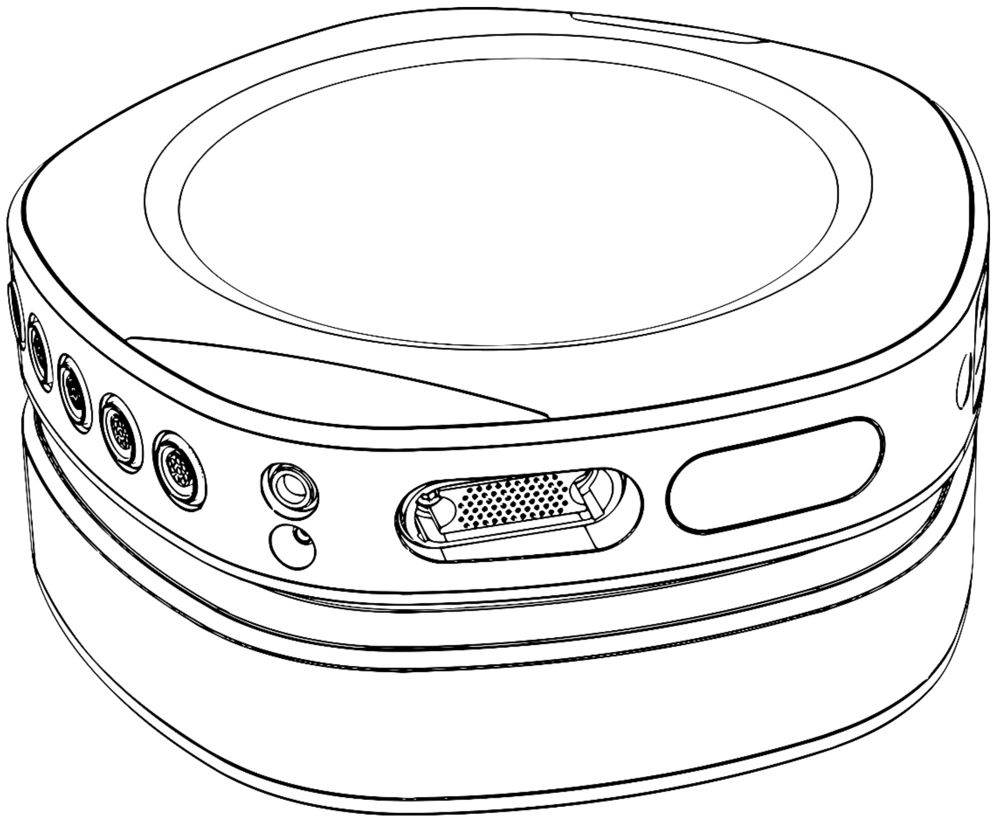

API DOCUMENTATION

TMSI PYTHON INTERFACE

REF: 93-2304-0004-EN-2-0

REV 2 EN

Last update: 2021-12-08

TABLE OF CONTENTS

1 INTRODUCTION

1.1 About this document

This document describes the TMSi Python Interface (*TMSiPy*), what it does, how it should work and what you can and cannot expect from TMSi regarding this interface. Please read carefully through this document.

1.2 TMSi Python Interface

The TMSi Python Interface is a library for Python written by TMSi to interface the SAGA Device Driver to Python. This interface allows you to acquire data, change device configurations and write and read .Poly5 or .XDF files via Python, as well as writing to an LSL stream outlet.

Functionality that is included in the TMSi Python Interface is limited to the signal acquisition part of the driver and loading data which was previously recorded in TMSi Polybench, the TMSi SAGA interface for MATLAB, or the TMSi Python Interface. All functionality needed to perform on-board memory recordings (so-called ambulatory measurement) is **NOT** implemented.

1.3 License and Support

The TMSi Python Interface is **free of charge** and distributed under the Apache License, Version 2.0. The complete text of the license is included in LICENSE.txt, which can be found in the root folder of the TMSi Python Interface.

TMSi has tested the interface, but cannot guarantee that it works under every circumstance, let alone that it will function in the experimental setup that you have in mind.

1.4 TMSi Support

The TMSi Python Interface can be obtained through our website's Support page or directly via our GitLab page. We cannot help you write Python code, but any questions about (using) the TMSi Python Interface can be asked via support@tmsi.com.

2 DESCRIPTION OF THE TMSI PYTHON INTERFACE

2.1 Features

The code in the TMSi Python Interface is a Python library that can be used to control TMSi SAGA devices. The goal of this library is to provide an easy and intuitive way of accessing the TMSi devices using Python. You can configure your experiments, write your own code or turn to supported libraries.

The library provides the following functionality:

- Sampling over USB.
- Sampling over electrical (docked) and optical interface.
- (Limited) live plotting of sampled data.
- (Limited) live filtering of sampled data.
- Sampling in impedance mode and (limited) live plotting of impedance values.
- Changing the SAGA device configuration, including saving and loading the configuration using an XML-structured format.
- Directly saving sampled data to Poly5 file format or XDF file format.
- Loading data from Poly5 or XDF file format.
- Catching device related errors. However, no typical error handling has been included.
- Writing to an LSL stream outlet
- (Limited) live plotting of the sampled data in a heatmap, specifically designed for HD-EMG applications.

The library does not (yet) support the following features:

- Recovery of lost samples when measuring over wireless interface.
- Recording data on SAGA’s onboard memory.

The library has been tested on the following Python version:

- Python 3.8.6 (Windows 10 / 64bit).
- Python 3.6.9 (Ubuntu 18.04 LTS / 64bit).

2.2 Installation requirements

The TMSi Python Interface requires the following for Windows computers:

- A computer with Windows 10, 64bit.
- SAGA Device Driver 2.0.0.

- Python version 3.6 or higher
    - Windows: WinPython64 3.8.6.0 (https://winpython.github.io/)

The TMSi Python Interface requires the following for Linux computers:

- A computer with Ubuntu 18.04 LTS, 64bit.
- SAGA Device Driver 2.1.0.
- Python version 3.6 or higher:
    - Ubuntu: Spyder3 and Python3 libraries

2.3 Installation

When using the Python library, make sure that the TMSi SAGA Device Driver has been successfully installed. Before using the library, copy the folder (including TMSiSDK and examples) to the working directory and start by having a look at the example code provided in the 'examples' folder (explained in chapter 3).

To use the plotters, additional steps need to be taken on Windows, as the correct graphics backend needs to be configured. First, open 'Control Panel → Edit System Environment Variables → Environment Variables'. Next, below User variables, click 'New…'. In the 'Variable name:' field, you should add *PYQTGRAPH_QT_LIB* and in the 'Variable value:' filed, you should add *PySide2*.
When the Spyder IDE is used, it is necessary to configure the plots to change the IPython Graphics Backed to 'inline'. This can be configured from Spyder's menu bar as follows: 'Tools → Preferences → IPython Console → Graphics → Graphics backend → Inline'.

2.4 Additional libraries

Possibly, additional Python libraries need to be installed using the Python package manager. A list of relevant libraries that have been used during development, including version number, can be found in the requirements text-files for Windows and Linux. These packages can either be installed directly or used in a virtual environment. Installing can be done using the Python's package manager `pip`:

```
pip install -r /path/to/tmsi_python_interface/requirements_Windows.txt
```

The use of virtual environments is advised to avoid version dependency issues. More information about the use of virtual environments can be found in the general Python documentation (https://docs.python.org/3/tutorial/venv.html).

2.5 LabStreamingLayer integration

LabStreamingLayer (LSL) is separate program used to stream and record data. In order to use the TMSi Python Interface in combination with LSL, the liblsl-library has to be installed separatly. More information on LSL, including ‘getting started’ information, can be found in the documentation of LabStreamingLayer itself (https://labstreaminglayer.readthedocs.io/info/intro.html).

When using the TMSi Python Interface in combination with LSL, the following warning can show up*: ‘Couldn't create multicast responder for 224.0.0.1 (set_option: A socket operation was attempted to an unreachable host)’*. This warning originates form the liblsl-library and can unfortunately not be solved by TMSi. The warning does not influence the connection between the TMSi Python interface and LSL and can simply be ignored.

2.6 Usage

Below, a code snippet is given that shows basic usage of the TMSi Python Interface.

```
1     from TMSiSDK import tmsi_device
2     from TMSiSDK.device import DeviceInterfaceType, ChannelType
3
4     tmsi_device.initialize()
5     dev = tmsi_device.create(tmsi_device.DeviceType.saga,
          DeviceInterfaceType.docked, DeviceInterfaceType.usb)
6
7     dev.open()
8     dev.start_measurement()
9
10    dev.stop_measurement()
11    dev.close()
```

The first two lines import two different subclasses of the TMSiSDK which are required to run the code snippet. Line 4 of the code snippet initialises the SDK, after which a device object can be created using the required Interface Types. This is shown on line 5 of the code snippet. After successful initialisation of the device object, the connection to the device can be opened (line 7). Next, the user may want to change the configuration, initialise a `file_writer` object or `plotter` object. The use of these functionalities is not shown in the code snippet, but can be found in the different examples that are described in Chapter 3. The code snippet shows how to start and stop a measurement on lines 8 and 10. Finally, it is important that the connection to the device is closed at the end of each code execution, which is shown on line 11 of the code snippet.

2.7 Content

The library contains the following:

TMSiSDK/

Folder containing all SDK files. This should be copied to the directory from which you want to use the TMSi Python Interface.

TMSiSDK/device

This class is the main device interface and explicitly defines how the classes, properties and methods of TMSi devices are specified in the TMSi Python Interface.

TMSiSDK/error

This class contains TMSi Error codes that can be retrieved in the application.

TMSiSDK/file_writer

This class explicitly states how the classes, properties and methods of file writers for specific data formats need to be defined.

TMSiSDK/sample_data

This class defines the properties of received samples.

TMSiSDK/sample_data_server

This class puts data, retrieved from a specific device, into a queue. Different consumer types, such as file writers or plotters, are registered to the created queues. The module keeps track of which consumers are registered to which device.

TMSiSDK/settings

This class specifies the ‘locally’ used settings of the SDK module.

TMSiSDK/tmsi_device

This class is a starting module to initialise the TMSi Python Interface and to instantiate device objects.

TMSiSDK/_resources

Folder containing EEG channel locations of our headcaps. Files are used for plotting purposes and electrode locations in .XDF files.

TMSiSDK/configs

Folder containing different xml-configuration files. This directory was previously located in the examples folder.

TMSiSDK/devices/

Folder containing device-specific implementations of the SDK. Currently, only the SAGA device is supported.

TMSiSDK/devices/saga/

This folder contains all SAGA-specific device interface implementations, such as the TMSi_Device_API. Furthermore, this folder includes the saga_device and saga_types class, which contain device-specific classes and methods based on the general TMSiSDK/device class. Finally, this folder contains the XML-based reading and writing of SAGA's configuration.

TMSiSDK/file_formats/

This folder contains all different types of specific file writers. This version of the TMSi Python Interface contains a class which writes data to TMSi Polybench native .Poly5 files, .XDF files or an LSL-stream outlet.

TMSiSDK/file_readers/

This folder contains all different types of specific file readers. This version of the TMSi Python Interface contains a class which reads data from TMSi Polybench native .Poly5 files as well as a reader for .XDF files. This makes it possible to review/analyse recordings in the Python environment. The XDF file format enables loading and processing data in the analysis library "MNE-Python" (https://mne.tools/stable/).

TMSiSDK/filters/

This folder contains the different type of pre-configured filters that can be used in combination with a plotter object. Currently, only a semi-real-time filter is implemented that can be used to retrieve and filter data, which can be passed to the plotter. Different (Butterworth) filters can be generated for the three analogue channel types (BIP, UNI and AUX). The user can configure a high-pass, low-pass and band-pass filter for these channel groups.

TMSiSDK/plotters/

This folder contains the two types of plotters that are available to the user.

TMSiSDK/plotters/impedance_plotter

This class creates a GUI window that displays the measured impedance values in either a grid or head layout. Next to the plot, the specific values per channels are displayed. When the GUI is closed, the device stops the impedance measurement. When file storage is enabled, the measured impedance values are saved to a .txt-file in the measurements folder.

TMSiSDK/plotters/plotter

This class creates a GUI window that displays the sampled time-series data online. For reasons of performance, the visualised data is downsampled to 500 Hz. The plotter allows for autoscaling of the different channels, observing different time ranges (from 1 to 10 seconds) and making a selection on which channels to display during the recording. When the GUI is closed, the device stops sampling data. This class loads the plotter_gui.py file that has been created using Qt Designer.

TMSiSDK/plotters/plotter_hd_emg

This class provides a GUI window that displays the high-pass filtered RMS values in a grid layout. To improve the resolution, linear interpolation is applied to neighbouring data points. The plotter is both available for 32-channel as well as 64-channel configurations. The user can set the orientation of the grid, as well as the upper limit for the colorbar, prior to starting the application. Note: This plotter is not compatible with the current version of the TMSi Python Interface on Linux (Ubuntu 18.04 LTS).

3 EXAMPLES SAGA

The easiest way to start using the interface is to follow the examples. With the examples you will be guided through the main features of the TMSi Python Interface. Connect a TMSi SAGA device to the PC as you would normally do and run the different examples. All examples use the USB and electrical (docked) interface as default interface types.

Example_bipolar_and_auxiliary_measurement.py

This example shows the functionality to display the output of an AUX sensor and the output of a simultaneously sampled BIP channel on the screen. To do so, the channel configuration is updated and a call is made to update the sensor channels. The data is saved to a .Poly5 file.

Example_changing_channel_list.py

This example shows the manipulation of the active channel list and demonstrates how the ChannelName property can be changed.

Example_changing_reference_method.py

This example shows how to change the reference method. The configurable options are common reference (ReferenceMethod.common) and average reference (ReferenceMethod.average). When using common reference mode, automatic switching from common to average reference, in case the common reference electrode is out of range, can also be disabled or enabled. This can be done configured using ReferenceSwitch.fixed and ReferenceSwitch.auto.

Example_changing_sample_rate.py

This example shows how to change the Base Sample Rate of SAGA, as well as how the sample rate of individual channels can be changed.

Example_config_settings.py

This example shows a brief overview of the different configuration settings of the SAGA device and how to change these settings. More elaborate explanations are given in the examples of the separate configuration items.

Example_EEG_workflow.py

This example shows the functionality of the impedance plotter and the data stream plotter. The example is structured as if an EEG measurement is performed, so the impedance plotter is displayed in head layout. The channel names are set to the name convention of the TMSi EEG cap using a pre-configured EEG configuration. The data is saved to either the Poly5 or XDF file format, depending on user input.

Example_EMG_workflow.py

This example shows the functionality of both the impedance plotter and the HD-EMG heatmap plotter. It is similar to the EEG example, but is modified for a HD-EMG workflow. Note: This plotter is not compatible with the current version of the TMSi Python Interface on Linux (Ubuntu 18.04 LTS).

Example_factory_defaults.py

This example shows how to initiate a factory reset of the SAGA device.

Example_filter_and_plot.py

This example shows how to couple an additional signal processing object to the plotter. The application of a bandpass filter on the first 24 UNI channels is demonstrated. The filter is only applied to the plotter for displaying purposes, the saved data does not contain any filtered data.

Example_impedance_plot.py

This example shows the functionality of the impedance plotter.

Example_load_save_configurations.py

This example shows how to load/save several different configurations from/to a file (in the “TMSiSDK/configs” directory).

Example_read_data.py

This example shows how to read data from a .Poly5 file.

Example_stream_lsl.py

This example shows how to set-up an LSL-stream for SAGA, which can be used in combination with other LSL applications.

Example_switch_dr_interface.py

This example shows how to change the interface between SAGA's Docking Station and Data Recorder.

Example_xdf_to_MNE.py

This example shows how to read data from an .XDF-file and convert it in a data object that can be used in MNE-Python.

4 CHANGES WITH RESPECT TO PREVIOUS VERSION

The current version of the TMSi Python interface includes some new functionalities with respect to the previous version. These changes include:

Added functionality:

- Added possibility to save data to the XDF file format
- Added possibility to save impedance values, both as .txt-file and included in an .XDF file
- Added LSL stream writer
- Added options to control the full device configuration
- Added an HD-EMG heatmap plotter

Improvements and changes

- Improved performance of the interface
- Improved error handling of file-writer
- Improved performance of different plotters for different screen resolutions
- Changed handling of relative imports
- Changed location of configuration and location files
- Prepared plotter and examples for multi device experiments

5 FAQ

An unexpected error occurred. Is there anything I should do before I can start a new measurement?

A: When the script stops running, this does not necessarily mean that the device has stopped sampling. Restarting your Python kernel should ensure that the device stops sampling and is disconnected, after which you can reopen the connection to the device in a new Python kernel.

Where can I find the channel name and units of my data?

A: The channel name of the data can be found in device/data channel property: (`data.channels[i]._Channel__name`) with `i` the desired channel number. The units of the data can be found in the device/data channel property: (`data.channels[i]._Channel__unit_name`) with `i` the desired channel number.

Why do I have to set a divider for a different sample rate?

A: The sample rate of SAGA device can only be set by the base sample rate divided by a power of 2. For example, with a base sample rate of 4000 Hz, the possible sample rates are 4000, 2000, 1000 or 500 Hz. To get these you will have to set the divider to respectively 1, 2, 4, 8.

`Sample_rate=base_sample_rate/n` with `n=[1, 2, 4, 8]`

What do I need to do if I get "docking station response time-out" after an error?

A: First thing to do is to repower both docking station and data recorder and try again. If this does not work, restart the Python Console, else restart the computer. Also make sure you have followed the steps mentioned in the User Manual of your device, for correct installation.

I want to control configurations that are not shown in the examples. How do I know how to control them?

A. Not all functionality of the interface is shown in the examples. More information can be found in the inline documentation of the interface. More information about general control of the SAGA device can be found in the SAGA Device API.

6 CHANGELOG

June 10th, 2021 – Initial release of the TMSi Python Interface.

December 8th, 2021 – Second release of the TMSi Python Interface, see chapter 4 for an overview of changes and added functionality.