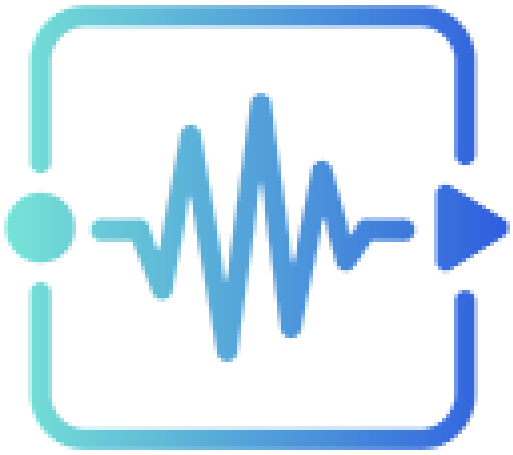
PyAWECore Library Installation Guide

(C) DSP Concepts



PyAWECore Library Installation Guide

This document describes how to install **PyAWECore Library** on your PC.

It also briefly mentions what PyAWECore is and what can be done with it.

Content of This Directory

The installation directory contains:

- `awecorelib-*` wheels for Linux and for Windows
- `pyawe_awb` wheel - this is a dependency package to `awecorelib`
- `Designs` and `Audio` directories with some data to run some basic scripts (see below)
- `Samples` directory with additional Python scripts using **PyAWECore Library**

What is PyAweCore Library?

PyAWECore Library is a Pythonic wrapper or abstraction for AWE Core. It allows the use of *AWE Core* in Python scripts or programs.

This enables the processing of AWE signal flows conveniently on the PC, without the need to open them in AWE Designer or in Matlab.

Once **PyAWECore Library** is installed, a command line script can be used for passing audio data to AWE Core. By using **PyAWECore Library**'s API other use-cases can be implemented too.

This little sample code shows how to process a WAV file.

```
import awecorelib
import wave

signalflow_file = "my_signalflow.awb"
wave_input_file = "my_inputfile.wav"

# Initialization of AWE Core instances and loading an AWB file
awe = awecorelib.init_from_awb(signalflow_file)
awb = awe.load(signalflow_file)

collected_data = bytearray(b'')
with wave.open(wave_input_file, 'r') as wav_in:
    data_in = wav_in.readframes(awb.layout_info.fundamental_blocksize)
    while data_in:
        data_out = awe.pump(data_in, 2)  # assuming a stereo-wav file here!
        collected_data.extend(data_out)
        data_in = wav_in.readframes(awb.layout_info.fundamental_blocksize)

print(f"Processing done: {len(collected_data)} bytes of audio data obtained")
```

Limitations

Note: For license protection purposes, default builds of PyAWECore Library impose a 3-hour timeout upon the AWE Core instance. Hence, in the code example above, users would need to re-instantiate the line `awe = awecorelib.init_from_awb(signalflow_file)` at least once every three hours. Please contact your sales representative for a custom build of PyAWECore Library if you need AWE instances to run for longer durations.

Note 2: As of the 2.1.0 release, the TensorFlow Lite Micro module is not supported with the Linux (x86) PyAWECore Library wheel. However, this module is supported with the Windows wheel. This issue will be

fixed in a future release.

Installation

Currently, **PyAWECore Library** is provided as -so called- Python wheel files. Those files are part of the Windows AWE Designer installation and will be placed into a directory `NAME_ME_CORRECTLY` inside the base installation directory, for example under `C:\DSP Concepts\AWE Designer 8.D.3 Pro`.

Those wheel files have to be used with a Python package manager.

Prerequisites

A Python installation is required. Please use a Python version that corresponds to the Python version mentioned in the wheel name, e.g., `cp311`.

For Windows, **PyAWECore Library** is built using Python 3.11, as this also is compatible with the underlying Matlab engine. See python matlab compatibility for more details.

For WSL (Windows Linux Subsystem) and other Linux systems, Python 3.10 is currently used.

Installation of Wheels

Typically, Python packages are installed into "environments" and not into the system wide Python installation. There are many ways to handle those environments, like pyenv or conda or simply pip - which also comes with the Python installation by default.

Perform the following steps in a command terminal/console to create a Python environment and install the wheels:

Windows:

- Run `py -3 -m venv aweenv` to create a local environment; possibly you can use `python` instead of `py` too. This depends on your Windows installation.
- `.\aweenv\Scripts\activate` to activate this environment

Linux:

- Run `python -m venv aweenv`
- `source ./aweenv/bin/activate`

Once the environment is activated, you will have a change in the console prompt. You can then install the wheels into this environment:

- Windows: `(aweenv) pip install pyawe_awb-*.whl awecorelib-*-win_amd64.whl`
- Linux: `(aweenv) pip install pyawe_awb-*.whl awecorelib-*-linux_x86_64.whl`

Please note that you need to be connected to the internet when running this installation command as further Python package dependencies are retrieved.

Commands Available

Once the Python environment is activated and the **PyAWECore Library** packages are installed, some scripts are available:

- `(aweenv) awecorelib-docs` - this shows the documentation in HTML (in your system's browser)
- `(aweenv) awecorelib-filepump` - this can read audio data from a WAV file and "pump it" through a design (see below)
- `(aweenv) awecorelib-config` - this shows the underlying AWE Core build and version information

Processing Audio

As mentioned above, **PyAWECore Library**'s API can be used to implement programs covering many different use-cases. One of the typical tasks is to process an audio file through an AWE design. **PyAWECore Library** already provides a rudimentary implementation of such a program, called `awecorelib-filepump`.

Sine Wave Generator

A simple sine wave generator layout can be used to generate a stereo WAV file. This first channel contains a clean sine tone, and the second channel is distorted with a pink noise.

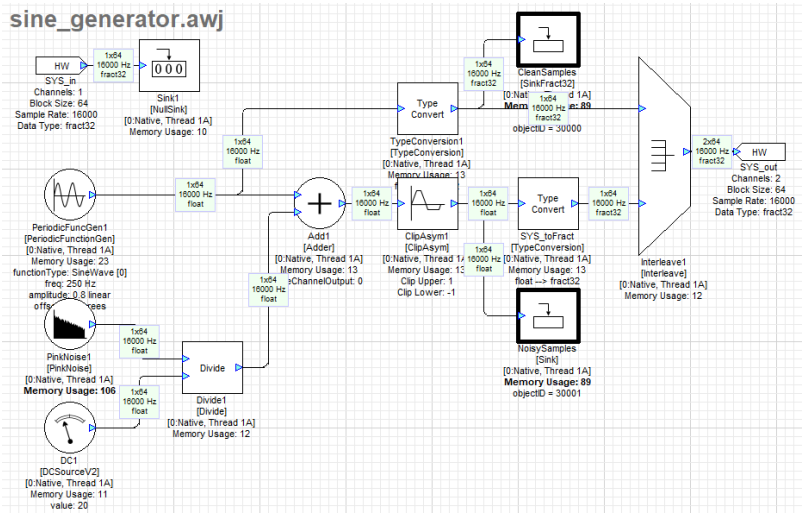


Figure 1: sine_generator.png

To run this with **PyAWECore Library** use:

- `(aweenv) awecorelib-filepump ./Designs/sine_generator.awb -c 1000 -o my_sine_output.wav`

This generates a 4-second-long audio signal (16000 / 64 = 4ms * 1000 = 4s) in the file `my_sine_output.wav`.

Passthrough with Attenuation

A sine stereo input signal is passed through a design which attenuates the first channel by minus 10dB.

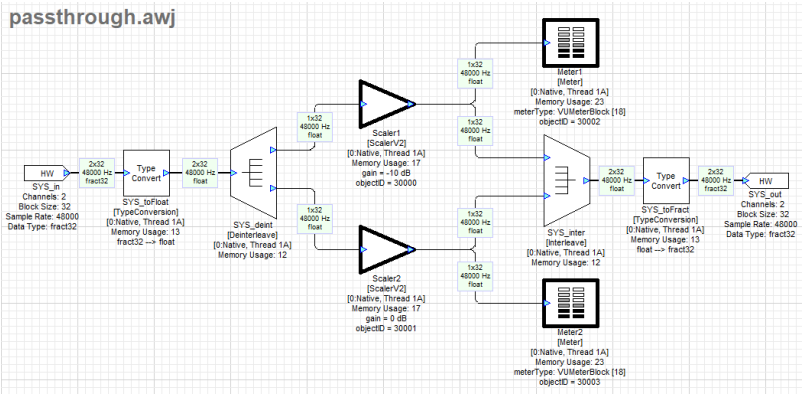


Figure 2: passthrough.png

To run this with **PyAWECore Library** use:

- `(aweenv) awecorelib-filepump ./Designs/passthrough.awb -w ./Audio/input_audio.wav -o my_passthrough_output.wav`

This generates the file `my_passthrough_output.wav` which should have the same content as `./Audio/output_processed_audio.wav` then.

Passthrough and Controlling the Attenuation

The same passthrough design is used now, but programmatically the gain on the second channel is decreased during processing.

To run this with **PyAWECore Library** use:

- `(aweenv) python ./Samples/attenuation_control.py`

You should see a step-wise decrease of amplitude in the second channel in file `passthrough_output_controlled.wav`.

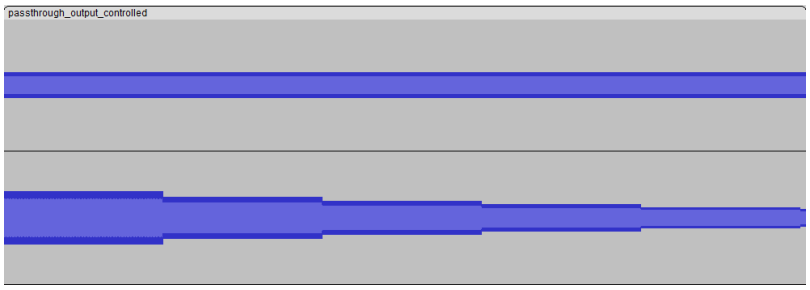


Figure 3: output_processed_with_control.png