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

**Bigya Bajracharya; ed. Phil Pfeiffer**

**Last Updated: April 11, 2024**

Table of Contents

[1 Introduction 1](#_Toc163694938)

[1.1 Purpose 1](#_Toc163694939)

[1.2 System Requirements 1](#_Toc163694940)

[2 Operation 1](#_Toc163694941)

[2.1 Overview 1](#_Toc163694942)

[2.2 Inputs 2](#_Toc163694943)

[2.3 Outputs 2](#_Toc163694944)

[2.4 Command Line Arguments 2](#_Toc163694945)

[3 Internal Design: Notes 3](#_Toc163694946)

[3.1 Architecture 3](#_Toc163694947)

[3.2 Supporting Applications 3](#_Toc163694948)

# Introduction

## Purpose

The AudioDispatcher app manages the transcription of audio recordings in a user-specified directory structure: i.e., a directory, together with that directory’s subdirectories.

A variety of supporting options can be used to specify different aspects of AudioDispatcher’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**: AudioDispatcher 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**
  + **CoreAudioProcessor.py:**  Supporting application for transcribing and diarizing individual files.

# Operation

## Overview

As input, AudioDispatcher accepts the name of a directory and an optional list of extensions of files to transcribe and diarize within that directory and its descendant directories. The AudioDispatcher walks the directory structure, using a supporting application, CoreAudioProcessor,py, to transcribe and diarize that structure’s files. Those files are then written to a user-specifiable directory whose structure mirrors that of the source directory structure.

As part of this process, AudioDispatcher, by default, produces three logs:

* a summary list of files it processed during its execution, along with the status of those operations
* a detailed list of outputs generated from incomplete and failed attempts to process recordings
* an XML-formatted file that characterizes the time needed to process each file, together with data on the degree to which processing succeeded.

AudioDispatcher offers four ways to customize its operation:

* **Baseline Configuration**: This is a set of default settings built into the program.
* **Configuration File**:
  1. AudioDispatcher 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 AudioDispatcher’s built-in configuration file and its baseline configuration.
* **Command-line Parameters**: AudioDispatcher also accepts options on its command line. One such option, the source directory, is required. Other options, if specified, override AudioDispatcher’s default and configuration file options.

## Inputs

* **Audio Directory Path**: This is the source directory for files to transcribe. AudioDispatcher will transcribe all audio files found in this directory and its subdirectories, subject to the option, if enabled, that suppresses the overwriting of existing transcriptions.
* **Command Line Parameters:** These parameters, which control how AudioDispatcher operates, are described below. More information can be obtained from AudioDispatcher’s header comment.

## Outputs

As AudioDispatcher processes the files in the specified source directory, it creates a transcription of each audio recording in the specified output directory. If enabled, it also diarizes each recording and logs application activity and errors. Final performance data is logged if enabled.

## Command Line Arguments

AudioDispatcher accepts the following command-line arguments:

* **Operating parameters**
  + **--audiodir, -au:** Path to the directory containing the audio files to be processed. (**Required**)
  + **--config\_dispatcher, -cxds**: Path to an alternative XML config file for this application.
  + -**-extensions, -ex**: List of file extensions to be treated as audio recordings.
* **System Environment**
  + **--python, -py**: Path to the Python interpreter executable.
  + **--audio\_processor, -pr**: Path to the single-file audio processor application.
  + **--enable\_logfile, -el**: If **True**, enables the logging of application activities and errors.
  + **--logfile\_dir, -ld**: Directory for user-visible log files.
  + -**-summary\_logfile\_name, -lnsm**: Summary log file name.
  + **--detailed\_logfile\_name, -lndt**: Detailed log file name.
  + **--performance\_logfile\_name, -lnpf**: Performance data log file name.
* **Audio Processor Configuration**
  + **--config\_processor, -cxpr**: Path to an alternative XML config file for the audio processor.
* **Transcription Settings**
  + **--batch\_size, -bs**: The batch size for transcription.
  + **--compute\_type, -ct**: Specifies the computation type.
  + **--device, -dv**: Hardware device for diarization.
  + -**-enable\_diarization, -ed**: If **True**, enables diarization after transcription
  + **--hf\_token, -ht:** The user token needed for diarization.
  + -**-model\_size, -ms**: The model size for transcription.
  + **--output\_dir, -od**: The directory to store transcriptions.
  + **--overwrite, -ov:** If set to true, overwrites existing output. (Default: True).

# Internal Design: Notes

## Architecture

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

* main.py: AudioDispatcher’s entry point. Its header comment describes the application’s parameters, giving design notes.
* Transcribe.py: Handles the transcription process by calling the CoreAudioProcessor app. This module interfaces with the app, directing it to processes audio data and write it to a spot in the user-specified target directory tree. The module also recovers CoreAudioProcessor’s output, using it to build AudioProcessor’s logs.
* 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.

## Supporting Applications

ApplicationDispatcher.py invokes CoreApplicationProcessor.py to process individual transcripts, capturing that program’s standard output, standard error output, and logs for generating its log files.