**CoreAudioProcessor.py – Usage and Design Notes**

**Bigya Bajracharya; ed. Phil Pfeiffer**

**Last Updated: April 11, 2024**

Table of Contents

[1. Introduction 1](#_Toc163696789)

[1.1 Purpose 1](#_Toc163696790)

[1.2 System Requirements 1](#_Toc163696791)

[2. Operation 1](#_Toc163696792)

[2.1 Overview 1](#_Toc163696793)

[2.2 Inputs 2](#_Toc163696794)

[2.3 Outputs 2](#_Toc163696795)

[2.4 Command Line Arguments 2](#_Toc163696796)

[3. Internal Design: Notes 2](#_Toc163696797)

[3.1 Architecture 2](#_Toc163696798)

# Introduction

## Purpose

The CoreAudioProcessor app transcribes and optionally diarizing a user-specified recording, optionally generating logs that characterize its operation.

A variety of supporting options can be used to specify different aspects of CoreAudioProcessor’s operation, including the names of the files it generates, the size of the model it uses to do transcription, whether to do diarization following transcription. whether to overwrite existing files in its output directory, and whether it generates logs.

## System Requirements

* **Python Environment**: CoreAudioProcessor requires Python 3.12 or later to run. Python can be obtained for free from the official Python website, <https://www.python>.org
* **Required PyPi Packages**
  + **xmlschema**: For using schemas to validate XML content.
* **Other**
  + **WhisperX**: The transcription and diarization utility. For more information on how to obtain, install, and configure WhisperX for use, see the installation guide for the appropriate target platform.

# Operation

## Overview

As input, CoreAudioProcessor accepts the name of a file and optional, assorted parameters for configuring its operation. CoreAudioProcessor transcribes, then optionally diarizes, the specified file, generating a transcript.

As part of this process, CoreAudioProcessor, by default, produces two logs:

* output generated from an incomplete or failed attempt to process the specified recording
* an XML-formatted file that characterizes the time needed to process this recording, together with data on the degree to which processing succeeded.

CoreAudioProcessor offers four ways to customize its operation:

* **Baseline Configuration**: This is a set of default settings built into the program.
* **Configuration File**:
  1. CoreAudioProcessor provides an XML file that can be changed to override the default settings in the baseline configuration.
  2. Alternatively, a configuration file can be specified on an execution’s command line. These configuration-file’s values will override the values specified in the CoreAudioProcessor’s built-in configuration file and its baseline configuration.
* **Command-line Parameters**: CoreAudioProcessor also accepts options on its command line. One such option, the source file, is required. Other options, if specified, override CoreAudioProcessor’s default and configuration file options.

## Inputs

* **Audio File Path**: This is the source directory for files to transcribe.
* **Command Line Parameters:** These parameters, which control how CoreAudioProcessor operates, are described below. More information can be obtained from CoreAudioProcessor’s header comment.

## Outputs

CoreAudioProcessor, if successful, creates a transcription of the specified audio recording in the specified output directory. If enabled, it also diarizes this recording and logs application activity and errors. Final performance data is logged if enabled.

## Command Line Arguments

CoreAudioProcessor accepts the following command-line arguments:

* + **-au, --audio\_file**: Path to the input audio file or directory. (Required)
  + **-cx, --config\_processor**: Path to an alternative XML configuration file.
  + **-bs, --batch\_size**: The batch size for transcription.
  + **-ct, --compute\_type**: Specifies the computation type.
  + **-dv, --device**: Hardware device for diarization.
  + **-ed, --enable\_diarization**: If true, enable diarization after transcription.
  + **-el, --enable\_logfile**: If true, enable logfile output.
  + **-ht, --hf\_token**: The user token needed for diarization.
  + -**ld, --logfile\_dir**: Directory to which to write the log file.
  + **-ln, --logfile\_name**: Name of the log file.
  + **-ms, --model\_size**: The model size for transcription.
  + **-od, --output\_dir**: Directory for storing the transcriptions.
  + -**ov, --enable\_overwrite**: If true, overwrite any earlier output.
  + **-lp, --performance\_log\_name**: Name for the performance log file.

# Internal Design: Notes

## Architecture

CoreAudioProcessor.py is structured as a stack of the following modules, which have been layered as shown below:

* main.py: CoreAudioProcessor’s entry point. Its header comment describes the application’s parameters, giving design notes.
* WhisperxTranscriber.py: Handles the transcription process by using the WhisperX speech recognition library to transcribe and optionally diarize audio files. WhisperX is a third-party library that augments OpenAI's transcription library, Whisper, with support for diarization.
* TranscriptionConfig.py: Handles the configuration settings for the application. It reads and validates configuration parameters from various sources, such as command-line arguments, configuration files, or default settings, and ensures that transcription-related parameters are properly configured.
* CommandLineParser.py: Provides command-line parsing for the application using the argparser library.
* XMLProcessor.py: Deals with reading and writing XML files, particularly for configuration purposes. It provides functionalities to parse and manipulate XML data, facilitating the management of configuration settings.
* LogWrapper.py: The LogWrapper class is used to configure and initialize logging with predefined settings. It provides logging functionality to monitor and troubleshoot the application. It logs various events, errors, and performance data, aiding in debugging and understanding the application's behavior.
* DEFAULTS.py: Contains definitions for values and settings for an audio transcription system, including paths, file extensions, logging configurations, model types, and other constants for the system's default behavior.
* CONSTANTS.py: Contains values for various keys used throughout the application. These constants include values for default settings, logging codes, or other keys that are reused across modules.
* StatusManager.py: Manages the overall status and errors of the application. It supports the generation of program status messages.