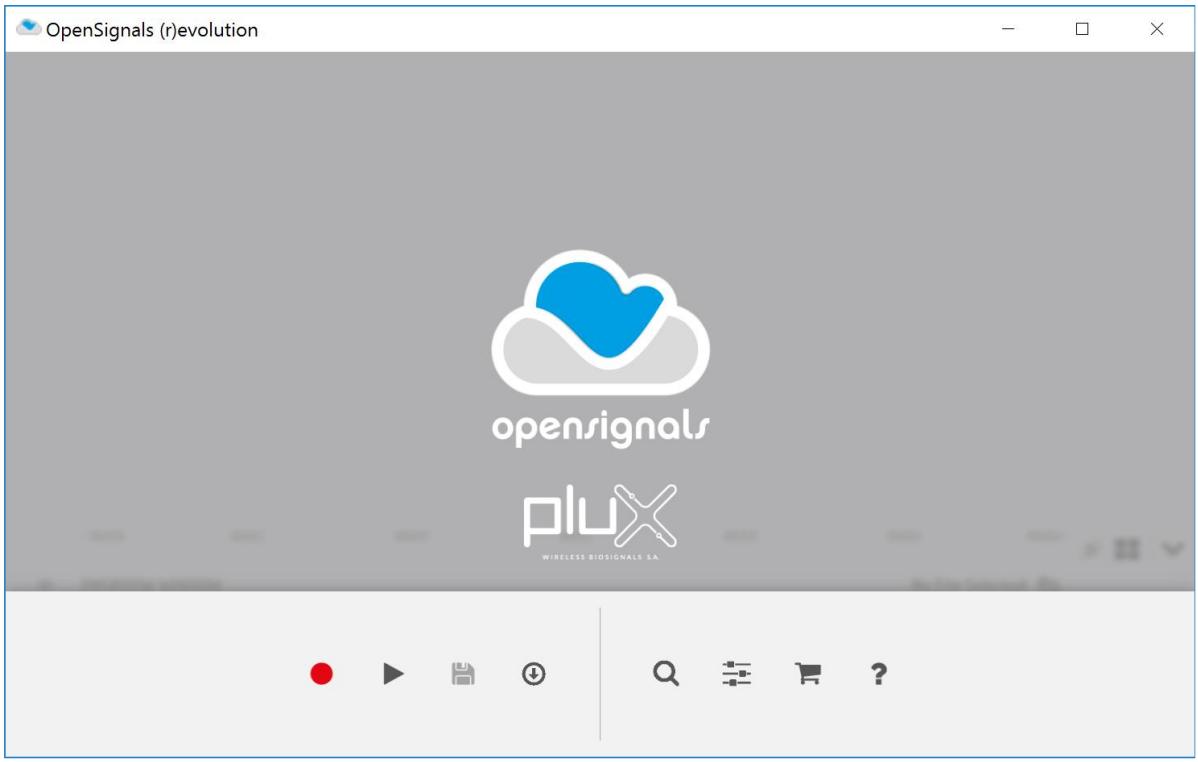
biosignal acquisition tool-kit for high-level research applications

Lab Streaming Layer Guide – Receiving OpenSignals Streams with Python™



ATTENTION
Please read this manual before
using your PLUX product(s) and
this software

The information contained in this manual has been carefully checked and were made every effort to ensure its quality. PLUX reserves the right to make changes and improvements to this manual and products referenced at any time without notice.

The word Bluetooth and its logo are trademarks of Bluetooth SIG Inc. and any use of such marks is under license. Other trademarks are the property of their respective own.

This module is part of the *OpenSignals (r)evolution* software (introduced with the release of December 2018). Reading the software's user manual is highly recommended:

http://biosignalsplux.com/downloads/OpenSignals_(r)evolution_User_Manual-print.pdf

PLUX Wireless Biosignals S.A.
email: plux@plux.info
web: http://www.plux.info

Headquarters
Zona Industrial das Corredouras, Lt. 14 – 1°
2630-369 Arruda dos Vinhos
Portugal
tel.: +351 263 978 572
fax: +351 263 978 902

Lisbon Office
Av. 5 de Outubro, n° 79 – 8°
1050-059 Lisboa
Portugal
tel.: +351 211 956 542
fax: +351 211 956 546

DISCLAIMER

OpenSignals (r)evolution, biosignalsplux & ***BITalino*** products are intended for use in life science education and research applications only; **they are not medical devices, nor medical software solutions, nor are they intended for medical diagnosis, cure, mitigation, treatment or prevention of disease and is provided to you "as is"**.

OpenSignals (r)evolution uses Google Chrome as the rendering engine for the graphical user interface. Google Chrome is a web browser developed by Google Inc. PLUX is neither responsible for its content, nor for its functionality.

We expressly disclaim any liability whatsoever for any direct, indirect, consequential, incidental or special damages, including, without limitation, lost revenues, lost profits, losses resulting from business interruption or loss of data, regardless of the form of action or legal theory under which the liability may be asserted, even if advised of the possibility of such damages.

TABLE OF CONTENTS

1 Introduction

The Lab Streaming Layer (LSL) module of the *OpenSignals (r)evolution* software is aimed to facilitate the support and data exchange between the 3rd party and the *OpenSignals (r)evolution* software. It has been introduced with the *OpenSignals* release of December 2018 and is based on the open-source LSL system which can be found on GitHub:

https://github.com/sccn/labstreaminglayer

As found in its official description, "*LSL is a system for the unified collection of measurement time series in research experiments that handles both the networking, time-synchronization, (near-) real-time access as well as optionally the centralized collection, viewing and disk recording of the data"*. This system enables *OpenSignals (r)evolution* to stream multi-channel sensor data acquired using *biosignalsplux* and *BITalino* kits to third 3rd party applications were only a few lines of code are required to receive real-time sensor data.

This guide is intended to demonstrate and guide the proper configuration of the module to enable real-time signal acquisition and streaming between *OpenSignals (r)evolution* and a LSL compatible 3rd party software. Additionally, examples are provided in the In this configuration, *OpenSignals (r)evolution* will act as a server while the 3rd party software will act as the client.

2 OpenSignals Configuration

The information below guides you through the setup process to activate the stream in the *OpenSignals (r)evolution* software.

2.1 Configuring Acquisition Devices in OpenSignals (r)evolution

Before using any device for acquisition via the LSL it is necessary to establish a Bluetooth connection with your computer and the PLUX device(s) first and to configure the acquisition devices in the *OpenSignals (r)evolution* software.

Follow the instructions in the *OpenSignals (r)evolution* user manual to learn how to properly set up your devices for signal acquisitions (Section 2.2):

http://biosignalsplux.com/downloads/OpenSignals_(r)evolution_User_Manual-print.pdf

2.2 Lab Streaming Layer Configuration (OpenSignals (r)evolution)

Open the settings panel of the *OpenSignals (r)evolution* software by clicking on the following icon which can be found in the software's main screen.


OpenSignals (r)evolution settings

In the settings panel, click on the *INTEGRATION* tab and select the *Lab Streaming Layer* checkbox to start the server as seen in Figure 1.

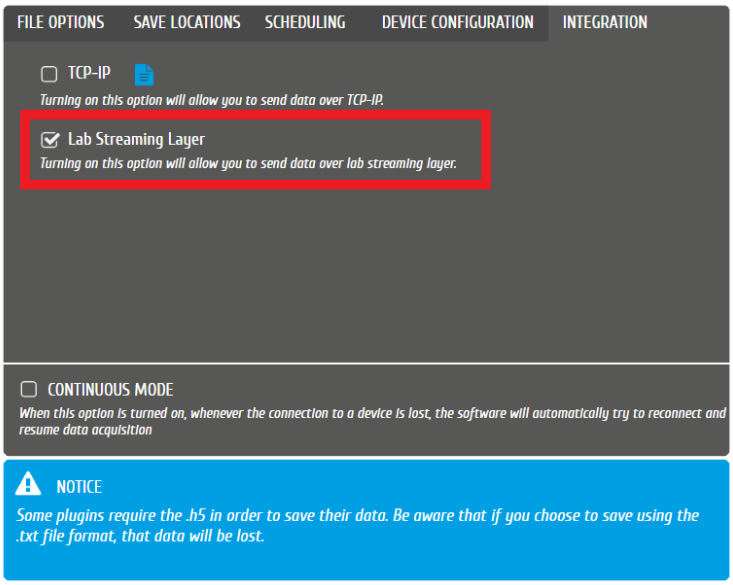


Figure 1: Settings panel with the activated LSL module.

NOTE
The LSL module has to be reactivated as described in this section after errors occur as a proper connection has to be re-established.

After this step, you can start the acquisition when you are ready to receive data in your 3rd party application.

3 Receiving OpenSignal Stream with Python

The LSL system allows you to receive signal streams using different identifiers of your choice. In this section, 3 different options are presented which can be useful for different use cases.

Option 1: Receive data from an unspecified OpenSignals stream
Use case: Only one instance of *OpenSignals* is being used, there are no other machines in the network using *OpenSignals* & the LSL.

Option 2: Receive data from a specific *biosignalsplux* or *BITalino* device using the device's MAC-address
Use case: When multiple devices are being used & you need access to the stream of a specific device.

Option 3: Receive data from a specific host
Use case: Multiple machines in your network are running OpenSignals & you need access to the stream of a specific machine.

The different code example for the different cases is provided in the following cases.

NOTE
Python needs access to your network in order to receive data. Please ensure that the Python is not being blocked by your firewall.

NOTE
The examples shown in this section are based on the official pylsl *ReceiveData.py* example which can be found on GitHub:

https://github.com/labstreaminglayer/liblsl-Python/blob/d95f40e878111620600f7d6dc6b45b62ac961776/pylsl/examples/ReceiveData.py

The example scripts presented in this document are available in the .ZIP file where this document can be found.

3.1 Receiving Data From an Unspecified OpenSignals Stream

Use Case
Only one instance of *OpenSignals* is being used, there are no other machines in the network using *OpenSignals* & the LSL.

First, we need to import the necessary *StreamInlet* class and *resolve_stream* function from the *pylsl* which are required to resolve the signal stream.

```
# Imports
from pylsl import StreamInlet, resolve_stream
```

Code Snippet 1: Importing the pylsl package.

Specify the name of the stream using *pylsl's resolve_stream* function. In the case of *OpenSignals,* the stream name is set to *OpenSignals*.

```
# Resolve an available OpenSignals stream
print("# Looking for an available OpenSignals stream...")
os_stream = resolve_stream("name", "OpenSignals")
```

Code Snippet 2: Resolving an available OpenSignals stream.

This function will block the script from running until an *OpenSignals* stream has been resolved. When found, the script will proceed to the next step by creating an inlet (data receiver) using the *StreamInlet()* class.

```
# Create an inlet to receive signal samples from the stream
inlet = StreamInlet(os_stream[0])
```

Code Snippet 3: Creating an inlet to revice the streamed samples.

The inlet is now ready to receive data which. A simple example of how to receive signal samples from *OpenSignals* using a *while* loop is shown below.

```
while True:
    # Receive samples
    sample, timestamp = inlet.pull_sample()
    print(timestamp, sample)
```

Code Snippet 4: Simple example loop for continuously receiving incoming samples.

The entire, summarized script can be found on the on the next page.

```
"""
OpenSignals Lab Streaming Layer - Receiving data OpenSignals
------------------------------------------------------------

Example script to show how to receive a (multi-)channel signal stream from
OpenSignals (r)evolution using the Lab Streaming Layer (LSL).
"""

# Imports
from pylsl import StreamInlet, resolve_stream

# Resolve an available OpenSignals stream
print("# Looking for an available OpenSignals stream...")
os_stream = resolve_stream("name", "OpenSignals")

# Create an inlet to receive signal samples from the stream
inlet = StreamInlet(os_stream[0])

while True:
    # Receive samples
    sample, timestamp = inlet.pull_sample()
    print(timestamp, sample)
```

Code Snippet 5: Example code showing how to receive samples from an OpenSignals LSL stream.

3.2 Receiving Data From a Specific PLUX Device in an OpenSignals Stream

Use Case
When multiple devices are being used & you need access to the stream of a specific device, you can use the device's MAC-address to identify the stream. The device's MAC-address can be found on the back of the device.

First, we need to import the necessary *StreamInlet* class and *resolve_stream* function from the *pylsl* which are required to resolve the signal stream.

```
# Imports
from pylsl import StreamInlet, resolve_stream
```

Code Snippet 6: Importing the pylsl package.

Specify the MAC-address of the device using *pylsl's resolve_stream* function.

```
# Define the MAC-address of the acquisition device used in OpenSignals
mac_address = "A1:B2:C3:D4:E5:F6"

# Resolve stream
print("# Looking for an available OpenSignals stream from the specified
device...")
os_stream = resolve_stream("type", mac_address)
```

Code Snippet 7: Resolving an available OpenSignals stream using the acquisition device's MAC-address.

This function will block the script from running until an *OpenSignals* stream has been resolved. When found, the script will proceed to the next step by creating an inlet (data receiver) using the *StreamInlet()* class.

```
# Create an inlet to receive signal samples from the stream
inlet = StreamInlet(os_stream[0])
```

Code Snippet 8: Creating an inlet to revice the streamed samples.

The inlet is now ready to receive data which. A simple example of how to receive signal samples from *OpenSignals* using a *while* loop is shown below.

```
while True:
    # Receive samples
    sample, timestamp = inlet.pull_sample()
    print(timestamp, sample)
```

Code Snippet 9: Simple example loop for continuously receiving incoming samples.

The entire, summarized script can be found on the on the next page.

```
"""
OpenSignals Lab Streaming Layer - Receiving data from a specific PLUX device
-----------------------------------------------------------------------------

Example script to show how to receive a (multi-)channel signal stream from
OpenSignals (r)evolution & a specific PLUX device using the Lab Streaming Layer
(LSL) and the device's MAC-address.

"""

# Imports
from pylsl import StreamInlet, resolve_stream

# Define the MAC-address of the acquisition device used in OpenSignals
mac_address = "A1:B2:C3:D4:E5:F6"

# Resolve stream
print("# Looking for an available OpenSignals stream from the specified
device...")
os_stream = resolve_stream("type", mac_address)

# Create an inlet to receive signal samples from the stream
inlet = StreamInlet(os_stream[0])

while True:
    # Receive samples
    samples, timestamp = inlet.pull_sample()
    print(timestamp, samples)
```

Code Snippet 10: Example code showing how to receive samples from an OpenSignals LSL stream using a device's MAC-address.

3.3 Receiving Data From a Specific Host Providing the OpenSignals Stream

Use Case
Multiple machines in your network are running OpenSignals & you need access to the stream of a specific machine. The hostname is the name of the computer streaming the data.

First, we need to import the necessary *StreamInlet* class and *resolve_stream* function from the *pylsl* which are required to resolve the signal stream.

```
# Imports
from pylsl import StreamInlet, resolve_stream
```

Code Snippet 11: Importing the pylsl package.

Specify the hostname of the host machine using *pylsl's resolve_stream* function.

```
# Define the name of the host streaming the sensor data
hostname = "HOSTNAME"

# Resolve stream
print("# Looking for an available OpenSignals stream from the specified host...")
os_stream = resolve_stream("hostname", hostname)
```

Code Snippet 12: Resolving an available OpenSignals stream using the name of the host machine.

This function will block the script from running until an *OpenSignals* stream has been resolved. When found, the script will proceed to the next step by creating an inlet (data receiver) using the *StreamInlet()* class.

```
# Create an inlet to receive signal samples from the stream
inlet = StreamInlet(os_stream[0])
```

Code Snippet 13: Creating an inlet to revice the streamed samples.

The inlet is now ready to receive data which. A simple example of how to receive signal samples from *OpenSignals* using a *while* loop is shown below.

```
while True:
    # Receive samples
    sample, timestamp = inlet.pull_sample()
    print(timestamp, sample)
```

Code Snippet 14: Simple example loop for continuously receiving incoming samples.

The entire, summarized script can be found on the on the next page.

```
"""
OpenSignals Lab Streaming Layer - Receiving data from a specific host computer
-------------------------------------------------------------------------------

Example script to show how to receive a (multi-)channel signal stream from
OpenSignals (r)evolution from a specific host using the Lab Streaming Layer (LSL)
and the hostname.

"""

# Imports
from pylsl import StreamInlet, resolve_stream

# Define the name of the host streaming the sensor data
hostname = "HOSTNAME"

# Resolve stream
print("# Looking for an available OpenSignals stream from the specified host...")
os_stream = resolve_stream("hostname", hostname)

# Create an inlet to receive signal samples from the stream
inlet = StreamInlet(os_stream[0])

while True:
    # Receive samples
    samples, timestamp = inlet.pull_sample()
    print(timestamp, samples)
```

Code Snippet 15: Example code showing how to receive samples from an OpenSignals LSL stream using a specified host.

3.4 Receiving Stream Metadata

After resolving a stream (as presented on the previous pages) you can get the stream metadata using the *info()* method of the *inlet()* object.

```
# Get information about the stream
stream_info = inlet.info()
```

Code Snippet 16: Get all the available information about the OpenSignals LSL stream using the info() method.

Afterwards, you can use the methods below to get general information about the stream such as the stream name, the MAC-address of the device (type), the host name and the number of streamed channels.

```
# Get individual attributes
stream_name = stream_info.name()
stream_mac = stream_info.type()
stream_host = stream_info.hostname()
stream_n_channels = stream_info.channel_count()
```

Code Snippet 17: Get specific channel info attributres.

The channel configuration (channel number, sensor type, and unit) can be accessed by using the *desc()* method. The example below shows how to get all the channel information while storing the information in a Python dictionary.

```
# Store sensor channel info & units in the dictionary
stream_channels = dict()
channels = stream_info.desc().child("channels").child("channel")

# Loop through all available channels
for i in range(stream_n_channels - 1):

    # Get the channel number (e.g. 1)
    channel = i + 1

    # Get the channel type (e.g. ECG)
    sensor = channels.child_value("sensor")

    # Get the channel unit (e.g. mV)
    unit = channels.child_value("unit")

    # Store the information in the stream_channels dictionary
    stream_channels.update({channel: [sensor, unit]})
    channels = channels.next_sibling()
```

Code Snippet 18: Example snippet showing how the sensor channel, type, and unit from all the streamed sensor channels.

4 Regulatory & Legal Information

4.1 Disclaimer

All mentioned *OpenSignals (r)evolution*, *biosignalsplux,* and *BITalino* products in this manual are intended for use in life science education and research applications only; they are not medical devices, nor medical software solutions, nor are they intended for medical diagnosis, cure, mitigation, treatment or prevention of disease and is provided to you “as is”.

We expressly disclaim any liability whatsoever for any direct, indirect, consequential, incidental or special damages, including, without limitation, lost revenues, lost profits, losses resulting from business interruption or loss of data, regardless of the form of action or legal theory under which the liability may be asserted, even if advised of the possibility of such damages.

4.2 Certification

OpenSignals (r)evolution and any PLUX device connected to this software do not have a medical device certification and are, therefore, not a medical device.

PLUX research products are intended for use in life science education and research applications with humans and not intended for diagnostics, cure, mitigation, treatment or prevention of disease.

4.3 Contact & Support

Contact us if you’re experiencing any problems that cannot be solved with the information given in the biosignalsplux or OpenSignals (r)evolution manual. We’ll get back to you as soon as possible to find the best solution for your problem.

Please send us an e-mail with precise information about the error occurrence, device configuration, and, if possible, screenshots of the problem to support@plux.info.