apex

API DOCUMENTATION

TMSI PYTHON INTERFACE FOR APEX

TMSi

REF: 93-2404-0004-EN-1-0

REV 1 **EN**

Last update: 2022-12-20

TABLE OF CONTENTS

1 INTRODUCTION

1.1 About this document

This document describes the (APEX-specific implementation of 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 APEX Device Driver to Python. This interface allows you to acquire data, change device configurations and write and read .Poly5 or .XDF files via Python. Also, functionality to convert files from .Poly5 to .EDF file formats is included.

All of APEX' functionalities are available in the TMSi Python Interface.

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 APEX devices. The goal of this library is to provide an easy and intuitive way of accessing 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 or Bluetooth.
- (Limited) live plotting of sampled data.
- (Limited) live filtering of sampled data.
- Sampling in impedance mode and (limited) live plotting of impedance values, for both pre-measurement and live impedance acquisition.
- Changing the 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.
- Offline conversion of sampled data from .Poly5 file format to .EDF file format.
- Loading data from .Poly5, .XDF or .EDF file format.
- Catching device related errors. However, no typical error handling has been included.
- Downloading recordings stored on onboard memory.
- Configuring the device for a card recording

The library has been tested on the following Python version:

- Python 3.9.12 (Windows 10 / 64bit).
- Python 3.10.8 (Windows 10 / 64bit).

2.2 Installation requirements

The TMSi Python Interface requires the following for Windows computers:

- A computer with Windows 10, 64bit.
- APEX Device Driver 1.0.0.0.
- Python version 3.9 or higher
    - Python for Windows: (https://www.python.org/downloads/release/python-3912/).

Other Python versions have not been tested with this release (v4.0.0.0).

2.3 Installation

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

Additional Python libraries need to be installed using the Python package manager. Doing so is done best using virtual environments. The Python interface is provided with a PowerShell script that enables the creation of a virtual environment, and the installation of the required libraries (stored in the *requirements3x_Windows.txt*). In order to run this script, you need to ensure that Python is available on the system's path, which can be confirmed during Python installation or manually via Windows' Environment Variables. To run the PowerShell script, it is needed to give permission to PowerShell. Allowing the installer script to run can be done by opening a command prompt in the directory of the Python interface (type *cmd*), and typing the following line:
PowerShell.exe -ExecutionPolicy UnRestricted -File .\installer39_Windows.ps1

When the Spyder IDE is used, it's needed to change the Python interpreter to the virtual environment's Python installation. The custom environment can be configured in *Preferences → Python interpreter.* The specific file to select can be found in the following path: *.../tmsi_python_interface/.venv/Scripts/python.exe*
A final step in using Spyder is to configure 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'. Plots can be made interactive again, for instance for analysis purposes, using the *ipython* library that comes with Spyder. This can be done using the two lines of code below:

```
ipython = get_ipython()
ipython.magic("matplotlib qt")
```

2.4 LabStreamingLayer integration

The TMSi Python interface provides functionalities to create LSL streams from Python. With the initial market release of APEX, the version which is described in this document, no accurate results can be guaranteed when writing to LSL streams.

In a later release of the TMSi Python interface, this functionality is also to be included for APEX, so that accurate synchronization of streams in an .XDF file can be obtained.

2.5 Usage

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

```
1    from apex_sdk.tmsi_sdk import TMSiSDK, DeviceInterfaceType, DeviceType
2    from apex_sdk.device.tmsi_device_enums import MeasurementType as
         ApexMeasurementType
3
4    TMSiSDK().discovery(DeviceType.apex, DeviceInterfaceType.usb)
5    discoveryList = TMSiSDK().get_device_list()
6
7    dev = discoveryList[0]
8    dev.open()
9    dev.start_measurement(ApexMeasurementType.APEX_EEG)
10
11   dev.stop_measurement()
12   dev.close()
```

The first two lines imports the required classes to run the code snippet. Line 4 of the code snippet initialises the SDK, and performs a device discovery with specific Interface Types. Line 5 shows how to retrieve a handle to this result. After successful discovery of the available device, a device object can be created (line 7), after which the connection to the device can be opened (line 8). 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 9 and 11. 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.

2.6 Content

The library contains the following directories and classes, which is different compared to the SAGA-specific version of the TMSi Python Interface. In a next release, the SAGA-specific version will be updated to the methodology outlined in the APEX SDK structure.

apex_sdk/

Folder containing all APEX-related SDK files.

TMSiFileFormats/

Folder containing file writers and file readers.

TMSiPlotters/

Folder containing the different plotter objects and a GUI framework in which the plotting windows can be embedded.

TMSiProcessing/

Folder containing additional processing capacities. Currently, this contains a class that creates a framework for real-time filtering of incoming data.

apex_sdk/tmsi_sdk

This class is a starting module to initialize the TMSi Python Interface and to instantiate APEX device objects. The device object provides a handle to all of the device-specific implementations that are available in the interface.

apex_sdk/device

This folder contains device-generic classes that provide an interface to specific devices in the various examples. One can find generic implementations for channels, devices, dongles, event-readers, measurements and the device's sampling threads.

apex_sdk/sample_data_server

This folder contains all functionality related to the sample_data_server, which is responsible for putting 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 and contains methods to register and unregister consumers from the server.

apex_sdk/tmsi_errors

This folder contains a wrapper to display device-related errors cleanly in the application.

apex_sdk/tmsi_utilities

Folder containing various utility functions and abstractions that are used by the SDK. The functions in this folder are not to be used directly, except for the "*apex_structure_generator*". This class provides a method to create abstract device-

specific structures which are needed to specify (for instance) the card configuration, in a non-abstract manner.

apex_sdk/device/devices/apex

This folder contains all APEX-specific implementations of the generic structures and classes that are outlined in the "apex_sdk/device" folder. All (publicly) available methods for APEX, that are used in the examples, can be found in the *apex_device* file. Also, this folder contains the wrapper for the C-based APEX device API.

TMSiSDK/_resources

Folder containing EEG channel locations of our headcaps and conversion files for displaying the Textile HD-EMG grids correctly. Files are used for plotting purposes and electrode locations in .XDF files.

TMSiSDK/configs

Folder containing different xml-configuration files.

TMSiFileFormats/file_writer

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

TMSiFileFormats/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, as well as conversion of .Poly5 data files to .EDF files. When using APEX with an LSL-stream outlet, an error will be thrown, as no accurate synchronization results can be guaranteed at this point.

TMSiFileFormats/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 readers for .XDF and .EDF files. This makes it possible to review/analyze recordings in the Python environment. The .XDF and .EDF file format enables loading and processing data in the analysis library "MNE-Python" (https://mne.tools/stable/).

TMSiPlotters/gui/

This folder contains the GUI with which the user can interact when plotting data. All different plotter objects are embedded in this GUI. Opening and closing of the GUI controls starting and stopping of measurements.

TMSiPlotters/plotters/

This folder contains the different plotters that are available to the user.

TMSiPlotters/plotters/impedance_plotter

This class creates a viewer for the measured impedance values in either a grid or head layout. Next to the plot, the specific values for each channel is displayed. When file storage is enabled, the measured impedance values are saved to a .txt-file in the measurements folder.

TMSiPlotters/plotters/plotter

This class creates a viewer that displays the sampled time-series data online. For reasons of performance, the visualised data is downsampled to 500 / 512 Hz. The plotter allows for autoscaling or changing the displayed range 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.

TMSiPlotters/plotters/plotter_hd_emg

This class is part of the TMSi Python Interface, but is not intended to be used with APEX. This type of plotter is only to be used with SAGA.

TMSiProcessing/filters/

This folder contains the different types 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.

TMSiProcessing/synchronisation/

This folder contains a class that makes it possible to synchronise different streams in an .xdf file, based on the timestamps that the LSL framework registered during acquisition. This file is to be used with .XDF files acquired with SAGA only.

3 EXAMPLES APEX

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 APEX device to the PC and run the different examples. Except for the Bluetooth example, all examples use the USB interface as default interface type.

Example_bluetooth_measurement.py

This example shows how to discover and connect to a device over the Bluetooth interface

Example_card_configuration.py

This example shows how to change the card recording configuration.

Example_changing_channel_list.py

This example shows how to change which channels are included in the reference and shows how to change specific channel names.

Example_changing_configuration.py

This example shows how to change the device sampling configuration. The example shows how to change the sampling frequency, disable the live-impedance measurement and set an impedance limit for the headcap indicator.

Example_config_settings.py

This example shows how to load and save .xml configurations.

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_factory_defaults.py

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

Example_filter_and_plot.py

This example shows how to connect an additional signal processing object to the plotter. The application of a bandpass filter on the CAP 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 how to perform an impedance measurement and plot the data. The data can be saved in a .txt-file.

Example_SD_card_download.py

This example shows how to download card recordings that are stored on the onboard memory.

3.1 Examples reading data

Aside from the examples that show how to specifically operate APEX, there are also some generic examples that show how to load or process acquired data. This is to be found in the “examples_reading_data” folder. This folder contains the examples below.

Example_edf_to_MNE.py

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

Example_poly5_to_edf.py

This example shows the offline conversion of .Poly5 files to .EDF files. Please note that EDF is a 16-bit file format, which requires pre-processing to remove the offset. The data can be either low-pass or bandpass filtered, in order to retain as much relevant information as possible in the conversion from the 32-bit to the 16-bit file format. Also note that the conversion requires the entire data file to be loaded in memory. The available RAM can be a limiting factor in the conversion of large data files.

Example_poly5_to_edf_batch_conversion.py

This example shows the offline conversion of .Poly5 files to .EDF files in batch, by converting all files in a selected folder (and its subfolders).

Example_read_data.py

This example shows how to read data from a .Poly5 file. Next to this, the reordering strategy for the Textile HD-EMG grid is demonstrated. Finally, conversion to an MNE-Python object is demonstrated.

Example_synchronise_xdf.py

This function shows how to synchronise two data streams in an .xdf file, that have been collected with the LSL-based LabRecorder application. This example is intended for use with SAGA.

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 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. Please reset the device (removing the USB cable or performing a hardware reset in the case of Bluetooth), after which you can reopen the connection to the device in a new Python kernel.

What parameters/(meta)data can I access when loading a .Poly5 file?

A: The following parameters can be retrieved when a Poly5Reader object is created: `samples` (containing the acquired data), `channels` (list containing all information from the Channel class), `sample_rate`, `num_samples`, `num_channels`, `ch_names` (containing all channel names) and `ch_unit_names` (containing all unit names of the used channels).

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

A: First thing to do is to repower the device and try again. If this does not work, restart the Python Console, otherwise 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 device can be found in the Device API.

5 CHANGELOG

December 20, 2022 – Initial release of APEX-specific documentation of the TMSi Python Interface

www.tmsi.com