# OledScreen.h

Text, letter

Description automatically generated

Includes Wire.h, a default Arduino library for using devices on the SDA and SCL. This is to ensure that the Oled screen has a fixed refresh rate. **SCL** is the clock line, **giving the refresh rate**. **SDA** is the data line, so the **data that is fed to the screen**, in this SCL refresh rate.

Adafruit\_GFX is used for sending bitmaps (image) to the screen

Adafruit\_SSD1306 is the driver to talk to the screen

ScreenBitmaps.h are the images that generate the animation. Each image is represented in a byte array. So ScreenBitmaps.h stores these arrays.

We define the screen width and height to tell Adafruit\_GFX how much space the image should fill up.

Some Oled screens have a reset pin, but we don’t need it, so we set it to -1 (we are forced to set this to something to initialize the Oled screen)

SCREEN\_ADDRESS is the hardware id for the i2c bus (**SDA+SDL = i2c bus**). You can add up to 127 devices to this bus, and the hardware id is used to know which data is for which device. The id 0x3C is the one corresponding to the type of screen we have.



This line initializes the Adafruit\_SSD1306 class with the properties SCREEN\_WIDTH, SCREEN\_HEIGHT, &Wire, OLED\_RESET, and assigns it to the useOLED variable. Then, we are able to use the useOLED instance later on in our code.

The following function is always run when turning on the Arduino:

Graphical user interface, text, application

Description automatically generated

Begin the useOLED instance (Adafruit begins to try to connect to the screen). If this fails, then an error message is printed in the Serial Monitor (under tools menu of Arduino IDE), and the function stops (because of the return).

If it is true, it’s because it was able to find the screen, so it proceeds running the rest of the function.

useOled.clearDisplay() removes everything from the screen

useOLED.drawBitmap() has the following parameters:

x, y, bitmap array, width, height, a color map array (or 1 for black and white)

useOLED.display() draws the bitmap on the screen

Text

Description automatically generated

activeBmp is the current bitmap (image) being shown for the talking animation. We have 3 images that together form the animation. If activeBmp = 0, that means that it is showing the first image, activeBmp = 1 means it is showing the second image and so on.

moveMouth is a function that is run in a loop as long as the mp3 player is playing an mp3 file.

When we are at the last bitmap (when activeBmp==2), then we set the image back to 0. It returns (so that the current cycle doesn’t reach activeBmp++ immediately). However, the loop keeps on going.

activeBmp++ means that after every run, activeBmp = activeBmp + 1

Graphical user interface, text, application

Description automatically generated

Timer<>::Task is a type (like void, or Boolean for example) defined in the Timer library. We are stating that the talkInterval variable is of the type Timer<>::Task.

When the function startTalking() is triggered the global variable isTalking will be set to true which helps tell the other parts of the code whether the robot is talking or not.

talkInterval variable holds the timer task. It holds the information/action of triggering the moveMouth function every 100ms.

Graphical user interface, text, application

Description automatically generated

stopTalking is a function we trigger when we want the robot to stop talking. If the talkInterval holds a value, then it means that the timer is running.

So in this function, if the timer is running (if talkInterval holds something), then we cancel the timer by passing the talkInterval variable to timer.cancel()

We set isTalking to false, and set the activeBmp to 0 (to the first image) so that next time the robot starts talking, it starts showing the animation from the start.

The last three lines simply set the image to a static mouth (as we’ve done before in the oledScreenSetup() function).

Text

Description automatically generated with medium confidence

updateOledScreen is a function that is nonstop triggered on the main program loop (which you can see in the Arduino\_MEGA.ino file).

Every time this function is triggered, it checks if the mp3 player is currently playing an mp3 file using the isPlayingMP3 function (which was defined in the MP3Player.h file).

It also checks whether we are currently talking or not using the global variable isTalking.

If the robot is NOT playing an mp3 file and isTalking (is moving his mouth) then stopTalking() function is triggered to stop the mouth from moving

If the robot IS playing an mp3 file but the variable isTalking is false (it is not moving his mouth) then the function startTalking() is triggered.

# ScreenBitmaps.h

Table

Description automatically generated

This file holds the bitmap images defined in a bitmap array.  
It holds both the static mouth image and the images used during the talking animation.

The bitmap images where converted into a bitmap array using the following tutorial:  
<https://miliohm.com/how-to-draw-or-print-bitmap-to-oled-display-arduino/>

We define it as a “static const unsigned char PROGMEM”, this essentially means that these variables don’t change (static const), can hold any type of character (unsigned char) and is loaded on the arduino harddrive instead of directly in the RAM memory to safe some space (PROGMEM).

# MP3Player.h

Text

Description automatically generated

The DFRRobotDFPlayerMini is a library that allows us to communicate with the MP3 player using the RX/TX communication (these are communication pins)

We create an instance of the DFRobotDFPlayerMini class and name it useMP3Player.

We define the variable MP3\_PLAYING\_PIN which represents pin number 52 in the Arduino board.

SoftwareSerial is a library that allows serial communication on non-serial pins.  
On the Arduino we are using actual serial pins, however the DFRobotDFPlayerMini package enforces us to use SoftwareSerial. So we simply pass the actual serial pins to Software Serial library so that our serial pins can used with DFRRobotDFPlayerMini. Tdlr: we are enforced to use this in order to use the DFRRobotDFPlayerMini library.

Text

Description automatically generated

We define the function isPlayingMP3 which returns a Boolean. We are using digitalRead() which allows us to read the high or low voltage information from the pin number 52 from the Arduino board. This reading is either 1 or 0. If the reading of pin 52 is 0, then the function will return True, because it is playing an MP3 file. In other words, when pin 52 is 0, then it is playing sounds, when the pin 52 is 1, then it is playing nothing.

Graphical user interface, text

Description automatically generated with medium confidence

This is a function used in the timer to play the mp3 files with a pre-determined delay. It can take in multiple parameters (void\* means it can be anything). In this case we are inputting an int variable called trackToPlay (a number between 1 and 14) and plays the track using useMP3Player.play() function.

The playTrackWithDelay function returns false to make sure that the timer doesn’t continue running.

These are the audio that can be played:

Text

Description automatically generated

Text, application

Description automatically generated

This is a function we can use to play an mp3 file. It expects the track to play and a timeDelay for establishing at what point to play it.

-----------------------

We have some code holding some logic to bypass a weird bug, when there are more then 13 files, track 8 is read as track 1. However, everything passed track 13 (14 and up) is read normally. Everything below 8 is read as above 8. E.g. passing track 1 will play track 9.

We first create a new variable trackToPlay which holds the track that should be played. If the track number is less than 13 (which is where the problem is at), we create another variable (startPlayingAt) which defines at what track the MP3 reads as track 1 (track 8).

If the current track is less then this variable (startPlayingAt) we simply add the “(13 – startsPlayingAt) + 1” to the track (returning current track + 6).

This sounds quite confusing, so lets give an example:

If we pass 7, the MP3 will play track 13.

So by following this calculation we will do:  
7 + (13 – 8) + 1 = 13

If the current track is more then startPlayingAt (8) we do track – startPlayingAt(8) + 1.

Another example:  
If we pass 11, the MP3 will play track 4.

So we do the following calculation:

11 – 8 + 1 = 4

-----------------------

If there is a time delay (timeDelay>0), we use the timer to run a function after this delay has passed. In the timer function we will pass the delay, the function to trigger and any parameters we want to pass to this function. See the previous section where the “playTrackWithDelay” function is explained.

After all this is done, we pass the trackToPlay to the “useMP3Player.play()” function to play the MP3 file.

Graphical user interface, text, application

Description automatically generated

This initializes the MP3 module.

We first tell the Arduino that MP3\_PLAYING\_PIN is an INPUT pin we want to read a value from (See “isPlayingMP3()” function above).

After that we tell SoftwareSerial (MP3PlayerSerial.begin()) to start on BAUDRATE 9600, this is the communication port that the software will use to communicate with the MP3.

Now we can start the useMP3Player instance (useMP3Player.begin()) by passing the SoftwareSerial instance. We also pass “true” as a second parameter, this makes it so that the MP3 player waits till the last command is fully read until the next one can be send.

This prevents misreadings when multiple commands are send to the MP3 player in a row.

(For example, play track 1 and then directly play track2, or pause and after that directly play again).