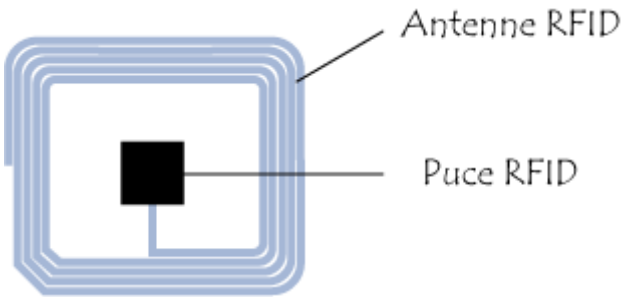
Compte rendu de la soirée du lundi 12 décembre 2016
à L'atelier MJC-Fablab de Château-Renault

Ce soir nous nous sommes intéressés à la technologie **RFID/NFC** dans un premier temps sur l'Arduino et ensuite sur RaspberryPi qui fera l'objet d'une autre soirée plus tard.

Commençons par un peu de théorie , pour savoir ou on met « les mains ».

Définition rapide des mots :

RFID signifie « *Radio Frequency IDentification* », en français, « *Identification par Radio Fréquence* ». Cette technologie permet d'identifier un objet, d'en suivre le cheminement et d'en connaître les caractéristiques à distance grâce à une étiquette émettant des ondes radio, attachée ou incorporée à l'objet. La technologie RFID permet la lecture des étiquettes même sans ligne de vue directe et peut traverser de fines couches de matériaux (peinture, neige, etc.).



On distingue 3 catégories d'étiquettes RFID :

- **Les étiquettes en lecture seule,** non modifiables,
- **Les étiquettes « écriture une fois**, lecture multiple »,
- **Les étiquettes en « lecture réécriture »**.

Par ailleurs, il existe deux grandes familles d'étiquettes RFID :

- **Les étiquettes actives, reliées à une source d'énergie embarqué (pile, batterie, etc.)**. Les étiquettes actives possèdent une meilleure portée mais à un coût plus élevé et avec une durée de vie restreinte,
- **Les étiquettes passives, utilisant l'énergie propagée à courte distance par le signal radio de l'émetteur.** Ces étiquettes à moindre coût sont généralement plus petites et possèdent une durée de vie quasi-illimitée. En contrepartie, elles nécessitent une quantité d'énergie non négligeable de la part du lecteur pour pouvoir fonctionner.

NFC :*Near Field Communication* en français La **communication en champ proche** est une technologie de communication sans fil à courte portée et à haute fréquence, permettant l'échange d'informations entre des périphériques jusqu'à une distance d'environ 10 cm dans le cas général.
Cette technologie est une extension de la norme ISO/CEI 14443 standardisant les cartes de proximité utilisant la radio-identification (RFID), qui combinent une carte à puce et un lecteur au sein d'un seul périphérique

les normes :

Basse fréquence ou LF (Low Frequency) : **à 125 ou 134 Khz ;**

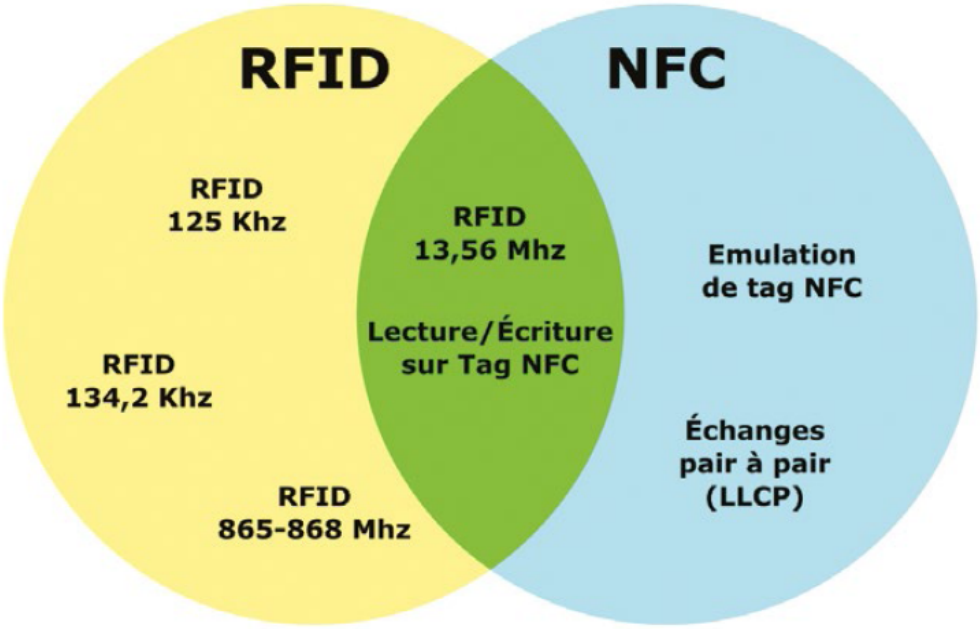
- Haute fréquence ou HF (High Frequency) : généralement **13,56 Mhz ;**

- Ultra-haute fréquence ou UHF (Ultra High Frequency) :
  865 MHz à 868 MHz (CE), et tantôt 2,4 Ghz ;
- Supra-haute fréquence ou SHF (Super High Frequency) :
  5,8 GHz (généralement pour les applications nécessitant une portée importante).

Les mots « **NFC** » et « **RFID** » ne sont pas interchangeables. Utiliser maladroitement l'un à la place de l'autre est tout à fait comparable au fait de confondre CD et DVD. .

La technologie NFC pour Near Field Communication ressemble au RFID, mais n'en est pas, car encadrée par d'autres standards et normes (regroupés sous les désignations **ISO/IEC 14443-1 à ISO/IEC 14443-4).**
Ces normes décrivent quelque chose s'approchant du RFID, mais imposent un certain nombre de restrictions tout en couvrant des points ignorés par les normes RFID. Ceci concerne les caractéristiques physiques (14443-1), l'interface d'alimentation et de gestion radiofréquence (14443-2), l'initialisation et l'anticollision (14443-3) et enfin le protocole de transmission (14443-4).

Un petit schéma vaut mieux qu'un long discours …



Principes :

Ces fréquences sont utilisées pour deux choses : transmettre des informations et alimenter ce que nous appellerons désormais le tag RFID ou simplement le tag.

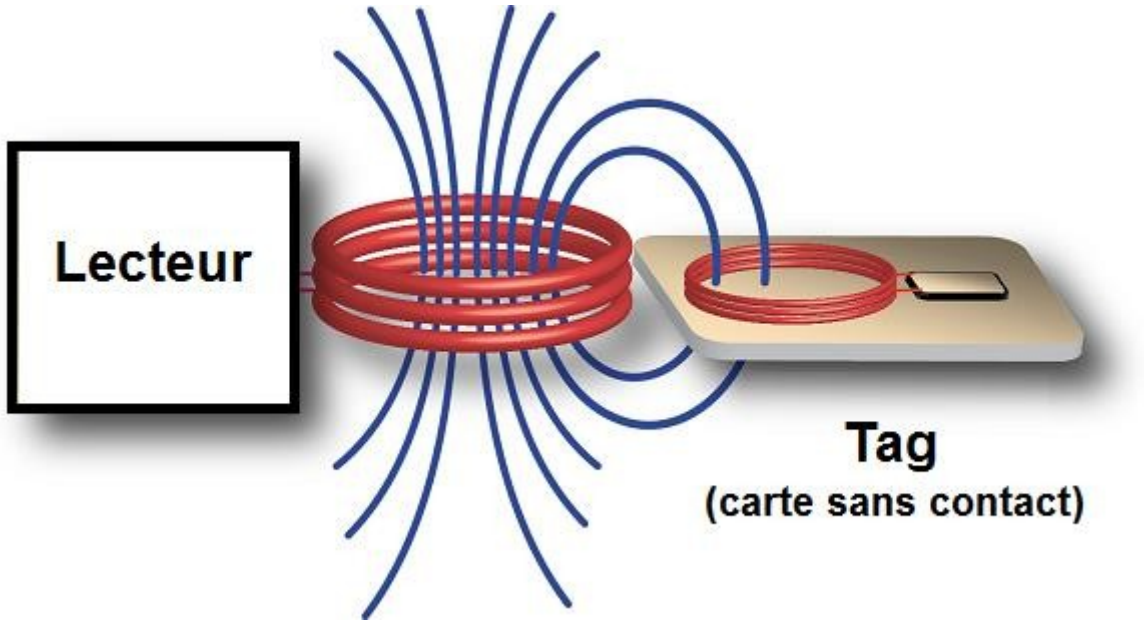
Ceux-ci sont constitués de deux principaux éléments : une antenne et une puce électronique . La puce est alimentée par induction électromagnétique selon un principe connu depuis plusieurs générations.

Prenez une bobine, faites-y passer un courant alternatif et placez une autre bobine à proximité. Le champ magnétique apparaissant et disparaissant dans la première bobine va induire un courant alternatif dans la seconde. C'est exactement le même principe de fonctionnement qu'un transformateur, à la différence que les deux bobines ne sont pas fixes et que le champ magnétique s'étend dans l'air et non dans une carcasse (ou tôles feuilletées). Ce principe qui est utilisé avec les smartphones modernes proposant un système de recharge sans fil.

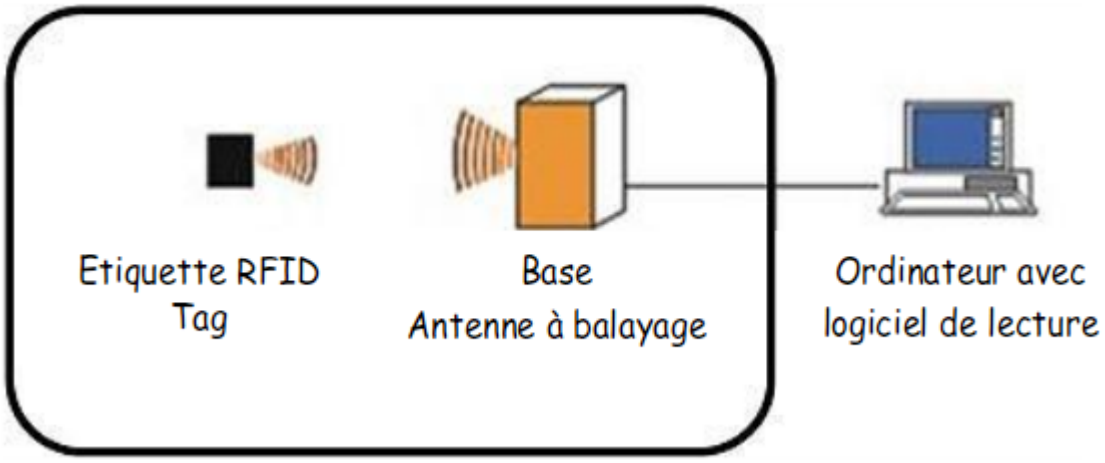
L'énergie ainsi transmise, qui est modulée par l'émetteur, permet d'alimenter une puce dédiée à cet usage qui en retour va pouvoir répondre en réfléchissant une partie du rayonnement électromagnétique (phénomène de rétrodiffusion) ou en l'absorbant. Nous avons donc une communication bidirectionnelle, half duplex (HDX) ou réception puis envoi pour le tag, et full duplex ou envoi et réception simultanés pour l'émetteur, qu'on désigne plutôt par le terme PCD pour Proximity Coupling Device (le tag étant dans la nomenclature un PICC ou Proximity Integrated Circuit Card).

Un tag est donc un véritable concentré de technologie reposant sur des décennies de recherches et d'expérimentations et bien plus qu'une simple puce dans une petite étiquette, une carte ou un porte-clés. En dehors de l'aspect énergétique, cette technologie doit également prendre en compte un problème majeur que l'on retrouve dans la documentation désigné par le terme « collision ».

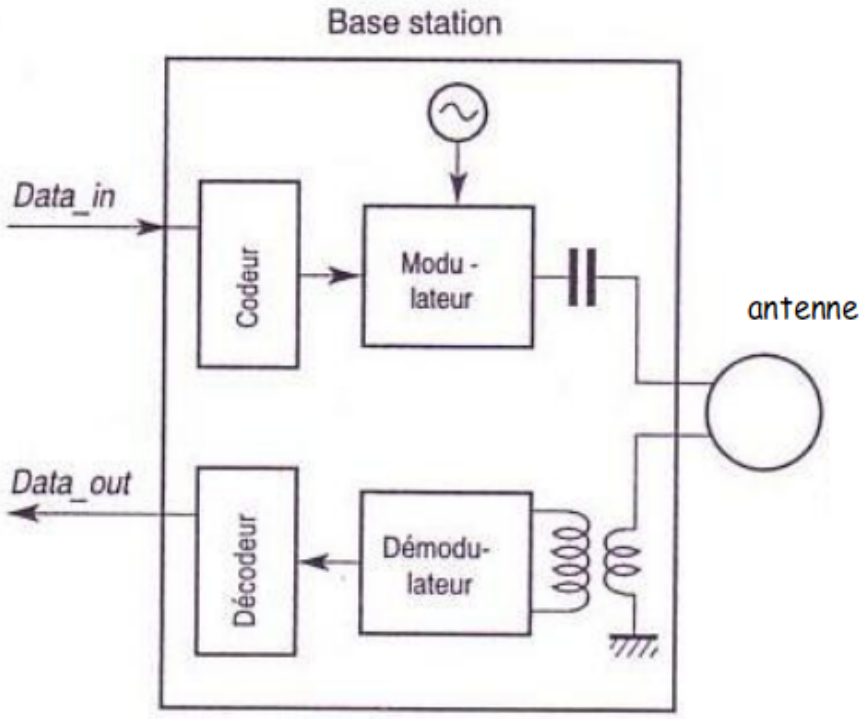
En effet, lorsque plusieurs tags (PICC) se trouvent dans le champ du lecteur (PCD) les communications se mélangent et des techniques particulières doivent être utilisées pour contourner ces effets : les algorithmes d'anti-collision. Les fréquences, les techniques de modulation de signaux, les algorithmes de communication et d'anti-collision et bien d'autres choses encore, forment ensemble une série de normes et de standards. C'est cet ensemble qui est désigné sous le terme RFID.

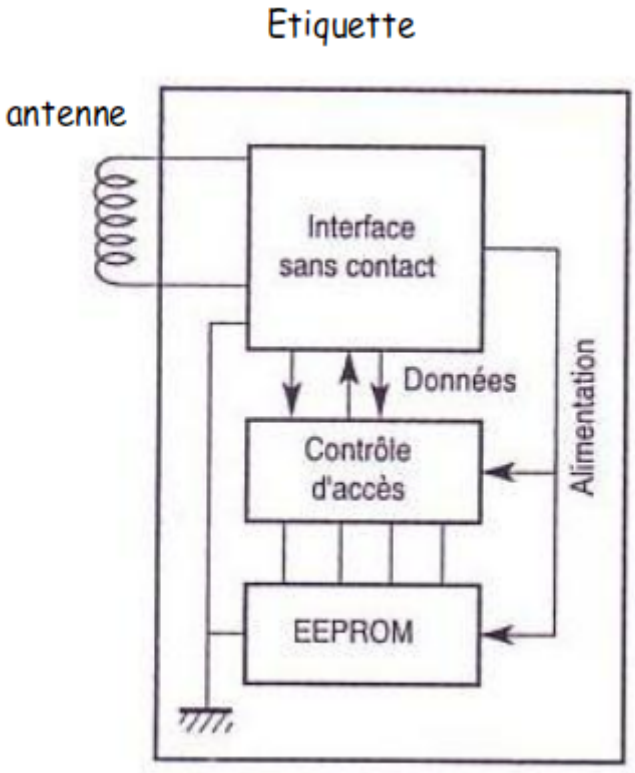


Le système d'identification comprend une base et une étiquette



Constitution de la partie RF de la station de base :



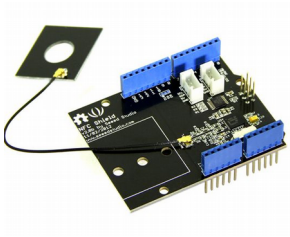


<u>Après la théorie, passons à la pratique :</u>

Le matériel :

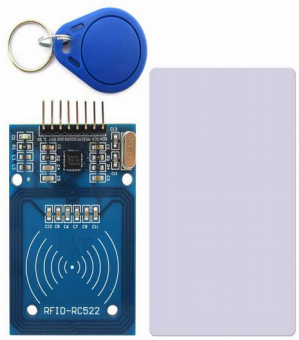
1 Arduino Uno ou Arduino Mega

1 Lecteur NFC Shield NFC V2.0 de chez Gotronic connexion SPI



Alimentation: 5 Vcc (3,7 à 5,5 Vcc)
Consommation: 150 mA maxi
Fréquence: 13,56 MHz
Antenne déportée
Connecteur pour le raccordement d'autres Shields
Portée de communication: 10 cm
Interface **SPI disponible**
Supporte les protocoles ISO14443 types A et B
Dimensions:
- module: 70 x 56 x 20 mm
- antenne: 28 x 30 mm

Ou 1 Module Lecteur RFID NFC - RC522 connexion SPI pouvant servir avec le raspberryPi



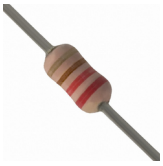
TThis is based on RF module RC522 near field communication module. Its working frequency is 13.56MHz and you can read and write a 13.56MHz tag with this module. Features: - Standard shield interface, perfectly compatible with Arduino UNO / MEGA2560. - Very easy to use with the library for Arduino: RFIDMaster. - Operating voltage: 4.5~5.2V. - Interface logic level: Onboard level converter chip, compatible with 3.3V / 5V. - Operating current: 13-~26mA. - Idle current: 10~13mA. - Sleep current: - Peak current: - Operating Frequency: 13.56MHz. - Supported card types: mifare1 S50, mifare1 S70 MIFARE Ultralight, mifare Pro, MIFARE DESFire. - Data transfer rate: maximum 10Mbit/s. - Application: Smart home project, Voice broadcast access control system, Attendance punch card system.

1 ou plusieurs badges NFC (13,56 MHz)

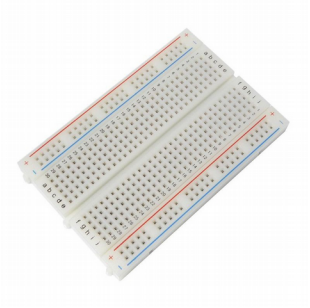
1 platine Relais

1 LED rouge , ou verte, ou bleue, ou blanche

1 résistance 220 Ohms rouge rouge marron



1 plaque Breadboard



Des fils de connexions

Le schéma de connexion pour le Shield NFC V2.0

Connexions SPI de la carte Arduino

Sur la carte Arduino UNO, Duemilanove et autres cartes basées sur les ATmega 168 / 328, le bus SPI utilise les broches :

- **10 : /SS ou RST**
- 11 : MOSI
- 12 : MISO
- et 13 : SCLK.

Sur l'Arduino Mega, ce sont les broches :

- 50 : MISO
- 51 : MOSI
- 52 : SCLK
- **et 53 : /SS ou RST**

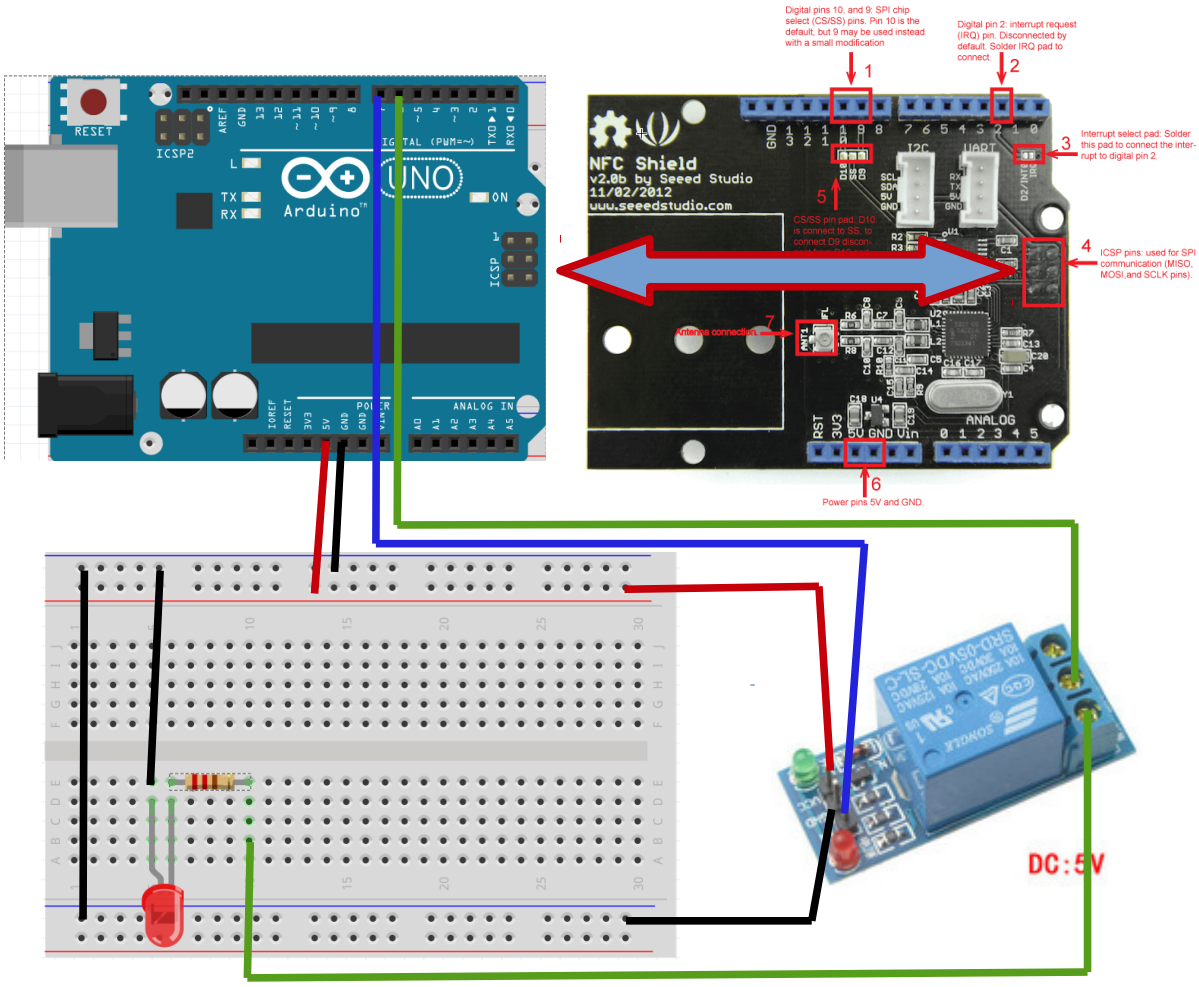
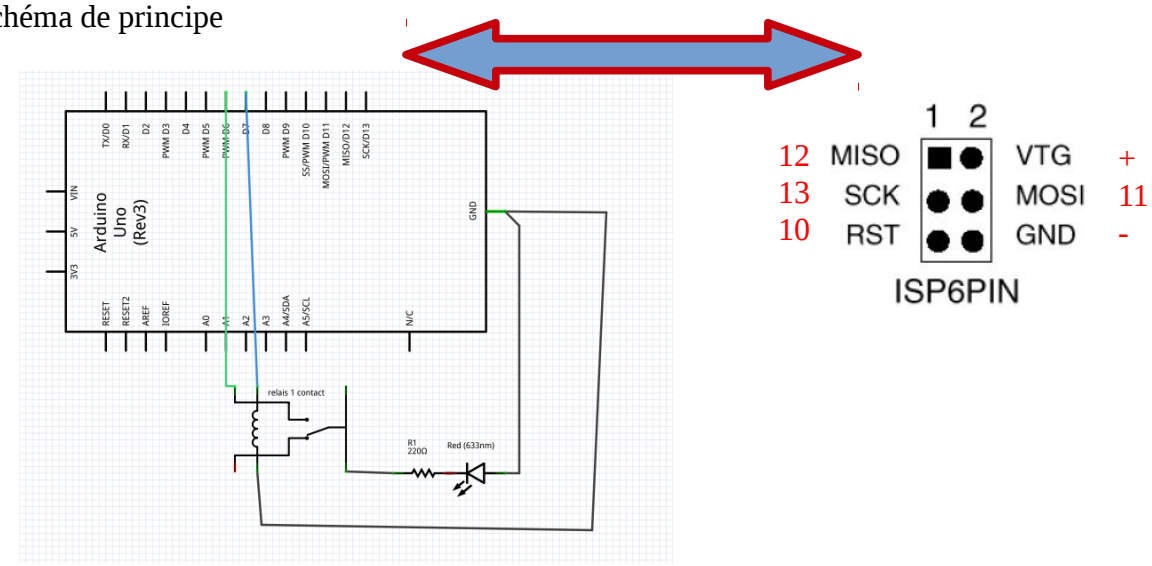


schéma de principe



L IDE Arduino et les librairies à utiliser :

On va considérer que l'on utilise la dernière version de Arduino (1.8.0. au 03/01/2017)

Pour les librairies sur le Shield NFC V2.0

Fermer l'IDE Arduino si vous utiliser le mode extraction des fichiers ZIP (ancienne méthode)
Télécharger PN532 library ZIP dans un dossier et l'extraire ou l'installer via l'installateur de l'arduino
Si vous avez chois le mode extraction copier les répertoires PN532, PN532_HSU, PN532_SPI,et PN532_I2C dans le dossier “librairies” de l'Arduino
Télécharger Don’s NDEF library ZIP dans un dossier et l'extraire ou l'installer via l'installateur de l'arduino
Si vous avez chois le mode extraction , ouvrir le dossier “NDEF-master” et le renommer “NDEF”. Copier “NDEF” dans le répertoire “librairies” de l'arduino.
Redémarrez l'IDE Arduino, vous devez voir les librairies “NDEF_Master” et “PN532_SPI_V1” et dans l'onglet « Fichier --> Exemples » voir les différents programmes d'exemples proposer PN532_SPI_V1 :

readAllMemoryBlocks
readMifareMemory
readMifareTargetID
writeMifareMemory

Pour les librairies sur le Lecteur RFID NFC - RC522

La bibliothèque à utiliser pour prendre en charge le module à base de PN532 n’est pas disponible (pour l’instant) via le gestionnaire de l’IDE Arduino. Il vous faudra vous rendre sur la page https://github.com/don/NDEF
Là, vous y trouverez le résultat du travail de Don Coleman (également co-auteur d’un ouvrage sur NFC chez O’Reilly) et en particulier sa bibliothèque NDEF qui, contrairement à ce que son nom laisse penser, ne se limite pas au traitement des données NDEF, mais à l’ensemble du support NFC à l’exception de la prise en charge directe du matériel.
Cette partie est l’affaire d’une autre bibliothèque, écrite par Yihui Xiong, mais diffusée à présent via le GitHub de Seeed Studio (cette société vend également shields et modules NFC compatibles) : https://github.com/Seeed-Studio/PN532.

Attention, il existe une bibliothèque PN532 dans le gestionnaire de bibliothèques. Celle-ci est

l’oeuvre d’Adafruit pour le support de son propre matériel.

Tâchez simplement de ne pas mélanger les deux supports PN532.

L’installation des bibliothèques est relativement simple, même si elle doit être faite manuellement et qu’il existe quelques subtilités. Commençons par le support PN532. Depuis la page GitHub, récupérez le fichier PN532-master.zip en cliquant sur le bouton Download ZIP en haut à droite. Cette archive contient un répertoire
PN532-master, contenant lui-même les éléments qui nous intéressent. Copiez alors les répertoires PN532, PN532_HSU, PN532_I2C et PN532_SPI, et leur contenu, dans le sous-répertoire libraries de votre dossier correspondant au carnet de croquis (spécifié dans les préférences de l’environnement Arduino).

Allez ensuite sur la page GitHub de la bibliothèque de Don Coleman et récupérez de la même façon le fichier NDEF-master.zip. Celui-ci contient également un répertoire, NDEF-master, possédant les fichiers utiles. Il vous faudra en revanche copier ce répertoire et son contenu dans votre libraries tout en le renommant Ndef.
Une fois toutes ces opérations effectuées, le résultat doit ressembler à ceci :

```
libraries/
PN532/
PN532_HSU/
PN532_I2C/
PN532_SPI/
```

Voilà , après avoir installer les librairies, c'est un gros morceau de fait , car les librairies font presque 80 % du programme.

Pour les programmes , je vous vais vous montrer ce que j'ai fait avec les exemples

readAllMemoryBlocks
readMifareMemory
readMifareTargetID
writeMifareMemory

Pour le Lecteur RFID NFC – RC522, vous essayer les programmes d’exemples, et vous noter ou cela ne va pas et vous me direz ce que vous avez comme résultat.

Pour le premier programme exemple : **readAllMemoryBlocks**
Ce programme permet de lire les 64 blocks de 16 octets ⇒ 64 x 16 = 1024 octets ou 1Ko de la carte NFC ou du badge NFC (13,56Mhz).
Téléversé le programme dans l’Arduino et poser la carte ou le badge sur l’antenne et regarder le résultat dans le terminal de l’Arduino. Cela donne ceci :

```
Hello!
Found chip PN532
Firmware ver. 1.6
Supports 7
Found 1 tags
Sens Response: 0x4
Sel Response: 0x8
 0x84 0x76 0x38 0x89Read card #2222340233

84 76 38 89 43 8  4  0  69 73 73 69 35 36 35 30 | Block  0 | Manufacturer Block
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  | Block  1 | Data Block
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  | Block  2 | Data Block
0  0  0  0  0  0  FF 7  80 69 FF FF FF FF FF FF | Block  3 | Sector Trailer
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  | Block  4 | Data Block
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  | Block  5 | Data Block
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  | Block  6 | Data Block
0  0  0  0  0  0  FF 7  80 69 FF FF FF FF FF FF | Block  7 | Sector Trailer
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  | Block  8 | Data Block
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  | Block  9 | Data Block
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  | Block 10 | Data Block
0  0  0  0  0  0  FF 7  80 69 FF FF FF FF FF FF | Block 11 | Sector Trailer
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  | Block 12 | Data Block
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  | Block 13 | Data Block
```

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 14 | Data Block
0 0 0 0 0 0 FF 7  80 69 FF FF FF FF FF FF | Block 15 | Sector Trailer
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 16 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 17 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 18 | Data Block
0 0 0 0 0 0 FF 7  80 69 FF FF FF FF FF FF | Block 19 | Sector Trailer
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 20 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 21 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 22 | Data Block
0 0 0 0 0 0 FF 7  80 69 FF FF FF FF FF FF | Block 23 | Sector Trailer
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 24 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 25 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 26 | Data Block
0 0 0 0 0 0 FF 7  80 69 FF FF FF FF FF FF | Block 27 | Sector Trailer
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 28 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 29 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 30 | Data Block
0 0 0 0 0 0 FF 7  80 69 FF FF FF FF FF FF | Block 31 | Sector Trailer
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 32 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 33 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 34 | Data Block
0 0 0 0 0 0 FF 7  80 69 FF FF FF FF FF FF | Block 35 | Sector Trailer
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 36 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 37 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 38 | Data Block
0 0 0 0 0 0 FF 7  80 69 FF FF FF FF FF FF | Block 39 | Sector Trailer
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 40 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 41 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 42 | Data Block
0 0 0 0 0 0 FF 7  80 69 FF FF FF FF FF FF | Block 43 | Sector Trailer
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 44 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 45 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 46 | Data Block
0 0 0 0 0 0 FF 7  80 69 FF FF FF FF FF FF | Block 47 | Sector Trailer
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 48 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 49 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 50 | Data Block
0 0 0 0 0 0 FF 7  80 69 FF FF FF FF FF FF | Block 51 | Sector Trailer
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 52 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 53 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 54 | Data Block
0 0 0 0 0 0 FF 7  80 69 FF FF FF FF FF FF | Block 55 | Sector Trailer
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 56 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 57 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 58 | Data Block
0 0 0 0 0 0 FF 7  80 69 FF FF FF FF FF FF | Block 59 | Sector Trailer
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 60 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 61 | Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block 62 | Data Block
0 0 0 0 0 0 FF 7  80 BC FF FF FF FF FF FF | Block 63 | Sector Trailer
```

En fait on n’a pas 1 Ko de disponible si l’on garde la structure à la norme NFC, mais 768 octets.
48 octets x 16 (voir plus bas) moins les données constructeur

l’information :
Read card #2222340233 ⇒ Donne l’identifiant unique de la carte

NFC c’est aussi un format de données :
Les tags Mifare Classic de NXP (ou des cloneurs chinois) sont tout simplement les plus courants, les plus faciles à trouver et souvent les moins chers. Mais ce sont aussi les tags qui ne sont pas 100% compatibles NFC et ne fonctionneront pas avec tous les lecteurs et smartphones.

Les Mifare Classic sont des tags particuliers dont la mémoire est **organisée en secteurs et en blocs. 16 secteurs de 4 blocs de 16 octets pour les 1K** (S50) et 40 secteurs de 4 ou 16 blocs de 16 octets pour les 4K (S70).
Chaque secteur possède un bloc particulier configurant la façon de lire ou d’écrire des données dans le bloc, dépendant de deux clés secrètes (A et B) dans le « sector trailer ».

La mémoire n’est donc pas organisée de façon linéaire et tous les secteurs peuvent être configurés différemment. Ceci pose problème pour les données au format NDEF et impose l’utilisation d’une technique particulière. Le premier secteur (et le 16ème pour une 4K) ne contient pas de donnée « utilisateur », mais un répertoire appelé MAD pour Mifare Application Directory.

Celui-ci référence quels secteurs sont utilisés pour certains types d'applications et donc quels secteurs contiennent ou non des données NDEF. Bien sûr, vous pouvez configurer vos tags Mifare Classic comme il vous plaît, mais le standard précise l'utilisation du MAD si vous voulez utiliser ce type de tags comme compatible NFC. Le MAD, et donc le secteur 0, doit respecter un format précis et pouvoir être lisible avec une clé prédéfinie (0xA0A1A2A3A4A5). Les secteurs dits NDEF, eux aussi, utilisent une clé par défaut 0xD3F7D3F7D3F7 et une configuration par défaut.

Cartes Mifare Classic sont divisés en **secteurs** et des **blocs** de section appelée. Chaque «secteur» a des droits d'accès individuels et contient un nombre fixe de «blocs» qui sont contrôlés par ces droits d'accès. Chaque bloc contient 16 octets et les secteurs contiennent soit 4 blocs (cartes 1K / 4K) pour un total de 64 octets par secteur, soit 16 blocs (cartes 4K uniquement) pour un total de 256 octets par secteur. Les types de cartes sont organisés comme suit:

- **1K Cartes** - 16 secteurs de 4 blocs chacun (secteurs 0..15)
- **4K Cartes** - 32 secteurs de 4 blocs chacun (secteurs 0..31) et 8 secteurs de 16 blocs chacun (secteurs 32..39)

4 Secteurs de blocs

Les cartes 1K et 4K utilisent chacune 16 secteurs de 4 blocs chacun, le fond 1K de mémoire sur les cartes 4K étant organisé de manière identique aux modèles 1K pour des raisons de compatibilité. Ces 4 secteurs de blocs individuels (contenant 64 octets chacun) disposent de fonctions de sécurité de base pouvant être configurées chacune avec un accès séparé lecture / écriture et deux clés d'authentification différentes de 6 octets (les clés peuvent être différentes pour chaque secteur). En raison de ces caractéristiques de sécurité (qui sont stockés dans le dernier bloc, appelé la **bande - annonce** du **secteur),** seul le bas 3 blocs de chaque secteur sont effectivement disponibles pour le stockage de données, ce qui signifie **que vous avez 48 octets par 64 octets secteur disponible pour votre propre usage.**

Chaque secteur de 4 blocs est organisé comme suit, avec quatre lignes de 16 octets chacun pour un total de 64 octets par secteur.

```
84 76 38 89 43 8  4  0  69 73 73 69 35 36 35 30 | Block  0 | Manufacturer Block
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  | Block  1 | Data Block
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  | Block  2 | Data Block
0  0  0  0  0  0  FF 7  80 69 FF FF FF FF FF FF | Block  3 | Sector Trailer
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  | Block  4 | Data Block
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  | Block  5 | Data Block
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  | Block  6 | Data Block
0  0  0  0  0  0  FF 7  80 69 FF FF FF FF FF FF | Block  7 | Sector Trailer
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  | Block  8 | Data Block
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  | Block  9 | Data Block
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  | Block 10 | Data Block
```

Sector	Block	Bytes															Description
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	[				Manufacturer Data											]	Manufacturer Block
	1	[				Data											]	Data
	2	[				Data											]	Data
	3	[	------KEY A------			]	[	Access Bits	]	[	------KEY B------					]	Sector Trailer

Le Secteur 0 Block 0 : Ce bloc contient des informations sur le tag , son modèle, le n° de série, le constructeur, et est normalement en lecture seule. En fonction du modèle exact du tag , les 7 premiers octets du bloc 0 n'ont pas la même signification.
Le secteur 0 est spécial puisqu'il contient le bloc constructeur. Il devrait être évité à moins que vous sachiez ce que vous faites.

Le Secteur 0 Block 1 et 2 : Blocs de données
(sur les autres Secteurs (1 à 15) 3 blocs de disponible 16x3 = 48o)
Les blocs de données ont une largeur de 16 octets et peuvent être lus et écrits en fonction des autorisations définies dans les bits d'accès. Vous êtes libre d'utiliser les 16 octets de données comme vous le souhaitez. Vous pouvez facilement stocker l'entrée de texte, stocker quatre valeurs entières 32 bits, un 16 caractères uri, etc.

Blocs de données en tant que "blocs de valeur"
Une alternative au stockage de données aléatoires dans les blocs de 16 octets est de les configurer comme «Blocs de valeur». Les blocs de valeurs permettent d'exécuter des fonctions de bourse électroniques (les commandes valides sont: lecture, écriture, incrémentation, décrémentation, restauration, transfert).

Chaque bloc de valeur contient une seule valeur 32 bits signée et cette valeur est stockée 3 fois pour l'intégrité des données et des raisons de sécurité. Il est stocké deux fois non inversé, et une fois inversé. Les 4 derniers octets sont utilisés pour une adresse de 1 octet, qui est stockée 4 fois (deux fois non inversée et deux fois inversée).

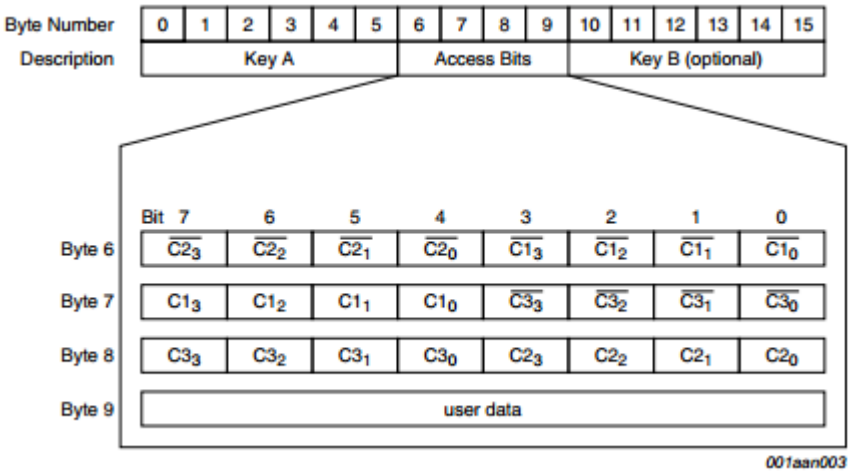
Les blocs de données configurés comme «blocs de valeur» ont la structure suivante:

Value Block Bytes
---
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
[Value] [~Value] [Value] [A ~A A ~A]

octets 0-3 : Valeur signée sur 4 octets (32bits)
octets 4-7 : La même valeur inversée
octets 8-11 : Copie la valeur
octet 12 : octet pouvant servir à spécifier une adresse (de bloc) pour la destination
octet 13 : le même octet inversée
octet 14 : copie de l'octet
octet 15 : copie de l'octet inversé

Sector	Block	Bytes	Description
		0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	
1	4	[Data]	Data
	5	[Data]	Data
	6	[Data]	Data
	7	[-------KEY A-------] [Access Bits] [-------KEY B-------]	Sector Trailer

Le block « sector trailer » contient les deux clés secrètes (touches A et B), ainsi que les conditions d'accès des quatre blocs. Il a la structure suivante:
Sector Trailer (exemple : Sector 1 Block 7)



Voilà pour la signification des données de la carte, il y aurait encore beaucoup à dire sur les mécanisme de sécurité…

Passons au deuxième programme d'exemple : **readMifareMemory**
Ce programme permet de lire le **Secteur 2 Bloc 8,** mais on peut lire un autre bloc, il suffit de changer les paramètres du programme : de 0x08 on passe en 0x05 pour le bloc 5 par exemple.
(valeur en Hex)

……...

```
if(nfc.authenticateBlock(1, id ,0x08,KEY_A,keys)) //authenticate block 0x08
  {
  //if authentication successful
  uint8_t block[16];
  //read memory block 0x08
  if(nfc.readMemoryBlock(1,0x08,block))
```

…….

voila ce que l'obtient dans le bloc 8 secteur 2 pour ma carte , cela pourra être différent dans votre cas

```
Hello!
Found chip PN532
Firmware ver. 1.6
Supports 7
Found 1 tags
Sens Response: 0x4
Sel Response: 0x8
 0xB0 0x2F 0x78 0x28Read card #2955900968
0 1 2 3 4 5 6 7 8 9 A B C D E F    Valeur des 16 octets du bloc 8 secteur 2
```

Passons au troisième programme d'exemple : **readMifareTargetID**

Ce programme permet de lire l'identifiant du tag : Secteur 0 Bloc 0 :

```
Hello!
Found chip PN532
Firmware ver. 1.6
Supports 7
Found 1 tags
Sens Response: 0x4
```

Sel Response: 0x8
0x84 0x76 0x38 0x89Read card #2222340233 un premier tag la carte

Found 1 tags
Sens Response: 0x4
Sel Response: 0x8
0xB0 0x2F 0x78 0x28Read card #2955900968 un deuxième tag le badge

Passons au quatrième programme d'exemple : **writeMifareMemory**

Ce programme est le plus intéressant car il permet de personnaliser comme on le veut les tags NFC, soit en respectant les normes NFC , soit en choisissant soit même les données à insérer dans les tags.

Dans le programme fournit , on programme le Bloc 8 secteur 2 avec les valeurs :
1 2 3 4 5 6 7 8 9 A B C D E F , ce qui nous fait bien 16 valeurs dans le bloc 8

…….

```
if(nfc.authenticateBlock(1, id ,0x08,KEY_A,keys)) //authenticate block 0x08
{
//if authentication successful

if(written == 0) //Not written
{
written = nfc.writeMemoryBlock(1,0x08,writeBuffer); // Write writeBuffer[] to block 0x08
if(written)
  Serial.println("Write Successful");
}


uint8_t block[16];
//read memory block 0x08
if(nfc.readMemoryBlock(1,0x08,block))
{
Serial.println("Read block 0x08:");
```

…….

On peut comme dans le programme précédent changer les valeurs du bloc à écrire .

Voilà ce que cela donne pour le bloc 5

found 1 tags
Sens Response: 0x4
Sel Response: 0x8
0xB0 0x2F 0x78 0x28Read card #2955900968

Read block 0x05:
0 1 2 3 4 5 6 7 8 9 A B C D E F valeur écrites dans le bloc 5 puis lues ensuite

j 'ai modifié le programme précédent pour écrire dans le bloc que je veux , le texte que je veux :

```
#include <PN532.h>

#if defined(__AVR_ATmega1280__)|| defined(__AVR_ATmega2560__)
#define SCK 52
#define MOSI 51
#define SS 53
#define MISO 50
#else
#define SCK 13
```

```
#define MOSI 11
#define SS 10
#define MISO 12
#endif

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

#define bloc 0x05

uint8_t written=0;

//char tab[16] = "Bonjour ca va !";
//char tab[16] = "BONJOUR       ";
//char tab[16] = "MJC-Fablab 37  ";
//int tab[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
  char tab[16] = "code ok       ";

void setup(void) {
  Serial.begin(9600);
  Serial.println("Hello!");

  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);
    Serial.println();

    uint8_t keys[]= {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF};
    uint8_t writeBuffer[16];
    for(uint8_t ii=0;ii<16;ii++)
    {
       writeBuffer[ii]=tab[ii]; //

    }
    if(nfc.authenticateBlock(1, id ,bloc,KEY_A,keys)) //authenticate block
    {
    //if authentication successful

    if(written == 0) //Not written
    {
    written = nfc.writeMemoryBlock(1,bloc,writeBuffer); // Write writeBuffer[] to block
    if(written)
      Serial.println("Write Successful");
    }


    uint8_t block[16];
    //read memory block
    if(nfc.readMemoryBlock(1,bloc,block))
    {
    Serial.println("Read block bloc:");
    //if read operation is successful
      for(uint8_t i=0;i<16;i++)
      {
        //print memory block
        Serial.print(block[i],HEX);
        /*int x = block[i];
```

```
      char lettre1 = char(x);*/
       Serial.print(" ");
       //Serial.print(lettre1);
      }
     Serial.println();

    }
   }
  }

 delay(500);
}
```

Et maintenant un début de programme **non optimisé** pour lire le tag NFC et ouvrir une porte ou du moins allumer une LED et déclencher un relais.

```
#include <PN532.h>

#if defined(__AVR_ATmega1280__)|| defined(__AVR_ATmega2560__)
#define SCK 52
#define MOSI 51
#define SS 53
#define MISO 50
#else
#define SCK 13
#define MOSI 11
#define SS 10
#define MISO 12
#endif

String ouvre = "code ok        ";// valeur à trouver dans le tag pour ouvrir la porte

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

void setup(void) {
   Serial.begin(9600);
   Serial.println("Hello!");

   pinMode (7, OUTPUT);
   pinMode (6, OUTPUT);

   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;
   String mesdonnees = "";
   // look for MiFare type cards
   id = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A);

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

      uint8_t keys[]= {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF};  // default key of a fresh card
      for(uint8_t blockn=8;blockn<9;blockn++) {
         if(nfc.authenticateBlock(1, id ,blockn,KEY_A,keys)) //authenticate block blockn
         {
            //if authentication successful
            uint8_t block[16];
```

```
            //read memory block blockn
            if(nfc.readMemoryBlock(1,blockn,block))
            {
               //if read operation is successful
               for(uint8_t i=0;i<16;i++)
               {
                 if (( (block[i] >= 0x2D && block[i] <= 0x2E) || (block[i] >= 0x20 && block[i] <= 0x21) ||( block[i] >= 0x31 && block[i] <= 0x39) ||
( block[i] >= 0x41 && block[i] <= 0x5A) ||( block[i] >= 0x61 && block[i] <= 0x7A)) && (blockn != 0)) // on ne lit que les valeurs ASCCII
                  {
                  int x = block[i];
                  char lettre1 = char(x);
                  Serial.print(lettre1);
                  Serial.print("   ");
                  mesdonnees = mesdonnees + char(x);
                  }
                  else
                   {
                   Serial.print(block[i],HEX);
                   Serial.print(" ");
                     if(block[i] <= 0xF) //Data arrangement / beautify
                     {
                        Serial.print("  ");
                     }
                     else
                     {
                        Serial.print(" ");
                     }
                   }

               }

               Serial.print("| Block ");
               if(blockn <= 9) //Data arrangement / beautify
               {
                  Serial.print(" ");
               }
               Serial.print(blockn,DEC);
               Serial.print(" | ");

               if(blockn == 0)
               {
                  Serial.println("Manufacturer Block");
               }
               else
               {
                  if(((blockn + 1) % 4) == 0)
                  {
                     Serial.println("Sector Trailer");
                  }
                  else
                  {
                     Serial.println("Data Block");
                  }
               }
            }
         }
      }
      Serial.print(mesdonnees);
      if ( ouvre == mesdonnees ) { // on test si les données sont bonnes
        Serial.print ( "c'est bon  ...");// c'est OK
        digitalWrite(7, HIGH); // on active le relais qui ouvre la porte
        digitalWrite(6, HIGH) ;// on allume la LED que si le relais est actif
        delay(3000);// on attend 3 secondes
        digitalWrite(7, LOW);// on désactive le relais d'ouverture de porte
        digitalWrite(6, LOW) ;// on eteind la LED
      }
      else
       {
        Serial.print ( " tu ne peux pas entrer ..");  // si la carte na pas les bonnes données … on n'entre pas
       }

   }
   delay(2000);
}
```

Voilà ce que la donne dans le terminal :

```
Hello!
Hello!
Found chip PN532
Firmware ver. 1.6
Supports 7
Found 1 tags
Sens Response: 0x4
Sel Response: 0x8
 0xB0 0x2F 0x78 0x28Read card #2955900968

c  o  d  e     o  k                        0   | Block  5 | Data Block
code ok       c'est bon  …
```

Normalement le relais est actionné est la porte s'ouvre :

voila pour aujourd'hui, c'est terminé. Mais il reste encore à voir les tags RFID 125Khz , pour lire les étiquettes des vêtements et autres …
A suivre le Lundi 9 Janvier 2017 à 20h