# Appendix A

**Arduino Part (Coding Summary)**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <Adafruit\_Fingerprint.h>

SoftwareSerial mySerial(2, 3);

Adafruit\_Fingerprint finger = Adafruit\_Fingerprint(&mySerial);

uint8\_t id;

void setup()

{

Serial.begin(9600);

while (!Serial);

delay(100);

// set the data rate for the sensor serial port

finger.begin(57600);

if (finger.verifyPassword())

{

Serial.println("Found fingerprint sensor!");

}

else

{

Serial.println("Did not find fingerprint sensor :(");

while (1) { delay(1); }

}

Serial.println(F("Reading sensor parameters"));

finger.getParameters();

finger.getTemplateCount();

Serial.print(F("Status: 0x")); Serial.println(finger.status\_reg, HEX);

Serial.print(F("Sys ID: 0x")); Serial.println(finger.system\_id, HEX);

Serial.print(F("Capacity: ")); Serial.println(finger.capacity);

Serial.print(F("Security level: ")); Serial.println(finger.security\_level);

Serial.print(F("Device address: ")); Serial.println(finger.device\_addr, HEX);

Serial.print(F("Packet len: ")); Serial.println(finger.packet\_len);

Serial.print(F("Baud rate: ")); Serial.println(finger.baud\_rate);

Serial.print(F("Total finger prints stored: ")); Serial.println(finger.templateCount);

}

uint8\_t readnumber(void)

{

uint8\_t num = 0;

while (num == 0)

{

while (! Serial.available());

num = Serial.parseInt();

}

return num;

}

void loop()

{

Serial.println("\n\nReady to enroll a fingerprint!");

Serial.println("Please type in -1 to delete all the stored finger prints...");

Serial.println("Please type in the ID # (from 1 to 127) you want to save the finger as...");

id = readnumber();

if (id == 0) // ID #0 not allowed, try again!

{

return;

}

if (id == uint8\_t(-1))

{

Serial.print("Cleared all finger prints from database");

finger.emptyDatabase();

return;

}

Serial.print("Enrolling ID #");

Serial.println(id);

while (! getFingerprintEnroll() );

}

uint8\_t getFingerprintEnroll()

{

int p = -1;

Serial.print("Waiting for valid finger to enroll as #"); Serial.println(id);

while (p != FINGERPRINT\_OK)

{

p = finger.getImage();

switch (p)

{

case FINGERPRINT\_OK:

Serial.println("Image taken");

break;

case FINGERPRINT\_NOFINGER:

break;

case FINGERPRINT\_PACKETRECIEVEERR:

Serial.println("Communication error");

break;

case FINGERPRINT\_IMAGEFAIL:

Serial.println("Imaging error");

break;

default:

Serial.println("Unknown error");

break;

}

}

// OK success!

p = finger.image2Tz(1);

switch (p)

{

case FINGERPRINT\_OK:

Serial.println("Image converted");

break;

case FINGERPRINT\_IMAGEMESS:

Serial.println("Image too messy");

return p;

case FINGERPRINT\_PACKETRECIEVEERR:

Serial.println("Communication error");

return p;

case FINGERPRINT\_FEATUREFAIL:

Serial.println("Could not find fingerprint features");

return p;

case FINGERPRINT\_INVALIDIMAGE:

Serial.println("Could not find fingerprint features");

return p;

default:

Serial.println("Unknown error");

return p;

}

Serial.println("Remove finger");

delay(2000);

p = 0;

while (p != FINGERPRINT\_NOFINGER)

{

p = finger.getImage();

}

Serial.print("ID "); Serial.println(id);

p = -1;

Serial.println("Place same finger again");

while (p != FINGERPRINT\_OK)

{

p = finger.getImage();

switch (p)

{

case FINGERPRINT\_OK:

Serial.println("Image taken");

break;

case FINGERPRINT\_NOFINGER:

break;

case FINGERPRINT\_PACKETRECIEVEERR:

Serial.println("Communication error");

break;

case FINGERPRINT\_IMAGEFAIL:

Serial.println("Imaging error");

break;

default:

Serial.println("Unknown error");

break;

}

}

// OK success!

p = finger.image2Tz(2);

switch (p)

{

case FINGERPRINT\_OK:

Serial.println("Image converted");

break;

case FINGERPRINT\_IMAGEMESS:

Serial.println("Image too messy");

return p;

case FINGERPRINT\_PACKETRECIEVEERR:

Serial.println("Communication error");

return p;

case FINGERPRINT\_FEATUREFAIL:

Serial.println("Could not find fingerprint features");

return p;

case FINGERPRINT\_INVALIDIMAGE:

Serial.println("Could not find fingerprint features");

return p;

default:

Serial.println("Unknown error");

return p;

}

// OK converted!

Serial.print("Creating model for #"); Serial.println(id);

p = finger.createModel();

if (p == FINGERPRINT\_OK) {

Serial.println("Prints matched!");

} else if (p == FINGERPRINT\_PACKETRECIEVEERR) {

Serial.println("Communication error");

return p;

} else if (p == FINGERPRINT\_ENROLLMISMATCH) {

Serial.println("Fingerprints did not match");

return p;

} else {

Serial.println("Unknown error");

return p;

}

Serial.print("ID "); Serial.println(id);

p = finger.storeModel(id);

if (p == FINGERPRINT\_OK) {

Serial.println("Stored!");

} else if (p == FINGERPRINT\_PACKETRECIEVEERR) {

Serial.println("Communication error");

return p;

} else if (p == FINGERPRINT\_BADLOCATION) {

Serial.println("Could not store in that location");

return p;

} else if (p == FINGERPRINT\_FLASHERR) {

Serial.println("Error writing to flash");

return p;

} else {

Serial.println("Unknown error");

return p;

}

  return true;

}

TO ENROLL FINGERPRINT

#include <Adafruit\_Fingerprint.h>

#include <SoftwareSerial.h>

SoftwareSerial mySerial(2, 3);

Adafruit\_Fingerprint finger = Adafruit\_Fingerprint(&mySerial);

void setup() {

Serial.begin(9600);

while (!Serial); // For Yun/Leo/Micro/Zero/...

delay(100);

Serial.println("fingertest");

pinMode(12, OUTPUT);

pinMode(11, OUTPUT);

// set the data rate for the sensor serial port

finger.begin(57600);

if (finger.verifyPassword()) {

Serial.println("Found fingerprint sensor!");

} else {

Serial.println("Did not find fingerprint sensor :(");

while (1) {

delay(1);

}

}

finger.getTemplateCount();

Serial.print("Sensor contains "); Serial.print(finger.templateCount); Serial.println(" templates");

Serial.println("Waiting for valid finger...");

}

void loop() {

getFingerprintIDez();

delay(50); // don't need to run this at full speed

digitalWrite(12, LOW);

digitalWrite(11, LOW);

}

uint8\_t getFingerprintID() {

uint8\_t p = finger.getImage();

switch (p) {

case FINGERPRINT\_OK:

Serial.println("Image taken");

break;

case FINGERPRINT\_NOFINGER:

Serial.println("No finger detected");

return p;

case FINGERPRINT\_PACKETRECIEVEERR:

Serial.println("Communication error");

return p;

case FINGERPRINT\_IMAGEFAIL:

Serial.println("Imaging error");

return p;

default:

Serial.println("Unknown error");

return p;

}

// OK success!

p = finger.image2Tz();

switch (p) {

case FINGERPRINT\_OK:

Serial.println("Image converted");

break;

case FINGERPRINT\_IMAGEMESS:

Serial.println("Image too messy");

return p;

case FINGERPRINT\_PACKETRECIEVEERR:

Serial.println("Communication error");

return p;

case FINGERPRINT\_FEATUREFAIL:

Serial.println("Could not find fingerprint features");

return p;

case FINGERPRINT\_INVALIDIMAGE:

Serial.println("Could not find fingerprint features");

return p;

default:

Serial.println("Unknown error");

return p;

}

// OK converted!

p = finger.fingerFastSearch();

if (p == FINGERPRINT\_OK) {

Serial.println("Found a print match!");

Serial.print("Found ID #"); Serial.print(finger.fingerID);

Serial.print(" with confidence of "); Serial.println(finger.confidence);

digitalWrite(11, HIGH);

delay(3000);

digitalWrite(11, LOW);

} else if (p == FINGERPRINT\_PACKETRECIEVEERR) {

Serial.println("Communication error");

} else if (p == FINGERPRINT\_NOTFOUND) {

Serial.println("Did not find a match");

digitalWrite(11, HIGH);

delay(3000);

digitalWrite(11, LOW);

} else {

Serial.println("Unknown error");

}

return p;

}

// returns -1 if failed, otherwise returns ID #

int getFingerprintIDez() {

uint8\_t p = finger.getImage();

if (p != FINGERPRINT\_OK) return -1;

p = finger.image2Tz();

if (p != FINGERPRINT\_OK) return -1;

p = finger.fingerFastSearch();

if (p != FINGERPRINT\_OK) return -1;

// found a match!

Serial.print("Found ID #"); Serial.print(finger.fingerID);

Serial.print(" with confidence of "); Serial.println(finger.confidence);

digitalWrite(12, HIGH);

delay(3000);

digitalWrite(12, LOW);

return finger.fingerID;

}