You can cancel the procedure at anytime by clicking on the `[Cancel]` button, the location of the log files recorded will not change.

Note: If the modification is done while data is logging, it will be taken into account when a new log file is created.

Tip:

Keyboard shortcut

- on Windows: <Ctrl> + <J>
- on OSX: <Cmd> + <J>

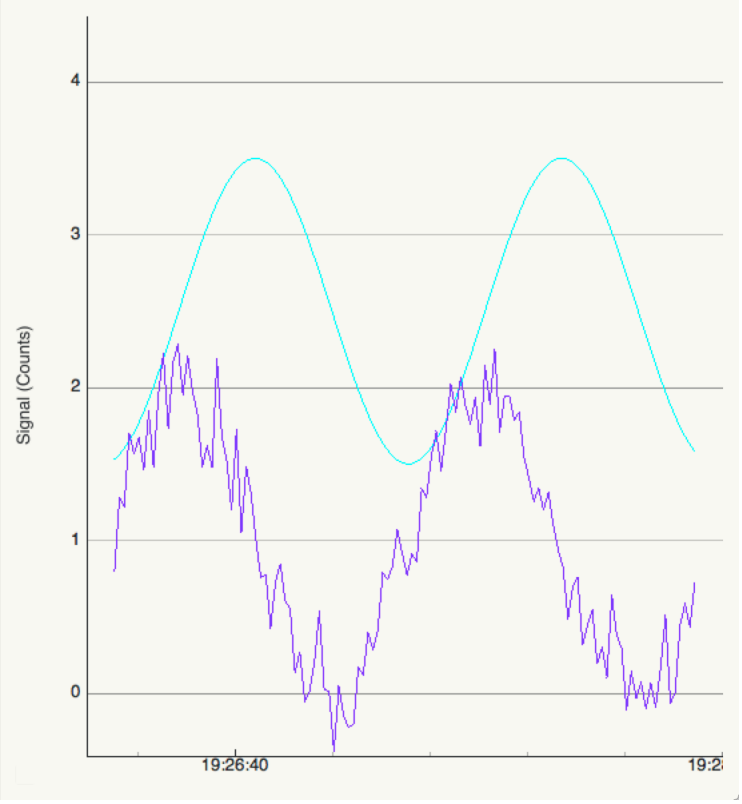
Advance configuration

Parameters included in the list below can be modified in the *Log* section.

- frequency: Frequency at which data is logged (in Hertz)
- interval_write: Interval at which data is written on the hard drive (in seconds), this is useful in case of power failure, only the last few minutes would be lost and not the entire file.
- length: Length of log files (in minutes), time after which a new log file is created.
- header: Default prefix of the filename of the logs.
- path: Default location to save logs.

1.4.3 Figure

The figure is updated in close to real-time (on the right of the window). It's the most intense part of the code in terms of computation requirements.



Freeze

The checkbox at bottom of the sidebar allow to freeze the figure to look at it or just to reduce CPU usage of the software if many variables are logged.

Note: Data is still logged normally in the background if the figure is frozen.

Y-Zoom

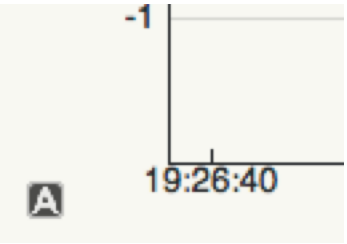
It's possible to zoom in/out on the y-axis of the figure by using the scrolling wheel of your mouse when the cursor is on top of the figure.

Y-Position

It's possible to move the graph on the y-axis by holding down the left button of the mouse and moving the cursor up or down.

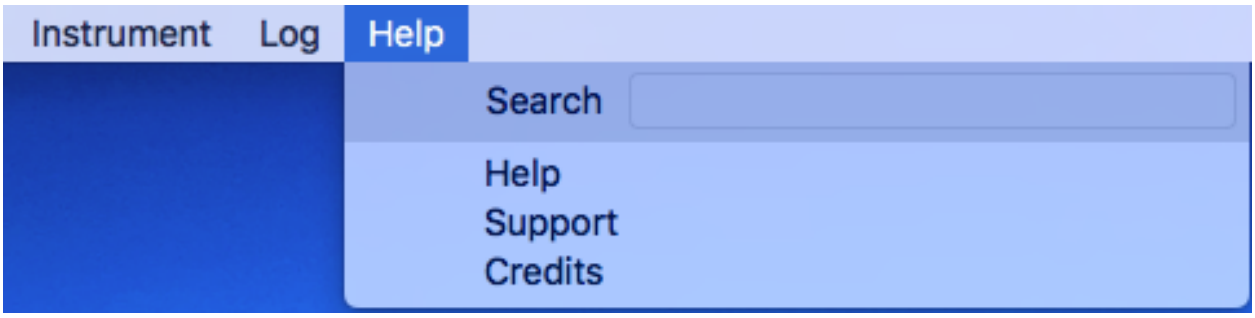
Note: Combine with the Y-Zoom this feature allow to look at the details of a specific curve.

Auto



In order to reset the setting of Y-axis to "auto", you can click on the `[A]` button at the bottom left of the figure, note that the icon will appear only if your cursor is on top of the graph and the Y-axis is not already set to "auto".

1.4.4 Help



A quick help is located in the menu bar. Support information as well as credits are available there too.

Note: On OSX, a search field appears too. It can help to find a feature in the menu bar. This element is added automatically by the OS.

1.5 Command Line Interface

The command line interface of Inlinino can be very handy when logging data on limited resources such as Raspberry Pi or when running on a server.

Start Inlinino (command `python -O src` in a Terminal) it will display a header and wait for a command.

```
Inlinino 2.0 alpha (May 16, 2016)
Type "help", "support", or "credits" for more information.
>>
```

Hint: Help can be found at anytime. Typing `help` will print a list of the command available. To get help on a specific command write `help <command>` replacing <command> with the name of the command you need some explanations.:

```
>>help

List of commands available (type help <topic>):
================================================
```

(continues on next page)

(continued from previous page)

```
EOF  credits  exit  help  instrument  log  shell  status  support

>>help log
log [arg]
  <start> logging data
  <stop> logging data
  <header> change file name header
  <filename> return current filename
>>
```

Tip: Autocompletion is enabled for each command pressing [tab] as you would do it on any bash or zsh terminal. A history of the command entered is kept in memory, they can be called by using the up arrow on your keyboard as you would do it on any bash or szh terminal. It's possible to run any command from bash by starting the command with `!`.

```
>>!pwd
/Users/nils/Documents/UMaine/Lab/Inline/inlinino

>>
```

1.5.1 Instrument

The set of command starting with `instrument` allow to connect/disconnect or display information regarding the instruments.

`instrument connect <instrument_name> [<instrument_port>]` Connect the instrument named `<instrument_name>`. If the instrument requires a port you should specify it in last argument `<instrument_port>`.

```
>>instrument connect SimulinoSin
>>
```

`instrument close <instrument_name>` Disconnect instruments named `<instrument_name>`.

```
>>instrument close SimulinoSin
>>
```

`instrument list [ports]` List all instruments If `ports` is specified in the command it will list all ports available on the computer.

```
>>instrument list
SimulinoSin
SimulinoRnd
>>instrument list ports
/dev/cu.Bluetooth-Incoming-Port: n/a
/dev/cu.iPhone-WirelessiAP: n/a
>>
```

`instrument read [<instrument_name>]` Read data from the instrument matching `<instrument_name>`. If no `<instrument_name>` is specified it will read from all instruments.

```
>>instrument read
{'sin': 2.212096683334974, 'sin_noise': 0.06595185215313082}
{'rnd2': 0.6670297159334724, 'rnd1': 1.4078267848958586}
>>instrument read SimulinoSin
{'sin': 1.6628582219802706, 'sin_noise': 0.3032862396665401}
>>
```

1.5.2 Log

The set of command starting with `log` allow to save data from instruments.

`log start` Start logging data.

```
>>log start
Start logging data.
>>
```

`log stop` Stop logging data.

```
>>log stop
Stop logging data.
>>
```

`log header <filename_prefix>` Change the log files prefix by the one specified in `<filename_prefix>`. You can check the modification with the command `log filename`.

```
>>log header Inlinino
>>
```

`log filename` Display the path to current logging file. Note that if the path is relative you can get the current directory from which Inlinino is running with `!pwd`

```
>>log filename
data/Inlinino_yyyymmdd_HHMMSS.csv
>>
```

1.5.3 Status

`status` Display current status of the application, few information regarding the verbosity of the software as well as if the instruments are connected or not are displayed.

```
>>status
[Configuration]
  verbosity:2
[Instruments]
  SimulinoSin[active]
  SimulinoRnd[active]
```

1.5.4 Exit

The application can be closed at anytime. Data is saved before exiting even if the user did not stop logging before exiting.

`exit` or `EOF` Exit command line interface and quit Inlinino. When application is closed properly:

```
>>EOF
(Inlinino)
```

When application is closed and logging or instruments are still running:

```
>>exit
Closing connection with SimulinoRnd
Stop logging data.
Stop buffer thread.
(Inlinino)
```

`[Ctrl]+[C]` Applications will try to exit properly, saving all the data and closing serial connection.

```
>>^CKeyboard Interrupt received.
Trying to close connection with instrument(s), to save data and close log file␣
↪properly.
Closing connection with SimulinoRnd
Stop logging data.
Stop buffer thread.
(Inlinino)
```

If you press several times `[Ctrl]+[C]` some python errors will show up and data might be lost.

1.6 PASC

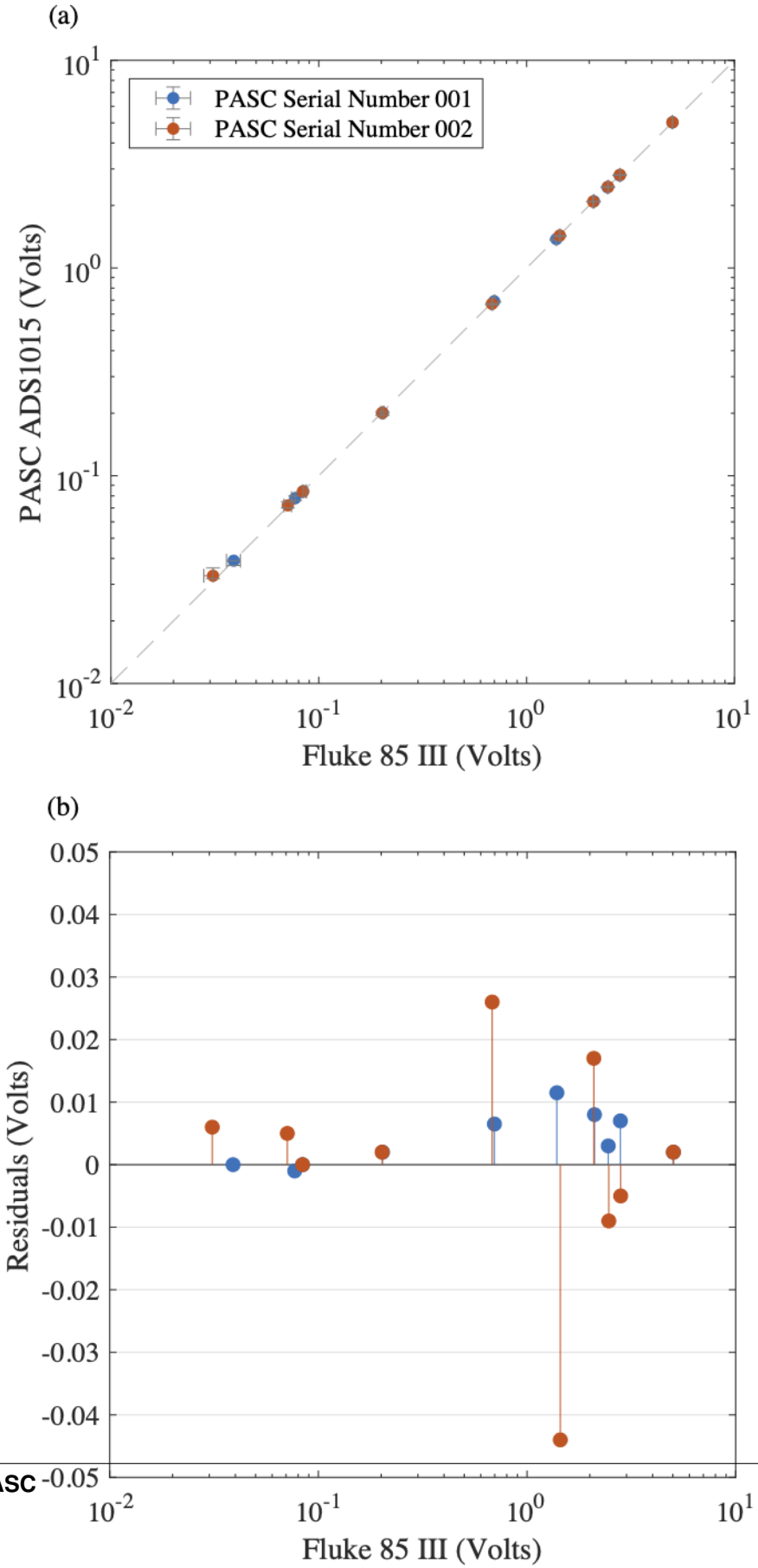
The precision analog to serial converter (PASC) is an optional data acquisition (DAQ) device. PASC is only required to log data from instruments communicating through analog channels. PASC can be built with an Arduino Uno type microcontroller and a precision analog to digital converter such as the Texas Instrument ADS1015 or ADS1115 developpement boards. The wiring instructions to build your own are available at Adafruit website.

We uploaded the firmware to the microcontroller following these instructions.

1. Load mcu_firmwares/PASC.cpp in the Arduino IDE:
    1. In ~/Documents/Arduino create a folder PASC/
    2. Copy and rename mcu_firmware/PASC.cpp to ~/Documents/Arduino/PASC/PASC.ino
    3. Load PASC.ino from Arduino Software (File > Open...)
2. Comment/uncomment appropriate lines in PASC.ino following instructions in comments of the file.
3. Compile and upload PASC to the microcontroller (button on the top left of Arduino IDE).

1.6.1 PASC Precision and Accuracy Validation

The precision and accuracy of the PASC serial number 001 and 002 were assessed with a Fluke 85 III Voltmeter. We found no significant bias and a reasonable root mean square error of (5.3 mV), the data is presented in the Figure below.


(a)
PASC Serial Number 001
PASC Serial Number 002
PASC ADS1015 (Volts)
Fluke 85 III (Volts)
(b)
Residuals (Volts)
Fluke 85 III (Volts)