Programmer's Documentation
Author: Cole Vikupitz
Last Modified: 10/9/2017

Introduction

In this document, I will cover at a high level how the auditory clock setting program works. This application was designed to allow users to set the day and time using single keystrokes, and receive output with audio feedback and queues. This application was written using Python 3, and uses the readchar and Pyaudio packages. The sound files the application uses were recorded and edited with Audacity.

File Structure

This application accesses the sound files within the directory *program_wav_files*, and its sub-folders, and the sound files within these sub-folders must not be moved. Therefore, it is important that this folder remains in the same directory as the source files. The application uses 2 source files total: sound.py and set_clock.py, both of which are covered below.

sound.py

This source file is what handles all of the sound files within the application, and was provided by Anthony Hornof. There are 3 things in this file for the programmer to understand: the class *Play*, the method *cleanup*, and the method *combine_wav_files*.

The Play class allows the programmer to play single audio files. To do this, the programmer just needs to instantiate the Play class (no need to assign it to anything). The programmer passes in the name of the file to play, and the file is played in the constructor. The sound file keeps track of the last file played (which is None if no previous file was played), which is an audio stream; therefore the programmer must invoke *sound.cleanup()* at the end of the program to close the last stream. Example of proper use shown below.

```
import sound

if __name__ == "__main__":

	sound.Play("test_file.wav")

	# Do other stuff here ...

	sound.cleanup()
```

When the application needs to combine and play multiple audio source files together, it uses the combine_wav_files method. This method takes 2 or more arguments; the first argument is the name of the audio file that will be created, and the second and following arguments are the audio files that are to be combined together. The files are all combined and saved into one bigger audio file with the name the programmer gives it in the first parameter. Example of proper usage is shown below.

```
import sound

if __name__ == "__main__":
```

```
	wav_file = "temp.wav"
	sound.combine_wav_file(wav_file, "one.wav", "two.wav", "three.wav")
	sound.Play(wav_file)
	sound.cleanup()
```

set_clock.py

The set_clock.py application is split into 4 large functions, and is composed of several smaller functions. These four functions are: *create_sound_filenames*, *verify_sound_filenames*, *create_menu_globals*, and *run_menu*. These 4 functions are invoked inside another function called *main*, which is invoked at the very bottom of the source file.

The method *create_sound_filenames* is first invoked in main. This method declares several global variables the application uses, including several audio source files to access. The application also declares a global variable for a temporary audio file name to use for the *combine_wav_files* method in sound.py. Next, *verify_sound_names* simply calls sound.Play() on all of the global audio files to ensure that they work. The introductory audio file is played last since the previous files canceled each other out. Then, create_menu_globals creates additional global variables for the menu system. This include characters for possible menu options, and a minimal string message to display on the command prompt at startup.

After these three methods are ran, *run_menu* is invoked which is an endless loop that gets the next keystroke using *readchar.readchar()*. The program then evaluates the character received and reacts appropriately, either scrolls through a variable and announces the selection, moves to another state, provide a help message, or quits the program. These actions that the program takes are divided into sub-functions that are invoked in each keystroke case. The pseudocode for this looping menu is shown below.

```
while True:
	next = readchar.readchar()  # Obtain next keystroke
	if (next == FORWARD):
		if (mode == 0):
			# Increment the weekday
		elif (mode == 1):
			# Increment the hour
		elif (mode == 2):
			# Increment the minute
		elif (mode == 3):
			# Switch AM to PM, or vice versa
		elif (mode == 4):
```

```
                # Report the time
        if (next == BACKWARD):
                if (mode == 0):
                        # Decrement the weekday
                elif (mode == 1):
                        # Decrement the hour
                elif (mode == 2):
                        # Decrement the minute
                elif (mode == 3):
                        # Switch AM to PM, or vice versa
                elif (mode == 4):
                        # Report the time
        if (next == SELECT):
                # Move to the next state, announce new state
        if (next == HELP):
                # Play the help message
        if (next == QUIT):
                # Terminate the application
```