eego™ amplifier Software Interface SDK

User Manual

SDK revision: 1.3.19
Document revision: 7.1

[UDO-SM-0124] SDK User Manual Revision 7.1 EN, Nov 2018

MANUFACTURER

eemagine Medical Imaging Solutions GmbH

Gubener Str. 47

D-10243 Berlin, Germany

Phone: +49 (0)30 2904 8404

E-Mail: support@eemagine.com

TABLE OF CONTENTS

1 INTRODUCTION

This document will describe how to use our Software Development Kit (SDK) for interfacing with the **eego** amplifier. The design goal of the SDK is to provide an interface which is easy to use and which can connect to all the supported **eego** amplifiers. The key concept to achieve this goal is the provision of a high-level C++ interface which takes care of the direct binding to the DLL.

Please be aware that only this interface is supported and the lower levels are accessible but can change at any time. You will not be supported when your application relies on any custom access you have implemented to those lower layers and you cannot rely on the devices performance and specification in this case.

In Chapter 3 the standard use case of receiving data from the amplifier is described in textual form. After reading this document you should have no problem to achieve integration of the eego amplifier in short time in your own software application. The technical details necessary for this will be described in chapter 4.

A short introduction of the possible data modes supported by the amplifier and the API is given in Chapter 5. The last chapter will give an overview of how errors from the hardware and the user will be handled by the SDK and how to resolve them.

2 IMPORTANT NOTICES, DISCLAIMER AND LICENSE

This user guide describes the use of our software interface to the **eego** amplifier.

The eego amplifier comes with its own user manual, which must be read carefully before working with this manual describing direct access to the data recorded by the amplifier.

You also must read the API description along with the latest release notes for the SDK.

This document is written as accurately as possible. However, mistakes are bound to occur, and we reserve the right to make changes to the products, which may render parts of this document invalid.

Any application you write based on this manual or the interface in general is under your full responsibility with regard to quality performance and accuracy of parameters.

You need to run full testing and qualification based on your own requirements to claim any performance of the combined system.

Please be especially advised that any certification holding for the **eego** amplifier is not valid for a combined system of your application software and the **eego** amplifier. You must obtain your own certification for a combined system of amplifier and software.

The following amplifiers (revisions 1.0 and above) are compatible with the interface/SDK:

Product code	Description
EE-211	**ee**go **amplifier**, 64ch referential, 2kHz
EE-212	**ee**go **amplifier**, 32ch referential, 2kHz
EE-213	**ee**go **amplifier**, 16ch referential, 2kHz
EE-214	**ee**go **amplifier**, 32ch referential, 24ch bipolar, 2kHz
EE-215	**ee**go **amplifier**, 64ch referential, 24ch bipolar, 2kHz
EE-221	**ee**go **amplifier**, 16ch referential, 16kHz
EE-222	**ee**go **amplifier**, 32ch referential, 16kHz
EE-223	**ee**go **amplifier**, 32ch referential, 24ch bipolar, 16kHz

EE-224	**ee**go **amplifier**, 64ch referential, 16kHz
EE-225	**ee**go **amplifier**, 64ch referential, 24ch bipolar, 16kHz
EE-410	**ee**go **amplifier**, 8ch bipolar, 2kHz
EE-411	**ee**go **amplifier**, 8ch referential, 2kHz
EE-430	**ee**go **amplifier**, 8ch referential, 512Hz

License:

3 USE CASE

A typical use case for the SDK:

1. The end user requests one amplifier.
2. From that amplifier an EEG stream object is requested, with user selectable parameters.
3. For as long as the end-user wants to process data, it will loop:
    a. Call the GetData method on the stream object. If the duration between the polls is small the buffer may be empty.
    b. Process any data returned.
    c. Wait until new data is expected. The maximum interval between two polls should be below one second. The driver uses an internal buffer that is limited in size. Waiting too long for getting new data may result in sample loss. The one second limit is a safe marging based on empirical data.
4. Close the stream.
5. Either go back to (2.) to open any other stream, or release the amplifier.

4 HOW TO USE THE SDK

In the following chapter you will find a description of how to integrate the SDK into your project. All paths beginning with SDK_ROOT are relative to the base directory of the SDK.

4.1 Supported Platforms

4.1.1 Windows

To use the amplifier, the drivers need to be installed. You will find them in "SDK_ROOT\windows\driver".

After connecting an amplifier for the first time Windows will search for drivers. It will not find any, so you have to guide Windows to the driver location. Tell Windows that you have a disk, or that you know where to find the files.

The libraries for Windows are found under "SDK_ROOT\windows" and are named eego-SDK.lib and eego-SDK.dll

Libraries for both 32 and 64 bit are released and available.

4.1.2 Linux

By default, USB devices are only accessible by root. To allow normal users to use the eego amplifier, it is advisable to add a configuration to UDev(linux device management system). Create a file named:

- /etc/udev/rules.d/90-eego.rules

and add the line:

- ATTRS{idVendor}=="2a56", ATTRS{idProduct}=="ee01", SYMLINK+="eego3.%n", MODE:="0666", ENV{DEVTYPE}=="usb_device"

Afterwards, restart the udev system (or reboot the system). Then, when the amplifier is connected, every user has permissions to use it.

The library for Linux can be found under "SDK_ROOT\linux". The library is named libeego-SDK.so.

Note that hot-plugging is not supported yet on Linux. When the library is loaded, it will look for connected devices. Those are the only devices supported during the lifetime of the application. If an amplifier is removed or added, the library will be able change the list of available amplifiers.

Libraries for both 32 and 64 bit are released and availalbe.

4.2 Header Files

In “SDK_ROOT\eemagine” you will find the headers of the SDK. Copy them into your project. Make sure that the folder where the eemagine folder is copied into is in the header search path of the project.

To start the development, include the reference to the SDK into one of your development files:

```
#include <eemagine/sdk/factory.h>
```

If compilation fails because it could not find a header file, the project settings have to be adapted.

4.3 Library Binding

With correct path settings and all the header files in the correct place, the compilation will still fail with an error message like this:

```
"Either EEGO_SDK_BIND_DYNAMIC or EEGO_SDK_BIND_STATIC must be defined!"
```

The reason for this is the following; The SDK provides two ways of binding the library to the end-user's code. The two ways are named static binding and dynamic binding, and refer to the way in which the program locates and loads the library that is necessary for the SDK to function. The end-user must specify which one of the two binding methods is intended to be used by defining either EEGO_SDK_BIND_STATIC or EEGO_SDK_BIND_DYNAMIC during compilation.

The differences between the two ways of binding are described below.

4.3.1 Static

By using static binding, the runtime environment will load the library at program startup. During compile time it will check that all the referenced symbols exists in the library.

Note that:

- For Windows, you have to integrate the file “windows\eego-SDK.lib” into the project compilation and have to make sure that “windows\eego-SDK.dll” can be found in the DLL searchpath on program startup.
- For Linux, you have to link against the libeego-SDK.so file.

4.3.2 Dynamic

When using dynamic binding, the SDKs library will be loaded during the first usage of the SDK. The first call to the SDK will always be the creation of the factory object. In the case of dynamic binding, the factory creation gets an additional argument to specify the path from where the library should be loaded. The user has to provide this path to the location where “SDK_ROOT\windows\eego-SDK.dll”

can be found on first usage of the SDK if using Windows. Likewise, for Linux, the path to SDK_ROOT\linux\libeego-SDK.so should be used.

4.4 Compilation of the Wrapper

A low level C interface to the library is wrapped by the high level C++ interface described in this document. Please be aware that the low level C interface is not specified to the end user and can change any time. You must not implement any shortcut interfacing into this layer!

To be able to use the C++ interface, it is necessary to include the file "SDK_ROOT\eemagine\sdk\wrapper.cc" into the compilation of your project. This can be done by either simply adding it to the source files - which should be compiled as part of the project - or by including it into exactly one of your own source files by using the #include statement.

4.5 Namespace

All public usable methods for the SDK are found inside the eemagine::sdk namespace.

4.6 Creating an Amplifier Object

(The following example is Windows oriented. For Linux, the filename libeego-SDK.so should be used instead of eego-SDK.dll)

This is the only place where the method of binding makes a difference in how to use the SDK. In the following example, dynamic binding is being used to get an amplifier object.

```
#define EEGO_SDK_BIND_DYNAMIC // How to bind
#include <eemagine/sdk/factory.h> // SDK header
#include <eemagine/sdk/wrapper.cc> // Wrapper code to be compiled.

int main(int argc, char **argv)
{
	using namespace eemagine::sdk;

	factory fact("eego-SDK.dll"); // Make sure that eego-SDK.dll resides in the
working directory
	amplifier* amp = fact.getAmplifier(); // Get an amplifier
	delete amp; // Make sure to delete the amplifier objects to release resources

	return 0;
}
```

The example for the static case looks a bit simpler (see below).

```
#define EEGO_SDK_BIND_STATIC // How to bind
#include <eemagine/sdk/factory.h> // SDK header
#include <eemagine/sdk/wrapper.cc> // Wrapper code to be compiled.

int main(int argc, char **argv)
{
	using namespace eemagine::sdk;

	factory fact;
	amplifier* amp = fact.getAmplifier(); // Get an amplifier
	delete amp; // Make sure to delete the amplifier objects to release resources

	return 0;
}
```

Please note that the factory constructor is missing the path argument.

You will always get a valid amplifier object ready to use, or an exception will be thrown. If multiple amplifiers are connected, you will get any of those without any defined order. If you want to connect to a specific amplifier, you have to use the method “getAmplifiers()” to retrieve a list of connected and available amplifiers to select the one you want to use.

Please always remember to delete the amplifier object to release the acquired resources.

From now on, we assume the static binding case.

4.7 Opening a Stream and Retrieve Data from it

(The below example is for Windows, as it uses the *<conio.h>* header and the *_kbhit()* function call.)

To retrieve data from the amplifier, you have to open a stream and poll for any data that may have been acquired in the meantime. The following example creates an amplifier object and starts a data stream. The result is printed to the terminal until a key is pressed.

```
#define EEGO_SDK_BIND_STATIC // How to bind
#include <eemagine/sdk/factory.h> // SDK header
#include <eemagine/sdk/wrapper.cc> // Wrapper code to be compiled.

// For sleeping
#include <chrono>
#include <thread>

#include <iostream> // console io

#include <conio.h> // For _kbhit();

int main(int argc, char **argv)
{
	using namespace eemagine::sdk;

	factory fact;
	amplifier* amp = fact.getAmplifier(); // Get an amplifier
	stream* eegStream = amp->OpenEegStream(500); // The sampling rate is the only
argument needed

	while (!_kbhit()) // Loop until any key gets pressed
	{
		buffer buf = eegStream->getData(); // Retrieve data from stream
		std::cout << "Samples read: " << buf.getSampleCount() << std::endl;
		std::this_thread::sleep_for(std::chrono::milliseconds(100)); // Need to
sleep less than 1s otherwise data may be lost
	}

	// release Resources
	delete eegStream;
	delete amp;

	return 0;
}
```

Please note that the eegStream has to be deleted, too. Only one stream can be active at any time. For a description of the different streaming types, please have a look at chapter 5. For a detailed description of the possible arguments and further documentation, please see the reference guide.

5 STREAMING MODES

The purpose of the SDK is to stream measured data to the user. Currently there are three different streaming modes which will be explained in this chapter.

5.1 Key Concepts

Generally the amplifier can be in two states: Streaming or not streaming. It can change between the two states by opening streams and closing these streams by deletion. Two streams cannot be active at the same time. If the user tries to open a second stream an exception will be thrown.

The default state of any amplifier is not streaming.

5.2 EEG Stream

EEG Stream is the most common streaming modality of the SDK. The data is measured at the EEG and (if available) AUX inputs, and can be polled via the GetData() method of the stream. No samples are lost if the GetData() method gets polled every second or faster. The internal buffers may store data for longer, but the SDK only guarantees one second of data. If the user waits too long and internal buffers overflow, then the next call to GetData() will throw an incorrectValue exception. The stream will simply continue and the next call to GetData() will return valid data(provided that the buffers don't overflow again).

5.2.1 Units

Calling GetData() on an EEG stream returns a buffer of samples. The values in this buffer are measured in Volts.

5.2.2 Channels

Apart from the channels that are selected in the mask when creating the stream, there are also two extra channels at the end of the channel list. These are:

- Trigger channel. This contains the triggers as measured by the amplifier
- Sample Counter. This is a channel containing the index if the sample as measured by the firmware. This counter does not always start at zero. It can be negative as well. The value in this channel will increment by one for each sample, thus is it can be used to measure if there are gaps in the data.

5.3 Impedance Stream

Although it is called a stream, it is actually a state. Calling GetData() on an impedance stream returns the latest known impedance state. This function can be called anytime and does not require to be called once per second.

5.3.1 Units

Calling GetData() on an Impedance stream returns a buffer of one sample. The values in this buffer are measured in Ohm.

6 ERROR HANDLING

All erroneous situations are described by four error types. When such an error occurs, the user will be notified by the SDK, which will throw an exception on the next call to the SDK. In the following chaper, those different error types and the reasons of their occurences will be explained.

Any exception thrown is regarded as a critical error. The only way to resolve it savely is to close all handles and restart from the factory again.

6.1 Not Connected

The exception notConnected is an unrecoverable error state. It happens when the amplifier is physically disconnected from the computing device during operation. It is unrecoverable, because the software (neither the end-user's application, nor the SDK) can do anything to make it connect again.

6.1.1 Resolution

The resolution to solve this error state is to simply reconnect the device. It will be recognized by the SDK as a device with a new handle, although it has the same serial number as a device that was previously seen. The proper resolution would be for the end-user to close the amplifier and start from the beginning again.

6.1.2 Occurrence

When trying to open a stream, or when getting data from a stream, this exception will occur if the device has been disconnected.

6.2 Not Found

If the end-user asks for information that is not available, a notFound exception will be thrown.

6.2.1 Resolution

There is no resolution to overcome the situation where something is not found.

6.2.2 Occurrence

The notFound exception typically occurs when accessing an amplifier this not there (anymore).

6.3 Incorrect Value

If the end-user provides parameters that are not valid, the SDK will throw an incorrectValue exception. The exception may also occur when samples were lost in the underlying layers (usb, driver). The exception itself will contain a string describing what parameter it was, and optionally why it was rejected.

6.3.1 Resolution

The resolution would be to review the incorrect value and try again with a correct value in case incorrect parameters were passed. If there was data loss, there is no resolution.

6.3.2 Occurrence

The most likely places where this exception could occurs, are:

- when trying to open an amplifier with the wrong id/handle
- when trying to open a stream with the wrong parameters (incorrect sampling rate for example)
- when passing an incorrectly sized buffer to the GetData function.

The end-user is responsible for calling the GetData function on a regular interval. If it fails to do so, then data will build up in the SDK itself. To prevent memory usage from building up, the SDK will throw out old data after a certain threshold. The next call to GetData will then throw this exception to notify the user that some data was not found.

Although it should not happen in production environment, there is a chance that USB communication fails for long or short periods. As a result, there may be a gap in the stream. Similar to the scenario above. If this happens, the incorrectValue exception will be thrown the next time the GetData function is called.

6.4 Already exists

This exception, alreadyExists, is currently not used in the SDK. It is reserved for future use.