Introduction to RFID

1. Terminology

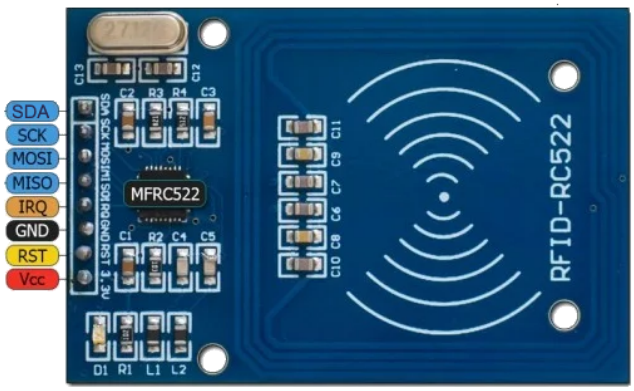
1.1. RFID: Radio-frequency identification. RFID uses electromagnetic fields to automatically identify and track tags attached to objects. An RFID system consists of a tiny radio transponder, a radio receiver and transmitter. A transponder, embedded in the tag (or card or PICC), is a type of radio transmitter that transmits signals automatically when it receives particular signals.
When triggered by an electromagnetic interrogation pulse from a nearby RFID reader device, the tag (card or PICC) transmits digital data, usually an identifying inventory number, back to the reader. This number can be used (like the barcode) to track inventory goods.
But unlike a barcode, the tag does not need to be within the line of sight of the reader. The RFID tag may be embedded in the tracked object.
RFID is one method of Automatic Identification and Data Capture (AIDC).

1.2. PICC: Proximity Integrated Circuit **Card**. That's the RFID card. It's also referred to as RFID tag. By proximity we mean a reading distance which is a short range of 10 cm or less for passive PICC or a greater range up to hundreds of meters for an active PICC.

Tag or PICC ->

1.3. PCD: Proximity Coupling **Device**. That's the reader of RFID card. It's also known as the **Interrogator** /ɪnˈtɛrəgeɪtə/ because it interrogates data from the card. NXP MFRC522 Contactless Reader IC.

PCD ->



1.4. MIFARE: While dealing with RFID card, the word MIFARE, also abbreviated MF will be coming often

used in contactless smart cards and proximity cards. ***[Read more at the end of this page.]***

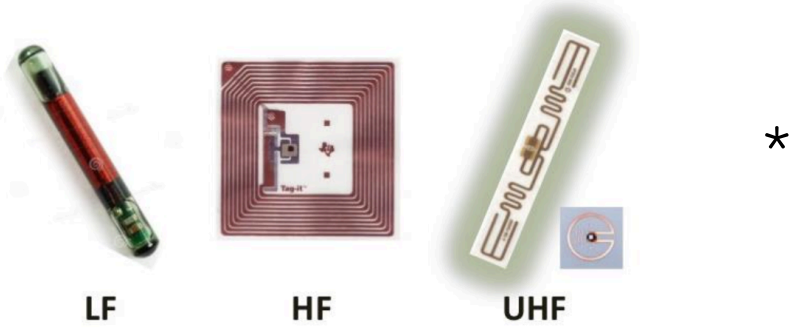
NXP Semiconductors is a Dutch-American semiconductor manufacturer with headquarters in Eindhoven, Netherlands and Austin, United States. The brand name MIFARE covers proprietary solutions based upon various levels of the ISO/IEC 14443 Type A 13.56 MHz contactless smart card standard. **ISO** (the **I**nternational **O**rganization for **S**tandardization) and **IEC** (the **I**nternational **E**lectrotechnical **C**ommission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a **j**oint **t**echnical **c**ommittee, ISO/IEC JTC 1.
The ISO/IEC 14443 series of standards describes the parameters for identification cards or objects for international interchange. The ISO/IEC 14443 series of standards is intended to allow operation of proximity cards in the presence of other contactless cards or objects conforming to the ISO/IEC 10536 series of standards and the ISO/IEC 15693 series of standards and near field communication (NFC) devices conforming to ISO/IEC 18092 and ISO/IEC 21481.

1.5. MFRC522 stands for MIFARE RC522. As I write I have not yet found out what RC in the card brand name means. I also couldn't find out the significance of the number 522. But it is assumingly **R**FID **C**ard or something very close to that. That being said, let's agree to take MFRC522 as **MI**FARE **R**FID **C**ard 522.

2. Active and Passive RFID Tags

According to their power supply source, RFID tags can be classified as passive, semi-passive, and active.

2.1. Passive RFID tags (mostly used) are powered by energy from the RFID reader's interrogating radio waves.



*

Passive RFID tags typically do not possess an onboard source of power. They are batteryless, so they use the incoming signal from the interrogator to supply the embedded chip. That's know as **power harvesting**.

Passive RFID tags do not have any RF emitters on board so they cannot create their own RF signals. Thus, the only way that allows a passive RFID tag to communicate with a reader

interrogator. Backscattering is the reflection of waves, particles, or signals back to the direction from which they came.

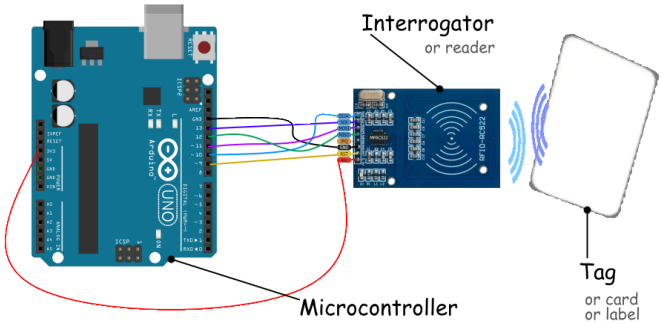
2.2. Semi-passive RFID Tags

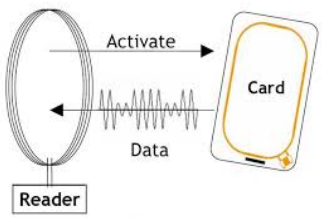
Semi-passive RFID tags have an embedded battery used to NOT generate RF signal BUT to supply internal circuitry or connected sensors.

Semi-passive RFID tags typically make use of backscatter modulation and reflect electromagnetic energy from the RFID reader to generate a tag response similar to that of standard passive tags. Common applications of semi-passive RFID tags include for example cold storage management, vehicle asset tracking, security access systems, supply chain automation, etc.

2.3. Active RFID Tags are powered by a battery and thus can be read at a greater range from the RFID reader, up to hundreds of meters.

3. How does an RFID system work?





A RFID system is made up of two parts: **a tag (or label or card)** and **a reader**. RFID tags are embedded with a **transmitter** and a **receiver**.
The RFID component on the tags have two parts: a **microchip** that stores and processes information, and an **antenna** to receive and transmit a signal. The tag **contains the specific serial number** for one specific object.
To read the information encoded on a tag, a two-way radio transmitter-receiver called an interrogator or reader emits an electromagnetic signal to the tag via an antenna. The emitted

powering up the chip.
The powered chip inside the tag then responds by sending its stored information back to the reader in the form of another radio signal different from what it received. This is called backscatter and its made possible by a transponder embedded on the tag. The reader (or interrogator) will then transmit the read data to an RFID computer or microcontroller to be processed by a written program.

4. Specifications of MFRC522 chip based board

- Operating frequency: **13.56MHz** (**HF**)
- Supply Voltage: 3.3V or 5V
- Current: 13-26mA
- Read Range: Approx. 3cm with supplied card and fob
- SPI Interface
- Max Data Transfer Rate: 10Mbit/s

5. Involved Data Types

In Arduino, byte, **uint8_t** and **unsigned char**, they are basically the same thing. These data types often cause confusions to new programmers. So is there any difference in them?
A **byte** stores an 8-bit unsigned number, from 0 to 255. For example for the number 0, the binary form is 00000000, there are 8 zeros (8 bits in total). for the number 255, the binary form is 11111111, there are 8 ones (8 bits in total).
A uint8_t data type is basically the same as byte in Arduino. Writers of embedded software often define these types, because systems can sometimes define int to be 8 bits, 16 bits or 32 bits long. The issue doesn't arise in C# or Java, because the size of all the basic types is defined by the language.
An **unsigned char** data type that occupies 1 byte of memory. It is the same as the byte datatype. The unsigned char datatype encodes numbers from 0 to 255. For consistency of Arduino programming style, the byte data type is to be preferred.

6. How is data stored on an RFID tag?

Data is typically stored in user memory on a tag. This is separate from the field for the unique serial number, which can be **pre-programmed** or **assigned by a user**.

7. Read Only, Read-write, and write-once RFID Tags?

7.1. Read-Only Tags have a unique serial number associated with them i.e they just have this serial number stored in the RFID Tag IC. This serial number is added to the tag at the time of manufacturing and cannot be updated or modified. These tags do not have any additional memory. They are widely used in applications that required simple identification like clothing stores and other applications which just require the tag to identify a product.

7.2. Read-Write Tags give users the ability to update or re-write the information stored on the tag. The serial number of the tag can be updated as required, readers can also write information

on to the tag when required, providing some sort of read log. These tags usually have EEPROM memory which can be updated as and when required

overwriting of data or tag tampering. These tags are significantly more expensive than Read-Only tags, however, have a number of use cases which can justify the higher price.

7.3. Write Once is another type of tag that is in-between Read-Only and Read-Write tags is the **Write Once**, Read Many (WORM) Tags - These tags can be encoded by the user one time, after this the tags gets locked and the information stored can't be altered. This tends to be more secure than Read-Write tags.

8. Dumping RFID Card Information

When the Microcontroller and the MFRC522 module are connected properly, and the code below is uploaded to the Microcontroller, we can present a PICC at reading distance of the MFRC522 Reader, and the unique ID (UID) will be shown on the Serial Monitor. Note that this may take some time as all data blocks are dumped, so the PICCs should be kept at reading distance until complete.

8.1. Library

Download the Library from here

8.2. Code

```
#include <SPI.h>
#include <MFRC522.h>

#define RST_PIN    9
#define SS_PIN    10

MFRC522 mfrc522(SS_PIN, RST_PIN);

void setup() {
    Serial.begin(9600);
    while (!Serial);
    SPI.begin();
        //Enhance the MFRC522 Receiver Gain to maximum value of some 48 dB
        mfrc522.PCD_SetRegisterBitMask(mfrc522.RFCfgReg, (0x07<<4));
    mfrc522.PCD_Init();
    delay(4);
    mfrc522.PCD_DumpVersionToSerial();
    Serial.println(F("DISPLAYING UID, SAK, TYPE, AND DATA BLOCKS:"));
}

void loop(){
    if(!mfrc522.PICC_IsNewCardPresent()){
        return;
    }

    if(!mfrc522.PICC_ReadCardSerial()){
        return;
    }

    /*
    Dump debug info about the card. Worry not. PICC_HaltA() will be automatically called at the e
    */
    mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
```

Let's explain how the code above works:

```
#include <SPI.h>
```

This line includes the SPI library, which is needed to communicate with the MFRC522 module using SPI.

```
#include <MFRC522.h>
```

This line includes the MFRC522 library, which provides functions for interfacing with the MFRC522 module.

```
#define RST_PIN    9
#define SS_PIN    10
```

These two lines define the reset (RST) and slave select (SS) pins for the MFRC522 module. The RST pin is connected to digital pin 9, and the SS pin is connected to digital pin 10.

```
MFRC522 mfrc522(SS_PIN, RST_PIN);
```

This line creates an instance of the MFRC522 class, called "mfrc522", and initializes it with the SS and RST pins.

```
void setup() {
```

This line starts the "setup" function, which is called once when the Arduino is powered on or reset.

```
Serial.begin(9600);
```

This line initializes the serial communication at a baud rate of 9600 bits per second. This allows the Arduino to send and receive data over a serial connection to a computer or other device.

```
while (!Serial);
```

This line waits until a serial connection is established before continuing with the rest of the setup function.

```
SPI.begin();
```

This line initializes the SPI communication between the Arduino and the MFRC522 module.

```
mfrc522.PCD_SetRegisterBitMask(mfrc522.RFCfgReg, (0x07<<4));
```

This line sets the receiver gain of the MFRC522 module to the maximum value. The PCD_SetRegisterBitMask function is a function provided by the MFRC522 library, which sets a bit mask for a particular register. In this case, the function sets the bit mask for the RFCfgReg register, which controls the receiver gain of the module.

```
mfrc522.PCD_Init();
```

This line initializes the MFRC522 module by configuring the necessary registers and preparing it for communication.

```
delay(4);
```

This line adds a delay of 4 milliseconds to ensure that the module is fully initialized before proceeding with the rest of the setup function.

```
mfrc522.PCD_DumpVersionToSerial();
```

useful for debugging purposes and ensuring that the module is functioning properly.

```
Serial.println(F("DISPLAYING UID, SAK, TYPE, AND DATA BLOCKS:"));
```

This line sends a string of characters to the serial port, indicating that the UID, SAK, card type, and data blocks of any detected RFID tags will be displayed on the serial port. SAK stands for "Select Acknowledge". It is a byte of data that is returned by an ISO 14443A compliant RFID tag after it has been selected by a reader. The SAK byte contains information about the tag's communication capabilities, including the tag type and the supported transmission rates. The SAK byte is used by the reader to determine the appropriate communication protocols and parameters to use when communicating with the tag. It is an important part of the initial communication between the reader and the tag, and is typically used in conjunction with the UID (unique identifier) of the tag to establish a communication link between the two devices.

```
void loop(){
```

This line starts the "loop" function, which is executed repeatedly after the setup function has completed.

```
if(!mfrc522.PICC_IsNewCardPresent()){
   return;
}
```

This block of code checks if a new RFID tag is present by calling the PICC_IsNewCardPresent function provided by the MFRC522 library. If no new tag is present, the function exits and waits for the next iteration of the loop function.

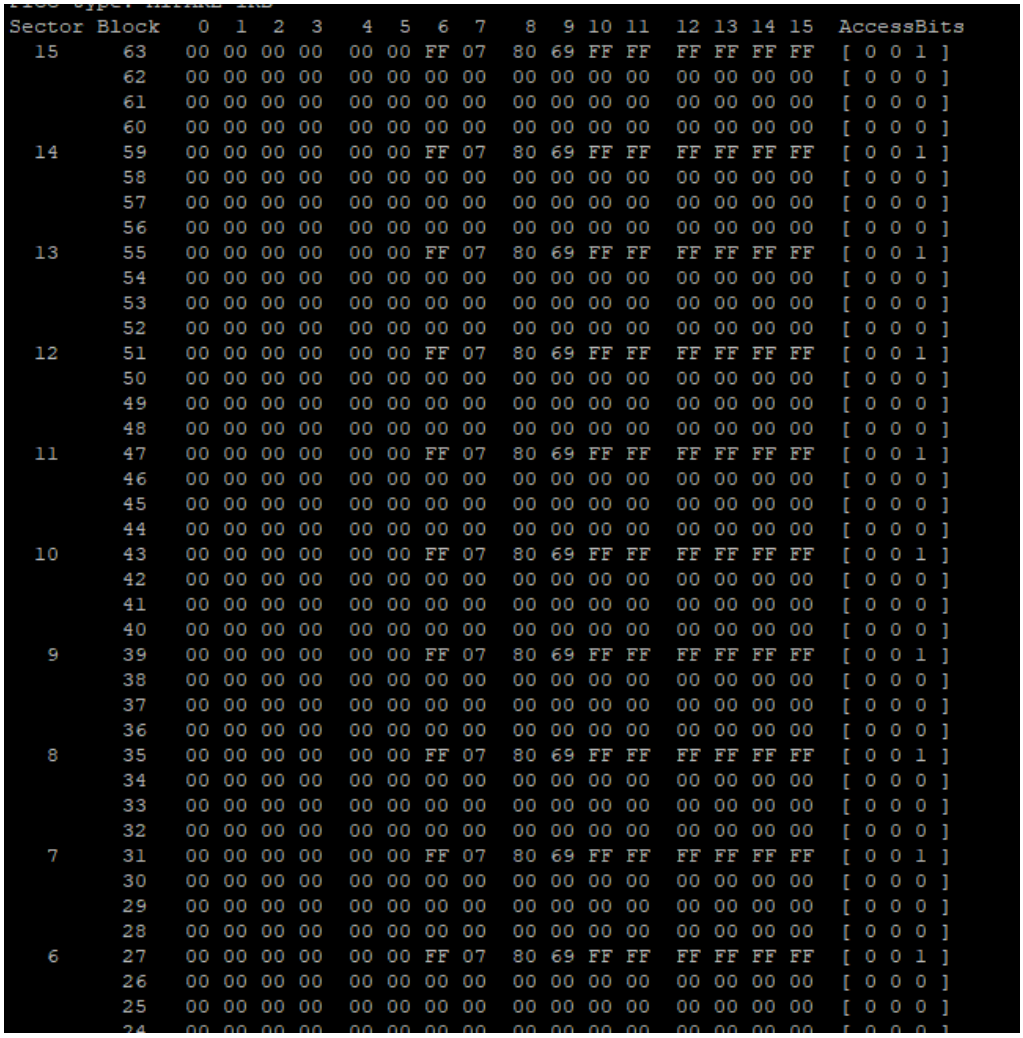
```
if(!mfrc522.PICC_ReadCardSerial()){
 return;
}
```

This block of code reads the serial number of the detected RFID tag using the PICC_ReadCardSerial function. If the read operation is unsuccessful, the function exits and waits for the next iteration of the loop function.

```
mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
```

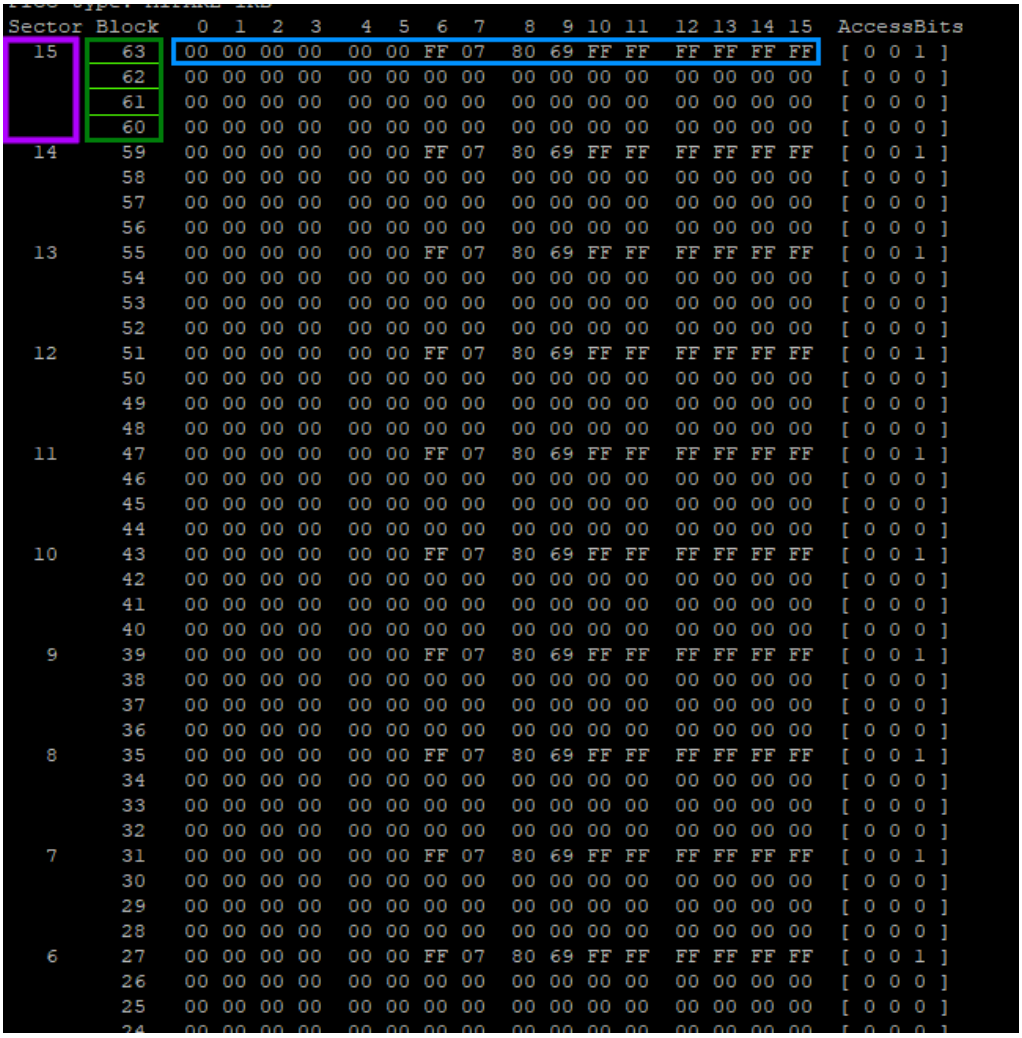
This line dumps the UID, SAK, card type, and data blocks of the detected RFID tag to the serial port using the PICC_DumpToSerial function. The "&" symbol is used to pass the address of the mfrc522.uid variable to the function, which is where the UID of the detected tag is stored.

8.3. Output: MIFARE Classic 1K Memory Layout



Sector	Block	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	AccessBits
15	63	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	62	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	61	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
14	59	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	58	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	57	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	56	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
13	55	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	54	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	53	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	52	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
12	51	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	49	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	48	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
11	47	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	46	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	45	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	44	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
10	43	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	42	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	41	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
9	39	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	38	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	37	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	36	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
8	35	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	34	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	33	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	32	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
7	31	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	29	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	28	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
6	27	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	26	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	25	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]

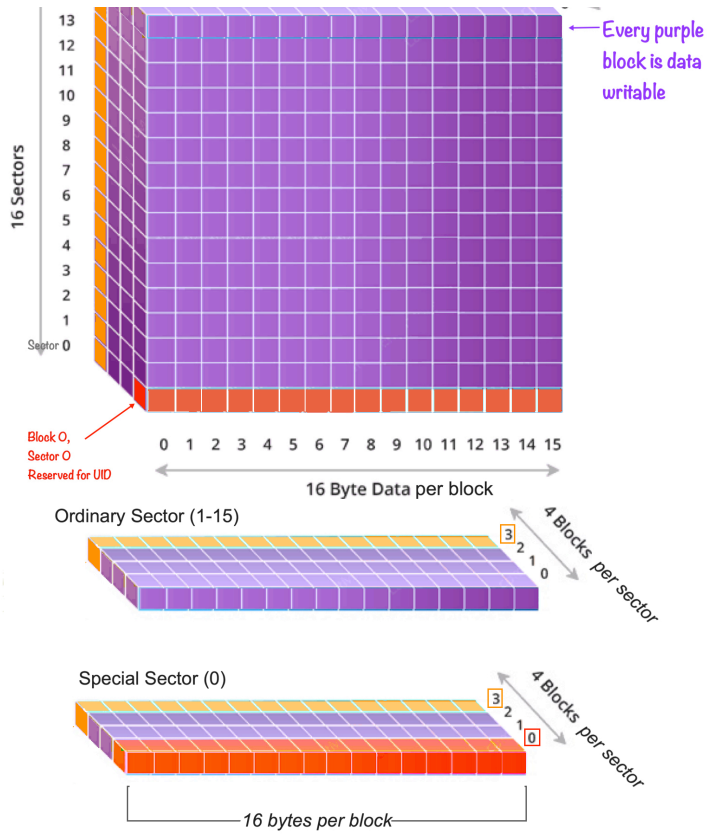
The 1K memory of the Tag is organized in 16 **sectors** (from 0 to 15)Each sector is further divided into 4 **blocks** (block 0 to 3). Each block can store 16 bytes of **data** (from 0 to 15).
That surely tells us we have 16 sectors x 4 blocks x 16 bytes of data = 1024 bytes = 1K memory.
The whole 1K memory with sectors, blocks and data is highlighted below.



Sector	Block	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	AccessBits
15	63	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	62	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	61	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
14	59	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	58	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	57	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	56	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
13	55	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	54	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	53	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	52	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
12	51	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	49	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	48	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
11	47	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	46	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	45	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	44	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
10	43	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	42	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	41	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
9	39	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	38	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	37	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	36	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
8	35	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	34	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	33	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	32	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
7	31	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	29	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	28	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
6	27	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	26	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	25	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]

Below is a 3D Representation of MIFARE Classic 1K Memory Map Layout

Block 3,
Every Sector
Reserved for Keys (A



1KByte consists of 16 sectors and 4 blocks per each sector.
- **Sectors from 0 to 3:** Read or write data by sector unit (Max.48Byte)
- **Sectors from 4 to 15:** Write data by block unit and read data by sector unit.

---

9. What's the Difference Between Proximity Cards and Smart Cards?

Proximity Cards store only a facility code and card number, while **Smart Cards** provide this information PLUS the ability to authenticate and store biographical information.

Many systems utilize proximity technology. These systems utilize readers that extract the ID number from the card and convey that ID number to the controller. The controller then searches the database for validity of that number, and instructs the reader whether to "allow" or "deny" entry based on the validity of the ID number in the software. If the card number was not a valid enrollee in the system, access would be denied.

As for Smart Cards, these cards can perform exactly as described above, but with even further capabilities of authentication and memory capacity. This additional storage can be used to store biographical information, pin codes or monetary values to be used in applications such as vending or pre-paid membership cards.

Some cards contain both 125kHz proximity technology as well as 13.56MHz smart card technology in a single card.
A couple of examples include the iCLASS 2020 and the RapidPROX Combi Card.

---

radio frequencies (RF) in the range of 30–300 kHz. Typically LF systems work at 125 KHz. Since its wavelengths range from 10–1 km, respectively, it is also known as the kilometre band or kilometre wave. The signal attenuation (loss) exhibited by LF radio waves is low, and that makes them suitable for long-distance communications.

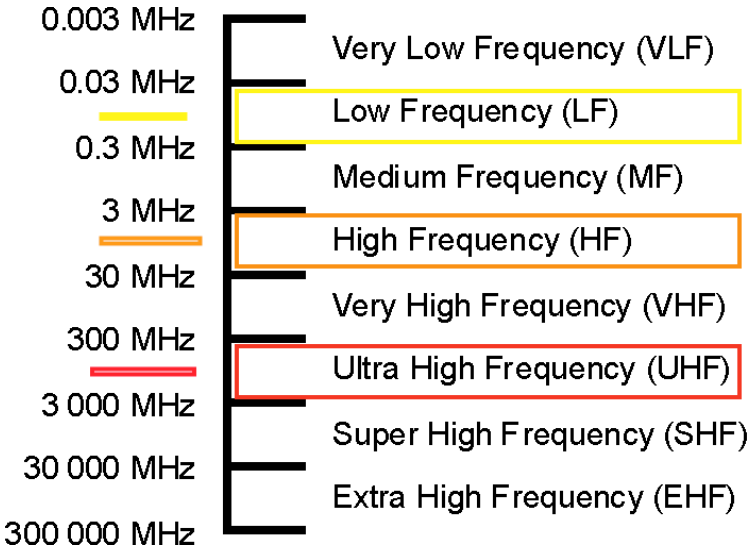
HF: High frequency (HF) is the ITU designation for radio frequencies (RF) in the range of 3-30 MHz. Typically HF systems work at 13.56 MHz. It is also known as the decameter band or decameter wave as its wavelengths range from one to ten decameters.

[On the picture above: Those tags we are familiar with are of 13.56MHz, thus High Frequency - HF]

UHF: Ultra High frequency (UHF) is the ITU designation for radio frequencies (RF) in the range of 300 MHz and 3 GHz. UHF Gen2 standard systems utilize the 860 to 960 MHz band. It is also known as the decimetre band as the wavelengths range from one meter to one tenth of a meter.

Lora Board that we use as a microcontroller uses UHF.

Below is a summary of frequency ranges:



** *The International Telecommunication Union is a specialized agency of the United Nations*

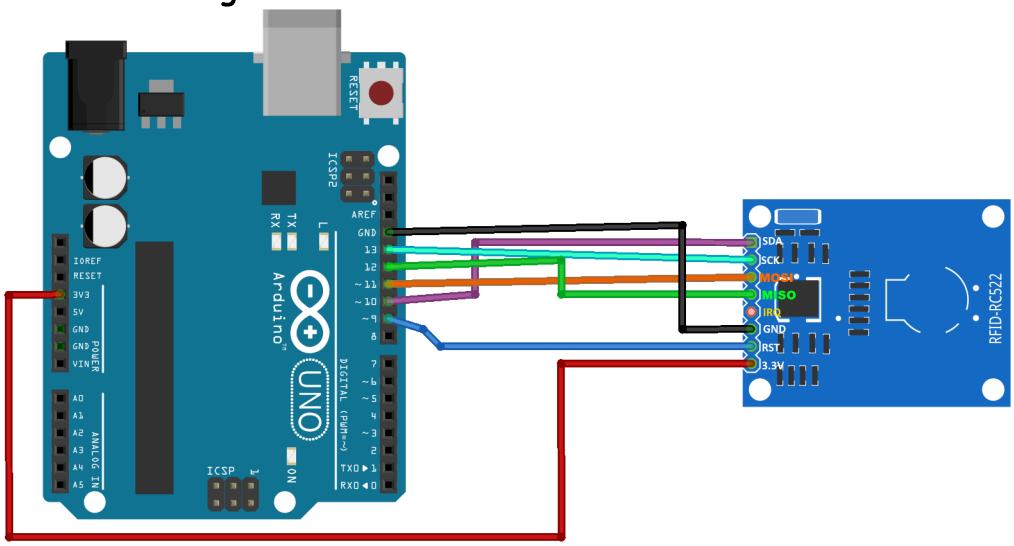
established on 17 May 1865 as the International Telegraph Union, making it the oldest UN agency. Wikipedia

---

Get RFID Card Unique ID

In the previous practice we got all the information from the card.
Amongst that information there was the UID of the card.
But what if we want to distinctively get the UID of the card?
This practice will guide us to do so.

Connection Diagram



Where is the Unique ID located?

The UID is located in
In the **Block 0** of the **Sector 0**

UID is made up of 4 bytes of data (from byte 0 to 3)

Code

```
#include <SPI.h>
#include <MFRC522.h>
#define SS_PIN 10
#define RST_PIN 9

MFRC522 rfid(SS_PIN, RST_PIN);
MFRC522::MIFARE_Key key;

byte nuidPICC[4];

void setup() {
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();
  Serial.println(F("READING THE CARD UNIQUE ID:"));

  for (byte i = 0; i < 6; i++) {

    key.keyByte[i] = 0xFF;
  }
```

```
  if(!rfid.PICC_IsNewCardPresent()){
    return;
  }

  if(!rfid.PICC_ReadCardSerial()){
    return;
  }


  if (rfid.uid.uidByte[0] != nuidPICC[0] ||
    rfid.uid.uidByte[1] != nuidPICC[1] ||
    rfid.uid.uidByte[2] != nuidPICC[2] ||
    rfid.uid.uidByte[3] != nuidPICC[3] ) {

    for (byte i = 0; i < 4; i++) {
      nuidPICC[i] = rfid.uid.uidByte[i];
    }


    Serial.println(F("********************"));
    printHex(rfid.uid.uidByte, rfid.uid.size);
    Serial.println(F("\n********************"));
  }

  else{
    Serial.println(F("This card was lastly detected."));
  }

 /*
  * Halt PICC
  * Stop encryption on PCD
 */

  rfid.PICC_HaltA();
  rfid.PCD_StopCrypto1();
}

void printHex(byte *buffer, byte bufferSize){
  for (byte i = 0; i < bufferSize; i++){
    Serial.print(buffer[i] < 0x10 ? " 0" : " ");
    Serial.print(buffer[i], HEX);
  }
}
```

Access Control Using RFID System

In this project, let's utilize Radio Frequency Identification (RFID) to identify the authorized card (tag) and then grant access to it.

We have a tag (PICC) that includes a microchip with an antenna, a reader (PCD) with an antenna, and a microcontroller (Arduino Board).

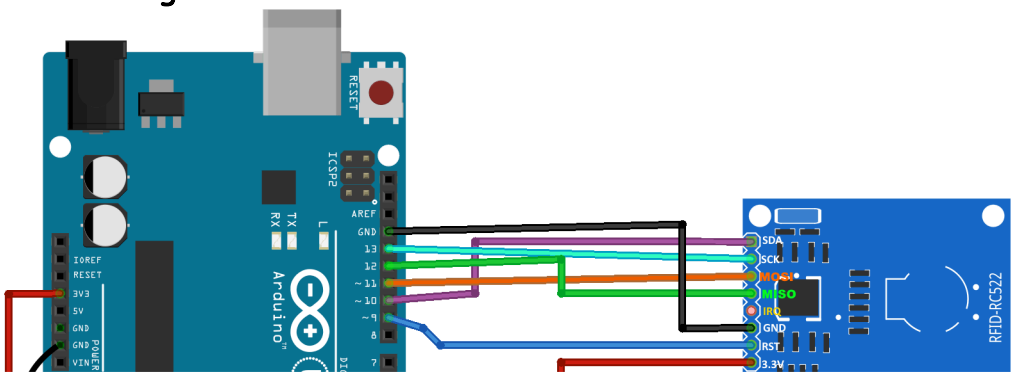
On the microcontroller, we'll write a program that accepts our authorized cards and rejects any other cards.

Prior to this project we have learnt to read the UID of a tag. Therefore we can easily know the UID of our card and we'll set that UID as authorized. We can set multiple card UIDs as authorized if we want. For example if every staff of an organization has her/his own card to have access to the organization's facilities. In that case we can create a database of our authorized cards.

Once a card (PICC) is brought in the proximity of the reader (PCD), the microcontroller, using the

not match, the access will be denied and the red LED will turn ON.

Circuit Diagram



Code

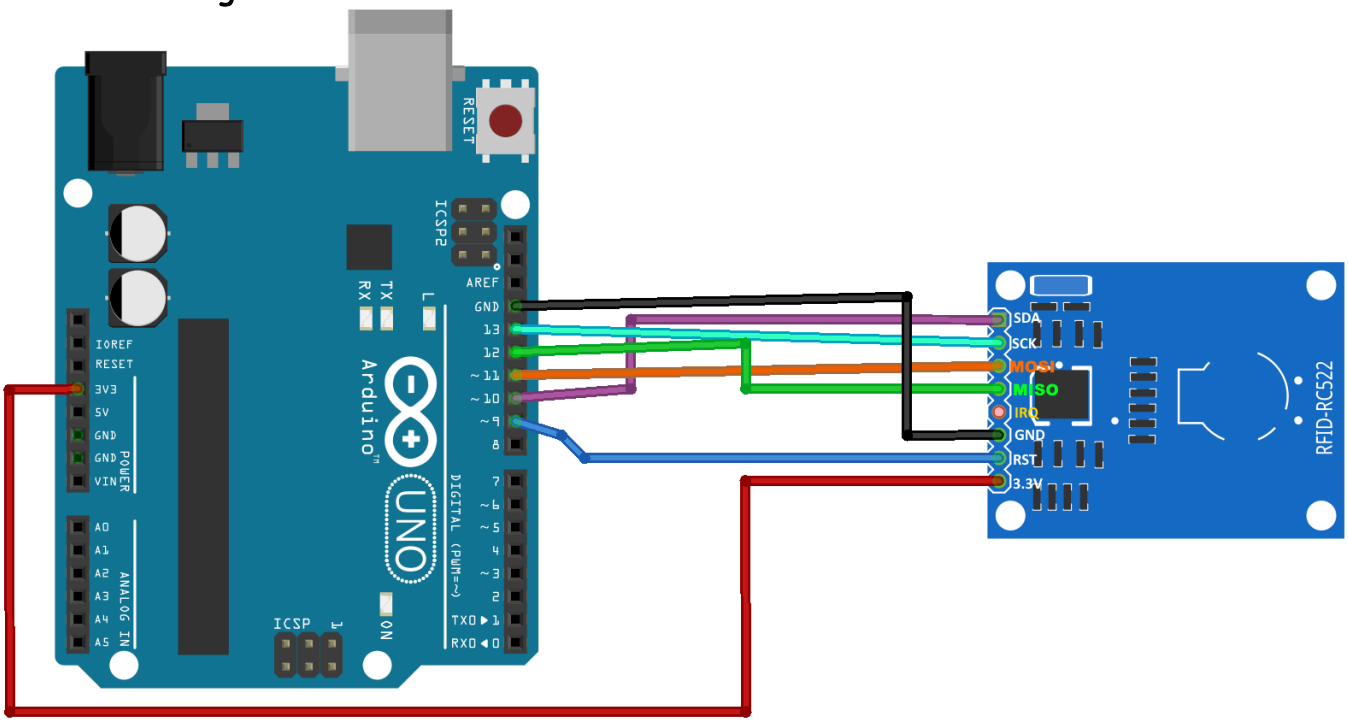
```
#include <SPI.h>
#include <MFRC522.h>

#define SS_PIN 10
#define RST_PIN 9
#define RED 2
#define Green 3
MFRC522 mfrc522(SS_PIN, RST_PIN);   // Create MFRC522 instance.

void setup(){
  Serial.begin(9600);   // Initiate a serial communication
  SPI.begin();      // Initiate  SPI bus
  mfrc522.PCD_Init();   // Initiate MFRC522
  Serial.println("Approximate your card to the reader...");
  Serial.println();
  pinMode(Green, OUTPUT);
  pinMode(RED, OUTPUT);
  digitalWrite(Green, LOW);
  digitalWrite(RED, LOW);
}
void loop(){
  // Look for new cards
  if ( ! mfrc522.PICC_IsNewCardPresent()){
    return;
  }
  // Select one of the cards
  if ( ! mfrc522.PICC_ReadCardSerial()){
    return;
  }
  //Show UID on serial monitor
  Serial.print("UID tag :");
  String content= "";
  byte letter;
  for (byte i = 0; i < mfrc522.uid.size; i++){
     Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
     Serial.print(mfrc522.uid.uidByte[i], HEX);
     content.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
     content.concat(String(mfrc522.uid.uidByte[i], HEX));
  }
  Serial.println();
  Serial.print("Message : ");
  content.toUpperCase();

  //We assume your card has UID 1A 7C 78 30
  //Please change the UID of the card/cards that you want to give access
  if(content.substring(1) == "1A 7C 78 30"){

    Serial.println("Authorized access");
    Serial.println();
    digitalWrite(Green, HIGH);
```

```
 else{
    Serial.println(" Access denied");
    digitalWrite(RED, HIGH);
    delay(10000);
    digitalWrite(RED, LOW);
  }
}
```

Writing Data to the RC522 Card {single block}

Previously we used RFID Card for access control. RFID badges can be used to control access to particular areas, for time keeping, or other applications.
In this section we'll write data to the RFID card. In Rwanda, the well known application of data writing is the operation of topping up money to Tap&Go cards.

Connection Diagram.



Code:

```
#include <SPI.h>
#include <MFRC522.h>
#define RST_PIN  9
#define SS_PIN  10
MFRC522 mfrc522(SS_PIN, RST_PIN);
MFRC522::MIFARE_Key key;
MFRC522::StatusCode card_status;

void setup(){
  Serial.begin(9600);
  SPI.begin();
  mfrc522.PCD_Init();
  Serial.println(F("Enter data, ending with #"));
  Serial.println("");
}


void loop(){
  for(byte i = 0; i < 6; i++){
    key.keyByte[i] = 0xFF;
```

```
    return;
  }

  if (!mfrc522.PICC_ReadCardSerial()){
    Serial.println("[Bring PIIC closer to PCD]");
    return;
  }

  byte buffr[16];
  byte block = 4;
  byte len;

  Serial.setTimeout(1000L);

  /*
   * Read data from serial
   */

  len = Serial.readBytesUntil('#', (char *) buffr, 16) ;

  Serial.println("[PIIC ready!]");
  if(len>0){
    for(byte i = len; i < 16; i++){
      buffr[i] = ' ';     //We have to pad array items with spaces.
    }

    String mString;
    mString = String((char*)buffr);

    Serial.println(" ");
    writeBytesToBlock(block, buffr);
    Serial.println(" ");
    mfrc522.PICC_HaltA();
    mfrc522.PCD_StopCrypto1();
    Serial.print("Saved on block ");
    Serial.print(block);
    Serial.print(":");
    Serial.println(mString);
    Serial.println("Note: To rewrite to this PICC, take it away from the PCD, and bring it closer again.");
  }
  delay(500);
}


void writeBytesToBlock(byte block, byte buff[]){
  card_status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block, &key, &(mfrc522.uid));

  if(card_status != MFRC522::STATUS_OK) {
    Serial.print(F("PCD_Authenticate() failed: "));
    Serial.println(mfrc522.GetStatusCodeName(card_status));
    return;
  }

  else{
    Serial.println(F("PCD_Authenticate() success: "));
  }
  // Write block
  card_status = mfrc522.MIFARE_Write(block, buff, 16);

  if (card_status != MFRC522::STATUS_OK) {
    Serial.print(F("MIFARE_Write() failed: "));
    Serial.println(mfrc522.GetStatusCodeName(card_status));
    return;
  }
  else{
    Serial.println(F("Data saved."));
  }
}
```

Let's explain what the code does:

We started by importing SPI and MFRC522 libraries

Then we instantiated MFRC522 class

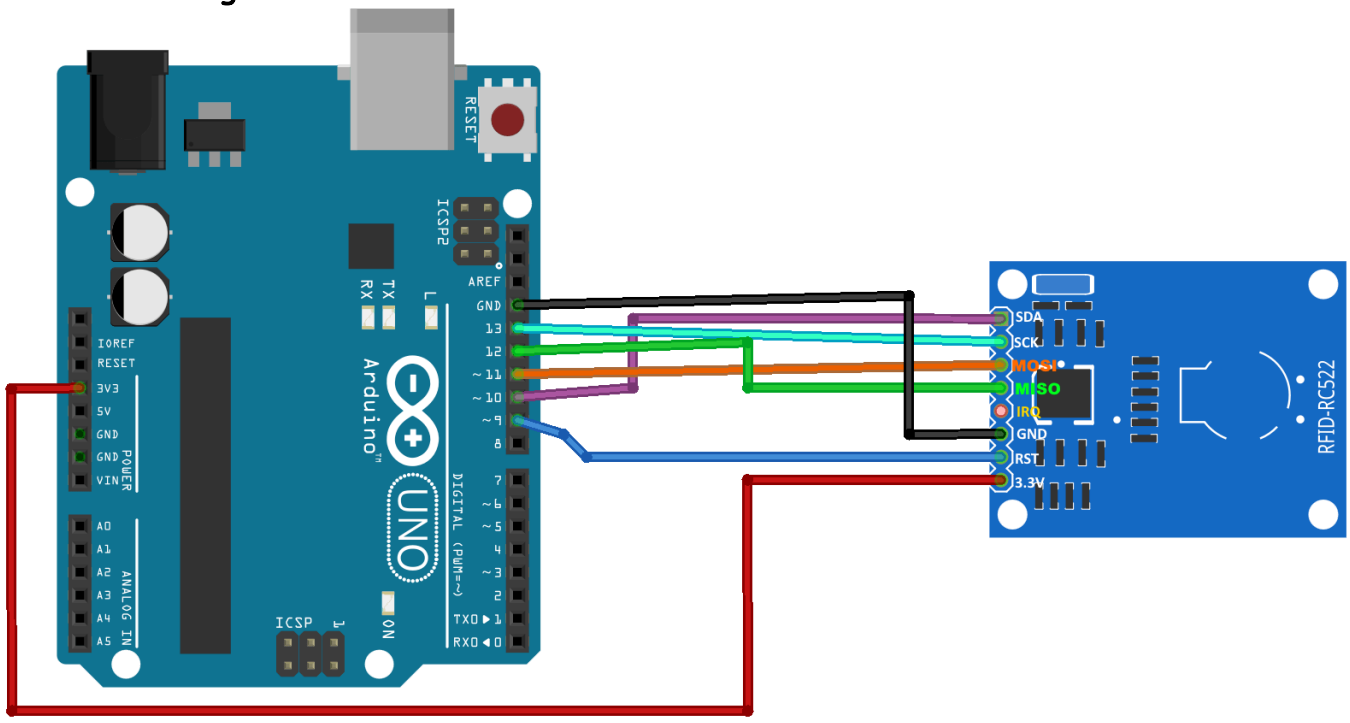
Also from MFRC522 we imported StatusCode as **card_status**
Amongst all the pins of RC522 Card Reader/Write only two pins need to be defined.
Those are RESET connected to digital pin 9 of the Arduino board and SDA pin connected to digital pin 10.
Remember, for Arduino board digital pin 10 serves as Slave Select (SS) or Chip Select (CS).
Take note that though digital pin 9 of the Arduino board is the one used for RESET in this code, any digital or analog pin apart from D13, D12, D11, D10 which serve as as Serial Clock (SCK or SCL), MISO, MOSI, and SS respectively.

Within the function setup():
- Serial Communication (UART) with the computer is initialized
- SPI bus is initialized
- MFRC522 card is initialized
- A heading is printed on the serial monitor: F("WRITING DATA ON RC522 CARD"). F() is used in order to move constant strings to the **program memory (or flash memory)** instead of the RAM. That will free up dynamic RAM though it will take up space that will decrease the amount of other code we can write. **Flash memory** is an electronic non-volatile computer memory storage medium that can be electrically erased and reprogrammed.

Reading Data From RFID Card

In this section we'll learn to read data saved on the RFID card.
This has an application in Tap&Go system where the passenger places the RFID card on the module located at the bus' entrance in order to get money deducted.

Connection Diagram:



Code:

```
#include <SPI.h>
#include <MFRC522.h>
```

```
MFRC522 mfrc522(SS_PIN, RST_PIN);

MFRC522::MIFARE_Key key;
MFRC522::StatusCode card_status;

void setup(){
    Serial.begin(9600);
    SPI.begin();
    mfrc522.PCD_Init();
    Serial.println(F("PCD Ready!"));
}

void loop(){
    for (byte i = 0; i < 6; i++){
        key.keyByte[i] = 0xFF;
    }
    if(!mfrc522.PICC_IsNewCardPresent()){
        return;
    }
    if(!mfrc522.PICC_ReadCardSerial()){
        return;
    }

    Serial.println(F("\n*** Balance on the PICC ***\n"));
    String balance = readBytesFromBlock();
    Serial.println(balance);
    Serial.println(F("\n**************************\n"));
    delay(1000);

    mfrc522.PICC_HaltA();
    mfrc522.PCD_StopCrypto1();
}

String readBytesFromBlock(){
    byte blockNumber = 4;

    card_status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, blockNumber, &key, &(
    if(card_status != MFRC522::STATUS_OK){
        Serial.print(F("Authentication failed: "));
        Serial.println(mfrc522.GetStatusCodeName(card_status));
        return;
    }

    byte arrayAddress[18];
    byte buffersize = sizeof(arrayAddress);
    card_status = mfrc522.MIFARE_Read(blockNumber, arrayAddress, &buffersize);
    if(card_status != MFRC522::STATUS_OK){
        Serial.print(F("Reading failed: "));
        Serial.println(mfrc522.GetStatusCodeName(card_status));
        return;
    }

    String value = "";
    for (uint8_t i = 0; i < 16; i++){
        value += (char)arrayAddress[i];
    }
```

```
}
```

How the code works

Import MFRC522, the RFID reader library	#include <MFRC522.h>
Instatiate a MFRC522 reader object with slave select pin and reset pin	MFRC522 mfrc522(SS_PIN, RST_PIN);
Create a MIFARE_Key struct named 'key', which will hold the card information	MFRC522::MIFARE_Key key;
Declare the block number to hold data written to the PICC	byte block=4;
Initialize MFRC522 PCD (Card Reader)	mfrc522.PCD_Init();

Prepare the security key for the read and write functions
whereby keyByte is defined in the "MIFARE_Key" 'struct' definition in the MFRC522.h library
	for (byte i = 0; i < 6; i++) {
	key.keyByte[i] = 0xFF;
	}

Identify if the PICC is new or has already been used. And if a new PICC placed to RFID reader continue.	mfrc522.PICC_IsNewCardPresent()
Select one of the cards located in the proximity	mfrc522.PICC_ReadCardSerial()
Stop reading the card	mfrc522.PICC_HaltA();
Stop encryption on PCD	mfrc522.PCD_StopCrypto1();
We need to define a variable with the read buffer size, since the MIFARE_Read method needs a pointer to the variable that contains the size, &buffersize is a pointer to the buffersize variable; MIFARE_Read requires a pointer instead of just a number.	byte buffersize = 18;

****

What does "struct" mean for "MFRC522::MIFARE_Key key;"?
Is this a static property in the class MFRC522?

---

int vs byte

A data type tells what a variable can hold in programming. A variable is a name we give to a piece of data in our memory. In general, we should pick the minimum for our task.
Because the number of blocks varies between 0 and 63

MIFARE Classic has 1 kilobytes of Memory

Each block can store 16 bytes of data (from 0 to 15).
That surely tells us we have 16 sectors x 4 blocks x 16 bytes of data = 1024 bytes = 1K memory.

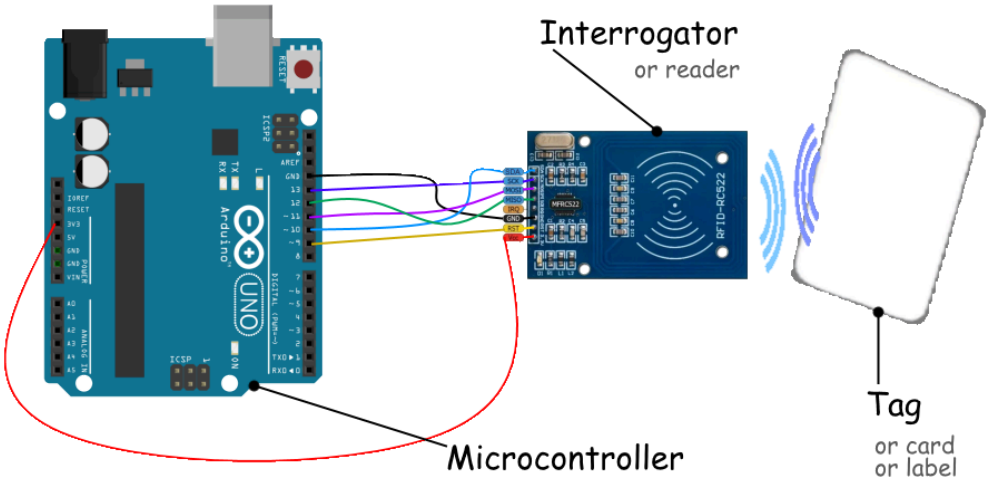
****

Transacting using RFID System

Prerequisite /priːˈrɛkwɪzɪt/

Before transaction, we should have written to the PICC a number like 5000 which represents money.

Connection Diagram



Code:

The code below does few things:
- Reading the initial Balance from the RFID card
- if the initial balance is more than the set transport fare then deduct set transport fare from the initial balance
- Save the new balance to the RFID card

```
#include <SPI.h>
#include <MFRC522.h>
#define RST_PIN 9
#define SS_PIN 10

MFRC522 mfrc522(SS_PIN, RST_PIN);
MFRC522::MIFARE_Key key;
MFRC522::StatusCode card_status;

void setup(){
  Serial.begin(9600);
  SPI.begin();
  mfrc522.PCD_Init();
  Serial.println(F("TRANSACTION:"));
}
```

```
  for (byte i = 0; i < 6; i++){
    key.keyByte[i] = 0xFF;
  }

  if(!mfrc522.PICC_IsNewCardPresent()){
    return;
  }


  if(!mfrc522.PICC_ReadCardSerial()){
    return;
  }

  String initial_balance = readBalanceFromCard(block_number, buffer_for_reading);
  operateData(block_number, initial_balance);

  mfrc522.PICC_HaltA();
  mfrc522.PCD_StopCrypto1();
}

String readBalanceFromCard(byte blockNumber, byte readingBuffer[]){
  card_status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, 4, &key, &(mfrc522.uid));

  if(card_status != MFRC522::STATUS_OK){
    Serial.print(F("Authentication failed: "));
    Serial.println(mfrc522.GetStatusCodeName(card_status));
    return;
  }

  byte readDataLength = 18;
  card_status = mfrc522.MIFARE_Read(blockNumber, readingBuffer, &readDataLength);

  if(card_status != MFRC522::STATUS_OK){
    Serial.print(F("Reading failed: "));
    Serial.println(mfrc522.GetStatusCodeName(card_status));
    return;
  }

  String value = "";
  for (uint8_t i = 0; i < 16; i++){
    value += (char)readingBuffer[i];
  }
  value.trim();

  return value;
}

bool saveBalanceToCard(byte blockNumber, byte writingBuffer[]){
  card_status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, blockNumber, &key, &(mfrc522.uid));

  if(card_status != MFRC522::STATUS_OK){
    Serial.print(F("PCD_Authenticate() failed: "));
    Serial.println(mfrc522.GetStatusCodeName(card_status));
    return;
  }
  else{
    //Serial.println(F("PCD_Authenticate() success: "));
  }

  // Write block


  card_status = mfrc522.MIFARE_Write(blockNumber, writingBuffer, 16);
```

```
    return;
  }
  else{
    //Serial.println(F("Data saved."));
    delay(5000);
    return true;
  }
}

void operateData(byte blockNumber, String initialBalance){
  int fareToKimironko = 450;
  float newBalance = initialBalance.toInt()-fareToKimironko;

  if(initialBalance.toInt()<fareToKimironko){
    Serial.print("Insufficient Balance: ");
    Serial.println(initialBalance);
    return;
  }

  String initial_balance_str;
  char writingBuffer[16];
  initial_balance_str = (String)newBalance;
  initial_balance_str.toCharArray(writingBuffer, 16);
  int strLeng = initial_balance_str.length()-3;
  /*
   * This servers to add spaces to the typed text in order to fill up to 16 characters
  */

  for(byte i = strLeng; i < 30; i++){
    writingBuffer[i] = ' ';
  }

  Serial.println("\n**********************");
  Serial.println("Processing...");
  if(saveBalanceToCard(blockNumber, writingBuffer)==true){
    Serial.print("Initial Balance: ");
    Serial.println(initialBalance);
    Serial.print("Fare to Kimironko: ");
    Serial.println(fareToKimironko);
    Serial.print("New Balance: ");
    Serial.println(newBalance);
    Serial.println("Transaction Succeeded.\n");
  }
  else{
    Serial.print("Transaction Failed.\nPlease try again.\n");
  }
  Serial.println("**********************\n");
}
```

Time-stamping to RFID Transaction

Now that we have mastered the use of RTC module, it's time to apply what we have just learned to RFID transacting. Time at which the transaction is done is very important. It helps us know what happened and when exactly it happened.

1. Wiring Diagram

Code:

This code stamps time to every transaction.

```
#include <SPI.h>
#include <MFRC522.h>
#include "DS1302.h"
#define RST_PIN 9
#define SS_PIN 10
#define ResetPin 7
#define DataPin 6
#define ClockPin 5

DS1302 rtc(ResetPin, DataPin, ClockPin); //Create a DS1302 object
MFRC522 mfrc522(SS_PIN, RST_PIN);
MFRC522::MIFARE_Key key;
MFRC522::StatusCode card_status;

void setup(){
    Serial.begin(9600);
    rtc.writeProtect(false); // turn off write protection
    rtc.halt(false); // clear the clock halt flag
    /*
    rtc.time(t);  is only used to set time.
    Other times it will be commented out.

    The format for Time t() is t(year,month,date,hour,minutes,seconds,day)
    */
    Time t(2022, 6, 6, 14, 57, 10, Time::kMonday);

    rtc.time(t); //use it ONLY to set time. Otherwise, comment it out.
    SPI.begin();
```

```
}

void loop(){
    byte block_number = 4;
    byte buffer_for_reading[18];
    for (byte i = 0; i < 6; i++){
        key.keyByte[i] = 0xFF;
    }

    if(!mfrc522.PICC_IsNewCardPresent() || !mfrc522.PICC_ReadCardSerial()){
        return;
    }

    String initial_balance = readBalanceFromCard(block_number, buffer_for_reading);
    operateData(block_number, initial_balance);

    mfrc522.PICC_HaltA();
    mfrc522.PCD_StopCrypto1();
}

String readBalanceFromCard(byte blockNumber, byte readingBuffer[]){
    card_status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, 4, &key, &(mfrc522.ui

    if(card_status != MFRC522::STATUS_OK){
        Serial.print(F("Authentication failed: "));
        Serial.println(mfrc522.GetStatusCodeName(card_status));
        return;
    }

    byte readDataLength = 18;
    card_status = mfrc522.MIFARE_Read(blockNumber, readingBuffer, &readDataLength);

    if(card_status != MFRC522::STATUS_OK){
        Serial.print(F("Reading failed: "));
        Serial.println(mfrc522.GetStatusCodeName(card_status));
        return;
    }

    String value = "";
    for (uint8_t i = 0; i < 16; i++){
        value += (char)readingBuffer[i];
    }
    value.trim();

    return value;
}

bool saveBalanceToCard(byte blockNumber, byte writingBuffer[]){
    card_status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, blockNumber, &key, &(

    if(card_status != MFRC522::STATUS_OK){
        Serial.print(F("PCD_Authenticate() failed: "));
        Serial.println(mfrc522.GetStatusCodeName(card_status));
        return;
    }
    else{
```

```
        // Write block

        card_status = mfrc522.MIFARE_Write(blockNumber, writingBuffer, 16);
        if(card_status != MFRC522::STATUS_OK){
            Serial.print(F("MIFARE_Write() failed: "));
            Serial.println(mfrc522.GetStatusCodeName(card_status));
            return;
        }
        else{
            //Serial.println(F("Data saved."));
            delay(5000);
            return true;
        }
    }

    void operateData(byte blockNumber, String initialBalance){
        int fareToKimironko = 450;
        float newBalance = initialBalance.toInt()-fareToKimironko;

        if(initialBalance.toInt()<fareToKimironko){
            Serial.print("Insufficient Balance: ");
            Serial.println(initialBalance);
            return;
        }

        String initial_balance_str;
        char writingBuffer[16];
        initial_balance_str = (String)newBalance;
        initial_balance_str.toCharArray(writingBuffer, 16);
        int strLeng = initial_balance_str.length()-3;
        /*
        * This servers to add spaces to the typed text in order to fill up to 16 characters
        */

        for(byte i = strLeng; i < 30; i++){
            writingBuffer[i] = ' ';
        }

        Serial.println("_____________________________");
        Serial.println("New transaction processing...");
        Serial.println("********************");
        if(saveBalanceToCard(blockNumber, writingBuffer)==true){
            Serial.print("Initial Balance: ");
            Serial.println(initialBalance);
            Serial.print("Fare to Kimironko: ");
            Serial.println(fareToKimironko);
            Serial.print("New Balance: ");
            Serial.println(newBalance);
        }
        else{
            Serial.print("Transaction Failed.\nPlease try again.\n");
        }

        char * timearray = readTime();
        Serial.print("Timestamp: ");
```

```
    Serial.println("Status: Success");
    Serial.println("_____________________________");
}


String dayAsString(const Time::Day day) { // function that converts the day ouptput into a string
    switch (day) {
        case Time::kSunday: return "Sunday";
        case Time::kMonday: return "Monday";
        case Time::kTuesday: return "Tuesday";
        case Time::kWednesday: return "Wednesday";
        case Time::kThursday: return "Thursday";
        case Time::kFriday: return "Friday";
        case Time::kSaturday: return "Saturday";
    }
    return "(unknown day)";
}

char * readTime(){
    Time t = rtc.time(); // get the time and date from the chip.
    const String day = dayAsString(t.day); // obtain text for the day of the week
    static char CurrentTime[50]; // initialise a character array to hold the date text
    snprintf(
        CurrentTime,
        sizeof(CurrentTime),
        "%s %04d-%02d-%02d %02d:%02d:%02d",
        day.c_str(),
        t.yr,
        t.mon,
        t.date,
        t.hr,
        t.min,
        t.sec
    ); // format the time into the character array
    return CurrentTime; // return the current time
}
```

Note: Avoid Double Printing on Serial Monitor
Set the line ending menu to "No line ending".

Storing Transactions on a host PC

A non-standalone embedded system connected to a host like a PC can store data to it [the host]. Our Embedded System is connected the the host PC using a USB cable and **the communication protocol used is UART.**

Let's first do the wiring according to wiring diagram below:

And make sure the code below is running:

This code stamps time to every transaction.

```
#include <SPI.h>
#include <MFRC522.h>
#include "DS1302.h"
#define RST_PIN 9
#define SS_PIN 10
#define ResetPin 7
#define DataPin 6
#define ClockPin 5

DS1302 rtc(ResetPin, DataPin, ClockPin); //Create a DS1302 object
MFRC522 mfrc522(SS_PIN, RST_PIN);
MFRC522::MIFARE_Key key;
MFRC522::StatusCode card_status;

void setup(){
    Serial.begin(9600);
    rtc.writeProtect(false); // turn off write protection
    rtc.halt(false); // clear the clock halt flag
    /*
    rtc.time(t);  is only used to set time.
    Other times it will be commented out.

    The format for Time t() is t(year,month,date,hour,minutes,seconds,day)
    */
    Time t(2022, 6, 6, 14, 57, 10, Time::kMonday);
    rtc.time(t); //use it ONLY to set time. Otherwise, comment it out.

    SPI.begin();
    mfrc522.PCD_Init();
```

```
void loop(){
    byte block_number = 4;
    byte buffer_for_reading[18];
    for (byte i = 0; i < 6; i++){
        key.keyByte[i] = 0xFF;
    }

    if(!mfrc522.PICC_IsNewCardPresent() || !mfrc522.PICC_ReadCardSerial()){
        return;
    }

    String initial_balance = readBalanceFromCard(block_number, buffer_for_reading);
    operateData(block_number, initial_balance);

    mfrc522.PICC_HaltA();
    mfrc522.PCD_StopCrypto1();
}

String readBalanceFromCard(byte blockNumber, byte readingBuffer[]){
    card_status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, 4, &key, &(mfrc522.ui

    if(card_status != MFRC522::STATUS_OK){
        Serial.print(F("Authentication failed: "));
        Serial.println(mfrc522.GetStatusCodeName(card_status));
        return;
    }

    byte readDataLength = 18;
    card_status = mfrc522.MIFARE_Read(blockNumber, readingBuffer, &readDataLength);

    if(card_status != MFRC522::STATUS_OK){
        Serial.print(F("Reading failed: "));
        Serial.println(mfrc522.GetStatusCodeName(card_status));
        return;
    }

    String value = "";
    for (uint8_t i = 0; i < 16; i++){
        value += (char)readingBuffer[i];
    }
    value.trim();

    return value;
}

bool saveBalanceToCard(byte blockNumber, byte writingBuffer[]){
    card_status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, blockNumber, &key, &(

    if(card_status != MFRC522::STATUS_OK){
        Serial.print(F("PCD_Authenticate() failed: "));
        Serial.println(mfrc522.GetStatusCodeName(card_status));
        return;
    }
    else{
        //Serial.println(F("PCD_Authenticate() success: "));
```

```
    // Write block

    card_status = mfrc522.MIFARE_Write(blockNumber, writingBuffer, 16);
    if(card_status != MFRC522::STATUS_OK){
        Serial.print(F("MIFARE_Write() failed: "));
        Serial.println(mfrc522.GetStatusCodeName(card_status));
        return;
    }
    else{
        //Serial.println(F("Data saved."));
        delay(5000);
        return true;
    }
}

void operateData(byte blockNumber, String initialBalance){
    int fareToKimironko = 450;
    float newBalance = initialBalance.toInt()-fareToKimironko;

    if(initialBalance.toInt()<fareToKimironko){
        Serial.print("Insufficient Balance: ");
        Serial.println(initialBalance);
        return;
    }

    String initial_balance_str;
    char writingBuffer[16];
    initial_balance_str = (String)newBalance;
    initial_balance_str.toCharArray(writingBuffer, 16);
    int strLeng = initial_balance_str.length()-3;
    /*
    * This servers to add spaces to the typed text in order to fill up to 16 characters
    */

    for(byte i = strLeng; i < 30; i++){
        writingBuffer[i] = ' ';
    }

    Serial.println("_____________________________");
    Serial.println("New transaction processing...");
    Serial.println("********************");
    if(saveBalanceToCard(blockNumber, writingBuffer)==true){
        Serial.print("Initial Balance: ");
        Serial.println(initialBalance);
        Serial.print("Fare to Kimironko: ");
        Serial.println(fareToKimironko);
        Serial.print("New Balance: ");
        Serial.println(newBalance);
    }
    else{
        Serial.print("Transaction Failed.\nPlease try again.\n");
    }

    char * timearray = readTime();
    Serial.print("Timestamp: ");
    Serial.println(timearray);
```

```
    Serial.println("_____________________________");
}


String dayAsString(const Time::Day day) { // function that converts the day ouptput into a string
    switch (day) {
        case Time::kSunday: return "Sunday";
        case Time::kMonday: return "Monday";
        case Time::kTuesday: return "Tuesday";
        case Time::kWednesday: return "Wednesday";
        case Time::kThursday: return "Thursday";
        case Time::kFriday: return "Friday";
        case Time::kSaturday: return "Saturday";
    }
    return "(unknown day)";
}

char * readTime(){
    Time t = rtc.time(); // get the time and date from the chip.
    const String day = dayAsString(t.day); // obtain text for the day of the week
    static char CurrentTime[50]; // initialise a character array to hold the date text
    snprintf(
        CurrentTime,
        sizeof(CurrentTime),
        "%s %04d-%02d-%02d %02d:%02d:%02d",
        day.c_str(),
        t.yr,
        t.mon,
        t.date,
        t.hr,
        t.min,
        t.sec
    ); // format the time into the character array
    return CurrentTime; // return the current time
}
```

On the host PC, let's then process data from Embedded System in 5 steps:

Step 1: Make sure the library pyserial is installed

On the host PC the library pyserial must be installed using the command

```
pip3 install pyserial
```

Step 2: Using nano, let's create and then open the file "readSerial.py" using a single command:

```
nano readSerial.py
```

```
#!/usr/bin/env python3
import serial
ser = serial.Serial(
        port='/dev/tty.usbserial-10', #plz change this according to your port number
        baudrate=9600,

        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE,
```

```
)
ser.flush()
if __name__ == '__main__':
    while True:
        if ser.in_waiting > 0:
            line = ser.readline().decode('utf-8').rstrip()
            f=open("transactions.txt","a")
            f.write(line)
            f.write("\n")
            f.close()
            print(line)
```

The python* code above reads whatever is written the particular serial interface and then saves it onto a file.

Step 3: **After editing save the file**

3.1. Try exiting

CTRL+X

3.2. You'll then be prompted to save changes, say yes by hitting Y

Y

3.3. Now that you have saved changes you can exit nano by hitting Enter Key (↵)

↵

Step 4: Let's make the file executable

```
sudo chmod +x readSerial.py
```

Step 5: Let's then run the code

****

NOTICE:

Before running the code below, make sure the Serial Monitor of Arduino IDE is closed or is not being used by any other application like PuTTY.

****

```
./readSerial.py
```

After this, we already have a permanent copy of what was volatile before.

From here we can decide in which format and location to save our data.

Can we save it in a local database? Can we upload it to a remote server? It all depends on us.

**though we used python in this lesson, any programming language that supports UART can be used to do what we did with Python.*

Individual Assignment to be submitted before ...

Assuming that you have a web application that can accept data using HTTP methods like POST or GET, write a python program that uploads data saved on the file "transactions.txt" to your web

a) Print "Data uploaded" on the screen.
b) Write "Data uploaded" to the Serial Interface

Storing transactions to an SD card

Standalone Embedded Systems [*independent systems which can work by themselves they don't depend on a host system*] that has data to store needs a storage device on its own.

We are now going to use 2 communication protocols (SPI and I2C) in a single system.

SPI
We have 2 slaves on a single master:
Master: ATMEGA328P (also known as AVR) on an Arduino Development Board
Slave 1: RFID R/W module with SPI interfacing protocol where reset pin is used.
Slave 2: SD Card R/W module with SPI interfacing protocol where reset pin is not used

I2C
We have 1 slave on a single master:
Master: ATMEGA328P (also known as AVR) on an Arduino Development Board
Slave: RTC module with I2C interfacing protocol

To the previous wiring let's add RTC module.

Code:
This code time-stamps and store to an SD card every transaction.

```
#include <SPI.h>
#include <MFRC522.h>
#include <SD.h>

#include "DS1302.h"

#define RST_PIN 9
#define SS_PIN 10

#define SD_CHIP_SELECT 4

#define RTC_RST_PIN 7
#define RTC_DATA_PIN 6
#define RTC_CLK_PIN 5

MFRC522 mfrc522(SS_PIN, RST_PIN);
MFRC522::MIFARE_Key key;
MFRC522::StatusCode card_status;

File myFile;


DS1302 rtc(RTC_RST_PIN, RTC_DATA_PIN, RTC_CLK_PIN);
```

```
  rtc.writeProtect(false); // turn off write protection
  rtc.halt(false); // clear the clock halt flag

  SPI.begin();
  Serial.println("Initializing SD card...");
  if (!SD.begin(SD_CHIP_SELECT)) {
    Serial.println("SD card initialization failed.");
    while (1);
  }
  Serial.println("SD card initialized.\n");

  Serial.println(F("DEVICE READY FOR TRANSACTION:"));
}

void loop(){
  byte block_number = 4;
  byte buffer_for_reading[18];
  for (byte i = 0; i < 6; i++){
    key.keyByte[i] = 0xFF;
  }

  if(!mfrc522.PICC_IsNewCardPresent()){
    return;
  }


  if(!mfrc522.PICC_ReadCardSerial()){
    return;
  }

  String initial_balance = readBalanceFromCard(block_number, buffer_for_reading);
  operateData(block_number, initial_balance);

  mfrc522.PICC_HaltA();
  mfrc522.PCD_StopCrypto1();
}

String readBalanceFromCard(byte blockNumber, byte readingBuffer[]){
    card_status = mfrc522.PCD_Authenticate(
        MFRC522::PICC_CMD_MF_AUTH_KEY_A, 4, &key, &(mfrc522.uid)
    );

  if(card_status != MFRC522::STATUS_OK){
    Serial.print(F("Authentication failed: "));
    Serial.println(mfrc522.GetStatusCodeName(card_status));
    return;
  }

  byte readDataLength = 18;
  card_status = mfrc522.MIFARE_Read(blockNumber, readingBuffer, &readDataLength);

  if(card_status != MFRC522::STATUS_OK){
    Serial.print(F("Reading failed: "));
    Serial.println(mfrc522.GetStatusCodeName(card_status));
    return;
  }
```

```
  for (uint8_t i = 0; i < 16; i++){
    value += (char)readingBuffer[i];
  }
  value.trim();

  return value;
}

bool saveBalanceToCard(byte blockNumber, byte writingBuffer[]){
    card_status = mfrc522.PCD_Authenticate(
        MFRC522::PICC_CMD_MF_AUTH_KEY_A, blockNumber, &key, &(mfrc522.uid)
    );

  if(card_status != MFRC522::STATUS_OK){
    Serial.print(F("PCD_Authenticate() failed: "));
    Serial.println(mfrc522.GetStatusCodeName(card_status));
    return;
  }
  else{
    //Serial.println(F("PCD_Authenticate() success: "));
  }

  // Write block

  card_status = mfrc522.MIFARE_Write(blockNumber, writingBuffer, 16);
  if(card_status != MFRC522::STATUS_OK){
    Serial.print(F("MIFARE_Write() failed: "));
    Serial.println(mfrc522.GetStatusCodeName(card_status));
    return;
  }
  else{
    //Serial.println(F("Data saved."));
    delay(5000);
    return true;
  }
}

void operateData(byte blockNumber, String initialBalance){
  int fareToKimironko = 450;
  float newBalance = initialBalance.toInt()-fareToKimironko;

  if(initialBalance.toInt()<fareToKimironko){
    Serial.print("Insufficient Balance: ");
    Serial.println(initialBalance);
    return;
  }

  String initial_balance_str;
  char writingBuffer[16];
  initial_balance_str = (String)newBalance;
  initial_balance_str.toCharArray(writingBuffer, 16);
  int strLeng = initial_balance_str.length()-3;
  /*
   * This servers to add spaces to the typed text in order to fill up to 16 characters
  */
```

```
  }

  Serial.println("\n********************");
  Serial.println("Processing...");
  if(saveBalanceToCard(blockNumber, writingBuffer)==true){
    char * timearray = readTime();
    Serial.println(timearray);
    Serial.print("Initial Balance: ");
    Serial.println(initialBalance);
    Serial.print("Fare to Kimironko: ");
    Serial.println(fareToKimironko);
    Serial.print("New Balance: ");
    Serial.println(newBalance);
    if(writeToSDCard((String)initialBalance, (String)fareToKimironko, (String)newBalance)){
      Serial.println("Transaction Succeeded.\n");
    }
    else{
      Serial.println("Transaction Succeeded BUT data was not saved on SDCard.\n");
    }
  }
  else{
    Serial.print("Transaction Failed.\nPlease try again.\n");
  }
  Serial.println("********************\n");
}

bool writeToSDCard(String initial_balance, String fare, String new_balance){
    char * timearray = readTime();
    myFile = SD.open("ops.txt", FILE_WRITE);
    if(myFile){
        myFile.println("********************");
        myFile.println(timearray);
        myFile.print("Initial Balance: ");
        myFile.println(initial_balance);
        myFile.print("Fare to Kimironko: ");
        myFile.println(fare);
        myFile.print("New Balance: ");
        myFile.println(new_balance);
        myFile.println("********************\n");
        myFile.close();
        return true;
  }else{
    Serial.println("error opening ops.txt");
    return false;
  }
 }


String dayAsString(const Time::Day day){
    switch(day){
        case Time::kSunday: return "Sunday";
        case Time::kMonday: return "Monday";
        case Time::kTuesday: return "Tuesday";
        case Time::kWednesday: return "Wednesday";
        case Time::kThursday: return "Thursday";
        case Time::kFriday: return "Friday";
```

```
  return "(unknown day)";
}

char * readTime(){
    Time t = rtc.time();
    const String day = dayAsString(t.day);
    static char CurrentTime[50];
    snprintf(
        CurrentTime,
        sizeof(CurrentTime),
        "%s %04d-%02d-%02d %02d:%02d:%02d",
        day.c_str(),
        t.yr,
        t.mon,
        t.date,
        t.hr,
        t.min,
        t.sec
        ); // format the time into the character array
  return CurrentTime; // return the current time
}
```

3.3V vs. 5V regarding the PCD signal strength

I experimented with supplying 3.3V then with 5V. The following is my observation:

With 3.3V a PCD can read the keychain only

With 5V a PCD can read both the card and the keychain

Reading from a dump

35 35 30 20

00110101 00110101 00110000 00100000

128-64-**32**-**16**-8-**4**-2-**1**
128-64-**32**-**16**-8-**4**-2-**1**
128-**64**-**32**-16-8-4-2-1
128-**64**-32-16-8-4-2-1

NFC Tag

Description

Chip: MF1108 , compatible Mifare s50 1k
Diameter: 25mm,
Material: PVC / 3M adhesive sticker
Operation frequency: 13.56 MHz
Detecting distance: 3-10cm
Standard: ISO14443A
Capacity: 1024 bytes , 764 bytes available to use, manufacturer block is read only
Compatible with Door Entry system, NFC Mobile Phone, Tickets, Credit Cards, and more.
Compatible with all 13.56MHz RFID reader to gain access control.
It applies to access control, hotel locks, staff attendance and school campus access and payment control, identification and security systems, parking lot entry and payment, social security management, transportation payment, municipal and ancillary service payment.
Note: If you apply for Mobile phone NFC, you should know this tag may not support all the NFC phone.

What is NFC?

Near field communication (NFC) are protocols that electronic devices use to communicate and transfer data between each other. Near field communication devices have to be very near to each other, usually between 10cm, but the range can vary depending on the device that is transmitting and the size of the tag. NFC tags require no power input whatsoever. They use magnetic induction between two small loop antennas. The tags these days carry between 96 and 4,096 bytes of information.

How does NFC work?

Just like Bluetooth and Wi-Fi, and all manner of other wireless signals, NFC works on the principle of sending information over radio waves. he technology used in NFC is based on older RFID (Radio-frequency identification) ideas, which used electromagnetic induction in order to transmit information. The former can be used to induce electric currents within passive components as well as just send data. This means that passive devices don't require their own power supply. They can instead be powered by the electromagnetic field produced by an active NFC component when it comes into range.

To determine what sort of information will be exchanged between devices, the NFC standard currently has three distinct modes of operation. Perhaps the most common use in smartphones is the peer-to-peer mode. This allows two NFC-enabled devices to exchange various pieces of information between each other. In this mode, both devices switch between active when sending data and passive when receiving.

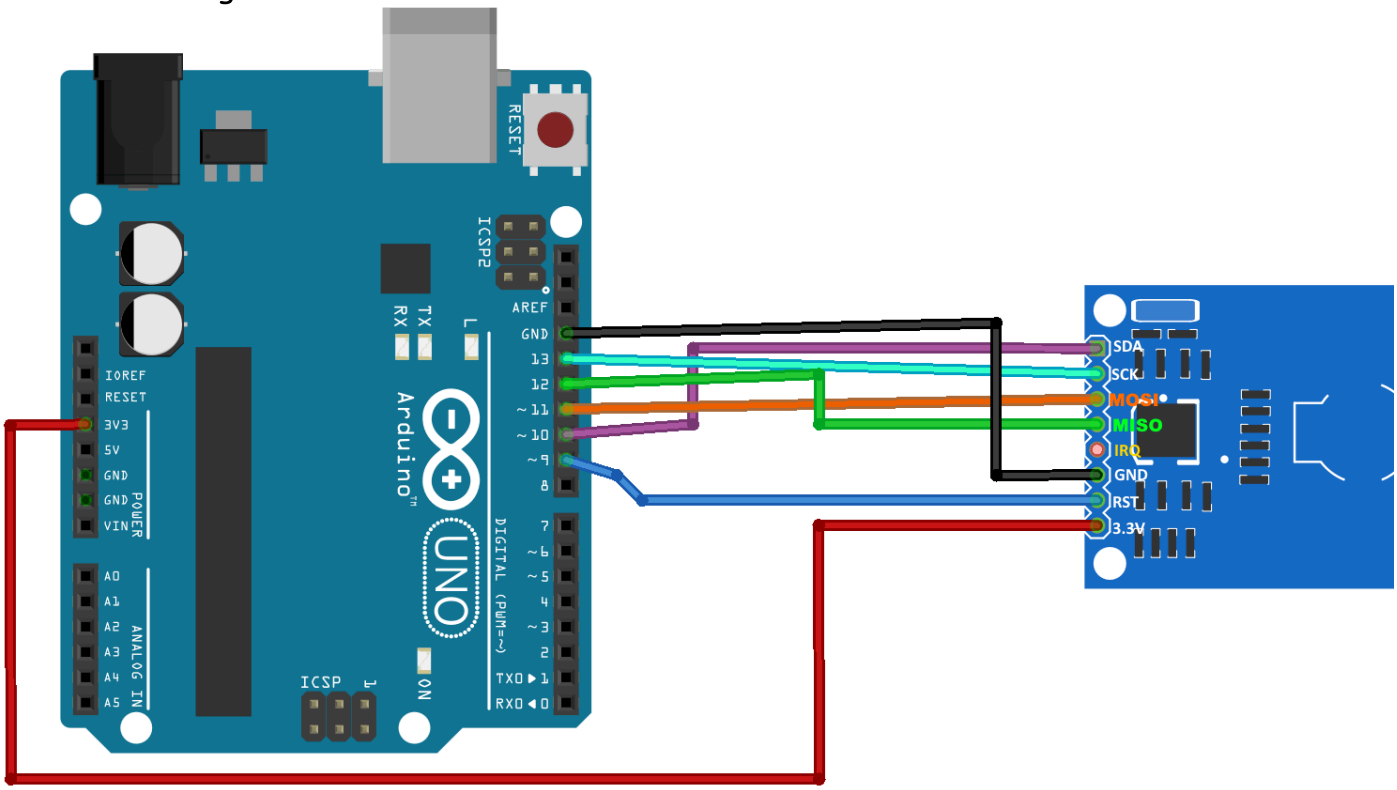
Read/write mode, on the other hand, is one-way data transmission. The active device, possibly your smartphone, links up with another device in order to read information from it. NFC advert tags use this mode.

The final mode of operation is card emulation. The NFC device can function as a smart or contactless credit card and make payments or tap into public transport systems.

Writing Data to the RC522 Card (multiple blocks)

Let s write to multiple blocks

Connection Diagram.



Code:

```
#include <SPI.h>
#include <MFRC522.h>
#define RST_PIN  9
#define SS_PIN  10
MFRC522 mfrc522(SS_PIN, RST_PIN);
MFRC522::MIFARE_Key key;
MFRC522::StatusCode card_status;

void setup(){
  Serial.begin(9600);
  SPI.begin();
  mfrc522.PCD_Init();
  Serial.println(F("Enter data, ending with #"));
  Serial.println("");
}


void loop(){
  for(byte i = 0; i < 6; i++){
    key.keyByte[i] = 0xFF;
  }

  if(!mfrc522.PICC_IsNewCardPresent()){
    return;
  }

  if (!mfrc522.PICC_ReadCardSerial()){
    return;
  }

  byte buffr[34];

  byte block = 4;
  byte len;
```

```
  /
   * Read data from serial
   */

  len = Serial.readBytesUntil('#', (char *) buffr, 30) ;

  if(len>0){
    for(byte i = len; i < 30; i++){
      buffr[i] = ' ';     //We have to pad array items with spaces.
    }
    Serial.println(" ");
    writeBytesToBlock(block, buffr);
    Serial.println(" ");
    mfrc522.PICC_HaltA();
    mfrc522.PCD_StopCrypto1();
    Serial.println("Note: To rewrite to this PICC, take it away from the PCD, and bring it closer again.");
  }
  else{
    delay(500);
    Serial.print(".");
  }
}


void writeBytesToBlock(byte block, byte buff[]){
  card_status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block, &key, &(mfrc522.uid));

  if(card_status != MFRC522::STATUS_OK) {
    Serial.print(F("PCD_Authenticate() failed: "));
    Serial.println(mfrc522.GetStatusCodeName(card_status));
    return;
  }

  else{
    Serial.println(F("PCD_Authenticate() success: "));
  }
  // Write block
  card_status = mfrc522.MIFARE_Write(block, buff, 16);

  if (card_status != MFRC522::STATUS_OK) {
    Serial.print(F("MIFARE_Write() failed: "));
    Serial.println(mfrc522.GetStatusCodeName(card_status));
    return;
  }
  else{
    Serial.println(F("Data saved."));
  }
}
```

Let's explain what the code does:
We started by importing SPI and MFRC522 libraries
Then we instantiated MFRC522 class.
From MFRC522 we imported MIFARE_Key as key
Also from MFRC522 we imported StatusCode as card_status
Amongst all the pins of RC522 Card Reader/Write only two pins need to be defined.
Those are RESET connected to digital pin 9 of the Arduino board and SDA pin connected to digital pin 10.
Remember, for Arduino board digital pin 10 serves as Slave Select (SS) or Chip Select (CS).
Take note that though digital pin 9 of the Arduino board is the one used for RESET in this code, any digital or analog pin apart from D13, D12, D11, D10 which serve as as Serial Clock (SCK or SCL), MISO, MOSI, and SS respectively.

Within the function setup():
- Serial Communication (UART) with the computer is initialized

- MFRC522 card is initialized
- A heading is printed on the serial monitor: F("WRITING DATA ON RC522 CARD"). F() is used in order to move constant strings to the program memory (or flash memory) instead of the RAM. That will free up dynamic RAM though it will take up space that will decrease the amount of other code we can write. Flash memory is an electronic non-volatile computer memory storage medium that can be electrically erased and reprogrammed.

Saving RFID Transaction to an SD Card [No RTC]

In this lesson let's write to an SD card the logs from RFID card usage.
Given that both RFID reader and SD Card module are SPI devices connected to a single Arduino board, we are dealing with the case of an "SPI master" connected with two "SPI slaves".

Let's take a time and observe how two SPI slaves RFID reader and SD card module are connected to a common master Arduino UNO board. When the SPI master has multiple slave select pins, the slaves can be wired in parallel like in the picture below.

For Arduino UNO though digital pin 10 is by default mentioned as CS/SS (chip select/slave select) pin any GPIO pin can be used for slave selection. So, when multiple SPI devices are connected to an Arduino UNO board, every SPI device connected should have its own CS/CC pin. In the current example the RFID reader is set to use digital pin 10 as SS while the SD Card module is set to use 4 as SS.

MFRC522 RFID reader pinout

Pin	Wiring to Arduino UNO
SDA	Digital 10
SCK	Digital 13
MOSI	Digital 11
MISO	Digital 12
IRQ	Not connected
GND	GND
RST	Digital 9

SD card module pinout

No Reset Pin Present

Pin	Wiring to Arduino UNO
VCC	3.3V
CS	Digital 4
MOSI	Digital 11
CLK	Digital 13
MISO	Digital 12
GND	GND

SD Library

Get the library from *this link*.

Code

In the code below we're going to write to SD Card the records of transactions done using RFID card.

```
#include <SPI.h>
#include <MFRC522.h>
#include <SD.h>

#define RST_PIN 9
#define SS_PIN 10

#define SD_CHIP_SELECT 4

MFRC522 mfrc522(SS_PIN, RST_PIN);
MFRC522::MIFARE_Key key;
MFRC522::StatusCode card_status;

File myFile;

void setup(){
  Serial.begin(9600);
  SPI.begin();
  Serial.println("Initializing SD card...");
  if (!SD.begin(SD_CHIP_SELECT)) {
    Serial.println("SD card initialization failed.");
```

```
  Serial.println("SD card initialized.\n");
  Serial.println(F("DEVICE READY FOR TRANSACTION:"));
}
void loop(){
  byte block_number = 4;
  byte buffer_for_reading[18];
  for (byte i = 0; i < 6; i++){
    key.keyByte[i] = 0xFF;
  }
  if(!mfrc522.PICC_IsNewCardPresent()){
    return;
  }
  if(!mfrc522.PICC_ReadCardSerial()){
    return;
  }
  String initial_balance = readBalanceFromCard(block_number, buffer_for_reading);
  operateData(block_number, initial_balance);
  mfrc522.PICC_HaltA();
  mfrc522.PCD_StopCrypto1();
}
String readBalanceFromCard(byte blockNumber, byte readingBuffer[]){
  card_status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, 4, &key, &(mfrc522.uid)
  if(card_status != MFRC522::STATUS_OK){
    Serial.print(F("Authentication failed: "));
    Serial.println(mfrc522.GetStatusCodeName(card_status));
    return;
  }
  byte readDataLength = 18;
  card_status = mfrc522.MIFARE_Read(blockNumber, readingBuffer, &readDataLength);
  if(card_status != MFRC522::STATUS_OK){
    Serial.print(F("Reading failed: "));
    Serial.println(mfrc522.GetStatusCodeName(card_status));
    return;
  }

  String value = "";
  for (uint8_t i = 0; i < 16; i++){
    value += (char)readingBuffer[i];
  }
  value.trim();
  return value;
}
bool saveBalanceToCard(byte blockNumber, byte writingBuffer[]){
  card_status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, blockNumber, &key, &(mf
  if(card_status != MFRC522::STATUS_OK){
    Serial.print(F("PCD_Authenticate() failed: "));
    Serial.println(mfrc522.GetStatusCodeName(card_status));
    return;
  }
  else{
    //Serial.println(F("PCD_Authenticate() success: "));
  }
  // Write block
  card_status = mfrc522.MIFARE_Write(blockNumber, writingBuffer, 16);
  if(card_status != MFRC522::STATUS_OK){
    Serial.print(F("MIFARE_Write() failed: "));
```

```
  }
  else{
    //Serial.println(F("Data saved."));
    delay(5000);
    return true;
  }
}

void operateData(byte blockNumber, String initialBalance){
  int fareToKimironko = 450;
  float newBalance = initialBalance.toInt()-fareToKimironko;

  if(initialBalance.toInt()<fareToKimironko){
    Serial.print("Insufficient Balance: ");
    Serial.println(initialBalance);
    return;
  }

  String initial_balance_str;
  char writingBuffer[16];
  initial_balance_str = (String)newBalance;
  initial_balance_str.toCharArray(writingBuffer, 16);
  int strLeng = initial_balance_str.length()-3;
  /*
   * This servers to add spaces to the typed text in order to fill up to 16 characters
  */

  for(byte i = strLeng; i < 30; i++){
    writingBuffer[i] = ' ';
  }

  Serial.println("\n********************");
  Serial.println("Processing...");
  if(saveBalanceToCard(blockNumber, writingBuffer)==true){
    Serial.print("Initial Balance: ");
    Serial.println(initialBalance);
    Serial.print("Fare to Kimironko: ");
    Serial.println(fareToKimironko);
    Serial.print("New Balance: ");
    Serial.println(newBalance);
    if(writeToSDCard((String)initialBalance, (String)fareToKimironko, (String)newBalance)){
      Serial.println("Transaction Succeeded.\n");
    }
    else{
      Serial.println("Transaction Succeeded BUT data was not saved on SDCard.\n");
    }
  }
  else{
    Serial.print("Transaction Failed.\nPlease try again.\n");
  }
  Serial.println("********************\n");
}

bool writeToSDCard(String initial_balance, String fare, String new_balance){
  myFile = SD.open("ops.txt", FILE_WRITE);
  if(myFile){
```

```
    myFile.println(initial_balance);
    myFile.print("Fare to Kimironko: ");
    myFile.println(fare);
    myFile.print("New Balance: ");
    myFile.println(new_balance);
    myFile.println("********************\n");
    myFile.close();
    return true;
  }else{
    Serial.println("error opening ops.txt");
    return false;
  }
}
```

How the code above works

The SD library provides useful functions for easily write in and read from the SD card. To write and read from the SD card, first you need to include the SPI and SD libraries:

```
#include <SPI.h>
#include <SD.h>
# define SD_CHIP_SELECT 4
```

You also have to initialize the SD card module at the Chip Select (CS) pin – in our case, pin 4.

```
SD.begin(SD_CHIP_SELECT);
```

To open a new file in the SD card, you need to create a file object that refers to your data file.

```
myFile = SD.open("data.txt", FILE_WRITE);
```

The first parameter of this function is the name of the file, data.txt, and the second parameter FILE_WRITE enables you to read and write into the file. This line of code creates a file called data.txt on your SD card. If the data.txt file already exists, Arduino will open the file instead of creating another one.

To write data to the currently open file, you use:

```
myFile.write(data);
```

In which the dataFile is the file object created previously and the data is what you want to write in the file.

You can also use the print() or println() functions to print data into the file:

```
dataFile.print(data);
dataFile.println(data); // followed by a new line
```

Develop a Handheld Cash In Device

Developing a Handheld Cash-In Device for recharging PICC (RFID cards) involves integrating several components to create a functional, portable device that an operator (agent) can use to add value to RFID cards using cash from customers. Here's a detailed explanation of the components, their roles, and the steps involved in the project:

Components Overview

Keypad:

LCD Display with I2C:
- Displays instructions, amounts, and messages.
- Requires only two data lines (SDA, SCL).

Arduino UNO:
- The main controller for the device.

Wires:
- For connecting components.

PCD (RFID Reader/Writer):
- For interacting with RFID cards (PICC).

PICC (RFID Card):
- For testing and usage in transactions.

System Design and Integration with I2C LCD

1. Keypad Integration

- **Connections**:
- Connect the keypad to digital pins on the Arduino.
- **Library**:
- Use the Keypad.h library.

2. I2C LCD Display Setup

- **Connections**:
- - Connect the I2C LCD to the Arduino using the SDA and SCL pins.
- - On the Arduino UNO, SDA is on A4, and SCL is on A5. There are also dedicated pins after AREF pin on the side of digital pins.
- **Library**:
- Use the LiquidCrystal_I2C.h library for managing the I2C LCD.

3. RFID Reader/Writer Configuration

- **Connections**:
- Connect the MFRC522 RFID module to the Arduino using SPI pins.
- **Library**:
- Use the MFRC522.h library.

4. Arduino Code Development

Sample code:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Keypad.h>
#include <SPI.h>
#include <MFRC522.h>

#define SS_PIN 10
#define RST_PIN 9

MFRC522 rfid(SS_PIN, RST_PIN);
LiquidCrystal_I2C lcd(0x27, 16, 2); // Adjust address if needed
```

```
char keys[ROWS][COLS] = {
 {'1','2','3','A'},
 {'4','5','6','B'},
 {'7','8','9','C'},
 {'*','0','#','D'}
};
byte rowPins[ROWS] = {9, 8, 7, 6}; // Connect to the row pinouts of the keypad
byte colPins[COLS] = {5, 4, 3, 2}; // Connect to the column pinouts of the keypad

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

String enteredAmount = "";

void setup() {
 lcd.init();
 lcd.backlight();
 SPI.begin();
 rfid.PCD_Init();
 lcd.setCursor(0, 0);
 lcd.print("Enter Amount:");
}

void loop() {
 char key = keypad.getKey();
 if (key) {
   if (key == '#') { // Assuming '#' is the confirmation key
     if (enteredAmount.length() > 0) {
       rechargeCard();
       enteredAmount = "";
       lcd.clear();
       lcd.setCursor(0, 0);
       lcd.print("Enter Amount:");
     }
   } else if (key == '*') { // Assuming '*' is used to clear input
     enteredAmount = "";
     lcd.clear();
     lcd.setCursor(0, 0);
     lcd.print("Enter Amount:");
   } else {
     enteredAmount += key;
     lcd.setCursor(0, 1);
     lcd.print(enteredAmount);
   }
 }
}

void rechargeCard() {
 lcd.clear();
 lcd.setCursor(0, 0);
 lcd.print("Recharging...");
 // RFID write logic here
 // For example:
 if (rfid.PICC_IsNewCardPresent() && rfid.PICC_ReadCardSerial()) {
   // Write the enteredAmount to the card
   // Use rfid.MIFARE_Write() or similar function to write data to the card
 }
 lcd.clear();
 lcd.setCursor(0, 0);
 lcd.print("Recharge Done");
 delay(2000);
}
```

Testing and Debugging

- **Initial Tests**:
- Ensure the keypad inputs are correctly registered and displayed on the I2C LCD.
- Verify the RFID reader/writer can read from and write to the RFID card.
- Test the entire flow: input amount, confirm, write to card, and display confirmation.
- **Error Handling**:
- Implement error messages for invalid card operations.
- Handle edge cases like entering invalid amounts or faulty hardware.

Final Steps

- **Encasing**:
- Design an ergonomic case to house all components