Project and Laboratory on Comunication System

Project Report

Vincenzo Costanzo (209895) Antonio Leo (209064)
Andrea Marcelli (209051) Sebastiano Miano (207006)
prof: Guido Albertengo

June 2015

Contents

1 Project Description

1.1 Goal

The project aims to develop an access control box based on a *NFC reader* with a two factor authentication: an *NFC card* and a user *personal PIN*. A secure communication with a Web Service is used to check the provided credentials. We decided to adopt the NFC technology for two main reasons:

- It is widely used in modern devices. Since 2012 the NFC technology started to gain popularity in all kind of devices, especially on smartphones. It is also used in payment services like in the innovative Google Wallet and Apple Pay.
- The challenge of using a non standard Fez Module and of writing a driver software to control it: GHI does not supply any Fez compatible NFC controller.

The data about accesses is collected in a database and it is easily accessible through a web page. It provides statistical informations about accesses and user management functionalities.

1.2 Use cases

The project aims to provide the following functionalities:

- User *authentication*.
- *Registration* of a new user.
- *Remote access control* through an administrative page.
- *NFC card management*: read, write and format of a NFC card.

1.2.1 Authentication

Providing an authentication mechanism is the main goal of the device. Figure 1 illustrates all the needed steps and all the actors that are involved to grant access to the user.

1.2.2 Registration

The registration of a new user is divided in two phases. In the first one the administrator registers a new user through a web administration page. This returns a *registration PIN* bound to the newly generated user that will be used in the second stage to associate the NFC card. Then, the administrator registers the NFC card by using the NFC Box and providing the previously generated PIN. Figure 2 illustrates the required steps for the registration.

1.3 Hardware components

The realization of the project required a lot of components and modules. Most of them come from the FEZ Spider Starter Kit with the addition of the *PN532 NFC/RFID Controller Shield* for the NFC communication. We also realized a 3D printed Box to provide a nice container for all the used modules. Section 5 is entirely dedicated to a detailed explanation of the design of the 3D Box.

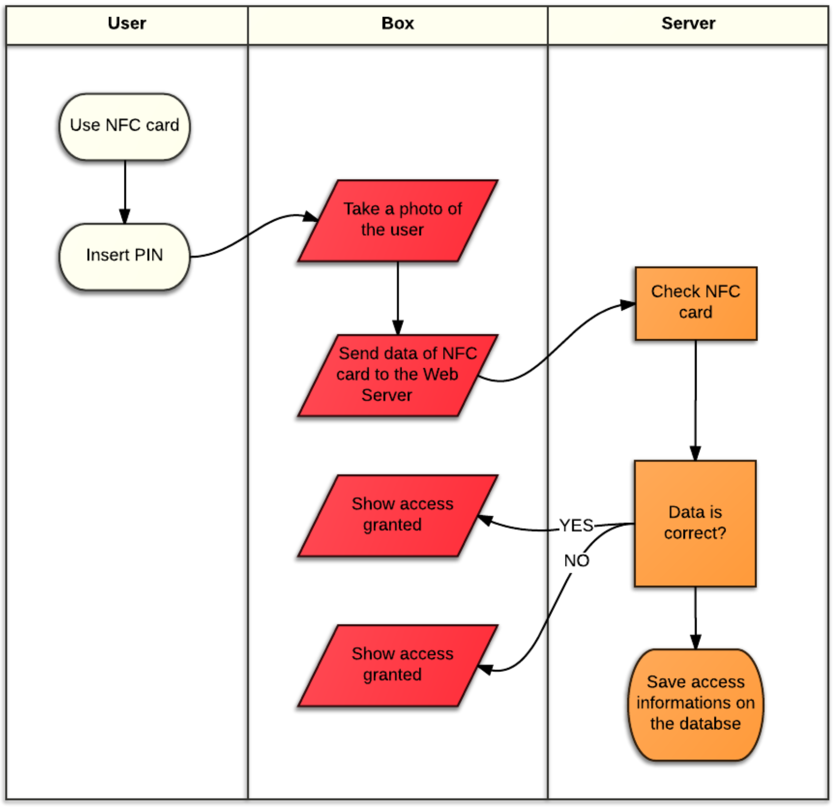


Figure 1: *Authentication.*

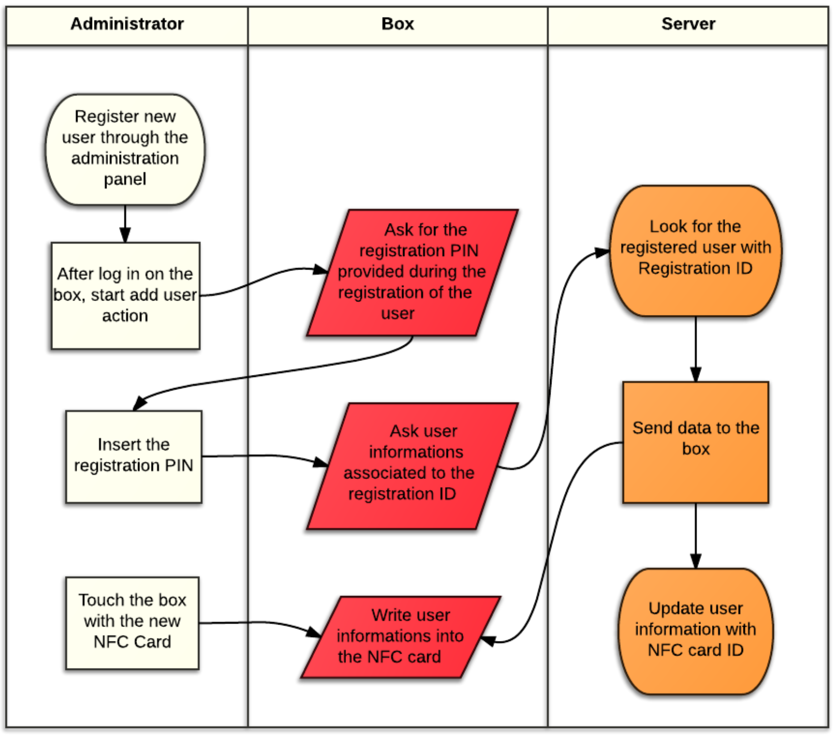

Administrator
Box
Server
Register new user through the administration panel
After log in on the box, start add user action
Ask for the registration PIN provided during the registration of the user
Insert the registration PIN
Ask user informations associated to the registration ID
Look for the registered user with Registration ID
Send data to the box
Touch the box with the new NFC Card
Write user informations into the NFC card
Update user information with NFC card ID

Figure 2: *Registration of a new user.*

1.3.1 Component list

- FEZ Spider Mainboard.
- A 3D printed White Box.
- Display T35 Module (3.5” with touchscreen).
- USB Client EDP Module.
- Camera Module.
- Multicolor LED Module.
- Ethernet J11D Module.
- WiFi RS21 Module.
- Extender Module.
- PN532 NFC/RFID Controller Shield.
- IDC cables.
- Dupont wire cables.

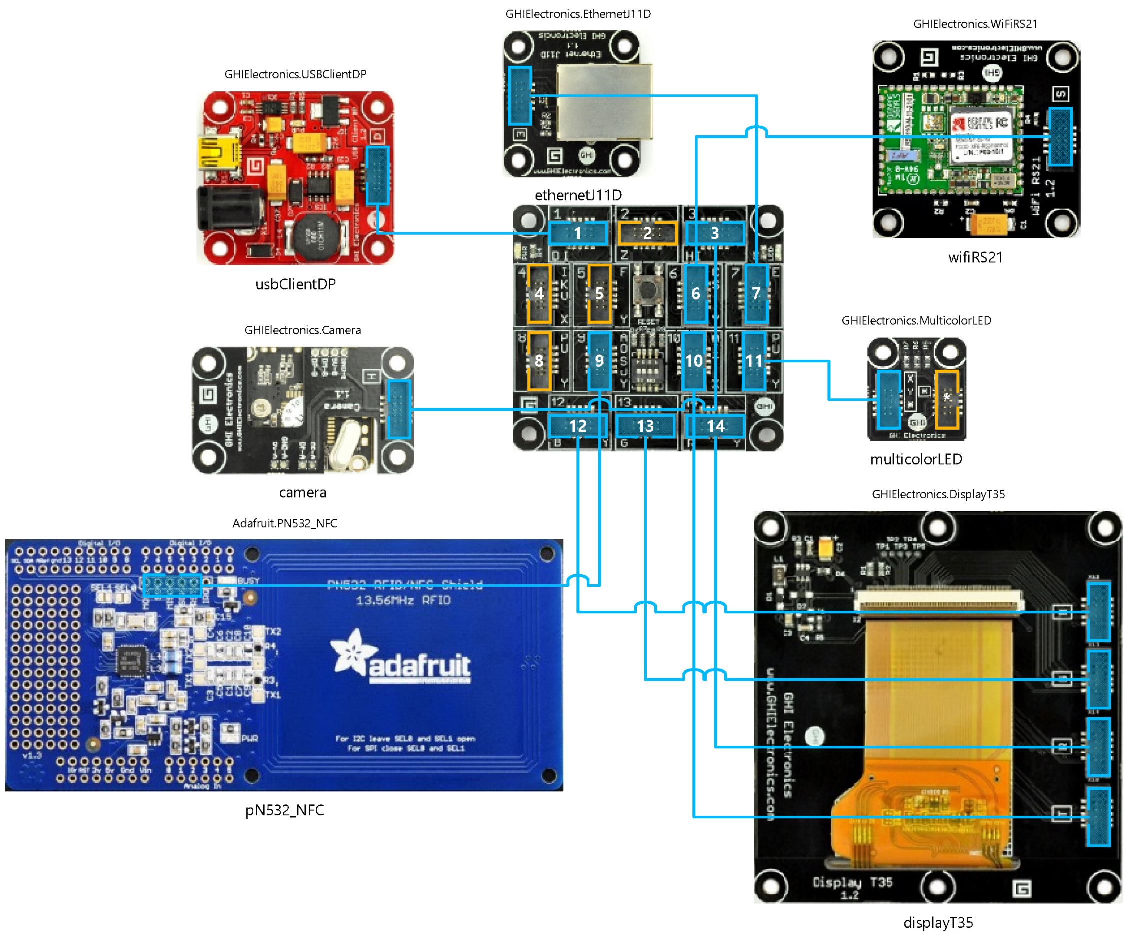


Figure 3: *Component list*

1.3.2 PN532 NFC/RFID Controller Shield

While GHI provides a Fez compatible RFID Reader Module[1], it does not supply any NFC controller. The shield that we have used in the project, Figure 4, was designed for Arduino and is based on the popular *NXP PN532 chip*. The board provides several selectable I/O modes including I2C, SPI, and HSU (High Speed Uart).

After soldering several pins on the shield, we have used Dupont wire cables to connect it with the Fez micro controller. For this purpose, the GHI Fez starter Kit provides a convenient Extender module for the breakout of cable signals.

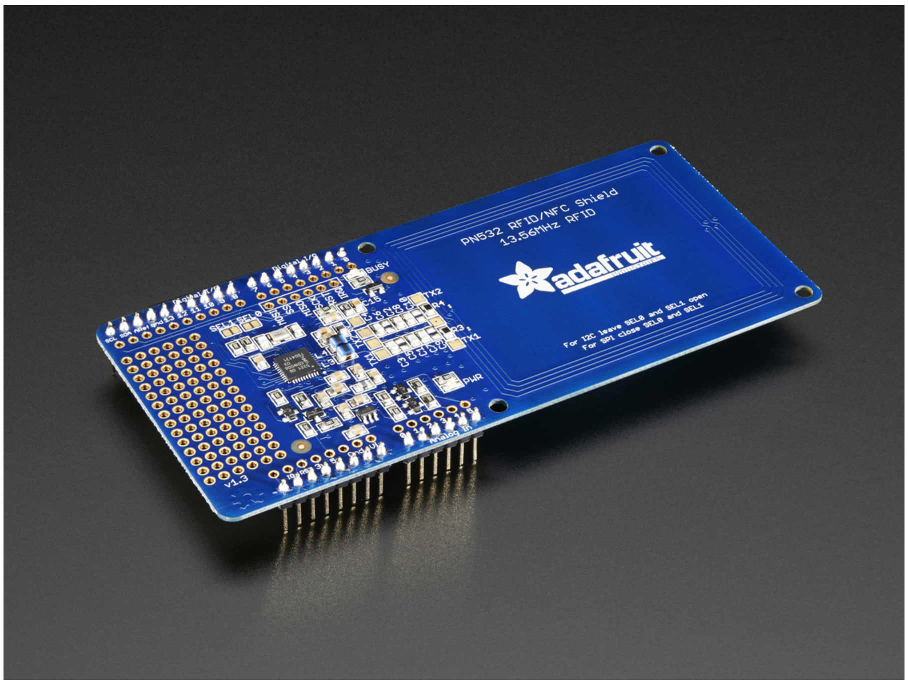


Figure 4: *Adafruit PN532 NFC/RFID Controller Shield for Arduino.*

1.4 Software architecture

The software architecture is composed of a *client* and a *server* side. Moreover, in order to support both the reading and writing of NFC tags, the driver of the NFC shield has been written from scratch. Section 2 is completely committed to the driver implementation detail.

The client side is represented by the Fez Spider micro-controller and all the connected modules. It operates as an *Access Control mechanism* with two-factor authentication using NFC tags. It implements all the necessary functions to support the modules management and server side communication. Section 3 provides a detailed description.

The server side provides a *set of REST Api* through a WCF Web Service and access to *statistical and management information* through a Web Server. The data storage is supplied by a Microsoft SQL Server. Section 4 is dedicated to the server implementation and working details.

The communication between the Fez Spider micro-controller and the WCF web Service is available with Ethernet or Wi-Fi interfaces.

[1]https://www.ghielectronics.com/catalog/product/366

2 PN532 Driver

The .NET platform lacks of a comprehensive support for the PN532 chip. For this reason, we decided to develop a brand new library to support the following operations:

- *Detection* of NFC card.
- *Reading* of Mifare Classic and Ultralight cards.
- *Wrinting* and *Formatting* a Mifare Classic card.

Before writing the software, it was required a detailed study about the SPI usage in the micro .NET platform, the communication between the host controller and the PN532 chip and the NDEF standard for writing data inside a NFC card. The most relevant information will be illustrated in Section 2.4.

The creation of a new comprehensive library started from an already available code for Arduino: the PN532 library from SeedStudio[2] and Don's NDEF implementation[3]. Although they have been very useful, some parts were incomplete. Finally, we used few components of the NDEF Android source code too[4] [5].

In order to simplify future usages of the Adafruit NFC Shiled with the Fez Spider mainboard, we decided to create a new compatible Gadgeteer software module. Further details are available in Section 2.5.

2.1 Driver Functionalities

The following is a list of functionalities that are provided by our driver:

- Communication with PN532 chip.
- Implementation of the NDEF standard.
- Reading Mifare Classic and Ultralight card.
- Initialisation of a Mifare Classic card.
- NDEF format of a Mifare Classic card.
- Memory dump of a Mifare Classic card.

[2]https://github.com/Seeed-Studio/PN532
[3]https://github.com/don/NDEF
[4]https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/nfc/NdefRecord.java
[5]https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/nfc/NdefMessage.java

2.2 Code organization

The code has been subdivided in three macro areas: the *PN532 communication*, the *NFC card management* and the *NDEF implementation*.

PN532 Communication

- *PN532_Command:* it manages the creation and the parsing of the commands that are inserted in the frames exchanged between the microcontroller and the NFC shield. Each command is generated as a byte array. Moreover, each method performs the proper error checking of the response. [Refer to Section 2.4.3 for more details.]
- *PN532_Communication:* it contains the definition of the constant fields and the implementation of the programming logic necessary for the creation and parsing of the frames that are exchanged. [Refer to Section 2.4.2 for more details.]
- *PN532_Comunication_SPI:* this is the code used to implement the SPI communication. It contains methods for reading and writing *Normal Frames*, *ACK* and *NACK*. [Refer to Section 2.4.2 for more details.]
- *PN532_const:* it contains a long list of byte constants used all over the driver implementation.
- *PN532Utility:* it contains the implementation of several debug functions and of a method for reversing the byte order.

NFC Card Management

- *NfcCard:* an *abstract class* that represents a generic NFC card.
- *MifareClassic:* a subclass of the NfcCard specifically created to manage the Mifare Classic NFC card: *read*, *write*, *memory dump*, *card initialization*, *NDEFFormat*, *NDEF Read record* and *NDEF Write record*.
- *MifareUltralight:* a subclass of the NfcCard specifically created to manage the reading of Mifare Ultralight NFC card.

NDEF Implementation

- *NdefMessage:* this is the NDEF Message implementation, according to the NFC/NDEF standard. It provides three constructors in order to manage different cases. One is the default no arguments constructors. Another one accepts a list of NdefRecord in order to support the write operation of a new NdefMessage in a NFC card. The last one receives a raw data byte array and this is used in the reading operation of a NdefMessage by parsing the content and creating a list of NdefRecord. [Refer to Section 6 for more details about the NDEF standard.]
- *NdefRecordArrayList:* it extends the functionalities provided by the micro .NET Framework ArrayList class in order to properly manage a list of NDEF Records.
- *NdefRecord:* this is the NDEF Record implementation, according to the NFC/NDEF standard. It provides several methods for the creation of URI and text records. Moreover it is incharged of the parsing of existing record during the reading operation. [Refer to Section 6 for more details about the NDEF standard.]

2.3 Driver Usage

The driver is distributed in the form of a Gadgeteer compatible software module. It extremely simplifies the usage of the *PN532 NFC/RFID Controller Shield* in conjunction with a .NET Gadgeteer mainboards. Further details about the creation of a new Gadgeteer compatible software module are illustrated in Section 2.5.

The driver internally implements two C# events and a delegate:

```
public event TagEventHandler TagDetected;
public event TagEventHandler TagLost;
public delegate void TagEventHandler(object sender, NfcCard e);
```

TagDetected corresponds to the detection of a new card and *TagLost*, is raised when the card is separated from the NFC antenna.

The following is a example of the usage of the driver software.

```
// Auto Generated Code
private Gadgeteer.Modules.Adafruit.PN532_NFC pN532_NFC;
this.pN532_NFC = new GTM.Adafruit.PN532_NFC(9);

// Subscribe to the delegates for handling events
// related to the card detection and lost.
pN532_NFC.TagDetected += nfc_TagDetected;
pN532_NFC.TagLost += nfc_TagLost;

private void nfc_TagLost(object sender, NfcCard e) {
        // NFC tag lost
}

private void nfc_TagDetected(object sender, NfcCard e) {
        // NFC tag Detected

        if (e is MifareClassic)
        {
                MifareClassic mifare = (MifareClassic)e;
                // Execute some command on the Mifare Card.
                mifare.formatNDEF()
        }
}

// Start listening.
pN532_NFC.Open();

// Do something..

// Stop listening.
pN532_NFC.Close();
```

Note Both *nfc_TagDetected* and *nfc_TagLost* methods receive a *NfcCard* object as argument. After checking which is the belonging subclass, it is possible to cast the argument to the proper class (MifareClassic or MifareUltralight) and directly call the methods for the corresponding operation on that card.

2.4 PN532 Communication

The system host controller can communicate with the PN532 by using the SPI, I2C or HSU (High Speed UART) serial link[1]. Only one link can be used at once, and the choice is done by a hardware configuration. In addition to the physical link used to communicate with the host controller, another dedicated IRQ line is used to inform the host controller when a response to a command is available.

2.4.1 SPI Interface

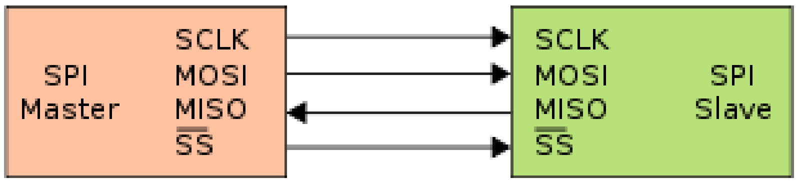


Figure 5: *SPI communication*

Serial Peripheral Interface The SPI (Serial Peripheral Interface) bus is a synchronous communication bus typically used to transfer data between a microcontroller and an external device[2]. It uses a clock signal to ensure the perfect synchronism in the transmission and reception between the two counterparts known as master and slave.

The SPI bus is characterized by the following signals:

- *SCLK (Serial CLocK):* it provides clock synchronization for the data exchange.
- *SS (Slave Select):* it is the signal for enabling the receiver slave.
- *MOSI (MasterOutSlaveIn):* it is the data line used for the transmission from master to slave.
- *MISO (MasterInSlaveOut):* it is the data line used for the transmission from slave to master.

At each clock pulse generated by the master, the *slave does* the following two things:

1. It samples the signal on the MOSI line for acquire data in reception.
2. It sets a logic level (0 or 1) on the MISO line to send data.

The SPI bus is full duplex: the transmission and the reception can take place simultaneously. However, the PN532 slave works in half duplex mode[1]. The SS signal is used by the master to enable the slave to start a communication session. The SS signal is *HIGH* when the slave is disconnected from the bus, the it is set to *LOW* from the master, when it wants to communicate with the slave. At the end of the communication, the signal is returned to the *HIGH* value.

PN532 and SPI Among the alternatives offered by the PN532 chip, we chose to use the SPI communication protocol. The .Net Micro Framework provides a convenient SPI class (Microsoft.SPOT.Hardware namespace) for the management of the SPI interface.

Configuration It is necessary to configure the SPI interface, by using the SPI.Configuration class. The following is the list of the required parameters:

- *ChipSelect_Port:* it is the pin (Cpu.Pin enum) which will be used as SS (Slave Select). We set it to GPIO_NONE, in order to drive this pin directly through the OutputPort class.
- *ChipSelect_ActiveState:* the active state of the chip select. Typically, the devices have an SPI chip select "active LOW", so it is necessary to set the logic level to 0 (false) in order to turn on the communication with the device.
- *ChipSelect_SetupTime:* it is the time that must elapse from when the chip select is activated and the clock signal is transmitted on its line.
- *ChipSelect_HoldTime:* it is the time that must elapse between the end of the read/write transaction and the instant when the chip select is inactive.
- *Clock_IdleState:* it indicates the clock idle condition on the line when the slave has not been activated.
- *Clock_Edge:* it indicates the rising edge or falling edge at which the data on the communication line (MOSI or MISO) is sampled.
- *Clock_Rate:* it is the clock frequency.
- *SPI_mod:* it indicates the physical SPI port of the CPU to use.

Reading and Writing The SPI class provides two main methods for reading and writing:

- *Write()*: it allows to perform a data transfer from the master to the slave.
- *WriteRead()*: it allows to perform a data transfer from master to slave and vice versa. The SPI bus is full duplex: at every clock pulse one bit is transmitted and another one is received.

Example The following portion of code illustrates the mechanism that is used to read and write from the SPI interface. The Slave Selection (NSS) signal is set to *false* to enable the slave to receive the data, before executing the Write() method. Afterwards it is reset to *true* at the end of the transmission. Figure 6 illustrates a diagram of the usage of the SPI signals for exchange commands with the PN532 NFC controller.

```
byte[] write = new byte[CMD_SIZE];
// prepare write buffer ...

// send frame
nssPort.Write(false);
spi.Write(write);
nssPort.Write(true);
```

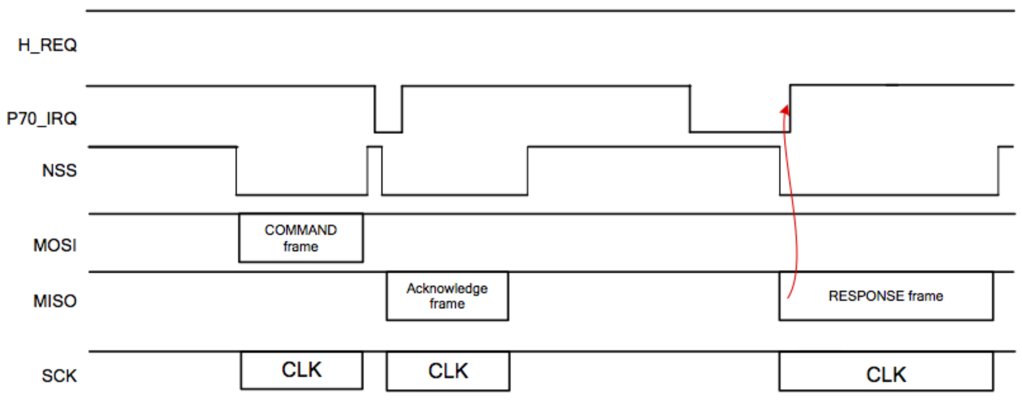


Figure 6: *SPI Handshake mechanism*

Byte ordering The Fez Spider mainboard uses a ARM7 LPC2478 processor with the little-endian byte order[3]. The SPI Fez Spider shift and transmits the output data in MSB first mode, which starts the transmission from the Most Significant Bit. In our case the PN532 shield expects to receive the bits in the reverse order (LSB Least Significant Bit), so we had to reverse the order of the bits before starting transmission.

2.4.2 Host controller communication protocol

Frames Communication between the host controller and the PN532 is performed through frames in a half-duplex mode. Three different types of frames are used:

Normal information frame

It is used to *exchange commands* from the host controller to the PN532 *and responses* to these commands from the PN532 to the host controller, Figure 7.

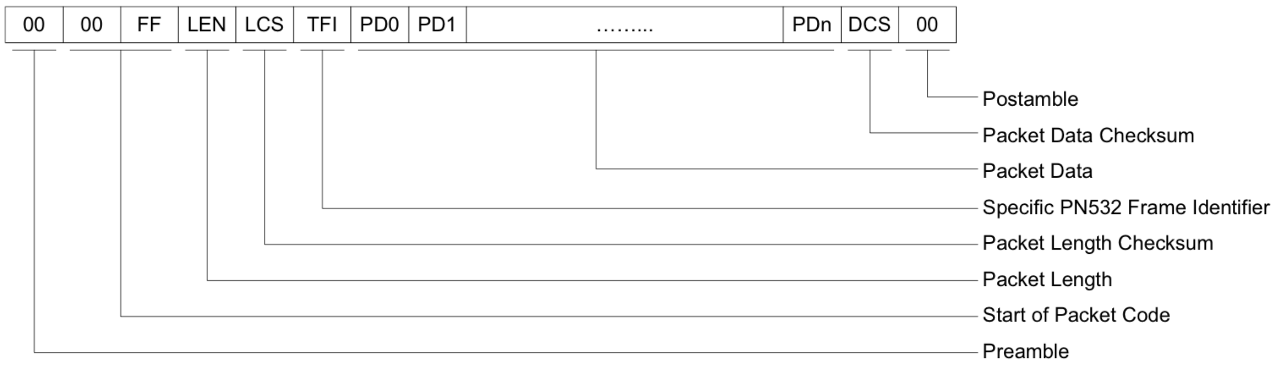


Figure 7: *Nornal Frame*

ACK frame

It is used to indicate that the previous frame has been successfully received, Figure 8.

NACK frame

It is used to indicate that the previous response frame has not been successfully received, then asking for the retransmission of the last response frame from the PN532 to the host controller, Figure 9.

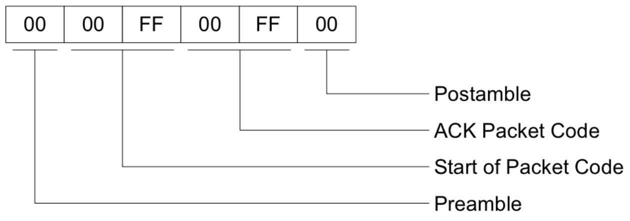


Figure 8: *Ack Frame*

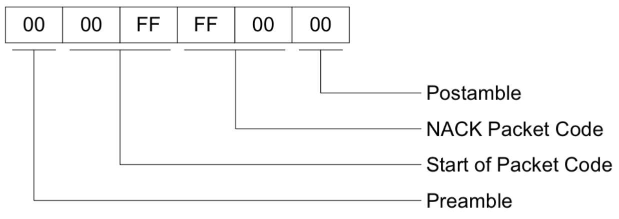


Figure 9: *Nack Frame*

IRQ As previously said, we used the IRQ pin to indicate when the PN532 is ready to send its frame, Figure 16. In that case, the host controller can wait for this line to be asserted by the PN532.

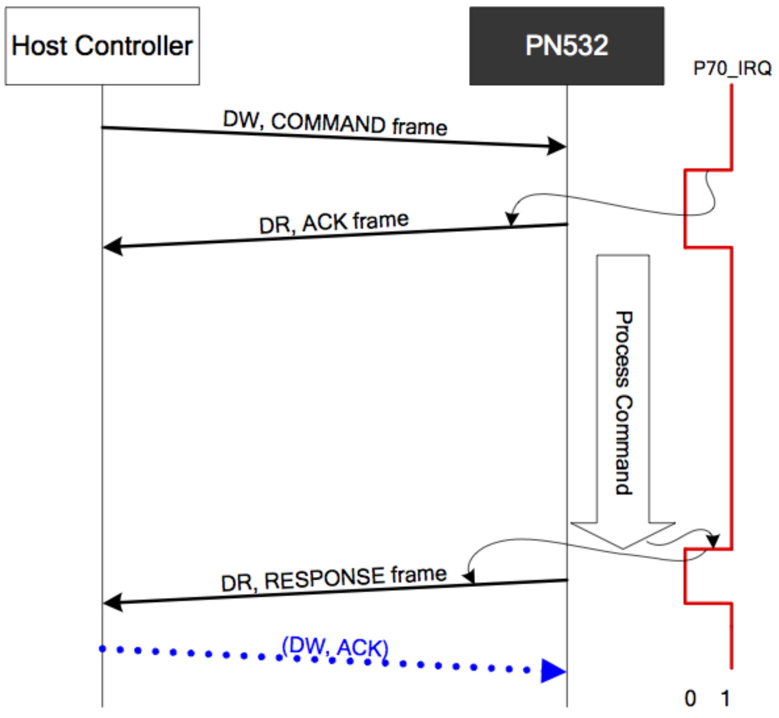


Figure 10: *Advanced SPI communication with Handshake mechanism.*

2.4.3 PN532 Commands

The following is a brief summary of the main commands that have been implemented in the driver software. Commands are exchanged through a *command-response pattern.* In each of the following pictures, the highlighted byte is the unique identifier of the command.

GetFirmwareVersion The PN532 sends back the version of the embedded firmware. Figure 11.

Input:

D4	02

Output:

D5	03	IC	Ver	Rev	Support

Figure 11: *GetFirmwareVersion Input and Output.*

OUTPUT packet detail

- IC: for PN532, the contain of this byte is 0x32.
- Ver: version of the firmware.
- Rev: revision of the firmware.
- Support: indicates which are the functionalities supported by the firmware. When the bits are set to 1, the functionality is supported, otherwise (bit set to 0) it is not.

SAMConfiguration This command is used to select the data flow path by configuring the internal serial data switch. Figure 12.

Input:

D4	14	Mode	Timeout	[IRQ]

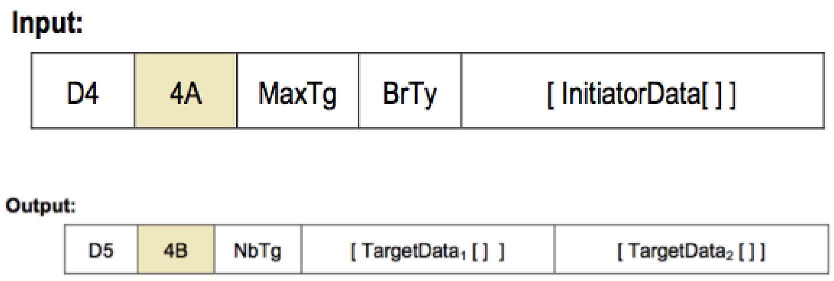
Output:

D5	15

Figure 12: *SAMConfiguration Input and Output.*

INPUT packet detail

- Mode: (0x01) Normal mode, the Security Access Module is not used.
- Timeout: (1 sec)
- IRQ: (0x01) The IRQ pin is driven by the PN532.



Input:

D4	4A	MaxTg	BrTy	[InitiatorData[ ]]

Output:

D5	4B	NbTg	[TargetData_1 [ ]]	[TargetData_2 [ ]]

Figure 13: *InListPassiveTarget Input and Output.*

InListPassiveTarget The goal of this command is to detect as many targets (maximum MaxTg) as possible in passive mode. Figure 13.

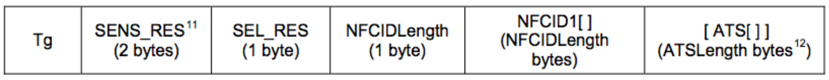
INPUT packet detail

- MaxTg: is the maximum number of targets to be initialized by the PN532. The PN532 is capable of handling 2 targets maximum at once. We set this to 0x1 in order to handle at maximum one card at once.
- BrTy: is the baud rate and the modulation type to be used during the initialization. We set this to 0x00 that correspond to 106 kbps (ISO/IEC14443 Type A).
- InitiatorData[ ]: is an array of data to be used during the initialization of the target(s). For the Baud Rate that we used, the content of this field is empty.

OUTPUT packet detail

- NbTg: is the Number of initialized Targets (minimum 0, maximum 1).
- TargetData[ ]: contains the information about the detected targets and depends on the Baud Rate that is selected. For the 106 kbps Type A (Mifare Classic and Ultralight), Figure 14 illustrates the payload structure. The following code snippet is an example of how to recognize a Mifare Classic and Ultralight based on some bytes value.

```
if (SENS_RES[0] == 0x00 && SENS_RES[1] == 0x04 && (SEL_RES == 0x08 || SAK == 0x18))
{
        //MifareClassic1K and MifareClassic4K
}
else if (SENS_RES[0] == 0x00 && SENS_RES[1] == 0x44 && SEL_RES == 0x00)
{
        //Mifare Ultralight
}
```



Tg	SENS_RES[11] (2 bytes)	SEL_RES (1 byte)	NFCIDLength (1 byte)	NFCID1[ ] (NFCIDLength bytes)	[ATS[ ]] (ATSLength bytes[12])

Figure 14: *106 kbps Type A*

Input:

D4	40	Tg	[DataOut [ ]]

Output:

D5	41	Status	[DataIn [ ]]

Figure 15: *InDataExchange Input and Output.*

InDataExchange This command is used to support protocol data exchanges between the PN532 as initiator and a target. Figure 15.

INPUT packet detail

- Tg: is a byte containing the logical number of the relevant target.
- DataOut: an array of raw data (from 0 up to 262 bytes) to be sent to the target (NFC tag) by the PN532.

OUTPUT packet detail

- Status: it is a byte indicating if the process has been terminated successfully or not.
- DataIn: is an array of raw data (from 0 up to 262 bytes) received by the PN532.

As illustrated in Figure 16, DataIn and DataOut arrays of bytes are exchanged between PN532 NFC shield and the NFC tag. A usage example will be analyzed in the Section 2.4.4.

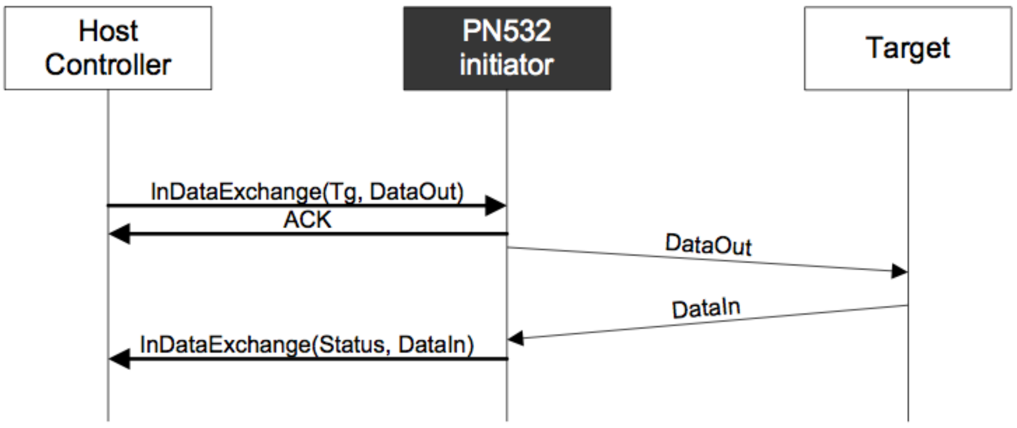


Figure 16: *In Data Exchange data flow.*

2.4.4 Mifare Interaction

Figure 17 illustrates an example that describes a short session with a Mifare Standard card. The first step consists of initializing the PN532, by using the *InListPassiveTargetCommand*, and waiting until a new target is detected. Then, in the second step, by using the *InDataExchange* command and by providing a proper Mifare command byte array, it is possible to Authenticate, Read and Write a Mifare card.

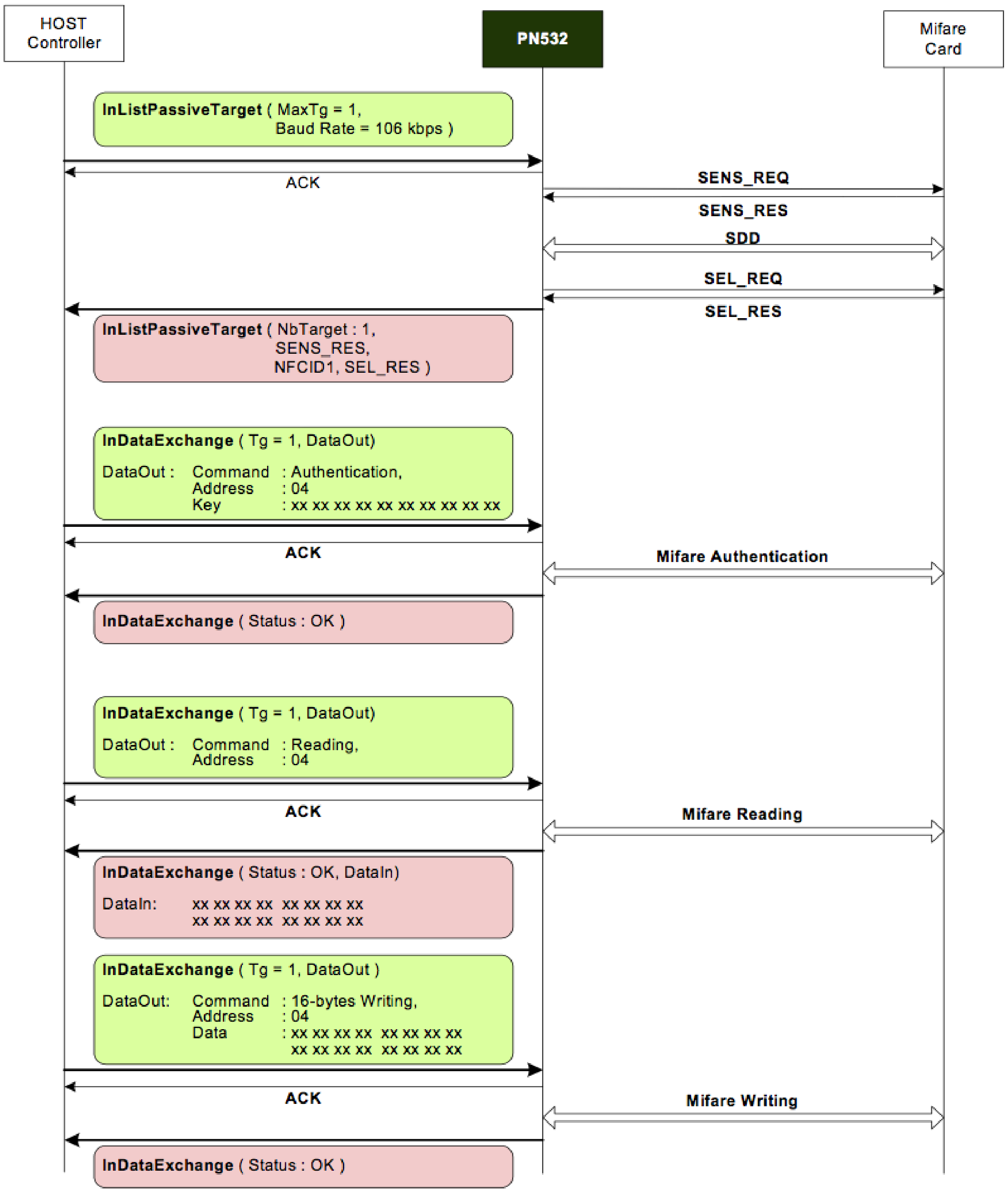

HOST Controller
PN532
Mifare Card
InListPassiveTarget (MaxTg = 1, Baud Rate = 106 kbps)
ACK
SENS_REQ
SENS_RES
SDD
SEL_REQ
SEL_RES
InListPassiveTarget (NbTarget : 1, SENS_RES, NFCID1, SEL_RES)
InDataExchange (Tg = 1, DataOut)
DataOut : Command : Authentication, Address : 04 Key : xx xx xx xx xx xx xx xx xx xx
ACK
Mifare Authentication
InDataExchange (Status : OK)
InDataExchange (Tg = 1, DataOut)
DataOut : Command : Reading, Address : 04
ACK
Mifare Reading
InDataExchange (Status : OK, DataIn)
DataIn: xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx
InDataExchange (Tg = 1, DataOut)
DataOut: Command : 16-bytes Writing, Address : 04 Data : xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx
ACK
Mifare Writing
InDataExchange (Status : OK)

Figure 17: *NFC session with a Mifare Standard card.*

2.5 Creation of a new .NET Gadgeteer Module

Due to the open standards of the hardware and software interfaces, including open source software for core libraries, it is possible to build peripheral modules compatible with the .NET Gadgeteer specification. The .NET Gadgeteer Module Builder's Guide [4] provides the guidelines to ensure that the new module is compatible with the widest range of .NET Gadgeteer mainboards. The main phases of the process are following summarized:

1. Install the .NET Gadgeteer Builder Templates[6].
2. Create a new .NET Gadgeteer Module Template in Visual Studio.
3. Implement the Module Driver using C#, following the recommended code pattern by exposing a constructor, some events and the respective delegate that are raised when something relevant occurs.
4. Compile the project and create an installer version of the module. (To build the installer MSI automatically, WiX Toolset 3.5 or newer is required[7].)
5. Install the new add-on module in Visual Studio.
6. Drag and Drop the Module from the Toolbar and connect to your favourite socket, like it is displayed in Figure 18.

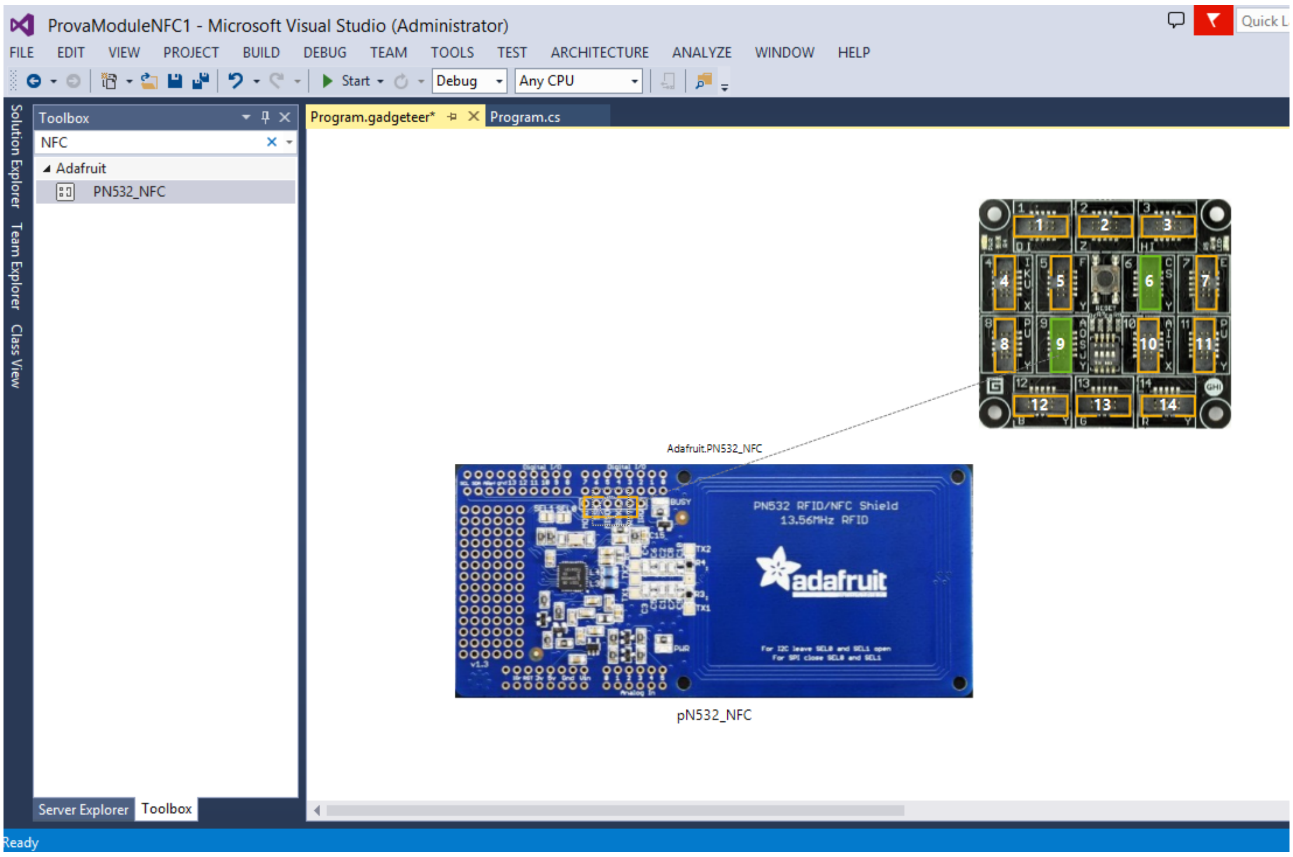


Figure 18: The Adafruit PN532 Module in Visual Studio.

[6]https://gadgeteer.codeplex.com/downloads/get/918082
[7]http://wixtoolset.org/

Gadgeteer event pattern The .NET Gadgeteer Module Builder's Guide [4] suggests the usage of the event pattern based on the following components:

- A delegate declaration for an event handler.
- An event.
- A private pointer to the event-raiser method.
- The event-raiser method itself.

NFC driver use a secondary thread to detect NFC Cards, but it is necessary to execute related event onto the Dispatcher thread. The following code snipped illustrates the technique that has been used for this purpose.

```
protected virtual void OnTagDetected(PN532_NFC sender, NfcCard card)
{
   if (this._OnTagDetected == null)
   {
        this._OnTagDetected = new TagEventHandler(this.OnTagDetected);
   }

   // Use Program.CheckAndInvoke to get onto the Dispatcher thread.
   // * If the event is null then it returns false.
   // * If it is called while not on the Dispatcher thread, it returns false,
   // but also re-invokes this method on the Dispatcher.
   // * If on the thread, it returns true so that the caller can execute the event.
   if (Program.CheckAndInvoke(TagDetected, this._OnTagDetected, sender, card))
   {
        this.TagDetected(sender, card);
   }
}
```

GadgeteerHardware.xml The following is a portion of *GadgeteerHardware.xml*, the xml file that has been used to define configuration parameter for the new module. It allows to specify an Image file, the compatible socket type and other properties that will be used by VisualStudio.

```
<ModuleDefinition Name="PN532_NFC"
                 UniqueId="2375e659-44ad-4bf7-b72f-332a301d60c0"
                 Manufacturer="Adafruit"
                 Description="A␣PN532_NFC␣module"
                 ModuleSuppliesPower="false"
                 Image="Resources\Image.jpg"
                 BoardHeight="53.3"
                 BoardWidth="117.7"
                 MinimumGadgeteerCoreVersion="2.42.500"

                  ...

<Socket Left="35" Top="10" Orientation="90" ConstructorOrder="2"
        TypesLabel="S" Optional="true">
        <Types>
           <Type>S</Type>
        </Types>
</Socket>
```

3 NFC Box Software

The box implements all use cases described in section 1.2. It allows us to detect when a user strips the card to the right side of the box and to send an authentication request to the web service in order to give or not access to the corresponding user. It also provides several functionalities to an administrative user in order to interact and performs some operations on the NFC card, addition to the fact of being able to register a new user card.

We used the .NET Micro Framework 4.3 and the latest GHI Electronics SDK (2015 R1 Pre-Release 2) which provides several fixes on known bugs as the *TakePicture()* method of the camera module and the *ConvertToFile()* function used to convert a NETMF Bitmap into a BMP file that can be later saved to a file or to the network.

3.1 The design

Given the complexity and the high number of callbacks required to manage the various hardware components inside the box, we decided to adopt a modular structure that allows the reading of the code simpler and, at the same time, easier its management.

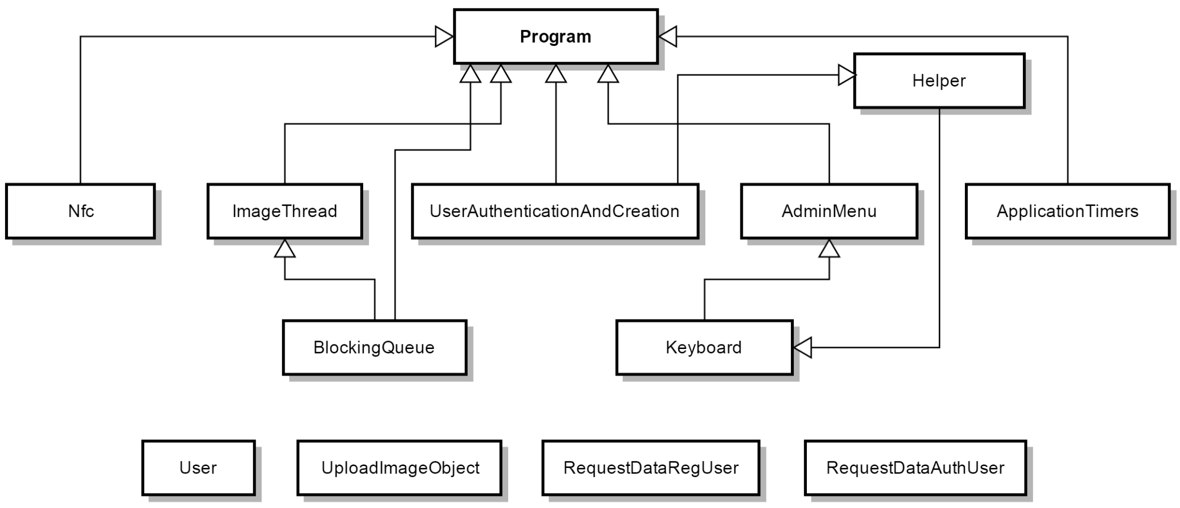


Figure 19: *Class organization*

The following is the list of used classes and for each of them there is a small description.

Program This is the main class of the program, it contains references to all modules and exposes the methods that will be called by other classes in order you to change the Graphical User Interface depending on the current context. It also contains all setup methods needed to initialize the hardware environment on which the system is based.

There is the initialization of:

- Wi-Fi or Ethernet module (depending on which you choose to use)
- The images that will be displayed on the screen
- The camera module and its callback that will be used to react properly when a new image is captured.

- The BlockingQueue and the ImageThread needed for implementing the producer/consumer pattern used in the image sending operation. See section 3.3.1 for more details.
- The NFC to start reading and discovering of new NFC tags

Nfc The class NFC allows the management of the communication with the PN532 driver [discussed in section 2]. When this class is instantiated, it immediately subscribes to the events provided by the driver in order to receive notifications when a new NFC tag is closer to the box.

When this occurs, the system reacts accordingly asking the Program class to display the keyboard and allowing the user to enter the PIN required for the authentication phase.

Figure 20: *First image after NFC initialization*

Keyboard The keyboard class allows to handle the keyboard shown to the user during the authentication phase. Instead of using the standard keyboard available with Glide, we decided to use a fixed image which provides a better visual effect. This has required to intercept every single touch event from the screen and parse it in order to figure out which button of the keyboard has been pressed.

AdminMenu This class is responsible for managing the menu that is presented to the administrator once authenticated within the box.

The administrator has the ability to:

- Register a new user
- Reset the card to its original state
- Read the content of a card
- Format it according to the NDEF standard

UserAuthenticationAndCreation This class is used to provide a simplified management of the methods needed for interacting with the web service.

In fact, it allows you to:

- Make an authentication request for the user who just swiped the card

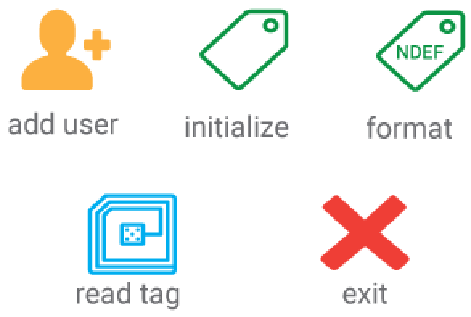


Figure 21: *Admin user menu*

- Register a new user
- Download the image associated with a particular user

ApplicationTimers This class allows you to manage the timers necessary for the application in order to:

- Reset some variables used during the access phase
- Provide a minimum time for viewing pictures before they are changed

User, UploadImageObject, RequestDataRegUser, RequestDataAuthUser These four classes are simply used as containers for data received from the web service as a result of a transaction.

Some of them are instead used to send a new request. Since the data exchange format required by the web service is JSON, we used a JSON serializer (provided in the Json.NetMF library[8]) that receives as input an object and provides its JSON representation as output.

3.2 Glide

Glide is a graphical library for .NET Micro Framework that uses a graphical screen designer. We choose to use it because provides a more responsive experience than the built-in Windows Presentation Foundation with many additional features, from buttons and lists to keyboards and message boxes.

[8]https://github.com/mweimer/Json.NetMF

3.3 Performance Optimization

There are also several enhancements we have made to harness the power of .NET Micro Framework.

3.3.1 Threads

We setup a multi-threading environment in order to separate some task that are conceptually independent and have different priorities. These tasks are:

- Manage the sending of the captured image to the Web Service during the authentication phase
- Make asynchronous HTTP requests
- Manage the GUI so that it is always available for interaction with the user

While the *HTTPHelper* class gives us the possibility to make asynchronous HTTP requests (GET and POST), handled by a secondary thread, and receiving only a callback when a reply has been received, for the first point we had to create a new thread deputy to carry out this task.

This is done by using two classes:

BlockingQueue This class is an implementation of a blocking queue which supports operations that wait for the queue to become non-empty when retrieving an element, and wait for space to become available in the queue when storing an element. The problem solved by this queue is commonly referred to as the producer/consumer problem. A common reason for having a separate producer and consumer is to disconnect the request sequence (usually existing on a thread interacting with the user) from the request processing effort. In simpler terms, the producer thread is not burdened with the effort of handling the request, and can instead focus on accepting requests and handing them off.

ImageThread The *ImageThread* acts as a consumer. It waits for a new image entered from the main thread in the BlockingQueue and, when inserted, it is woken up and immediately picks up this image sending it to the web service. We have assigned to it a below normal level priority in order to speed up the authentication process and possibly postpone sending of the image in a few moments later as it does not require an immediate interaction.

3.3.2 HTTPS

Our web service requires two information in the authentication request:

- The NfcID of the card
- The user's PIN

Since the security level of information contained in this message is high we decided to carry this request within a SSL tunnel and then use the HTTPS protocol, which guarantees that the data can not be compromised or sniffed during the transit on the network.

4 Server Side Communication

The server side represents a fundamental part in the realization of an authentication system. The two-factor authentication system ensures that both the NFC card and the inserted PIN must be valid.

NFC authentication systems can be categorized in two main typologies:

- *Offline Authentication Systems*: the secret is not shared between the user and the authentication server. Normally, it is just the NFC card that it is authenticated and the associated PIN is saved inside.
- *On-line Authentication Systems*: the secret information is shared between the user and the authentication server. Although this solution is more complex, it provides a higher level of security.

The solution that we adopted falls in the second category. The NFC Box accesses the database to check if the provided secret is the one that is associated with the NFC tag. Since the secret is not accessible to an attacker, duplicating the card does not ensure the access.

4.1 Web Services

The web service usage allows us to expose the functionality of our existing code over the network and to recall these operations in a easy way.

We chose to develop the web service using the WCF (Windows Communication Foundation) framework. Moreover, in order to secure the communication between the Web Service and FEZ Spider, we used the HTTPS [5].

The following table illustrates the list of the REST API provided through Web Services:

Type	**URI**	**Web Service Name**
POST	authenticateUser	GetAuthenticateUserAsJson
POST	UploadPhoto/{nfcID}	UploadLastPhoto
GET	DownloadPhoto/{nfcID}	DownloadUserPhoto
POST	registerNewUser	GetRegistrateUserAsJson

Table 1: *Web Services*

In each API we use the HTTP status code to indicate the success or failure of the operation.

AuthenticateUser

This method is used to authenticate a user given a card ID and the user's PIN. In case of success it returns some user information to display on the NFC box.

RegisterNewUser

This method is used during the second phase of user registration [1.2.2].

It receives the NfcID and the RegistrationID and it associate the given NFC card to the user identified by the RegistrationID.

UploadPhoto

It is used to load the photo captured during the authentication phase in the database in order to create an access log.

DownloadPhoto

This method is used to download the photo of an authenticated user in order to display it with the other information returned by the AuthenticateUser method.

4.2 Microsoft Sql Server

We chose to use Microsoft SQL 2012 Database for the data storage. It is highly integrated in the Visual Studio IDE and it allows a simple creation of methods that internally perform database queries.

The database is composed of two tables: one is used to store the *user information* and the other is for the *accesses log*.

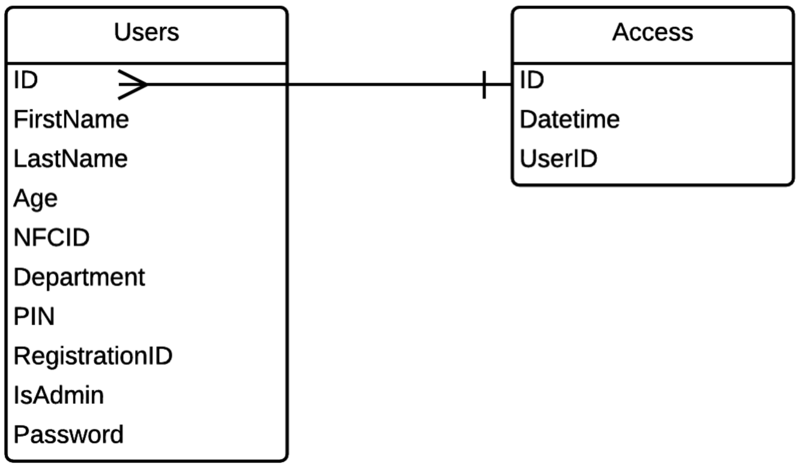


Figure 22: *ER Diagram of the Database.*

4.3 Web Server

The Web Server provides an administrative web page. It exposes some administrative functions such as the user management and statistical pages to monitor the accesses log. Furthermore it allows an administrator to add, modify and remove an user.

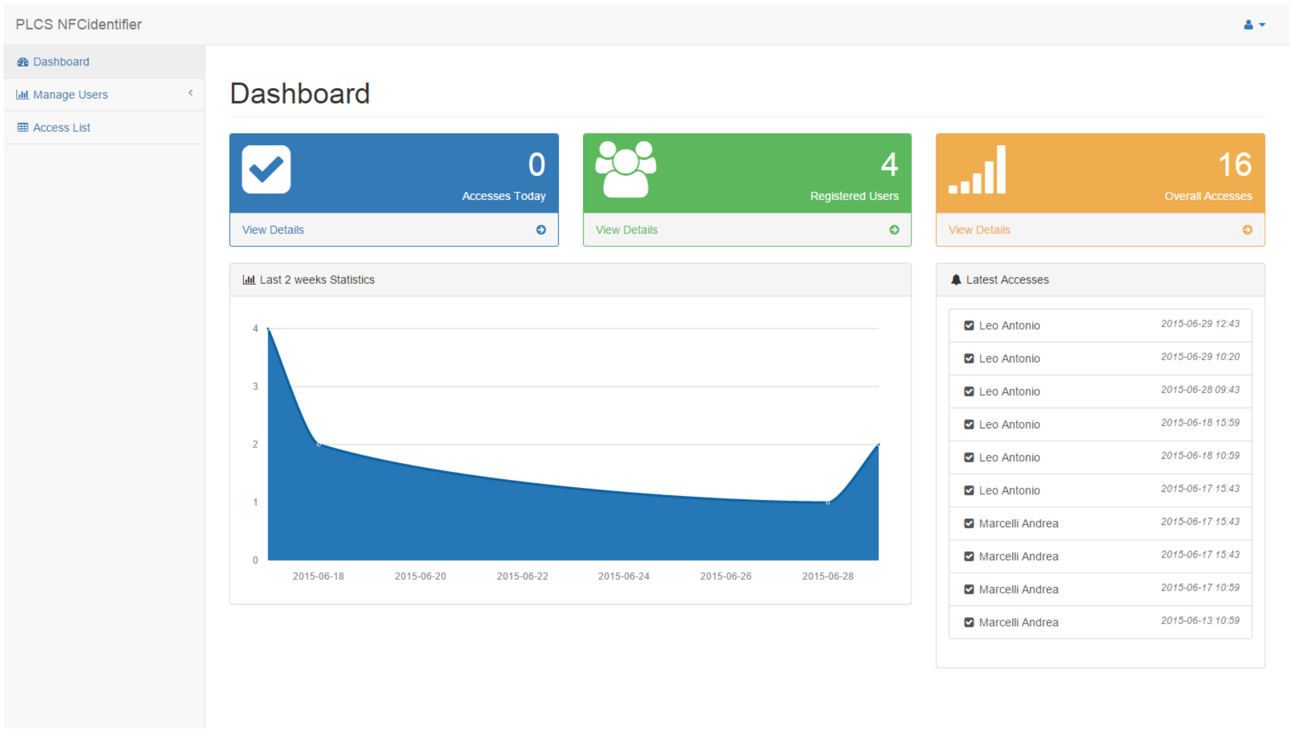
We used the Apache Web Server (v. 2.2) and the PHP (v. 5.6) script language. It is interesting to note that PHP, since version 5.3, dropped the support of the *MSSQL* set of functionalities, so we used the Microsoft *SQLSRV* extension for PHP to manage the interaction with the Microsoft SQL database.

The administrative Web Page provides two kinds of functionalities: *statistical information* and *users management*.

4.3.1 Statistical information

This is the first functionality accessible once the administrator logged in. The home page displays several statistical information about last logging events.

- Basic information such as today's accesses, number of registered users and overall accesses.
- Graph of accesses in the last 2 weeks.
- Last ten accesses with user information.



✓ Leo Antonio	2015-06-29 12:43
✓ Leo Antonio	2015-06-29 10:20
✓ Leo Antonio	2015-06-28 09:43
✓ Leo Antonio	2015-06-18 15:59
✓ Leo Antonio	2015-06-18 10:59
✓ Leo Antonio	2015-06-17 15:43
✓ Marcelli Andrea	2015-06-17 15:43
✓ Marcelli Andrea	2015-06-17 15:43
✓ Marcelli Andrea	2015-06-17 10:59
✓ Marcelli Andrea	2015-06-13 10:59

Figure 23: *Dashboard of the administrative panel.*

In the Access List page, the administrator can monitor the entire access log. It is also possible to view the photo uploaded during the access attempts.

4.3.2 User management

This section provides user management functionalities:

- Add user.
- Modify and delete user.
- List of registered users.

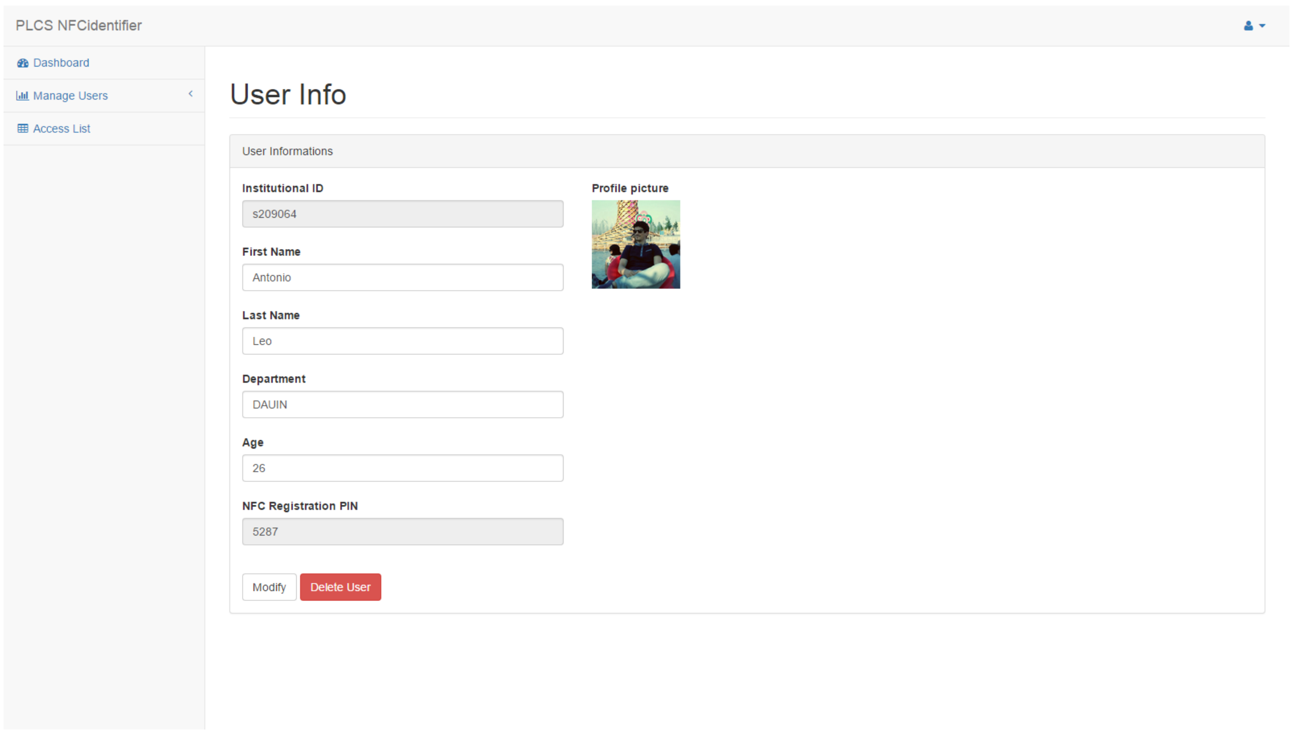
Users are registered using following institutional information:

- Profile picture
- Institutional ID (e.g. s123456)

- First name
- Last name
- Age
- Department

During the registration a temporary random number is generated: it is needed for the later binding of the NFC card with the new user. Then, it will be used by the administrator on the NFC box when a new user is registered.

The same procedure is applied if the user needs to change its associated NFC card.



User Info

User Informations

Institutional ID
s209064

Profile picture

First Name
Antonio

Last Name
Leo

Department
DAUIN

Age
26

NFC Registration PIN
5287

Figure 24: *Modify and delete page.*

5 3D Printed Box

The *GHI Fez Spider microcontroller* is used with a multitude of modules. From the beginning, it appeared the necessity of building a box to keep all component together during the development of the project. The availability of a 3D printer made possible to print a custom made 3D box.

5.1 Design

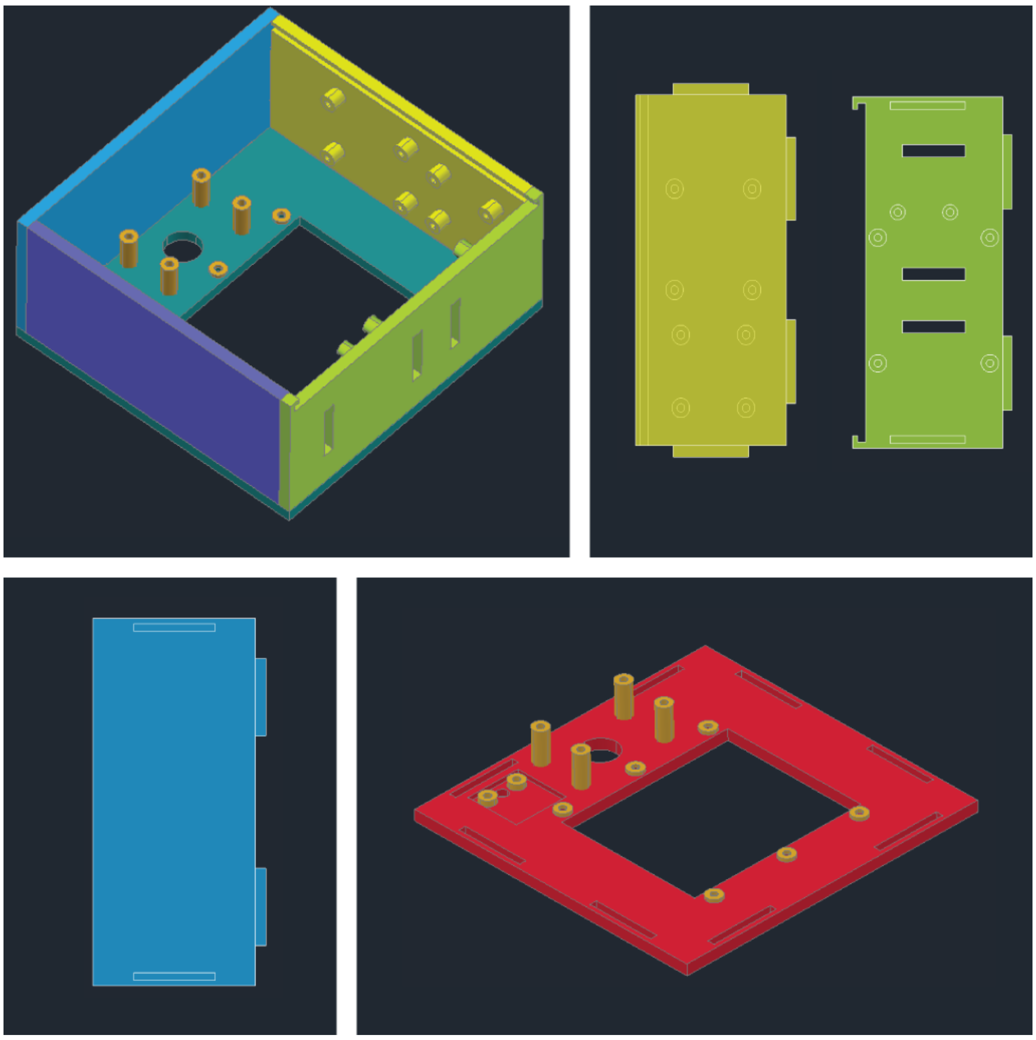


Figure 25: *Some technical drawings from the AutoCAD deisgned model.*

After different model proposals, we chose to design a modular box. Although it was theoretically possible to print a one piece 3D box, dividing it into different modules, made the construction more robust and precise. The presence of lateral support for the screws still represents a technical issue for the 3D printer. For this reason the final product was made of 6 different parts: the main side, an enclosing plate and four vertical walls. The project of the 3D model was developed using AutoCAD software and in figure 25 there are some technical drawings of the box.

The front face has some holes to fit the camera, the lcd monitor and a multicolour led. Moreover twelve cylinders provide the necessary support for the screw to fasten all the electronic modules. Lateral walls provide the support for the NFC and FEZ microcontroller, the expansion module and the USB and Ethernet connections. Three air intakes in correspondence of most hot zones allow the air to circulate. The back plate was designed to guarantee an easy access to the box interior during the development phase. Two rails are printed on the lateral walls in order to allow the cover to slide. A couple of holes allow the Ethernet and USB cables to enter and help in the cooling the board.

The dimension are 14,5cm(**w**), 15(**h**), 7(**d**).

5.2 Printing

Printing the entire model took more than 16 hours.

5.3 Assembling

Although the high precision of the printing, the tight joint sockets required a manual finishing touch. The box was assembled without glue. All modules have been fasten using small screws lightly tighten to the extruded cylinders. Figure 26 shows the box during the initial assembling phase.

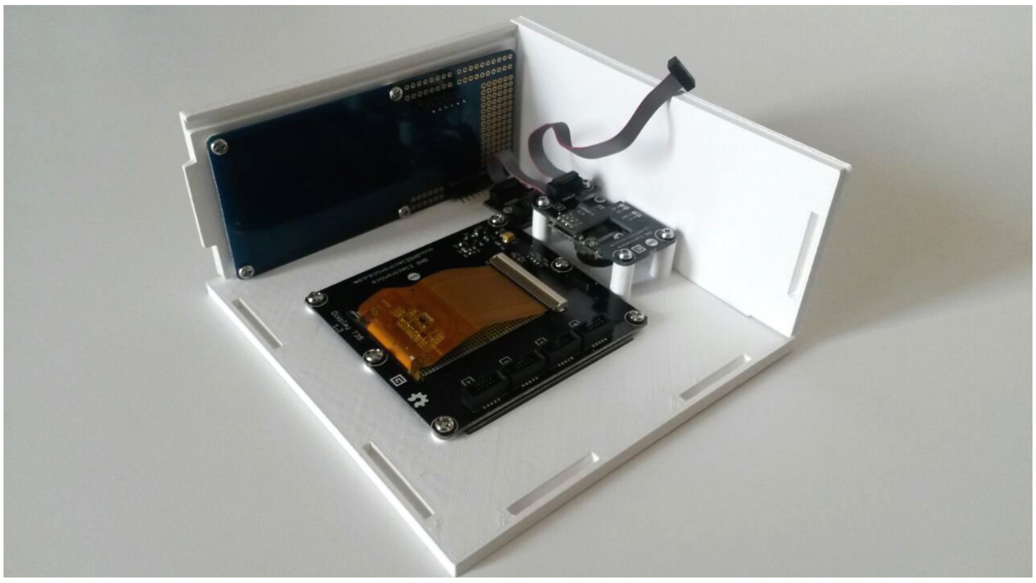


Figure 26: *Assembling the 3D printed box.*

6 Appendix

6.1 Near Field Communication

6.1.1 Introduction

NFC (Near Field Communication) is a set of short-range wireless communication technologies designed to offer light-weight and secure communication between two devices. The NFC Forum is responsible for publishing and maintaining a variety of standards relating to NFC technology. [6].

NFC operates at 13.56MHz, and there are two possible working models: *Passive Communication* and *Active communication*. In the first model, the "initiator" generates a small magnetic field that powers the target, meaning that the "target" does not require a power source. In the Active communication, or peer-to-peer, both devices are powered: each one alternately creates its own magentic field and the other act as a target, in continuous rotation.

6.1.2 Passive Communication with ISO14443A Cards

While the PN53x family of transceivers are compatible with a number of 13.56MHz RFID card standards, by far the most popular one is the ISO14443A. The most common ISO14443A compatible cards are the ones of the Mifare family, from NXP. Among them, Mifare Classic and Mifare Ultralight are the most frequently used NFC tags. While all ISO14443A cards share certain common characteristics defined by the standard, each set of Mifare chips (Classic, Ultralight, Plus, DESFire) has its own features and peculiarities. The two most common formats are described below.

- *Mifare Classic* cards are extremely common and have a 1K or 4K EEPROM. They have basic security features for each 64 byte (1K/4K cards) or 256 byte (4K cards) sector.
- *Mifare Ultralight* cards contains a 512 bytes EEPROM. These tags are inexpensive, often come in sticker format and are are frequently used for transportation, concert tickets, etc.

6.1.3 NFC Data Exchange Format

The NFC Data Exchange Format (NDEF) is a standardised data format that can be used to exchange information between any compatible NFC device and another NFC device or tag. The standard is maintained by the NFC Forum.

The NDEF format is used to store and exchange information like URI and plain text saved in the form of NDEF Records and incapsulated in NDEF Messages. NFC tags like Mifare Classic cards can be formatted as NDEF tags, ready to be accessed by any other NDEF compatible device. NDEF messages can also be used to exchange data between two active NFC devices in the "peer-to-peer" mode.

The NDEF standard includes a lot of different Record Type Definitions (RTDs) that define how information like URIs should be stored. Each NDEF device or tag can contained multiple RTDs. Standard RTD definitions are described in "NFC Record Type Definition (RTD) Specification? maintained by the NFC Forum.

NDEF Records

Bit 7	6	5	4	3	2	1	0
[MB]	[ME]	[CF]	[SR]	[IL]	[	TNF	]
[			TYPE LENGTH				]
[			PAYLOAD LENGTH				]
[			ID LENGTH				]
[			RECORD TYPE				]
[			ID				]
[			PAYLOAD				]

- TNF - Type Name Format Field The Type Name Format or TNF Field of an NDEF record is a 3-bit value that describes the record type, and sets the structure and content of the rest of the record. Possible record type names include:
  - 0x00 Empty Record
  - 0x01 Well-Known Record
  - 0x02 MIME Media Record
  - 0x03 Absolute URI Record
  - 0x04 External Record
  - 0x05 Unknown Record
  - 0x06 Unchanged Record
- IL - ID LENGTH Field The IL flag indicates if the ID Length Field is present or not. If this is set to 0, then the ID Length Field is ommitted in the record.
- SR - Short Record Bit The SR flag is set to one if the PAYLOAD LENGTH field is 1 byte (8 bits/0-255) or less. This allows for more compact records.
- CF - Chunk Flag The CF flag indicates if this is the first record chunk or a middle record chunk.
- ME - Message End The ME flag indicates if this is the last record in the message.MB: Message BeginThe MB flag indicates if this is the start of an NDEF message.
- Type Length Indicates the length (in bytes) of the Record Type field. This value is always zero for certain types of records defined with the TNF Field described above.
- Payload Length Indicates the length (in bytes) of the record payload. If the SR field (described above) is set to 1 in the record header, this value will be one byte long (for a payload length from 0-255 bytes). If the SR field is set to 0, this value will be a 32-bit value occupying 4 bytes.
- ID Length Indicates the length in bytes of the ID field. This field is present only if the IL flag (described above) is set to 1 in the record header.

- Record Type This value describes the 'type' of record that follows. The values of the type field must corresponde to the value entered in the TNF bits of the record header.
- Record ID The value of the ID field if an ID is included (the IL bit in the record header is set to 1). If the IL bit is set to 0, this field is ommitted.
- Payload The record payload, which will be exactly the number of bytes described in the Payload Length field earlier.

References

[1] Pn532 user manual. http://www.nxp.com/documents/user_manual/141520.pdf.

[2] .net micro framework: Spi, the high speed serial bus. http://social.technet.microsoft.com/wiki/contents/articles/22917.net-micro-framework-spi-the-high-speed-serial-bus.aspx.

[3] Lpc2478. http://www.nxp.com/documents/data_sheet/LPC2478.pdf.

[4] .net gadgeteer module builder's guide. https://gadgeteer.codeplex.com/downloads/get/665907.

[5] A guide to designing and building restful web services with wcf 3.5. https://msdn.microsoft.com/en-us/library/dd203052.aspx.

[6] About nfc. https://learn.adafruit.com/adafruit-pn532-rfid-nfc/about-nfc.