PROJET ETUDIANT 3

CAHIER DES CHARGES PERSONNEL

Pour la réalisation de notre projet, j'ai pour but de gérer une partie de la caisse, particulièrement la gestion du lecteur d'empreinte. Une fois à la caisse, le client validera son paiement par le biais de son empreinte digitale. Son identifiant se retrouvera enregistré à l'aide d'une IHM (Interface Homme Machine). La nouvelle information va alors être envoyée vers la base de données puis être comparé avec celle présent déjà présente. Ce qui permettra aux clients de pouvoir valider un potentiel paiement.

Pour ce faire, il ma fallu utiliser un ordinateur sous Windows 10, ainsi que le lecteur d'empreintes utilisant la norme RS232 branché directement à l'aide d'un câble USB sur l'un des ports souhaité de la machine.

PRINCIPAUX PROBLEMES RENCONTRES

- Si la communication avec le lecteur d'empreintes digitales n'est pas ferme lors de l'arrêt du processus, alors le capteur d'empreintes digitales cesse de fonctionner et il faudra attendre plusieurs minutes, ou effectuer plusieurs manipulation (débrancher/rebrancher câble USB du lecteur PC, redémarrer le PC afin de fermer tous les ports ouverts, changer de PC…) avant de pouvoir le réutiliser ce qui me ralentis considérablement pour les tests.

- Le lecteur d'empreintes est capricieux (il met du temps à envoyer une réponse s'il en envoie).

LOGICIEL DE PROGRAMMATION

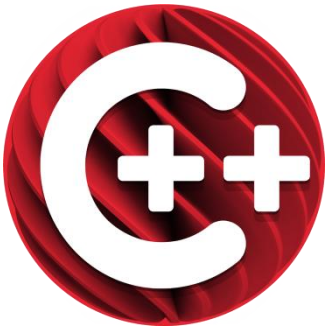
A) Delphi

Bien qu'il y avait des contraintes concernant les outils que nous allons utiliser, nous avons tout de même le choix de logiciels, tant qu'ils nous permettent d'effectuer exactement la même chose, voir mieux, que ceux recommandés.

J'avais tout d'abord opté pour Delphi afin de découvrir un nouvel environnement. Utilisant le langage Pascal et fournissant un IDE (Environnement de développement intégré) pour le développement d'application rapide de logiciel de bureau, mobile, web et console. Il est actuellement maintenu et développé par Embarcadero Technologies. Cependant, malgré une certaine volonté je n'ai pu réussir à faire ce que je voulais.

B) C++ Builder

J'ai alors décidé de rester sur le lange C++, et ainsi d'utiliser C++ Builder, venant également d'Embarcadero.L'ayant déjà utilisé plusieurs fois pour nos projet scolaire, je n'avais aucun mal à être à l'aise avec son interface.C'est d'ailleurs le logiciel que j'ai le plus utilisé tout au long de ce projet.



MATERIELS

A) Câble USB A mâle/micro USB B mâle

Afin de connecter le lecteur d'empreinte à la tour de l'ordinateur, j'ai utilisé un câble équipé d'un connecteur micro USB qui servira d'intermédiaire entre les deux appareils.

B) Lecteur d'empreinte digitale

Concernant le lecteur d'empreinte digitale celui-ci est le modèle GT511-C3.

Bien évidemment, j'avais à ma disposition un clavier, une souris ainsi qu'un écran. Ci-dessous, une vue d'ensemble concernant les éléments cités précédemment s'y trouve.

C) Vue d'ensemble

LECTEUR D'EMPREINTE DIGITALE

A) Caractéristiques

This device is one chip module with;

- fingerprint algorithm
- optical sensor

The major functions are the followings.

- High-accuracy and high-speed fingerprint identification technology
- Ultra-thin optical sensor
- 1:1 verification, 1:N identification
- downloading fingerprint image from the device
- Reading & writing fingerprint template(s) from/to the device
- Simple UART & USB communication protocol

* In the idle state, please turn off the whole fingerprint module. The sensor's metal frame can be integrated with a touch IC to wake up the module.

Technical Specification

Item	Value
CPU	ARM Cortex M3 Core
Sensor	optical Sensor
Effective area of the Sensor	14 x 12.5(mm)
Image Size	202 x 258 Pixels
Resolution	450 dpi
The maximum number of fingerprints	200 fingerprints
Matching Mode	1:1, 1:N
The size of template	496 Bytes (template) + 2 Bytes (checksum)
Communication interface	UART, default baud rate = 9600bps after power on USB Ver1.1, Full speed
False Acceptance Rate (FAR)	< 0.001%
False Rejection Rate(FRR)	< 0.1%
Enrollment time	< 3 sec (3 fingerprints)
Identification time	< 1.0 sec (200 fingerprints)
Operating voltage	DC 3.3~6V
Operating current	< 130mA

Le capteur d'empreintes utilise une liaison rs232, c'est une norme standardisant une voie de communication de type série, et permet une communication asynchrone et duplex entre deux équipements. Il est plus communément appelé « port série » et sous Windows, ils sont désignés par les noms COM1, COM2, COM3, etc.

B) Déroulement de la communication

Pour gérer la communication avec le capteur d'empreinte digital, j'ai utilisé trois éléments :

- La documentation du capteur
- Le logiciel de démonstration
- Le logiciel SerialPortMonitor

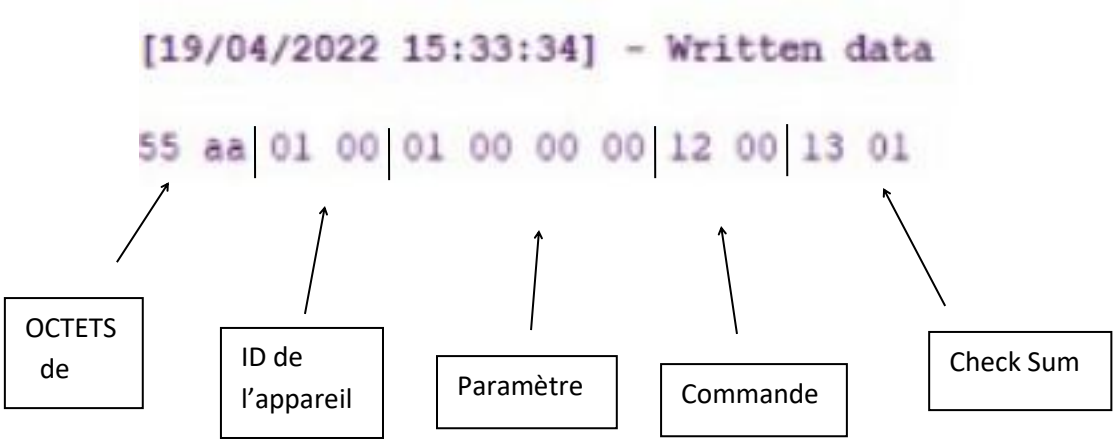
Tout d'abord il m'a fallu connaître grâce à la documentation du lecteur d'empreinte la structure des trames a envoyer. Les paramètres sont spécifiques pour chaque commandes. De plus, nous utilisons une notation en hexadécimal.

Structure général

Command Packet (Command)

OFFSET	ITEM	TYPE	DESCRIPTION
0	0x55	BYTE	Command start code1
1	0xAA	BYTE	Command start code2
2	*Device ID*	WORD	Device ID: default is 0x0001, always fixed
4	*Parameter*	DWORD	Input parameter
8	*Command*	WORD	Command code
10	*Check Sum*	WORD	Check Sum (byte addition) OFFSET[0]+...+OFFSET[9]=*Check Sum*

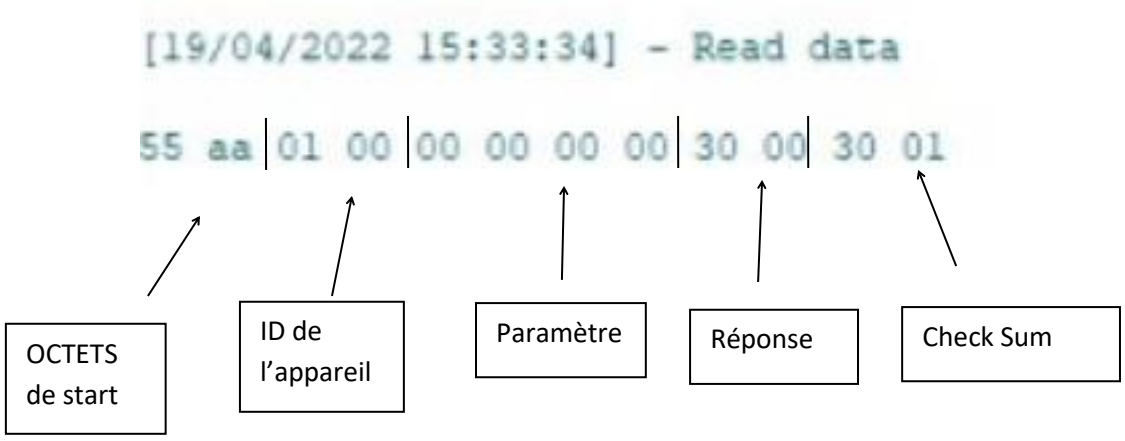
Ci-dessous, nous avons un exemple pour une trame d'envoi / requête :



Response Packet (Acknowledge)

OFFSET	ITEM	TYPE	DESCRIPTION
0	0x55	BYTE	Response start code1
1	0xAA	BYTE	Response start code2
2	*Device ID*	WORD	Device ID: default is 0x0001, always fixed
4	*Parameter*	DWORD	**Response == 0x30:** (ACK) Output Parameter **Response == 0x31:** (NACK) Error code
8	*Response*	WORD	**0x30:** Acknowledge (ACK). **0x31:** Non-acknowledge (NACK).
10	*Check Sum*	WORD	Check Sum (byte addition) OFFSET[0]+...+OFFSET[9]=*Check Sum*

Maintenant, ci-dessous nous avons un exemple pour une trame de réponse :



Simple rappel qui m'a tout de même été utile :

- BYTE ⟶ 1 OCTETS
- WORD ⟶ 2 OCTETS
- DWORD ⟶ 4 OCTETS

Il y a deux possibilités pour les octets de réponse :

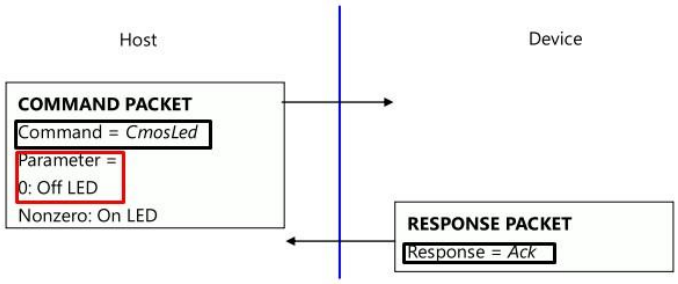
- 30 signifie Acknowledge (ACK)
- 31 signifie Non-acknowledge (NACK).

Donc dans l'exemple ci-dessous, la trame d'envoi correspond à la commande 12 qui est CMOS LED et a pour paramètre « 00 01 00 00 » qui permet d'allumer la LED du lecteur.

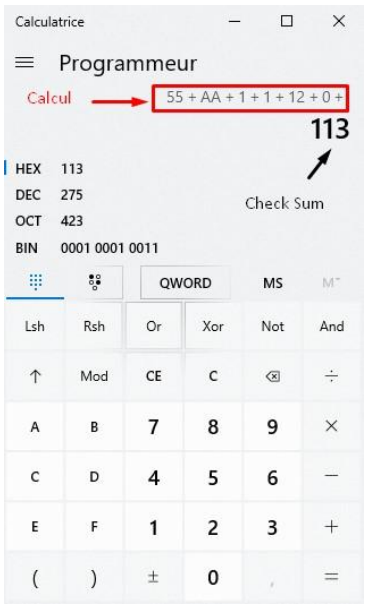
```
[19/04/2022 15:33:34] - Written data
55 aa 01 00 01 00 00 00 12 00 13 01
[19/04/2022 15:33:34] - Read data
55 aa 01 00 00 00 00 00 30 00 30 01
```

Tandis que la trame de réponse n'attend pas de paramètre donc il vaut 0 et a pour réponse 30 qui signifie ACK donc que la transmission est bel et bien réussi, effectuer correctement. Nous verrons plus en détaille l'utilité de la commandes CmosLed, qui est plus qu'important pour le bon déroulement de l'utilisation du lecteur d'empreinte.

5.4. CMOS LED control(CmosLed)

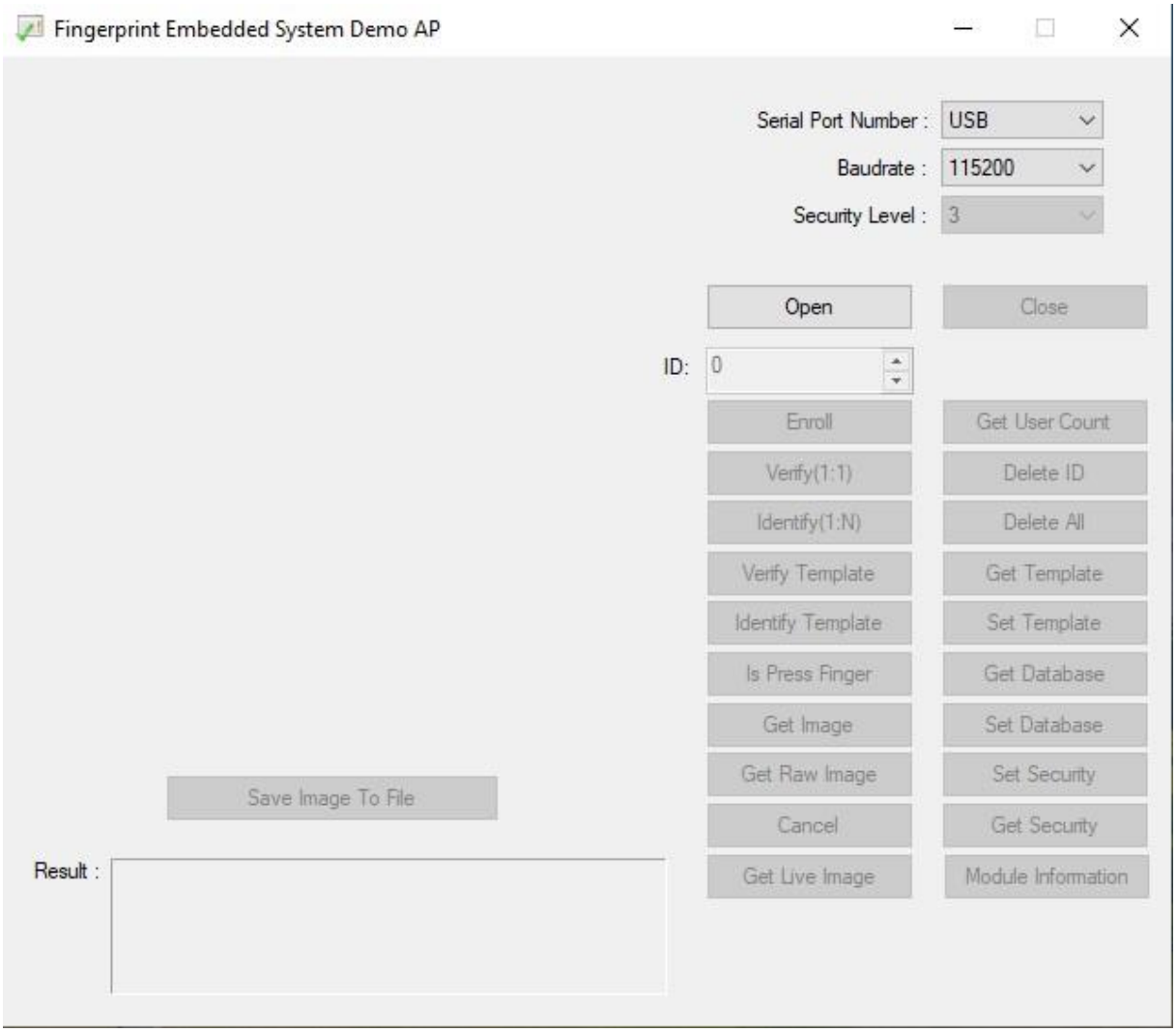


Il faut bien évidemment que pour la trame de requête, que le Check Sum correspond bien. Nous avons vu précédemment comment celui-ci se calcule, personnellement j'utilisais la calculatrice en mode programmeur présent chez les ordinateur disposant de Windows.



C) Logiciel démonstration du fournisseur

Une fois le lecteur d'empreintes digitale en notre possession autre le fait d'avoir accès à sa documentation, le fournisseur nous met à notre disposition un logiciel/application de démonstration, dévoilant le fonctionnement du lecteur. Pour l'utiliser, il suffira de choisir le numéro de port du capteur et sa vitesse ; puis de l'ouvrir grâce au bouton 'Open. Ensuite il suffira d'effectuer les commandes voulues tout en interagissant avec le lecteur d'empreinte digitale à portée de main.

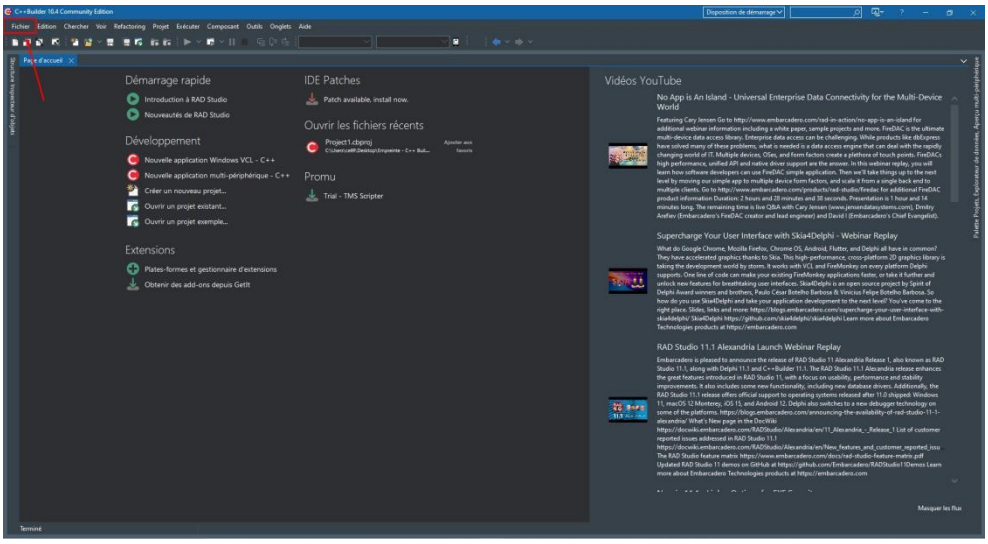


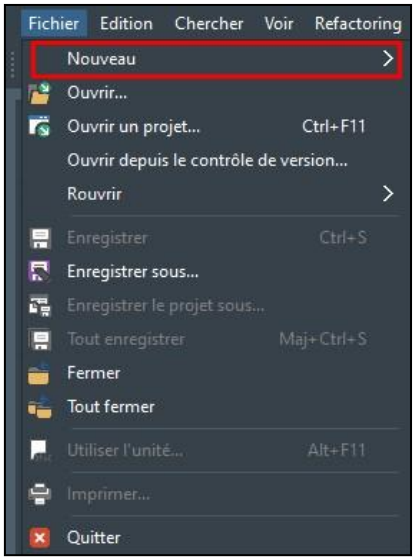
Avant de quitter l'application il faut impérativement cliquer sur 'Close' sinon il sera impossible de réutiliser le capteur avant un moment. Cette application ma été fortement utile afin de comprendre le fonctionnement du lecteur d'empreinte et d'éviter notamment les problèmes liés à l'appareil, et me faire gagner un certain temps.

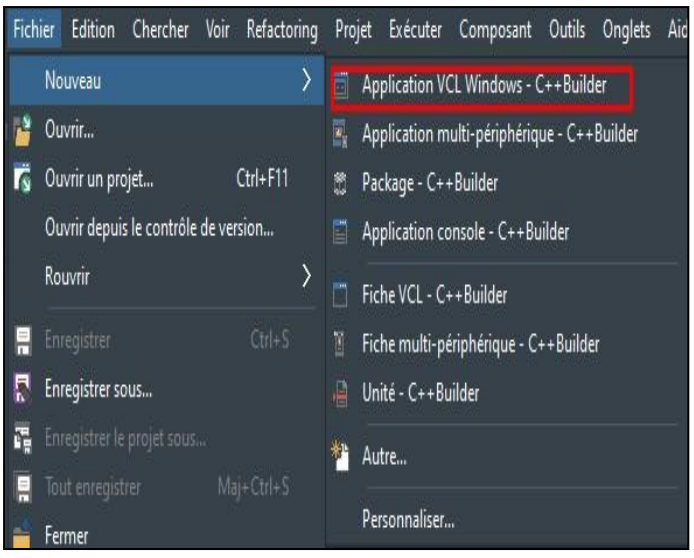
PRESENTATION DU CODE

A) Création du projet

Avant de commencer à coder il fallait que je choisisse le bon type de projet. Il y en existe une multitude. Pour ma part, j'ai opté pour une application VCL (Visual Component Library). Elle me permet d'utiliser un ensemble de composant visuels issu donc d'une librairie accessible à tous. Contenant une grande variété de classes visuelles/non visuelles et utilitaire, elle donne la possibilité de créer et développer une application Windows, web, base de données et console.





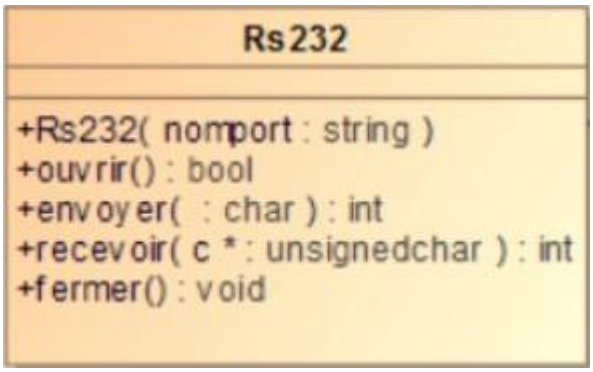


Voici ci dessus comment accéder à la création de ce genre de projet :

Fichier -> Nouveau -> Application VCL.

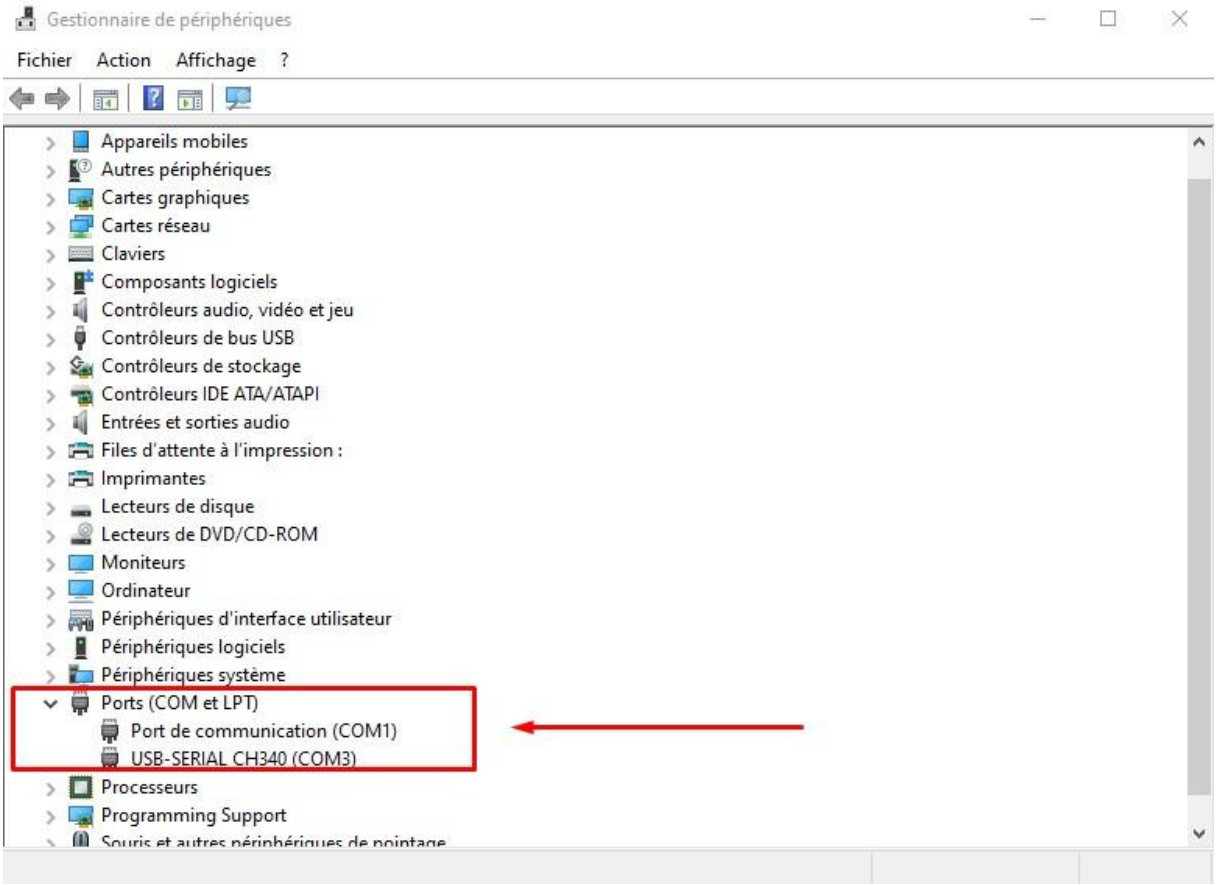
B) Gestion du code

Durant la conception de ce projet, j'ai utilisé la classe Rs232 donné par les professeurs.



I/Les Ports

Avant de commencer à interagir avec le lecteur d'empreinte. Il nous faut vérifier qu'on utilise le bon port. Pour se faire, nous allons directement confirmer cela au niveau du gestionnaire de périphériques de l'ordinateur. Sur l'image ci-dessous, nous voyons que le lecteur d'empreinte est branché au niveau du port COM3.



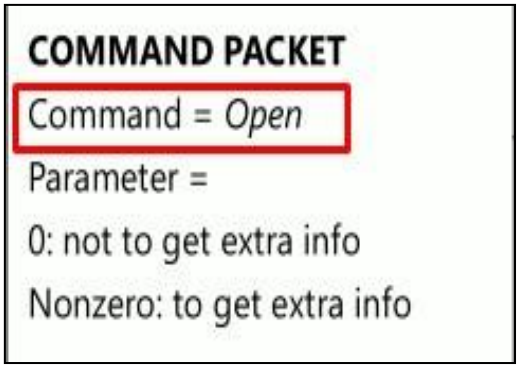
Nous allons pouvoir coder les entrées entrante et sortante au niveau de ce port COM3.

a) Port d'entrée

```
void __fastcall TTForm1::btnOuvrirClick(TObject *Sender)
{
    PortCom = new Rs232("COM3");
      PortCom->configurer(CBR_9600,8,NOPARITY,ONESTOPBIT);


  char trameOPEN[] = {0x55,0xaa,0x01,0x00,0x01,0x00,0x00,0x00,0x01,0x00,0x02,0x01}; //Initialisation

  for(int i=0; i<12; i++){

          PortCom->envoyer(trameOPEN[i]);
               textResultat->Text = "PORT OPEN";

  }

          for(int i=0; i<12; i++){

               PortCom->recevoir(trameOPEN);

          }

       PortCom->ouvrir();
}
 //---------------------------------------------------------------------------
```

L'événement btnOuvrirClick exécute le code présent ci-dessus. J'utilise l'objet dynamique PortCom de la classe Rs232 en indiquant bien que je veux agir sur le port COM3. Je configure ce port avec une vitesse de 9600 bauds, 8 bits, sans parité et un seul bit de stop.

Grâce à la documentation du lecteur d'empreinte, j'ai pu créer la trame d'ouverture du port en faisant bien attention au niveau de la commande ainsi que du check sum. Cette même trame qui est stocké dans une variable char appelé trameOPEN[].



Number (HEX)	Alias	Description
01	*Open*	Initialization

```
char trameOPEN[] = {0x55,0xaa,0x01,0x00,0x01,0x00,0x00,0x00,0x01,0x00,0x02,0x01}
```

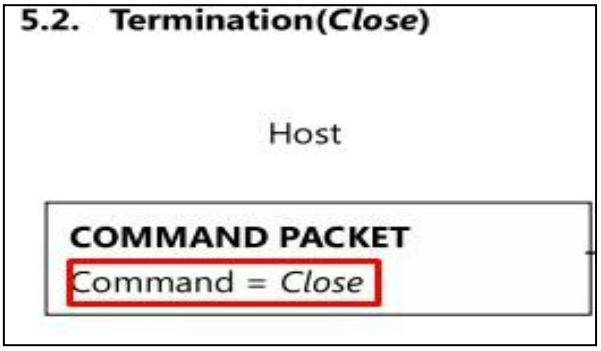
Puis j'ai fais appel à la méthode ouvrir() de la classe Rs232 grâce à l'objet PortCom crée.

b)Port de sortie

```
void __fastcall TTForm1::btnFermerClick(TObject *Sender)
{
    char trameCLOSE[] = {0x55,0xaa,0x01,0x00,0x00,0x00,0x00,0x00,0x02,0x00,0x02,0x01}; //Termination

  for(int i=0; i<12; i++){

        PortCom->envoyer(trameCLOSE[i]);

  }

        for(int i=0; i<12; i++){

              PortCom->recevoir(trameCLOSE);

        }

   PortCom->fermer();
   textResultat->Text = "PORT FERMER";

   delete PortCom;
}
```

Concernant la fermeture du port, l'événement btnFermerClick s'en occupe.

Grâce à la documentation du lecteur d'empreinte désormais, j'ai pu créer la trame d'ouverture du port en faisant bien attention au niveau de la commande de nouveau. Cette même trame qui est stocké dans une variable char appelé trameCLOSE[].




```
char trameCLOSE[] = {0x55,0xaa,0x01,0x00,0x00,0x00,0x00,0x00,0x02,0x00,0x02,0x01}; //Termination
```

Afin de fermer le port, je fais ici contrairement à la page précédente, appel à la méthode fermer() de la la classe Rs232, ceci est vraiment important car ça nous permet de ne pas avoir des soucis d'envoi de trame lors de la prochaine ouverture.

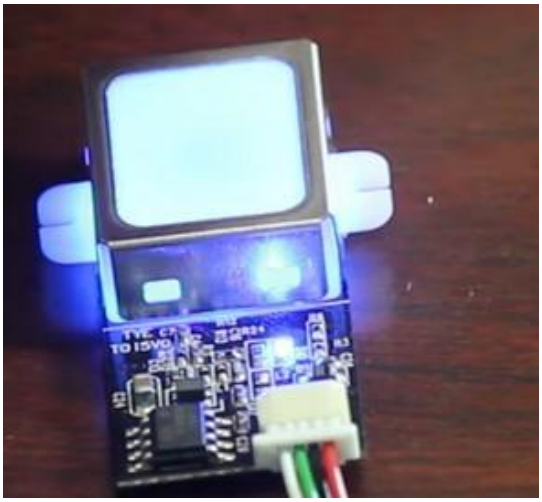
II/ Enregistrement de l'empreinte.

L'enregistrement de l'empreinte digitale se déroule en plusieurs étapes.

Tout d'abord, la LED du lecteur d'empreinte digitale doit être allumer avant chaque enregistrement. C'est la base fondamentale pour le bon déroulement de l'enregistrement. A savoir, que chaque enregistrement d'empreinte digitale se ressemblent. Seul la composition des trames changent. Il est nécessaire de répéter 3 fois l'enregistrement afin de valider l'empreinte.

```
//Allume la led
do{
    for(int i=0; i<12; i++){
        PortCom->envoyer(trameLedOn[i]);
    }
    for(int i=0; i<12; i++){
        PortCom->recevoir(&reponse[i]);
    }
    Sleep(1000);
}while(reponse[8]!=0x30);
```

On vérifié avant de passer à l'étape suivante que le 8 octet de la réponse reçu est bien égale à 30. C'est le tout premier exemple que j'ai effectué concernant l'analyse d'une trame au début de la rédaction de mon rapport. La LED nous avertit alors physiquement en devenant bleu, que la trame de requête à été reçu.



Cela fait, nous pouvons passer à l'étape suivante. Celle-ci démarrera l'enregistrement d'une nouvelle empreinte digitale. Pour cela nous avons besoin d'un identifiant d'empreinte libre. L'identifiant se trouve au niveau des paramètres de la trame, qui correspond au 4eme octet, modifiable jusqu'à 200. Ici, l'identifiant est le numéro 1. On envoie alors la trame EnrollStart.

```
char trameEnrollStart[] = {0x55,0xaa,0x01,0x00,0x01,0x00,0x00,0x00,0x22,0x00,0x23,0x01};
```

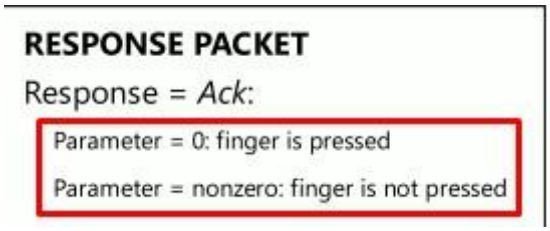
```
do{
    for(int i=0; i<12; i++){
    PortCom->envoyer(trameEnrollStart[i]);
    }
            for(int i=0; i<12; i++){
                PortCom->recevoir(&reponse[i]);
            }

     Sleep(1000);


}
while(reponse[8]!=0x30);
```

On place le doigt dont l'empreinte sera enregistré, sur le capteur par la suite. La trame «trameIsPressFinger» sera envoyée afin de vérifier qu'un doigt est présent sur le lecteur.

```
char trameIsPressFinger[] = {0x55,0xaa,0x01,0x00,0x00,0x00,0x00,0x00,0x26,0x00,0x26,0x01};
```

```
//Verifie si un doigt est present sur le lecteur
do{
    for(int i=0; i<12; i++){
        PortCom->envoyer(trameIsPressFinger[i]);
    }

    for(int i=0; i<12; i++){
    PortCom->recevoir(&reponse[i]);
    }
    Sleep(1000);
}while(reponse[4]!=0x00);
```

Il est important de répéter cette boucle tant que la réponse en paramètre n'est pas égale à 0. Cela signifierait qu'aucun doigt est présent sur le capteur du lecteur d'empreinte et que le client doit ainsi poser son doigt pour faire changer cette valeur afin qu'elle soit égale à 0.



Si le capteur reconnaît une pression, on envoie la trame correspondant au premier enregistrement d'empreinte.

```
// trame pour le 1er enregistrement
do{
    for(int i=0; i<12; i++){
    PortCom->envoyer(trameEnroll1[i]);
    }
            for(int i=0; i<12; i++){
            PortCom->recevoir(&reponse[i]);
            }

          Sleep(1000);

}while(reponse[8]!=0x30);
```

Si nous avons un réponse dont l'ACK (le 8ème octet) est égale à 30, nous pouvons passer à la capture de l'empreinte grâce à la trame «CaptureFinger».

```
char trameCaptureFinger[] = {0x55,0xaa,0x01,0x00,0x00,0x00,0x00,0x00,0x60,0x00,0x60,0x01};
```

```
do{
    for(int i=0; i<12; i++){
    PortCom->envoyer(trameCaptureFinger[i]);
    }
            for(int i=0; i<12; i++){
                PortCom->recevoir(&reponse[i]);
            }

     Sleep(1000);


}
while(reponse[8]!=0x30);
```

Afin d'avoir la certitude que le premier enregistrement est fini, on éteint la LED.

```
 //Eteindre la led
do{
    for(int i=0; i<12; i++){
        PortCom->envoyer(trameLedOff[i]);
    }
    for(int i=0; i<12; i++){
        PortCom->recevoir(&reponse[i]);
    }
    Sleep(1000);
}while(reponse[8]!=0x30);
```

Puis on vérifie qu'aucun doigt est présent sur le lecteur d'empreinte tant que les paramètres de la trame (4ème octet), n'est pas égale à 0.

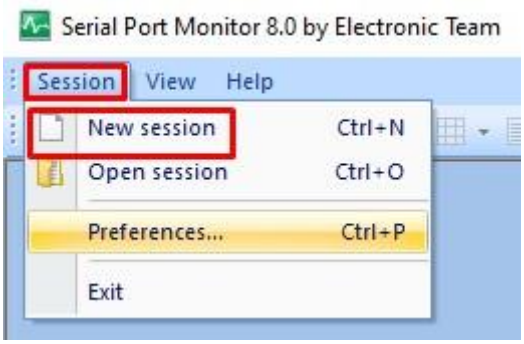
```
//Verifie si un doigt n'est pas present sur le lecteur
 do{
      for(int i=0; i<12; i++){
    PortCom->envoyer(trameIsPressFinger[i]);
    }

    for(int i=0; i<12; i++){
        PortCom->recevoir(&reponse[i]);
    }
  Sleep(1000);
}while(reponse[4]==0x00);
```

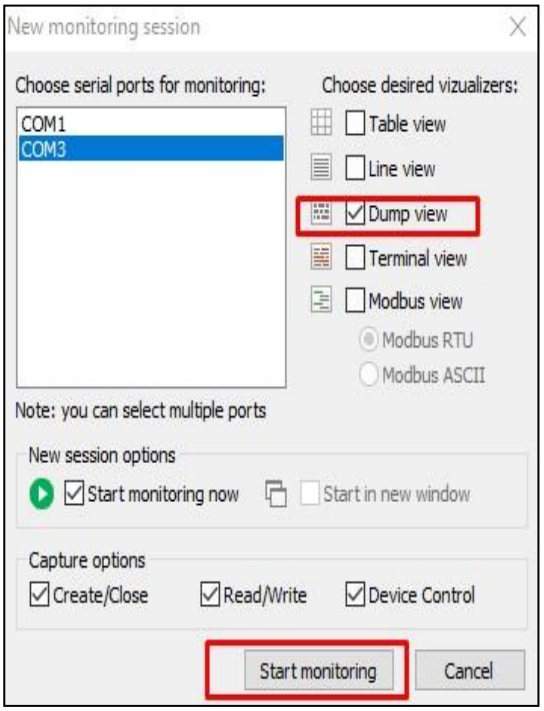
Puis on poursuit de nouveau ces étapes pour deux prochains enregistrement pour valider l'empreinte de l'individu.

C) Serial Port Monitor

Serial Port Monitor est un logiciel qui permet de voir les flux, ce qui se passe entre chaque port d'un ordinateur, un outil utilitaire pour Rs32/Rs422/Rs485. C'est un logiciel nouveau que j'utilise, et qui a été prépondérant lors du déroulement de l'enregistrement d'empreinte et dans la compréhension du fonctionnement du lecteur d'empreintes et dans la gestion des problèmes/erreurs.

Pour l'utiliser il faudra cliquer sur l'onglet session/new session puis remplir le formulaire qui s'affiche avec le bon port, puis coché Dump view et start monitoring now.





Puis on regarde les trames circulant lorsqu'on effectue une action sur le port en question.

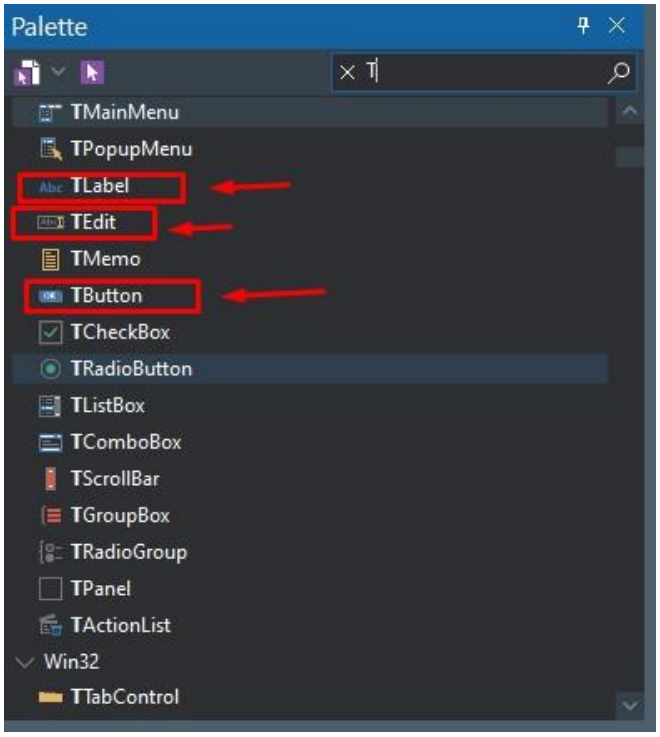
```
[30/05/2022 18:57:56] - Open port COM3 (C:

[30/05/2022 18:57:59] Written data (COM3)
    55 aa 01 00 00 00 00 00 02 00 02 01
[30/05/2022 18:57:59] - Close port COM3
```

D)IHM

L'interface Homme-Machine est le centre de tout le code. C'est en cliquant sur les différents boutons présent, que les méthodes codés s'exécutent. Grâce au faut que j'ai crée une application VCL, j'ai accès à une librairie de base, me permettant d'avoir déjà plusieurs composant à ma disposition.

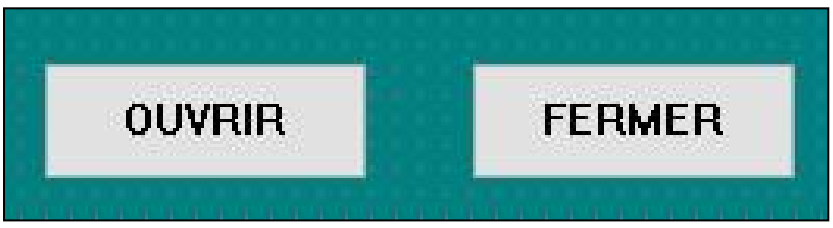
Afin d'avoir ces boutons en question, une section de la conception de l'IHM nous donnait accès à ces-derniers. Il suffisait tout simplement de saisi dans la barre de recherche : T + le type de bouton désiré.

Pour ma part, j'en ai utilisé principalement trois : TButton, TEdit ainsi qu'un TLabel.



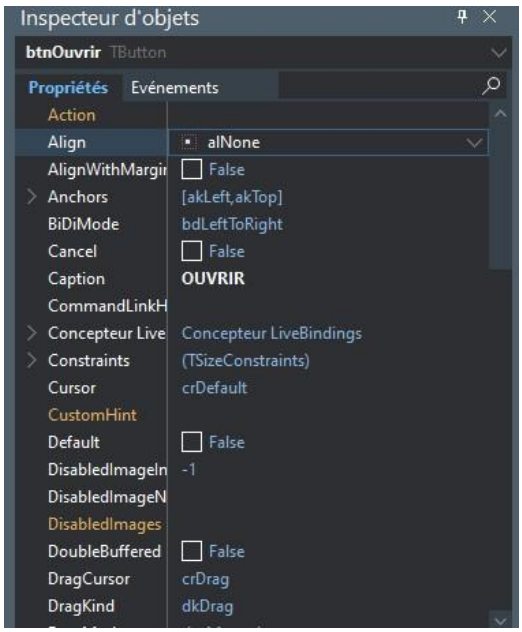
Ici par exemple, nous trouvons un TEdit.

Quant à ici, nous avons un exemple de deux TButton.

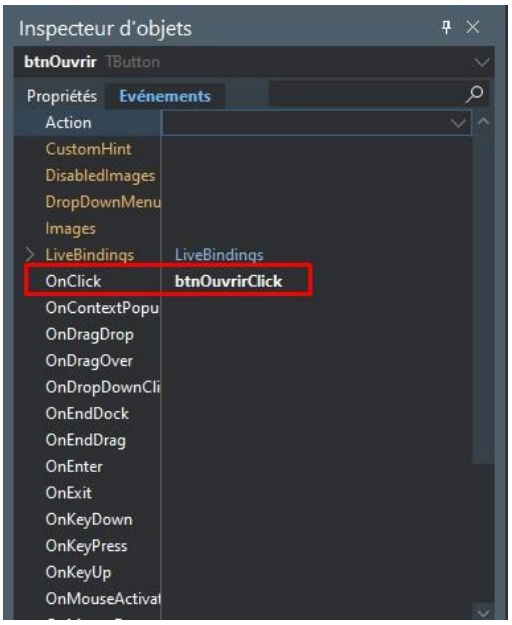


Chaque bouton possède des propriétés ainsi qu'un champ d'événement. Nous allons prendre l'exemple du TButton OUVRIR. Les propriétés impliquent directement sur le côté visuel du bouton. Que ce soit son nom, sa taille, sa couleur, où même la police d'écriture. Ils sont nombreux et modifiables comme bon nous semble.

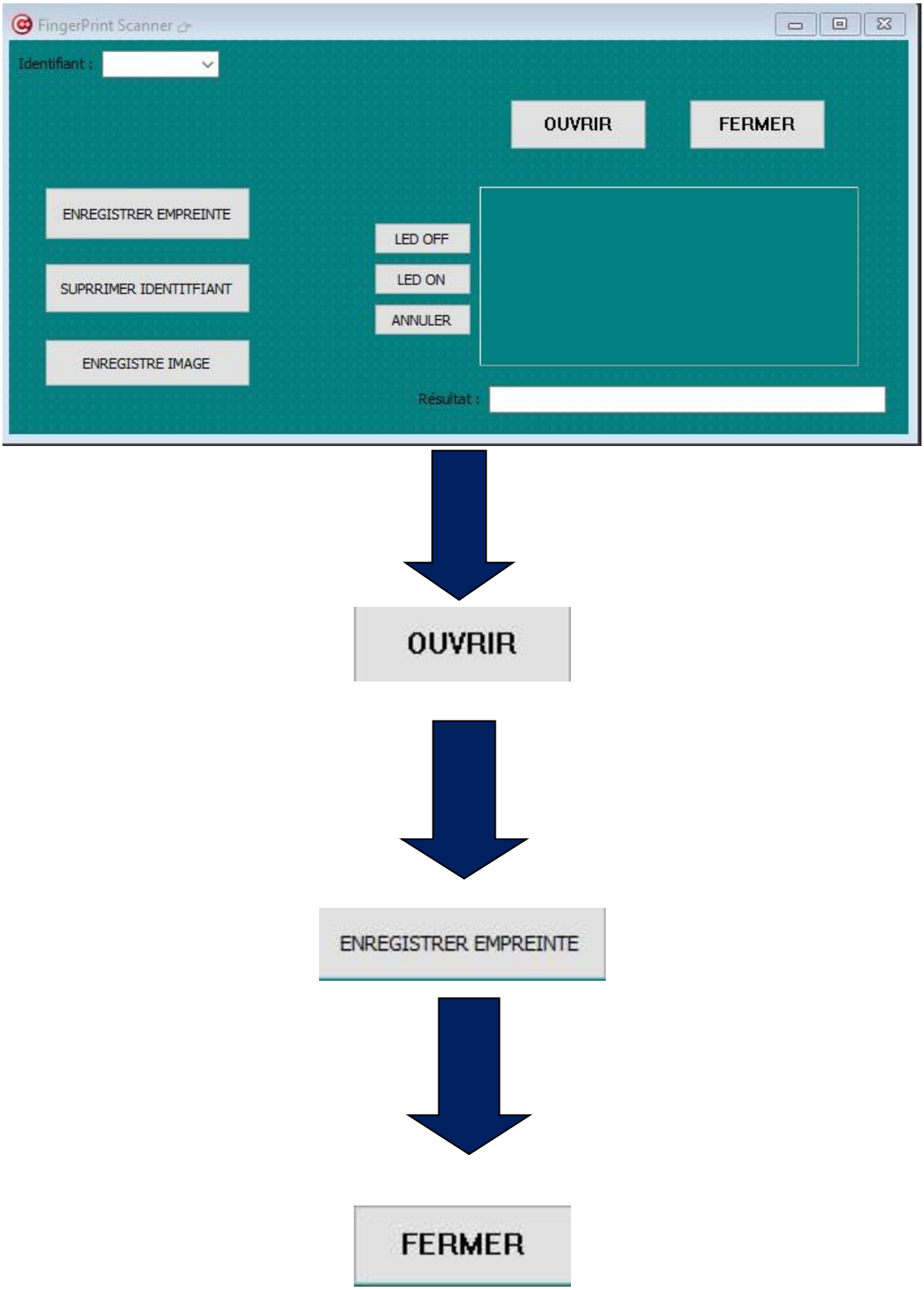
La majorité des boutons ont les mêmes propriétés générales cité ci-dessus. Il existe des exceptions bien évidemment, que je n'ai pas utilisé concernant ce projet.



Cela va de même pour les événements. Dans notre cas, celui utilisé est «OnClick», parlant de lui-même, il exécute l'événement dont le nom est «btnOuvrirClick» une fois qu'on clique dessus, et appliquera le code qui lui a été accordé.



Déroulement de l'IHM



CONSLUSION

Je suis ravi d'avoir pu travailler sur ce projet car c'est celui qui me plaisait le plus. La gestion d'un capteur d'empreinte digitale, la liaison rs232 est nouveau pour moi et très intéressant ; j'y ai beaucoup appris. De plus le fait que le projet soit en groupe, j'ai pu apprendre mes camarades via leur conseils et également en les soutenant dans leur partie tout en découvrant de nouvelles choses.

AMELIORATION POSSIBLE :

- Optimisation d'envoi de trame, essayer de réduire le temps entre chaque commande.

- Meilleure gestion des id d'empreintes libres.

- Gestion de la suppression des empreintes

- Gestion de l'enregistrement des empreintes

- Gestion des erreurs

- Esthétique de l'IHM.