Main Functions

```
#include <NewSoftSerial.h>
#include <EXROM.h>

NewSoftSerial rfid(2,3); //RX 2, TX 3 (from Arduino Perspective)

int redLED = 4;
int yellowLED = 5;
int greenLED = 6;

int lock = 13;

int key1 = 7;
int key2 = 8;
int key3 = 9;
int key4 = 10;
int key5 = 11;
int key6 = 12;
int key7 = 14;
int key8 = 15;
int key9 = 16;
int key0 = 17;
int keya = 18;
int keyb = 19;


void setup()
{
  Serial.begin(9600);
  rfid.begin(9600);
  initKeypad();
  Serial.println("RFID Reader initialising");
  createHead();
  Serial.println("Please type 'menu' to enter the menu");
  pinMode(lock, OUTPUT);
  pinMode(redLED, OUTPUT);
  pinMode(greenLED, OUTPUT);
  pinMode(yellowLED, OUTPUT);
  digitalWrite(yellowLED, HIGH);
}

void loop()
{
  checkRFID();
  enterMenu();
}

void openDoor()
{
```

```
  digitalWrite(lock, HIGH);
  digitalWrite(greenLED, HIGH);
  Serial.println("Access Granted");
  delay(3000);
  digitalWrite(lock, LOW);
  digitalWrite(greenLED, LOW);
  rfid.flush();
}

void closeDoor()
{
  digitalWrite(redLED, HIGH);
  Serial.println("Access Denied");
  delay(1000);
  digitalWrite(redLED, LOW);
  rfid.flush();
}
```

Database Functions

```
/*
  EXDB - Database functions based on the EXROM library

 Copyright(c) 2010 Tom "TBSliver" Bloor. All rights reserved.

 This library is free software; you can redistribute it and/or
 modify it under the terms of the GNU Lesser General Public
 License as published by the Free Software Foundation; either
 version 2.1 of the License, or (at your option) any later version.

 This library is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 Lesser General Public License for more details.

 You should have received a copy of the GNU Lesser General Public
 License along with this library; if not, write to the Free Software
 Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
 */


#include <EXROM.h>
#define HEADER_P 0
#define HBYTE 250
#define EBYTE 251
#define MAXMEM 512 // Max memory allocated to the database
```

```
//DO NOT TOUCH
union Header
{
  byte raw[3];
  struct Data
  {
    byte hByte;
    int noTag;
  }
  data;
}
header;
//END OF DO NOT TOUCH AREA

union Entry
{
  byte raw[19]; // must be the same size as the Data struct.
  //remember that ints are 2 bytes, floats, longs etc. are 4.
  struct Data
  {
    //Needed for functionality
    byte eByte;
    union EData
    {
      byte raw[18]; //must be same size as the PData Struct.
      struct PData
      {
        //Customize this bit!
        byte rfidTag[10];
        int pinNo[4];
        //End of custom area
      }
      pData;
    }
    eData;
  }
  data;
}
entry;

union PData
{
  byte raw[18]; //must be same size as the PData Struct.
  struct Data
  {
    //Customize this bit!
    byte rfidTag[10];
    int pinNo[4];
    //End of custom area
  } data;
}
```

```
pData;

//Customize to print the data inside the PData struct.
void printEData()
{
  Serial.print(" Entry Data: ");
  Serial.print(" RFID No: ");
  for(int i=0;i<10;i++)
    Serial.print(entry.data.eData.pData.rfidTag[i]);
  Serial.print(" Pin No: ");
  for(int i=0;i<4;i++)
    Serial.print(entry.data.eData.pData.pinNo[i]);
}



/*-----------------------------------------------------
 DB
 -----------------------------------------------------*/

void printDB()
{
  Serial.println("Printing DB");
  readHead();
  Serial.println(header.data.noTag);
  for(int i=0;i<header.data.noTag;i++)
    printEntry(i);
  if(header.data.noTag == 0)
    Serial.println("No Tags in Database");
}

/*-----------------------------------------------------
 Entry
 -----------------------------------------------------*/

int entryPointer(int entryID) //returns pointer for specified entryID
{
  return HEADER_P + sizeof(header.raw) + (sizeof(entry.raw)*entryID);
}

void writeEntry(byte entryW[sizeof(entry.data.eData.raw)], int entryID)
{
  entry.data.eByte = EBYTE;
  for(int i=0;i<sizeof(entry.data.eData.raw);i++)
  {
    entry.data.eData.raw[i]=entryW[i];
  }
  EXROM.write(entryPointer(entryID), entry.raw, sizeof(entry.raw));
  readHead();
  header.data.noTag++;
```

```
  updateHead();
}

boolean readEntry(int entryID)
{
  EXROM.read(entryPointer(entryID), &entry.data.eByte);
  if(entry.data.eByte==EBYTE)
  {
    EXROM.read(entryPointer(entryID), entry.raw, sizeof(entry.raw));
    return true;
  }
  else
    return false;
}

void printEntry(int entryID)
{
  if(readEntry(entryID))
  {
    Serial.print("EntryID: ");
    Serial.print(entryID);
    printEData();
    Serial.println();
  }
  else
  {
    Serial.print("No entry at ID ");
    Serial.print(entryID);
    Serial.println();
  }
}

boolean compEntry(byte entryA[sizeof(entry.data.eData.raw)], byte
entryB[sizeof(entry.data.eData.raw)])
{
  boolean flag = false;
  for(int i=0;i<sizeof(entry.data.eData.raw);i++)
  {
    if (entryA[i]==entryB[i])
      flag = true;
    else
    {
      flag = false;
      break;
    }
  }
  return flag;
}

int findEntry(byte entryF[sizeof(entry.data.eData.raw)])
{
```

```
  int flag = -1;
  readHead();
  for(int i=0;i<header.data.noTag;i++)
  {
    readEntry(i);
    if (compEntry(entryF, entry.data.eData.raw))
    {
      flag = i;
      break;
    }
  }
  return flag;
}

boolean addEntry(byte entryA[sizeof(entry.data.eData.raw)])
{
  if(findEntry(entryA)==-1 && (entryPointer(header.data.noTag) +
sizeof(entry.raw)) <= MAXMEM)
  {
    readHead();
    for(int i=0; i <= header.data.noTag; i++)
    {
      if(!readEntry(i))
      {
        writeEntry(entryA, i);
        Serial.println("Adding Entry to DB");
        return true;
      }
      else
      {
        Serial.print("Entry already present at ID");
        Serial.println(i);
      }
    }

  }
  else if((entryPointer(header.data.noTag) + sizeof(entry.raw)) > MAXMEM)
  {
    Serial.println("Database Full");
    return false;
  }
  else
  {
    Serial.println("Entry already in Database");
    return false;
  }
}

void delEntry(int entryID)
```

```
{
  readHead();
  byte eEntry[sizeof(entry.raw)];
  for(int i=0;i<sizeof(eEntry);i++)
  {
    eEntry[i] = 255;
  }
  if(entryID==header.data.noTag)
  {
    EXROM.write(entryPointer(entryID), eEntry, sizeof(eEntry));
  }
  else
  {
    readEntry(header.data.noTag-1);
    EXROM.write(entryPointer(entryID), entry.raw, sizeof(entry.raw));
    EXROM.write(entryPointer(header.data.noTag-1), eEntry, sizeof(eEntry));
  }
  header.data.noTag--;
  updateHead();
}

/*-----------------------------------------------------
 Header
 -----------------------------------------------------*/

//Writes a new header
void writeHead()
{
  header.data.hByte=HBYTE;
  header.data.noTag=0;
  EXROM.write(HEADER_P, header.raw, sizeof(header.raw));
}

//Reads a header if there is one present at HEADER_P
boolean readHead()
{
  EXROM.read(HEADER_P, &header.data.hByte);
  if(header.data.hByte==HBYTE)
  {
    EXROM.read(HEADER_P, header.raw, sizeof(header.raw));
    return true;
  }
  else
    return false;
}

//Updates the header if there is one present, with whatever noTag is set.
//use readHead() first to get an accurate update, for example:

//readHead();
//header.data.noTag++;
```

```
//updateHead();

//This will update the header with an increment of one in noTag.
boolean updateHead()
{
  byte hCheck;
  EXROM.read(HEADER_P, &hCheck);
  if(hCheck==HBYTE)
  {
    EXROM.write(HEADER_P, header.raw, sizeof(header.raw));
    return true;
  }
  else
    return false;
}

//Creates a header if there isnt one present.
void createHead()
{
  Serial.println("Checking for Database");
  if(readHead())
  {
    Serial.println();
    Serial.println("Database present");
  }
  else
  {
    Serial.println();
    Serial.println("Database not present: Creating Database");
    writeHead();
  }
}

//Does what it says on the tin.
void printHead()
{
  Serial.print("HeadID: ");
  Serial.println(header.data.hByte, DEC);
  Serial.print("HeadData: ");
  Serial.print(header.data.noTag);
  Serial.println("\r\n");
}
```

Keypad Functions

```
void initKeypad()
{
  pinMode(key1, INPUT);
```

```
  digitalWrite(key1, HIGH);
  pinMode(key2, INPUT);
  digitalWrite(key2, HIGH);
  pinMode(key3, INPUT);
  digitalWrite(key3, HIGH);
  pinMode(key4, INPUT);
  digitalWrite(key4, HIGH);
  pinMode(key5, INPUT);
  digitalWrite(key5, HIGH);
  pinMode(key6, INPUT);
  digitalWrite(key6, HIGH);
  pinMode(key7, INPUT);
  digitalWrite(key7, HIGH);
  pinMode(key8, INPUT);
  digitalWrite(key8, HIGH);
  pinMode(key9, INPUT);
  digitalWrite(key9, HIGH);
  pinMode(key0, INPUT);
  digitalWrite(key0, HIGH);
  pinMode(keya, INPUT);
  digitalWrite(keya, HIGH);
  pinMode(keyb, INPUT);
  digitalWrite(keyb, HIGH);
}

int readKeypad()
{
  if(digitalRead(key1)==LOW)
  {
    delay(500);
    return 1;
  }
  else if(digitalRead(key2)==LOW)
  {
    delay(500);
    return 2;
  }
  else if(digitalRead(key3)==LOW)
  {
    delay(500);
    return 3;
  }
  else if(digitalRead(key4)==LOW)
  {
    delay(500);
    return 4;
  }
  else if(digitalRead(key5)==LOW)
  {
    delay(500);
    return 5;
```

```
  }
  else if(digitalRead(key6)==LOW)
  {
    delay(500);
    return 6;
  }
  else if(digitalRead(key7)==LOW)
  {
    delay(500);
    return 7;
  }
  else if(digitalRead(key8)==LOW)
  {
    delay(500);
    return 8;
  }
  else if(digitalRead(key9)==LOW)
  {
    delay(500);
    return 9;
  }
  else if(digitalRead(key0)==LOW)
  {
    delay(500);
    return 0;
  }
  else if(digitalRead(keya)==LOW)
  {
    delay(500);
    return -1;
  }
  else if(digitalRead(keyb)==LOW)
  {
    delay(500);
    return -2;
  }
  else return -3;
}

int count = 0;

boolean getPin(int cPin[4]) //compares cPin to the input pin
{
  int iPin[4];
  int count = 0;
  int in;
  Serial.println("Please input correct Pin");
  Serial.println("Press * to delete last number");
  while(count < 4)
```

```
  {
    in = readKeypad();
    if(in > -1)
    {
      iPin[count] = in;
      count++;
      for(int i=0;i<count;i++)
      {
        Serial.print("*");
      }
      Serial.println(count);

    }
    if(in == -1 && count > 0)
    {
      count--;
      for(int i=0;i<count;i++)
      {
        Serial.print("*");
      }
      Serial.println();
    }
  }
  boolean flag = false;
  for(int i=0;i<4;i++)
  {
    if(cPin[i]==iPin[i])
      flag = true;
    else
    {
      flag = false;
      break;
    }
  }
  return flag;
}

boolean addPin()
{
  int count = 0;
  int in;
  Serial.println("Please input new Pin");
  Serial.println("Press * to delete last number");
  while(count < 4)
  {
    in = readKeypad();
    if(in > -1)
    {
      pData.data.pinNo[count] = in;
      count++;
      for(int i=0;i<count;i++)
```

```
      {
        Serial.print("*");
      }
      Serial.println(count);

    }
    if(in == -1 && count > 0)
    {
      count--;
      for(int i=0;i<count;i++)
      {
        Serial.print("*");
      }
      Serial.println();
    }
  }
}
```

Menu Functions

```
char menuInput[20];

void enterMenu()
{
  readString();
  if(!strcmp(menuInput, "menu"))
  {
    digitalWrite(redLED, HIGH);
    digitalWrite(greenLED, HIGH);
    mainMenuOptions();
    mainMenu();
    menuInput[0] = '\0';
    return;
  }
}

void readString()
{
  if(Serial.available() > 0)
  {
    delay(100);
    int serialNo = Serial.available();
    for(int i=0; i < serialNo; i++)
    {
      menuInput[i] = Serial.read();
    }
    menuInput[serialNo] = '\0';
    Serial.flush();
```

```
  }
}

void mainMenuOptions()
{
  Serial.println("Welcome to the Menu");
  Serial.println("Please choose an option");
  Serial.println("-----------------------");
  Serial.println("1. Open Door");
  Serial.println("2. Print Tag Database");
  Serial.println("3. Add RFID Tag to Database");
  Serial.println("4. Remove RFID Tag from Database");
  Serial.println("5. Re-print this Menu");
  Serial.println("6. Exit Menu");
  Serial.println();
}


void mainMenu()
{
  while(1)
  {
    byte serialIn = 0;
    if(Serial.available())
    {
      serialIn = Serial.read();
      Serial.flush();
    }
    if(serialIn == 49) // 1
    {
      openDoor();
    }
    else if(serialIn == 50) // 2
    {
      printDB();
    }
    else if(serialIn == 51) // 3
    {
      addRFID();
    }
    else if(serialIn == 52) // 4
    {
      delRFID();
    }
    else if(serialIn == 53) // 5
    {
      mainMenuOptions();
    }
    else if(serialIn == 54) // 6
    {
      Serial.println("Exiting Menu...");
```

```
      digitalWrite(redLED, LOW);
      digitalWrite(greenLED, LOW);
      return;
    }
    else if(serialIn != 0)
    {
      Serial.println("That is not an option");
      Serial.flush();
    }
  }
}
```

Tag Functions

```
union tagdata
{
  byte tagin[16];
  struct tag
  {
    byte stx;
    byte data[10];
    byte cs[2];
    byte cr;
    byte lf;
    byte etx;
  }
  tag;
}
tagdata;

boolean readRFID()
{
  if(rfid.available())
  {
    delay(100);
    Serial.println("Reading RFID Tag");
    for(int i=0; i < sizeof(tagdata.tagin); i++)
    {
      tagdata.tagin[i] = rfid.read();
    }
    printRFID();
    rfid.flush();
    return 1;
  }
  else return 0;
}

void printRFID()
```

```
{
  for(int i=0; i < sizeof(tagdata.tag.data);i++)
  {
    Serial.print(tagdata.tag.data[i]);
  }
  Serial.println();
}

boolean checkRFID()
{
  if(readRFID())
  {
    int tagNo = findRFID(tagdata.tag.data);
    if(tagNo != -1)
    {
      Serial.println("Tag in Database");
      readEntry(tagNo);
      if(getPin(entry.data.eData.pData.pinNo))
      {
      openDoor();
      return true;
      }
      else
      {
        Serial.println("Incorrect Pin. please scan card and try again");
        closeDoor();
      }
    }
    else
    {
      Serial.println("Tag not in Database");
      closeDoor();
    }
  }
  return false;
}

int findRFID(byte fTag[10])
{
  int flag = -1;
  readHead();
  for(int i=0;i<header.data.noTag;i++)
  {
    readEntry(i);
    if (compRFID(fTag, entry.data.eData.pData.rfidTag))
    {
      flag = i;
      break;
    }
  }
  return flag;
```

```
}

boolean compRFID(byte tagA[10], byte tagB[10])
{
  boolean flag = false;
  for(int i=0;i<10;i++)
  {
    if (tagA[i]==tagB[i])
      flag = true;
    else
    {
      flag = false;
      break;
    }
  }
  return flag;
}

void addRFID()
{
  Serial.println("Please Scan RFID Tag to add.");
  while(1)
  {
    if(readRFID())
    {
      addPin();
      for(int i=0;i<10;i++)
        pData.data.rfidTag[i]=tagdata.tag.data[i];
      addEntry(pData.raw);
      return;
    }
  }
}

void delRFID()
{
  Serial.println("Please Scan RFID Tag to delete.");
  while(1)
  {
    if(readRFID())
    {
      delEntry(findRFID(tagdata.tag.data));
      return;
    }
  }
}
```

Category:Madlab projects

From:
http://testwiki.hecatron.com/ - **Hacman DEMO ONLY**

Permanent link:
http://testwiki.hecatron.com/doku.php?id=old:hackspace:door-system:door-control:version-0-3

Last update: **2022/11/30 16:34**