ANQUETIN Romain
BLAIN Hugo

ESIREM
4A IT ILC

Compte Rendu
Communication sans fils

TP Systèmes de télécommunications sans-fils et applications:
Carte sans-contact MIFARE Classic

Table des matières

Table des figures

Introduction:

Notre objectif tout au long de nos séances de TP était de réaliser une carte multiservice pour les étudiants de l'ESIREM. Cette carte devait permettre de contenir l'identité de l'étudiant ainsi qu'un crédit pour la cafétéria ou les photocopies.
Pour ceci nous avions à notre disposition une carte sans contact type *NXP Mifare Classic* ainsi que de la documentation. Nous avions aussi un *coupleur USB CDC* ou un *coupleur TCP/IP*, fournis avec les librairies et la documentation nécessaires.
Avant tout, au cours des séances de TD en classe nous avons défini les architectures, applications et différentes clés de sécurité que nous allions utiliser.
Ce n'est qu'après cela, à l'aide du cours, de la documentation et du travail effectué en TD que nous avons mis en œuvre notre projet sous le logiciel *Qt Creator*.
Dans un premier temps nous parlerons de l'interface graphique pour l'utilisateur, de quoi elle est composée et comment s'en servir. Ensuite nous verrons l'architecture que nous suivrons pour stocker nos données dans la carte. Et pour finir nous verrons le fonctionnement de notre programme et comment fonctionne notre projet.

Partie 1: Développement de l'interface graphique

Introduction:

Dans cette première partie nous allons développer l'interface graphique pour la gestion de la carte sans-contact MIFARE Classic. Pour ceci, sous *Qt Creator*, nous allons utiliser l'onglet "Design". Nous pouvons préciser qu'en plus des informations disponibles sur cette interface, nous mettrons en place des indications visuelles sur le coupleur avec ses LEDs, elles correspondront au déclenchement ou à la fin de certaines actions.

1) Les attentes

Pour les besoins de ce projet, notre interface graphique devra permettre de:

- Ecrire et lire l'identité de la personne dans la carte (nom et prénom)
- Incrémenter et décrémenter des unités dans la carte avec la gestion d'un backup

2) Mise en place

Pour construire notre interface graphique nous devons dans un premier temps ajouter une Interface Utilisateur (UI) à notre projet, elle se situera dans le sous-dossier "Forms" de notre projet.

Figure 1: Architecture de notre projet

Ensuite, grâce à un simple glisser/déposer nous pouvons disposer les éléments qui nous seront nécessaires, comme par exemple:

- Des *QTextEdit*, utiles pour récupérer du texte saisis
- Des *QPushButton*, permettants de lancer certaines actions
- Des *QLabel*, pour afficher des informations
- Des *QSpinBox,* utilisées pour récupérer des nombres entrés

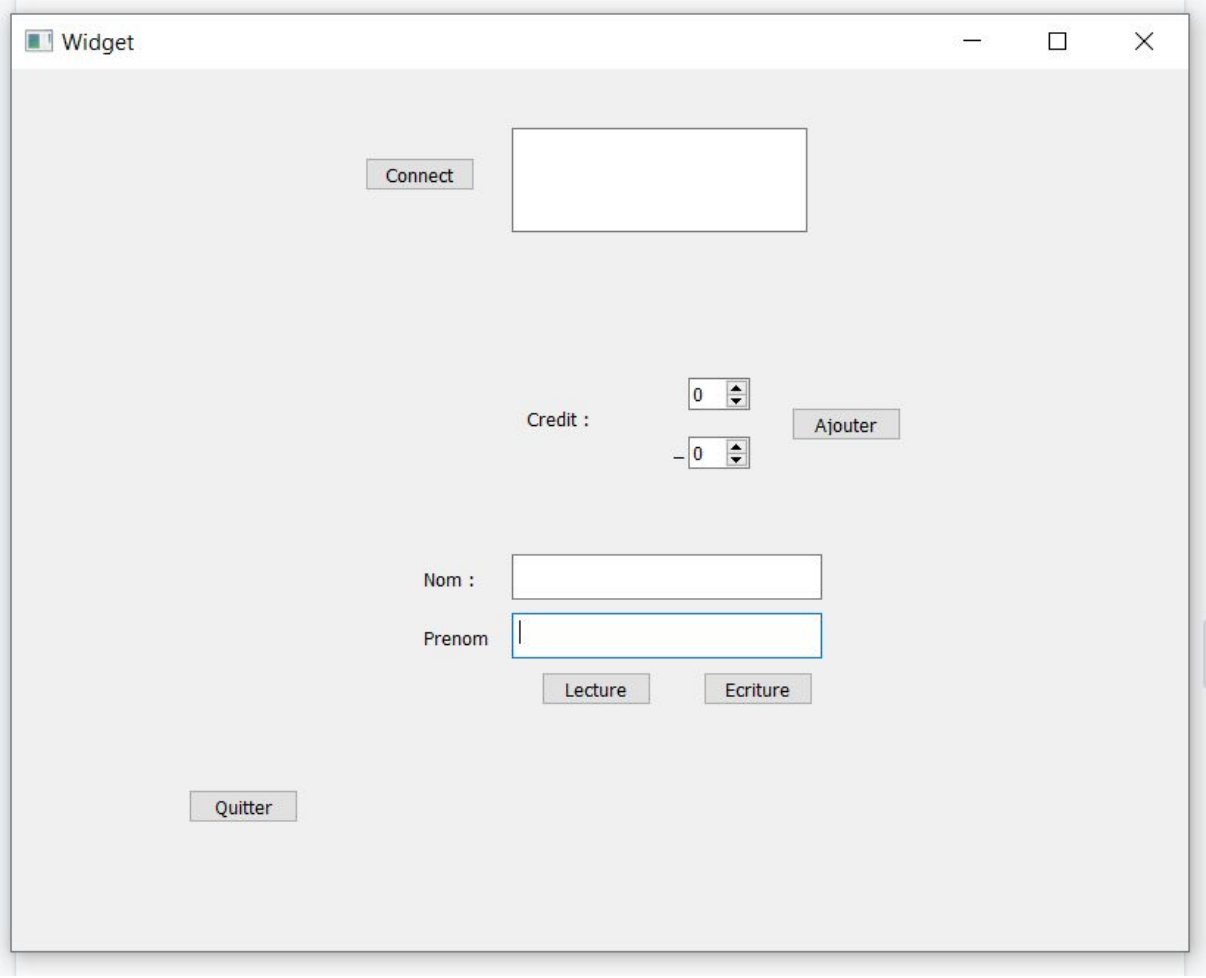

Figure 2: Interface Utilisateur (UI)

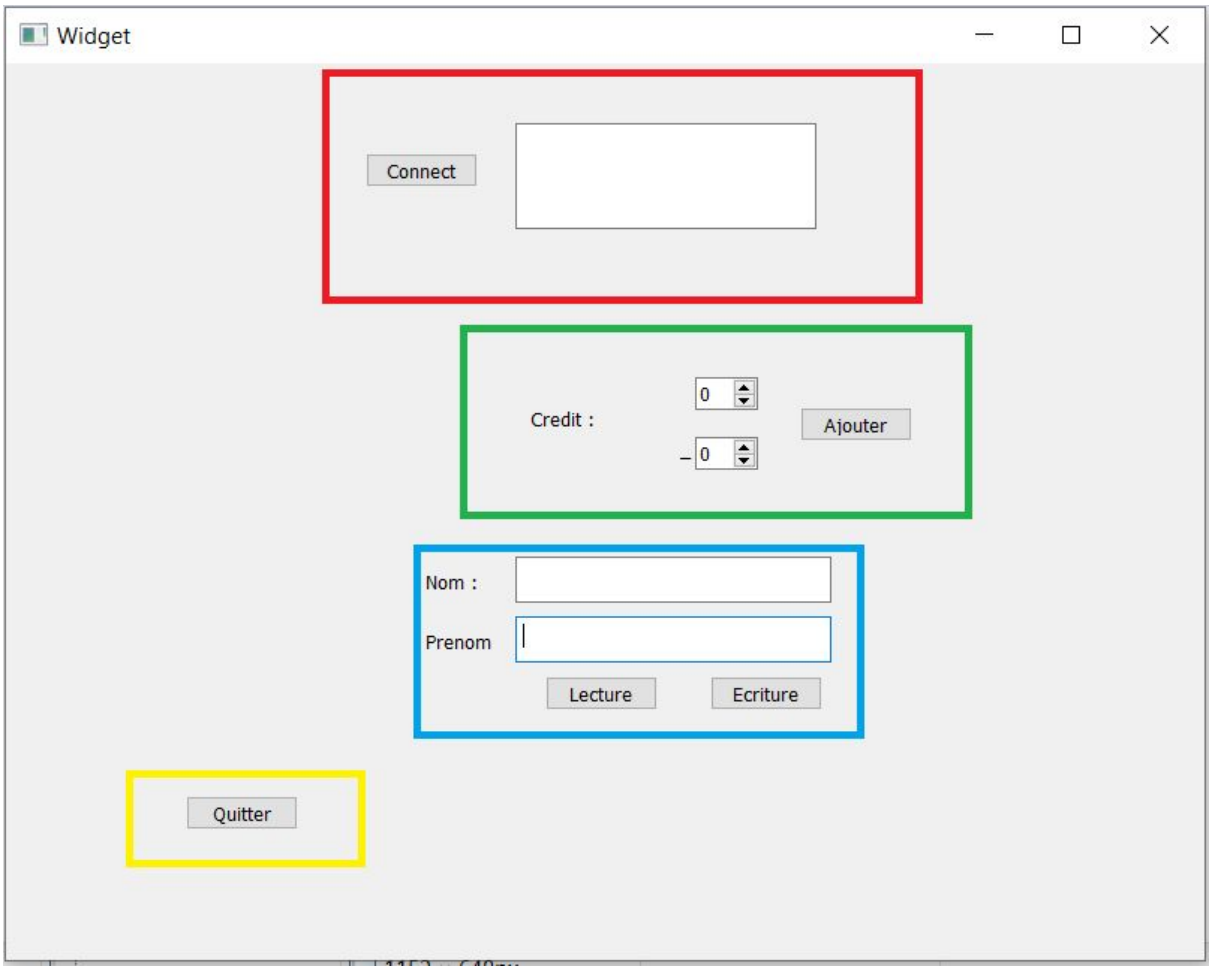

Figure 3: Interface Utilisateur découpée

Avant d'expliquer le fonctionnement on peut déjà remarquer que notre interface comporte 4 parties avec des fonctions différentes.

La première partie encadrée en rouge permet de lancer la communication grâce au bouton "Connect", la zone de texte à sa droite nous sert à afficher des informations.

Dans la zone verte juste en dessous, nous pouvons gérer tout ce qui est incrémentation et décrémentation des crédits de l'étudiant une fois la connexion établie. Le petit label "Crédit :" s'actualise pour afficher le nombre de crédits de l'étudiant. Les deux SpinBox recueillent le nombre de crédits que nous voulons ajouter ou enlever, celle du haut ajoute des crédits et celle du bas avec le petit symbole "-" en enlève. Enfin le bouton "Ajouter" permet de lancer l'incrémentation ou la décrémentation. Ici il faut préciser que nous avons choisis de n'utiliser qu'un seul bouton pour les deux actions car nous avons estimé que cela suffirait et nous ne voulions pas inutilement surcharger l'interface.

Dans la zone bleue on retrouve ce qui nous permet de gérer la lecture et l'écriture des nom et prénom. Elle dispose de deux zones distinctes pour le nom et prénom et de deux boutons pour lancer l'écriture et la lecture.

Enfin la zone jaune permet simplement de déconnecter la carte grâce au bouton "Quitter".

Partie 2: Architecture de la carte et fonctionnement

Introduction:

Après avoir vu dans la première partie, les attentes, les outils graphiques mis en place et comment les utiliser, nous verrons dans cette seconde partie comment cela fonctionne. Nous parlerons de l'architecture suivie et des commandes utilisées pour les différentes actions.

1) Identification du possesseur de la carte

Pour l'identification du propriétaire de la carte nous utiliserons le secteur 2 avec la configuration suivante:

- 2 blocs de données
- 1 blocs contenant le rôle du secteur "Identité"

Sector	Block	Data
2	11	KeyA+AccessBit+KeyB
	10	Nom
	9	Prenom
	8	« Indentite »

Figure 4: Secteur 2, Identité du propriétaire de la carte

L'authentification se fera avec les clés suivantes:

- Key A : A0 A1 A2 A3 A4 A5
- Key B : B0 B1 B2 B3 B4 B5

2) Compteur des crédits

Pour le compteur des crédits nous utiliserons le secteur 3 avec la configuration suivante:

- 2 blocs de compteur (dont un backup)
- 1 bloc contenant le rôle du secteur, “Porte monnaie"

Sector	Block	Data
3	15	KeyA+AccessBit+KeyB
	14	Compteur
	13	Backup Compteur
	12	« Porte Monnaie »

Figure 5: Secteur 3, porte monnaie du propriétaire de la carte

L’authentification se fera avec les clés suivantes :

- Key A : C0 C1 C2 C3 C4 C5
- Key B : D0 D1 D2 D3 D4 D5

3) Connexion

Au lancement de notre programme, dans le “main” on instancie un objet Widget de la classe que nous avons créée dans “widget.cpp”. C’est dans celle-ci que nous avons implémenté les méthodes que nous utiliserons.

```
5  int main(int argc, char *argv[])
6  {
7      QApplication a(argc, argv);
8      Widget w;
9      w.show();
10     return a.exec();
11 }
```

Figure 6: Partie de “main.cpp”

Une fois la carte sur le lecteur il faut cliquer sur le bouton “Connect”:

```
29  void Widget::on_Connect_clicked()
30  {
31
32      uint16_t status = 0;
33      MonLecteur.Type = ReaderCDC;
34      MonLecteur.device = 0;
35      status = OpenCOM(&MonLecteur);
36      qDebug() << "OpendCOM1" << status;
37
38      char version[30];
39      uint8_t serial[4];
40      char stackReader[20];
41
42      status = Version(&MonLecteur,version,serial,stackReader);
43      ui->Affichage->setText(version);
44      ui->Affichage->update();
45      std::string erreur_connexion = "64532";
46
47      if(erreur_connexion.compare(version) < 0 ){
48         lecture();
49         LireCredit();
50         LEDBuzzer(&MonLecteur,LED_ON);
51      }
52  }
```

Figure 7: Connect

Cette fonction nous permet de lancer la connexion entre l’ordinateur et le lecteur ligne 35 de la figure 7 avec la commande OpenCOM. Nous affichons les informations dans la zone texte à côté du bouton “Connect” grâce au lignes 43 et 44. Enfin, si la connexion est correctement établie nous lançons directement la lecture de l’identité du propriétaire de la carte et de ses crédits, lignes 48 et 49. Remarque, dans ce cas une information visuelle est activée sur le lecteur comme on peut le voir ligne 50, la LED du lecteur s’allume.

4) Lecture de l'identité

En connectant la carte comme vu juste avant ou en appuyant sur le bouton "Lecture", la fonction *lecture()* est lancée.

```
130 void Widget::on_Lecture_clicked()
131 {
132     lecture();
133 }
```

Figure 8: *on_Lecture_cliked()*

Figure 9: Fonction *lecture()*

La fonction *lecture()* se charge de la lecture du nom et prénom. On commence à activer le champ RF du lecteur avec la commande RF_Power_Control, ligne 89. On effectue la prise de contact de la carte selon la norme ISO14443A avec un Request standard avec la méthode ISO14443_3_A_PollCard, ligne 85 et on récupère la valeur de retour pour s'assurer qu'il n'y a pas eu d'erreur (status == 0).

Si aucune erreur n'a eu lieu, on commence par lire le nom. On charge les clés MIFARE nécessaires dans la mémoire du lecteur avec la méthode Mf_Classic_LoadKey, ligne 102 à 105. On récupère les données du bloc qui nous intéresse, ici secteur 2 bloc 10 pour le nom, avec la méthode Mf_Classic_Read_Block. On convertit ensuite le tableau de caractères récupéré en une variable de type String que nous avons appelée "result", pour ceci on utilise une boucle "for", voir ligne 107 à 109. Enfin on affiche le résultat dans l'interface graphique grâce aux widgets Nom.

Figure 10: Affichage du nom

Pour récupérer le prénom, la démarche est sensiblement la même, les clés sont déjà chargées et nous accédons toujours au secteur 2 mais cette fois au bloc 9. Nous affectons le résultat au widget correspondant au prénom et pour finir nous rafraîchissons l'affichage.

```
120         ui->Prenom->setText(result);
121         ui->Nom->update();
122         ui->Prenom->update();
123
124         LEDBuzzer(&MonLecteur,LED_GREEN_OFF);
```

Figure 11: Affichage, rafraîchissement et LED

On remarque qu'il y a aussi une indication visuelle pour cette action avec la commande LEDBuzzer.

5) Écriture de l'identité

Pour écrire dans la mémoire de la carte une nouvelle identité il faut commencer par remplir les deux champs texte en face de nom et prénom sur l'interface utilisateur. Ensuite, appuyer sur le bouton "Ecriture", ce qui lancera la fonction on_Ecriture_clicked.

```
64  void Widget::on_Ecriture_clicked()
65  {
66      RF_Power_Control(&MonLecteur,TRUE,0);
67      BYTE atq[2];
68      BYTE sak[1];
69      BYTE uid[12];
70      uint16_t uid_len = 12;
71      int status = ISO14443_3_A_PollCard(&MonLecteur,atq,sak,uid,&uid_len);
72      if(status==0){
73      QString stringdataNom = ui->Nom->toPlainText();
74      QString stringdataPrenom = ui->Prenom->toPlainText();
75      unsigned char dataNom[16];
76      unsigned char dataPrenom[16];
77      memcpy(dataNom,stringdataNom.toStdString().c_str(),16);
78      memcpy(dataPrenom,stringdataPrenom.toStdString().c_str(),16);
79      Mf_Classic_LoadKey(&MonLecteur,Auth_KeyA,key_A2,2);
80      Mf_Classic_LoadKey(&MonLecteur,Auth_KeyB,key_B2,2);
81      Mf_Classic_Write_Block(&MonLecteur,TRUE,10,dataNom,Auth_KeyB,2);
82      Mf_Classic_Write_Block(&MonLecteur,TRUE,9,dataPrenom,Auth_KeyB,2);
83      LEDBuzzer(&MonLecteur,BUZZER_OFF);
84      }
85  }
```

Figure 12: on_Ecriture_clicked()

Comme pour la lecture, on commence par activer le champ RF et effectuer la prise de contact de la carte avec les méthodes RF_Power_Control et ISO14443_3_A_PollCard et on récupère la valeur de retour pour s'assurer qu'il n'y a pas eu d'erreur (status == 0).

On récupère les données écrites dans les zones de textes de l'interface utilisateur sous forme de String et nous les transformons en tableau de caractère ligne 73 à 78. Nous devons les convertir avec la méthode memcpy car c'est sous cette forme que sont attendues les données dans la méthode pour écrire des données dans la carte. La méthode memcpy permet copier les valeurs de n octets à partir de l'emplacement pointé par la source directement sur le bloc de mémoire pointé par destination.

On charge les clés nécessaires avec la méthode Mf_Classic_LoadKey et on écrit les données recueillies avec la méthode Mf_Classic_Write_Block, ligne 79 à 82. Comme nous l'avons définie précédemment, nous stockons le nom dans le secteur 10, bloc 2 et le prénom dans le même secteur, bloc 9. Comme toujours une indication visuelle avec les LEDs est déclenchée, ligne 83.

6) Fonction *LireCredit()*

Etant donné que le fonctionnement de cette fonction est très semblable à celui de "*lecture()*", nous ne rentrerons pas trop dans les détails. Nous pouvons simplement dire que comme précédemment nous devons charger les clés dans la mémoire du lecteur, lire les données au bon endroit dans la mémoire de la carte (secteur 3, bloc 14 comme défini plus haut), convertir les données sous une variable de type String et afficher l'information. On indique que la lecture des crédits se termine à l'aide d'une LED du lecteur.

Cependant, petite spécificité, afin de lire les crédits nous utilisons la méthode Mf_Classic_Read_Value et non pas Mf_Classic_Read_Block comme pour le nom et le prénom. En effet, puisque la donnée que nous voulons récupérer est un nombre, il faut utiliser Mf_Classic_Read_Value qui permet de lire la valeur contenue dans un bloc de la carte.

```
135 void Widget::LireCredit(){
136     Mf_Classic_LoadKey(&MonLecteur,Auth_KeyA,key_C2,3);
137     Mf_Classic_LoadKey(&MonLecteur,Auth_KeyB,key_D2,3);
138        uint32_t data;
139     Mf_Classic_Read_Value(&MonLecteur,TRUE,14,&data,Auth_KeyA,3);
140     QString result = "Credit : "+ QString::number((int)data);//QString::fromUtf8((char*)data);
141
142     ui->Credit->setText(result);
143     LEDBuzzer(&MonLecteur,LED_GREEN_OFF);
144
145 }
```

Figure 13: Fonction *LireCredit()*

7) Bouton “Ajouter”, incrémentation/décrémentation

Dernière partie majeure de notre projet, l’incrémentation et la dépréciation des crédits. Pour cela, l’interface utilisateur comporte deux petite zones permettant à l’utilisateur d’indiquer combien de crédit il souhaite ajouter ou enlever et d’un bouton “Ajouter” pour valider.

```
147 void Widget::on_Ajouter_clicked()
148 {
149     uint32_t add = ui->boxadd->value();
150     uint32_t sub = ui->boxsub->value();
151     ui->boxsub->setValue(0);
152     ui->boxadd->setValue(0);
153     int tot = add - sub;
154 
155     RF_Power_Control(&MonLecteur,TRUE,0);
156     BYTE atq[2];
157     BYTE sak[1];
158     BYTE uid[12];
159     uint16_t uid_len = 12;
160 
161     Mf_Classic_LoadKey(&MonLecteur,Auth_KeyA,key_C2,3);
162     Mf_Classic_LoadKey(&MonLecteur,Auth_KeyB,key_D2,3);
163 
164     int status;
165     status = ISO14443_3_A_PollCard(&MonLecteur,atq,sak,uid,&uid_len);
166     if(status == 0){
167 
168         if(tot < 0){
169          tot = tot*-1;
170          uint32_t to=tot;
171             Mf_Classic_Decrement_Value(&MonLecteur,TRUE,13,to,14,Auth_KeyB,3);
172             Mf_Classic_Restore_Value(&MonLecteur,TRUE,14,13,Auth_KeyB,3);
173         }else{ //pb
174             uint32_t to=tot;
175             Mf_Classic_Increment_Value(&MonLecteur,TRUE,13,to,14,Auth_KeyB,3);
176             Mf_Classic_Restore_Value(&MonLecteur,TRUE,14,13,Auth_KeyB,3);
177         }
178         LireCredit();
179         LEDBuzzer(&MonLecteur,BUZZER_OFF);
180     }
```

Figure 14: Méthode lorsqu’on clique sur le bouton “Ajouter”

Lorsqu’on clique sur le bouton “Ajouter” c’est la méthode de la figure 14 qui est lancée. On commence par récupérer les valeurs des deux SpinBox (une pour ajouter et une pour enlever) et on les stocke dans deux variables “add” et “sub”. On en profite pour “vider” les SpinBox en les remettant à 0, ceci est plus agréable pour les utilisateurs. On calcule “add - sub” et on stocke le résultat dans une variable “tot”, ainsi on a le total que nous devons incrémenter ou décrémenter. Pour donner des exemples si add = 0 et sub = 5, tot = -5, on sait que nous devrons donc décrémenter 5 unités au crédit du propriétaire de la carte. Autrement si add = 3 et sub = 0 nous saurons que cette fois-ci nous devons incrémenter les crédits de 3.

Ensuite nous activons le champ RF du lecteur avec la commande RF_Power_Control, ligne 155 et nous chargeons les clés nécessaires avec Mf_Classic_LoadKey aux lignes 161 et 162. Nous effectuons la prise de contact de la carte selon la norme ISO14443A avec un Request standard avec la méthode ISO14443_3_A_PollCard et nous vérifions qu'il ne s'est pas produit d'erreur avec "status == 0".

Si aucune erreur ne s'est produite lors de la connexion, on regarde le signe de la valeur de la variable "tot", ligne 168. Comme dit plus tôt, une valeur positive signifie que nous devrons procéder à une incrémentation alors qu'une négative correspond à une décrémentation.

Après vérification, si nous devons faire une décrémentation, on commence par inverser la valeur de "tot" pour qu'elle redevienne positive. On convertit "tot" en uint32_t car c'est le type attendu pour la méthode qui va suivre, ligne 170. On utilise la méthode Mf_Classic_Decrement_Value afin de décrémenter de la valeur souhaitée le bloc indiqué, dans notre cas c'est le bloc 14 du secteur 3 qui joue le rôle de compteur. Il faut noter que cette méthode, en plus d'effectuer cette opération sur le bloc indiqué, stocke également le résultat de cette décrémentation dans une seconde zone mémoire, le backup, qui correspond au bloc 13 du même secteur. Ensuite, on restaure la valeur du compteur à partir de celle du backup avec la méthode Mf_Classic_Restore_Value. En effet, comme vu dans les TDs, on a besoin d'un backup pour le débit crédit d'unités. Il nous permet de toujours garder une bonne information dans la carte en cas de coupure de courant ou de dysfonctionnement quelconque.

Pour ce qui est de l'incrémentation, on utilise le même principe mais on utilise cette fois-ci la méthode Mf_Classic_Increment_Value plutôt que Mf_Classic_Decrement_Value, elle, permet à l'inverse de l'autre d'incrémenter de la valeur souhaitée.

Pour finir on appelle la fonction *LireCredit()* afin d'actualiser l'affichage sur l'interface utilisateur et on indique toujours avec la LED du lecteur que l'action vient bien de se terminer, ligne 178 et 179.

8) Bouton “Quitter”

Une fois que l’utilisateur a terminé il peut appuyer sur le bouton “Quitter”, celui-ci appelle la méthode on_Quitter_clicked().

```
55  void Widget::on_Quitter_clicked()
56  {
57      int16_t status =0;
58      RF_Power_Control(&MonLecteur,FALSE,0);
59      status= LEDBuzzer(&MonLecteur,LED_OFF);
60      status = CloseCOM(&MonLecteur);
61      qApp->quit();
62  }
```

Figure 15: Méthode on_Quitter_clicked

Cette méthode permet de mettre fin à la connexion. On désactive le champ RF du lecteur avec la méthode RF_Power_Control, on éteint la LED du lecteur avec LEDBuzzer et on ferme la connexion entre le lecteur et l’ordinateur avec CloseCOM.

Conclusion

Pour conclure, au cours de ces séances de TPs nous avons pu réaliser notre projet qui était de réaliser une carte multiservices permettant de contenir l'identité du propriétaire ainsi qu'un crédit d'unité pour la cafétéria. Tout au long de ce projet nous avons pu manipuler des notions vues en cours, mettre en œuvre le travail préliminaire fait en TD et acquérir de nouvelles connaissances. Nous avons pu voir comment établir la connexion entre carte/lecteur et ordinateur/lecteur. Comment lire et écrire des informations sur une carte. Comment incrémenter et décrémenter et l'importance d'un backup ainsi que les clés de sécurité. Bien que nous ayons déjà travaillé sous *Qt Creator*, cela nous aura aussi permis de nous familiariser encore plus avec cet environnement de développement. Enfin on peut dire que notre programme fonctionne mais que de notre améliorations restent possibles.

Annexe

Dépot GitHub:

https://github.com/Mortpo/TP_TD_Systeme_de_telecommunication_sans_fils

Organisation: