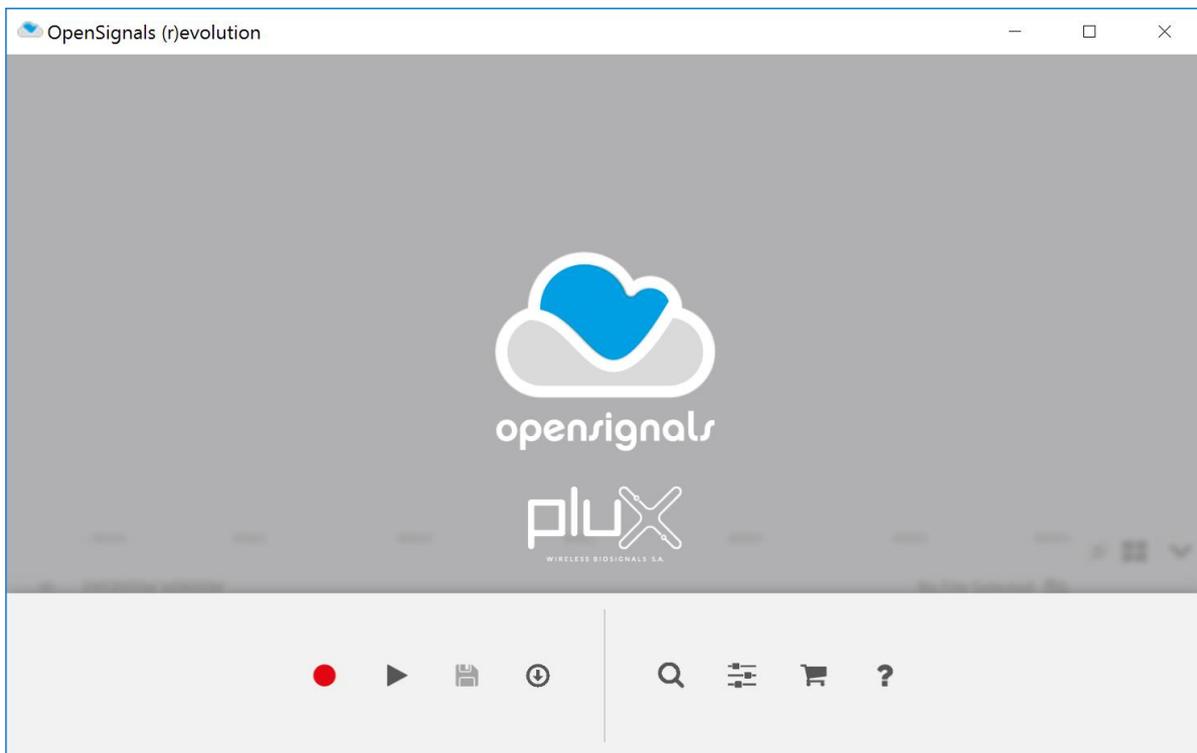![opensignals (r)evolution logo]

biosignal acquisition tool-kit for high-level research applications

Lab Streaming Layer Guide – Receiving OpenSignals Streams with MATLAB®

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

This document serves to provide guidance on using OpenSignals' Lab Streaming Layer feature with MATLAB®.

The information contained in this document has been carefully checked and we 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. MATLAB® is a registered® trademark of The MathWorks, INC. Use of this trademark does not imply any affiliation with or endorsement by them. Other trademarks are the property of their respective owners.

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° 70 – 2°
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"**.

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 (r)evolution* December 2018 release 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 MATLAB® 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 this configuration.

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.
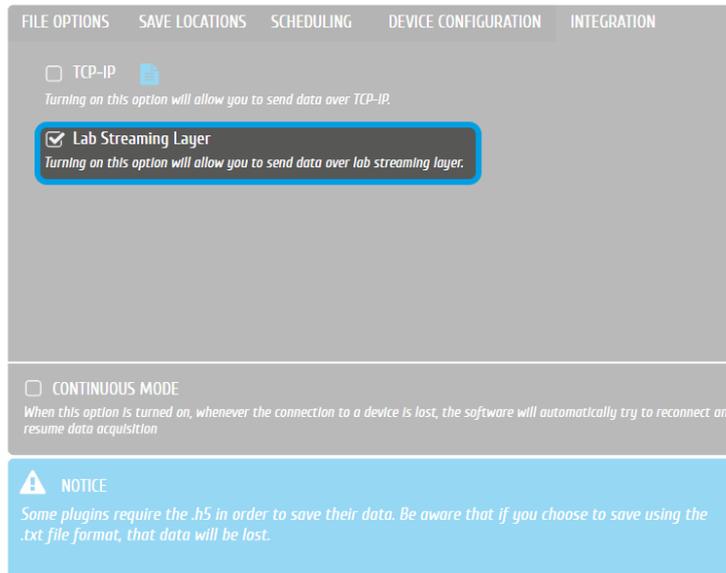

Figure 1: Settings panel with the activated LSL module.

NOTE
The LSL module must be reactivated as described in this section after errors occur as a proper stream 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 MATLAB®

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 (r)revolution* is being used, there are no other machines in the network using *OpenSignals (r)evolution* & 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 (r)evolution* & 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
MATLAB® needs access to your network in order to receive data. Please ensure that MATLAB® is not being blocked by your firewall.

NOTE
The examples shown in this section are based on the official liblsl-Matlab *ReceiveData.m* example which can be found on GitHub:

https://github.com/labstreaminglayer/liblsl-Matlab/blob/master/examples/ReceiveData.m

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

3.1 Downloading and Building the liblsl-Matlab Library

NOTE

The following instructions are taken from the official LSL for MATLAB® repository which can be found on GitHub. Visit the repository for up-to-date instructions.

Step 1

Download the newest liblsl-Matlab library from the official GitHub repository:

https://github.com/labstreaminglayer/liblsl-Matlab

Step 2

Build a version of the liblsl library for your computer & MATLAB® version by following the instructions on the following website:

https://github.com/sccn/labstreaminglayer/blob/master/doc/BUILD.md

Step 3

Extract the downloaded.ZIP file and add the extracted folder to the MATLAB® search path by clicking on *"File/Set Path…"* in MATLAB®.

Step 4

Follow the instructions on the following website to build the library for your operating system:

https://github.com/sccn/labstreaminglayer/wiki/INSTALL#build-instructions

Step 5

Afterwards, add an up-to-date build of the library for your MATLAB® version and operating system to the *bin/* directory of the downloaded and extracted in Step 2. Depending on the operating system, the following files should be placed in the *bin/* directory.

- Windows: liblsls64.dll
- macOS: liblsl64.dylib <u>and</u> liblsl64.1.4.0.dylib
- Linux: liblsl64.so <u>and</u> liblsl64.so.1.4.0

Step 6

Select one of the 3 available options on the following sections and pages to resolve a data stream.

3.2   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 (r)evolution* & the LSL.

First, we need to load the liblsl-Matlab library using the *lsl_loadlib()* function.

```
% Load liblsl-Matlab Library
lib = lsl_loadlib();
```
Code Snippet 1: Importing the liblsl-Matlab package.

Specify the name of the stream using the *lsl_resolve_byprop* function. In the case of *OpenSignals (r)evolution,* the stream name is set to *OpenSignals*.

```
% Resolve an available OpenSignals stream
disp('Looking for an available OpenSignals stream...');
streams = {};
while isempty(streams)
    streams = lsl_resolve_byprop(lib, 'name', 'OpenSignals');
end
```
Code Snippet 2: Resolving an available OpenSignals stream.

This code block 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 *lsl_inlet()* function.

```
% Create an inlet to receive signal samples from the stream
inlet = lsl_inlet(streams{1});
```
Code Snippet 3: Create an inlet to receive signal samples from the OpenSignals stream.

The inlet is now ready to receive data. A simple example of how to receive signal samples (*samples*) from *OpenSignals* and the LSL generated time stamps (*ts*)using a *while* loop is shown below.

```
while true
    % Receive samples
    [samples, ts] = inlet.pull_sample();
    fprintf('%.5f\n', ts);
    fprintf('%.2f\t', samples);
end
```
Code Snippet 4: Simple example loop for continuously receiving incoming samples.

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

```matlab
%% OpenSignals Lab Streaming Layer
%     Example script to show how to receive a (multi-)channel signal stream
%     from OpenSignals using the Lab Streaming Layer (LSL)

% Load liblsl-Matlab Library
lib = lsl_loadlib();

% Resolve an available OpenSignals stream
disp('Looking for an available OpenSignals stream...');
streams = {};
while isempty(streams)
     streams = lsl_resolve_byprop(lib, 'name', 'OpenSignals');
end

% Create an inlet to receive signal samples from the stream
inlet = lsl_inlet(streams{1});

while true
    % Receive samples
    [samples, ts] = inlet.pull_sample();
    fprintf('%.5f\n', ts);
    fprintf('%.2f\t', samples);
end
```

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

3.3 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 load the liblsl-Matlab library using the *lsl_loadlib()* function.

```
% Load liblsl-Matlab Library
lib = lsl_loadlib();
```
Code Snippet 6: Importing the liblsl-Matlab package.

Specify the MAC-address of the streaming device using the *lsl_resolve_byprop* function. In this case, the type is the device's MAC-address.

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

% Resolve an available OpenSignals stream
disp('Looking for an available OpenSignals stream...');
streams = {};
while isempty(streams)
    streams = lsl_resolve_byprop(lib, 'type', 'mac_address');
end
```
Code Snippet 7: Resolving an available OpenSignals stream.

This code block 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 *lsl_inlet()* function.

```
% Create an inlet to receive signal samples from the stream
inlet = lsl_inlet(streams{1});
```
Code Snippet 8: Create an inlet to receive signal samples from the OpenSignals stream.

The inlet is now ready to receive data. A simple example of how to receive signal samples (*samples*) from *OpenSignals* and the LSL generated time stamps (*ts*) using a *while* loop is shown below.

```
while true
    % Receive samples
    [samples, ts] = inlet.pull_sample();
    fprintf('%.5f\n', ts);
    fprintf('%.2f\t', samples);
end
```
Code Snippet 9: Simple example loop for continuously receiving incoming samples.

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

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

% Load liblsl-Matlab Library
lib = lsl_loadlib();

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

% Resolve an available OpenSignals stream
disp('Looking for an available OpenSignals stream...');
streams = {};
while isempty(streams)
    streams = lsl_resolve_byprop(lib, 'type', 'mac_address');
end
% Create an inlet to receive signal samples from the stream
inlet = lsl_inlet(streams{1});

while true
    % Receive samples
    [samples, ts] = inlet.pull_sample();
    fprintf('%.5f\n', ts);
    fprintf('%.2f\t', samples);
end
```

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

3.4   Receiving Data From a Specific Host Providing the OpenSignals Stream

> **Use Case**
> Multiple machines in your network are running *OpenSignals (r)evolution* & 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 load the liblsl-Matlab library using the *lsl_loadlib()* function.

```
% Load liblsl-Matlab Library
lib = lsl_loadlib();
```
Code Snippet 11: Importing the liblsl-Matlab package.

Specify the hostname of the host machine using the *lsl_resolve_byprop* function.

```
% Define the name of the host streaming the sensor data
hostname = 'HOSTNAME';

% Resolve an available OpenSignals stream
disp('Looking for an available OpenSignals stream...');
streams = {};
while isempty(streams)
    streams = lsl_resolve_byprop(lib, 'hostname', hostname);
end
```
Code Snippet 12: Resolving an available OpenSignals stream using the name of the host machine.

This code block 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 *lsl_inlet()* function.

```
% Create an inlet to receive signal samples from the stream
inlet = lsl_inlet(streams{1});
```
Code Snippet 13: Create an inlet to receive signal samples from the OpenSignals stream.

The inlet is now ready to receive data. A simple example of how to receive signal samples (*samples*) from *OpenSignals* and the LSL generated time stamps (*ts*) using a *while* loop is shown below.

```
while true
    % Receive samples
    [samples, ts] = inlet.pull_sample();
    fprintf('%.5f\n', ts);
    fprintf('%.2f\t', samples);
end
```
Code Snippet 14: Simple example loop for continuously receiving incoming samples.

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

```matlab
%% OpenSignals Lab Streaming Layer
%     Example script to show how to receive a (multi-)channel signal stream
%     from OpenSignals & a specific host machine using the Lab Streaming
%     Layer (LSL) and the hostname

% Load liblsl-Matlab Library
lib = lsl_loadlib();

% Define the name of the host machine streaming the sensor data
hostname = 'HOSTNAME';

% Resolve an available OpenSignals stream
disp('Looking for an available OpenSignals stream...');
streams = {};
while isempty(streams)
      streams = lsl_resolve_byprop(lib, 'hostname', hostname);
end

% Create an inlet to receive signal samples from the stream
inlet = lsl_inlet(streams{1});

while true
    % Receive samples
    [samples, ts] = inlet.pull_sample();
    fprintf('%.5f\n', ts);
    fprintf('%.2f\t', samples);
end
```

Code Snippet 15: Example code showing how to receive samples
from an OpenSignals LSL stream using the hostname of the host machine.

3.5 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.

```matlab
% 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), and the number of streamed channels.

```matlab
% Get individual stream attributes
stream_name = stream_info.name();
stream_mac = stream_info.type();
stream_n_channels = stream_info.channel_count();
```
Code Snippet 17: Get specific channel info attributes.

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 Map Container object.

```matlab
% Store sensor channel info & units in a containers.map
stream_channels = containers.Map('KeyType', double, 'ValueType', 'any');
channels = stream_info.desc().child('channels').child('channel');
channel = 0;

% Loop through all available channels
for k = 1:(stream_n_channels - 1)
    % Get the channel number (e.g. 1)
    channel = k + 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 stream_channels
    stream_channels(channel) = [sensor, unit];
    channels = channels.next_sibling();
end
```
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.