NFC Shield

Introduction

NFC Shield is a Near Field Communication interface for Arduino build around the popular NXP PN532 integrated circuit. NFC is a short-distance radio technology that enables communication between devices that are held close together. NFC traces its roots in RFID technology and is an open platform technology standardized in ECMA-340 and ISO/IEC 18092.

NFC is widely used like RFID to recognize cards/tags (NXP Mifare Cards / Tags). NFC can be used as an alternative to travel card using the read/write memory provided by cards/tags. Few mobile phones comes with NFC inbuilt - they are used as readers of cards, tags, smart posters with a Web URL(like a Mobile QR-Code reader). This technology is also being applied for smart cashless purchases.

Like many other standards, NFC technology is regulated by Near Field Communication Forum which standardizes NFC communication -- how they devices pair, share data and allow a secure transaction to happen.NFC Forum develops and certifies devices compliant with NFC standards.

NFC operate on unlicensed ISM(Industry Scientific Medical) band of 13.56 MHz Frequency. NFC communication range is up to 10 cm. But, this is limited by the antenna and power radiation design. Most devices work within a range of 10mm. NFC Shield antenna is designed to work within a range of 1cm. NFC Shield provides all necessary circuitry for PN532 like 27.12Mhz crystal, power supply.It also beaks-out the I/O pins of PN532 for easy access.

The communication between Arduino and NFC Shield is via SPI.

Features

• Arduino Shield compatible. No soldering required.

• SPI interface. Hence, most Arduino pins are available for other applications.

• Built in PCB Antenna.

• Supports both 3.3V and 5V operation using TI's TXB0104 level translator.

• Socket to connect other shields.

• The maximum communication range of this NFC Shield is about 5 cm

Application Ideas

• Use as a RFID reader with Mifare One tags (ISO14443 Type-A) and cards (13.56Mhz).

• Build visiting card sharing system.

• Build attendance systems.

• Design authentication systems.

• Read Smart Posters.

• Securely exchange small data with other NFC devices

• Use with Seeeduino ADK Main Board for creating mobile NFC applications.

• And other endless possibility.

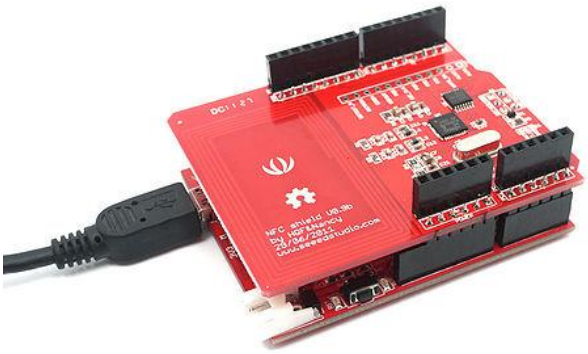
Usage

Hardware Installation

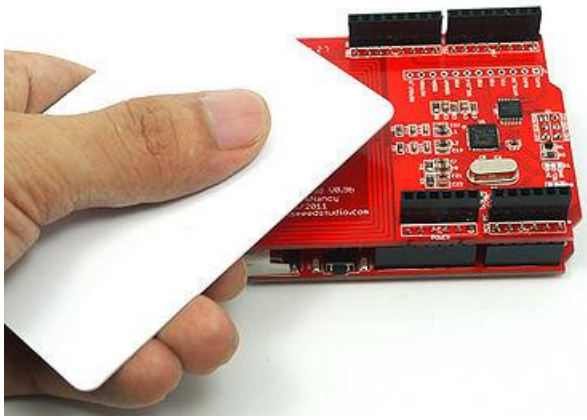
• Set Seeeduino power selection slide-switch to 3.3V.

• Connect NFC Shield to Seeeduino as shown below.

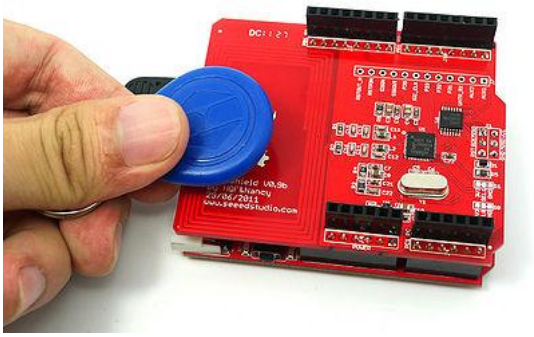
• Compile and upload the example sketch provided.



• Hold the MIFARE Card near the antenna. The NFC Shield will read the passive id data.



• Hold the MIFARE Tag near the antenna. The NFC Shield will read the passive id data.



Programming

The PN532 software library for NFC Shield is derived from Adafruits PN532 Library.The original library provides API for reading Passive Target ID of Mifare Card/Tags. This is enough for card/tag identification purpose. We have added APIs for authentication, reading from and writing to Mifare Cards/Tags. The software library only provides low level functionality. Users have to implement NFC application layer(if required).

Please Note: Arduino 1.0 users have to change the #include <WProgram.h> lines to #include <Arduino.h> in PN532.cpp and PN532.h.

Quick Start Demo

A simple sketch which reads the Passive Target ID from MIFARE cards and tags. Passive Target ID is an unique, permanent and read-only number programmed on to the MIFARE card by the manufacturer. This number is used to identify one card from another.

• Connect the NFC Shield to Seeeduino / Arduino as shown above.

• Compile and upload the program to Arduino.

• Bring a Mifare Card near the NFC Antenna as shown above.

```
#include <PN532.h>

/*
 * Corrected MISO/MOSI/SCK for Mega from Jonathan Hogg (www.jonathanhogg.com)
 * SS is the same, due to NFC Shield schematic
 */
#define SS 10
#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
```

```
  #define MISO 50
  #define MOSI 51
  #define SCK 52
#else
  #define MISO 12
  #define MOSI 11
  #define SCK 13
#endif

PN532 nfc(SCK, MISO, MOSI, SS);

void setup(void) {
  Serial.begin(9600);

  nfc.begin();

  uint32_t versiondata = nfc.getFirmwareVersion();
  if (! versiondata) {
    Serial.print("Didn't find PN53x board");
    while (1); // halt
  }
  // Got ok data, print it out!
  Serial.print("Found chip PN5"); Serial.println((versiondata>>24) & 0xFF, HEX);
  Serial.print("Firmware ver. "); Serial.print((versiondata>>16) & 0xFF, DEC);
  Serial.print('.'); Serial.println((versiondata>>8) & 0xFF, DEC);
```

```
  Serial.print("Supports "); Serial.println(versiondata & 0xFF, HEX);


  // configure board to read RFID tags and cards
  nfc.SAMConfig();
}



void loop(void) {
  uint32_t id;
  // look for MiFare type cards
  id = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A);


  if (id != 0) {
    Serial.print("Read card #"); Serial.println(id);
  }
}
```

Application Programming Interfaces

NFC is a secure technology (Meaning: Communication between NFC reader/writer and NFC card/tag happens in a encrypted and authenticated manner). The security and other complex handshaking are handled by PN532 firmware provided by NXP.

The APIs make use of the commands to invoke the interfaces provided by PN532 firmware via SPI. All these commands are documented in PN532 User Manual. The following APIs are provided by PN532 Library.

PN532(uint8_t cs, uint8_t clk, uint8_t mosi, uint8_t miso)

An object of PN532() is created with this. The digital pins of Arduino used as SPI (in AtMega328P or Mega) is specified as parameters.

Usage:

```
#define SCK 13

#define MOSI 11

#define SS 10

#define MISO 12


PN532 nfc(SCK, MISO, MOSI, SS);
```

begin()

begin() method has to be called to initialize the driver.

Usage:

```
nfc.begin();
```

boolean SAMConfig(void)

This API invokes the SAMConfiguration command of PN532 and sets it to Normal Mode. SAM stands for Security Access Module (i.e the PN532 system). PN532 system can work in Normal mode, Virtual Card mode, Wired Card mode and Dual Card mode.

Usage:

```
nfc.SAMConfig(); // Call this before any read/write operation
```

uint32_t readPassiveTargetID(uint8_t cardbaudrate)

This method reads the Passive Target ID and returns it as a 32-bit number. At the moment only reading MIFARE ISO14443A cards/tags are supported. Hence use PN532_MIFARE_ISO14443A as parameter. Returns 32 bit card number

Usage:

```
uint32_t cid;

// look for MiFare type cards/tags

cid = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A);
```

uint32_t authenticateBlock(uint8_t cardnumber, uint32_t cid, uint8_t blockaddress ,uint8_t authtype, uint8_t * keys)

This method is used to authenticate a memory block with key before read/write operation. Returns true when successful.

- cardnumber can be 1 or 2
- cid is 32-bit Card ID
- blockaddress is block number (any number between 0 - 63 for MIFARE card)
- authtype is which key is to be used for authentication (either KEY_A or KEY_B)
- keys points to the byte-array holding 6 keys.

Usage:

```
uint8_t keys[]= {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF};  // default key of a fresh card

nfc.authenticateBlock(1, id ,3,KEY_A,keys); ////authenticate block 3, id is 32-bit passive target id.
```

uint32_t readMemoryBlock(uint8_t cardnumber,uint8_t blockaddress, uint8_t * block)

This method reads a memory block after authentication with the key. Returns true when successful.

• cardnumber can be 1 or 2

• blockaddress is block number (any number between 0 - 63 for MIFARE card) to read. Each • block is 16bytes long in case of MIFARE Standard card.

• block points to buffer(byte-array)to hold 16 bytes of block-data.

Usage:

uint8_t block[16];

nfc.readMemoryBlock(1,3,block); //Read can be performed only when authentication was successful.

uint32_t writeMemoryBlock(uint8_t cardnumber,uint8_t blockaddress, uint8_t * block)

This method writes data to a memory block after authentication with the key. Returns true when successful.

• cardnumber can be 1 or 2

• blockaddress is block number (any number between 0 - 63 for MIFARE card) to write. Each • block is 16bytes long in case of MIFARE Standard card.

• block points to buffer(byte-array) which holds 16 bytes of block-data to write.

Usage:

```
uint8_t writeBuffer[16];
  for(uint8_t ii=0;ii<16;ii++)
   {
     writeBuffer[ii]=ii; //Fill buffer with 0,1,2....F
   }
```

nfc.writeMemoryBlock(1,0x08,writeBuffer); //Write writeBuffer[] to block address 0x08. Read can be performed only when authentication was successful.

readAllMemoryBlocks.pde

Compile and upload readAllMemoryBlocks.pde example provided with the library. This sketch reads the complete memory of a MIFARE Standard card using default authentication keys. The output gives typical memory layout of fresh MIFARE Standard card.

Blocks are classified as Manufacturer Block(read-only), Data Block (user/application writable area), and Sector Trailer(authentication and access bits for that sector)

Output