```
/*
 * Call for swallows and swifts
 * Code by Fabrizio Calderan, 02/17/2020
 * This call is set to work from mid-February to late March,
 * at sunrise and sunset. Sunrise times and sunset times (less
 * 2 hours) have been hardcoded in the functions below. They
 * are referred to the coordinates (45°N, 12°E).
 *
 *
 * Required components:
 *
 * - 1x ESP8266/12E
 * - 1x RTC Module DS3231
 * - 1x DFPlayer Mini
 * - 1x 3W/4Ω Speaker
 * - 1x On-Off switch
 * - 1x 10KΩ Trimmer
 * - 1x 120x80x50 IP56 enclosure box for outdoor use.
 */

#include "SoftwareSerial.h"
#include "DFRobotDFPlayerMini.h"
#include <Wire.h>
#include <RtcDS3231.h>

#define DELAY_ONPLAY 5000
#define DELAY_ONPAUSE 50000
#define countof(a) (sizeof(a) / sizeof(a[0]))

// RTC Library (I2C)
RtcDS3231<TwoWire> rtcObject(Wire);

// MP3 Serial communication
SoftwareSerial mySoftwareSerial(14, 12);
DFRobotDFPlayerMini myDFPlayer;

// Variables
int volume, y;
bool IDLE = true;

RtcDateTime currentTime, compileTime, stopTime;
RtcDateTime tsBegin, tsEnd;

// Playback state get information through the BUSY pin
bool playbackState = digitalRead(13);


void setup() {

    mySoftwareSerial.begin(9600);
    // Start serial connection
    Serial.begin(115200);
    // Start I2C
    rtcObject.Begin();

    /* Date and time first configuration
     *
     * Define a date and time object that accepts the format
     * yyyy, m, d, H, M, S. Then configure the RTC with the
     * object defined.
     */
```

```cpp
    stopTime    = RtcDateTime(1978, 4, 8, 10, 40, 0);
    compileTime = RtcDateTime(__DATE__, __TIME__);
    currentTime = rtcObject.GetDateTime();

    if (compileTime > currentTime) {
        Serial.print("Adjusting RTC datetime from ");
        Serial.print(printDateTime(currentTime));
        Serial.print(" to ");
        Serial.println(printDateTime(compileTime));
        rtcObject.SetDateTime(compileTime);
    }

    // Get the current Year
    y = currentTime.Year();

    Serial.println("Init MP3 module");
    // Use softwareSerial to begin communication with the MP3 module
    if (!myDFPlayer.begin(mySoftwareSerial, false)) {
      while(true) {
          delay(0); // ESP8266 watchdog needs this
      }
    }

    /*
     * Set the volume
     */
    setVolume();

    /*
     * Playback Test
     */
    myDFPlayer.play(1);
    delay(8000);
    myDFPlayer.stop();

    Serial.println("-------- /setup --------");
}


void loop() {

    setVolume();
    playbackState = digitalRead(13);

    // Get the time from the RTC
    currentTime = rtcObject.GetDateTime();

    Serial.print("Current datetime: ");
    Serial.println(printDateTime(currentTime));

    if (eventTS()) {

        IDLE = false;

        Serial.print("Playing until ");
        Serial.println(printDateTime(stopTime));

        /*
         *  LOW  = The MP3 module is busy
         *  HIGH = The MP3 module is available
```

```cpp
        */

        if (playbackState == HIGH) {
            myDFPlayer.stop();
            myDFPlayer.play(1);
        }
        delay(DELAY_ONPLAY);

    }
    else {

        if (IDLE == false) {
            myDFPlayer.pause();
            myDFPlayer.stop();
            Serial.println("End of playback");
        }

        IDLE = true;
        delay(DELAY_ONPAUSE);
    }

    Serial.println("-------- /loop --------");

}


bool eventTS() {

    if (currentTime < stopTime) {
        return true;
    }

    return (sunriseTS() || sunsetTS());
}


void setVolume() {

    // Get the value of potentiometer in the range of [3..30]
    volume = map(analogRead(A0), 0, 1023, 2, 30);
    volume = constrain(volume, 3, 30);

    // Set the player volume
    myDFPlayer.volume(volume);
    Serial.print("Volume level: ");
    Serial.println(String(volume));
}


bool _TS(int mth_, int day_, int h_, int m_) {

    tsBegin = RtcDateTime(y, mth_, day_, h_, m_, 0);
    tsEnd   = RtcDateTime(y, mth_, day_, h_ + 2, m_, 0);

    if (currentTime >= tsBegin && currentTime <= tsEnd) {
        stopTime = tsEnd;
        return true;
    }
    else {
```

```cpp
        return false;
    }
}


bool sunriseTS() {

    /* Sunrise datetimes from February 20 to March 27 */
    return (_TS(2, 20, 7, 8)  ||
        _TS(2, 21, 7, 6)  ||
        _TS(2, 22, 7, 4)  ||
        _TS(2, 23, 7, 3)  ||
        _TS(2, 24, 6, 1)  ||
        _TS(2, 25, 6, 59) ||
        _TS(2, 26, 6, 57) ||
        _TS(2, 27, 6, 56) ||
        _TS(2, 28, 6, 54) ||
        _TS(2, 29, 6, 52) ||
        _TS(3, 1, 6, 49)  ||
        _TS(3, 2, 6, 47)  ||
        _TS(3, 3, 6, 45)  ||
        _TS(3, 4, 6, 43)  ||
        _TS(3, 5, 6, 41)  ||
        _TS(3, 6, 6, 40)  ||
        _TS(3, 7, 6, 38)  ||
        _TS(3, 8, 6, 36)  ||
        _TS(3, 9, 6, 34)  ||
        _TS(3, 10, 6, 32) ||
        _TS(3, 11, 6, 30) ||
        _TS(3, 12, 6, 28) ||
        _TS(3, 13, 6, 27) ||
        _TS(3, 14, 6, 25) ||
        _TS(3, 15, 6, 23) ||
        _TS(3, 16, 6, 21) ||
        _TS(3, 17, 6, 19) ||
        _TS(3, 18, 6, 17) ||
        _TS(3, 19, 6, 15) ||
        _TS(3, 20, 6, 13) ||
        _TS(3, 21, 6, 11) ||
        _TS(3, 22, 6, 9)  ||
        _TS(3, 23, 6, 7)  ||
        _TS(3, 24, 6, 6)  ||
        _TS(3, 25, 6, 4)  ||
        _TS(3, 26, 6, 2)  ||
        _TS(3, 27, 6, 1));
}


bool sunsetTS() {

    /* Sunset datetimes from February 20 to March 27 (minus 2 hours) */
    return (_TS(2, 20, 15, 41) ||
        _TS(2, 21, 15, 42) ||
        _TS(2, 22, 15, 44) ||
        _TS(2, 23, 15, 45) ||
        _TS(2, 24, 15, 47) ||
        _TS(2, 25, 15, 48) ||
        _TS(2, 26, 15, 49) ||
        _TS(2, 27, 15, 51) ||
```

```
        _TS(2, 28, 15, 52) ||
        _TS(2, 29, 15, 54) ||
        _TS(3, 1, 15, 56)  ||
        _TS(3, 2, 15, 58)  ||
        _TS(3, 3, 15, 59)  ||
        _TS(3, 4, 16, 1)   ||
        _TS(3, 5, 16, 2)   ||
        _TS(3, 6, 16, 3)   ||
        _TS(3, 7, 16, 5)   ||
        _TS(3, 8, 16, 6)   ||
        _TS(3, 9, 16, 8)   ||
        _TS(3, 10, 16, 9)  ||
        _TS(3, 11, 16, 10) ||
        _TS(3, 12, 16, 12) ||
        _TS(3, 13, 16, 13) ||
        _TS(3, 14, 16, 14) ||
        _TS(3, 15, 16, 16) ||
        _TS(3, 16, 16, 17) ||
        _TS(3, 17, 16, 18) ||
        _TS(3, 18, 16, 20) ||
        _TS(3, 19, 16, 21) ||
        _TS(3, 20, 16, 22) ||
        _TS(3, 21, 16, 24) ||
        _TS(3, 22, 16, 25) ||
        _TS(3, 23, 16, 26) ||
        _TS(3, 24, 16, 28) ||
        _TS(3, 25, 16, 29) ||
        _TS(3, 26, 16, 30) ||
        _TS(3, 27, 16, 31));
}



String printDateTime(const RtcDateTime& dt) {
    char datestring[20];

    snprintf_P(datestring,countof(datestring),
            PSTR("%02u/%02u/%04u %02u:%02u:%02u"),
            dt.Month(), dt.Day(), dt.Year(),
            dt.Hour(), dt.Minute(), dt.Second());

    return datestring;
}
```