

Responsable pédagogique	AF	AM	OP	<b>PM</b>	<div><b>TER</b></div> <div><b>TP Client Qt/C++ Modbus over IP</b></div>
Période	Sem1	Sem2	Sem3	<b>Sem 4</b>	
Volume horaire	Cours		TP		
			<b>8-12</b>		

Indicateur temporel (hors rédaction du compte-rendu) :

questions	3-4h	4-12h
1-2		
3-4		

Documents à rendre : **Compte-rendu** selon modèle TP, contenant en plus les sources (en Annexe), les captures d'écran **pertinentes (requête / réponse pour un paramètre modifié par exemple)**. Des résultats obtenus sous forme de photos et / ou screen validant le fonctionnement du client et serveur pourront aussi être joints.

**BUT** : Etudier, coder des méthodes d'interface C++/Qt et mettre en œuvre des échanges client / serveur modbus sur IP. Etudier, modifier les différentes zones d'une cartographie mémoire (map) Modbus.

Note : Vous travaillerez notamment pour la mise au point (1<sup>er</sup> temps) de vos méthodes C++ sur un serveur émulant le système en adresse ip 127.0.0.1 ! (plus simple dans un premier temps) **Lancer l'appliquatif Qt ModipButler**. (Cf figure 2)

Dans un 2ème temps, une fois que votre classe sera fonctionnelle, vous testerez votre programme (client non graphique) avec le système serveur du kart (programme serveur embarqué sur l'automate d'adresse ip 192.168.20.205).

**Pré-requis : Installer Qt !! La dernière version de préférence. Suivre le poly QT5 Intro de M.Menu.**

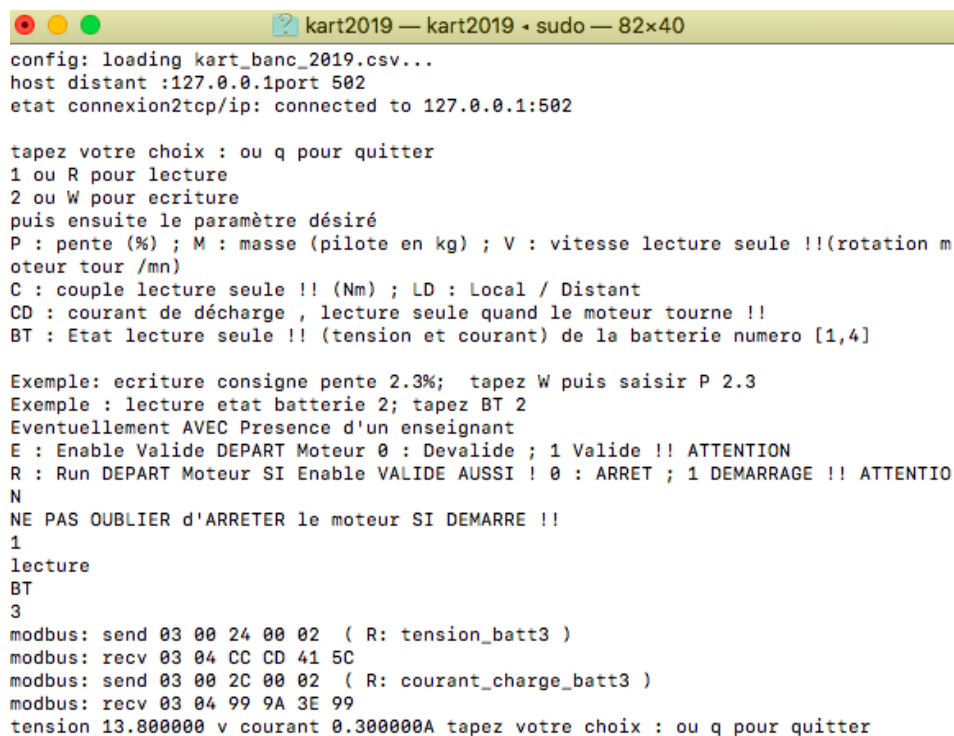
L'intégralité de librairie *QamModbus* n'est pas nécessaire pour ce TP, le minimum est normalement inclus dans le zip. Néanmoins, si vous voulez installer cette dernière aller sur rimatara Module RAD et prendre le fichier *modip\_src-151220171011.tar.gz*

## Partie 1 : Tests et prise en main des échanges clients/ serveur

### 1. Test avec le serveur (simulé par slave\_test ou Modipbutler ) et le client en mode texte

Lancer le serveur qui simule l'automate de préférence en ligne de commande par `sudo ./ModipButler`. (Cf figure 2). C'est lui le serveur qui répond aux requêtes clientes.

Lancer ensuite le client fourni, en tapant la commande : **sudo ./kart2019 kart\_banc\_2019.csv** Assurez-vous que le champ `host` du fichier `csv` contient bien l'adresse ip de la machine locale `127.0.0.1`



```
config: loading kart_banc_2019.csv...
host distant :127.0.0.1port 502
etat connexion2tcp/ip: connected to 127.0.0.1:502

tapez votre choix : ou q pour quitter
1 ou R pour lecture
2 ou W pour ecriture
puis ensuite le paramètre désiré
P : pente (%) ; M : masse (pilote en kg) ; V : vitesse lecture seule !!(rotation m
oteur tour /mn)
C : couple lecture seule !! (Nm) ; LD : Local / Distant
CD : courant de décharge , lecture seule quand le moteur tourne !!
BT : Etat lecture seule !! (tension et courant) de la batterie numero [1,4]

Exemple: ecriture consigne pente 2.3%; tapez W puis saisir P 2.3
Exemple : lecture etat batterie 2; tapez BT 2
Eventuellement AVEC Presence d'un enseignant
E : Enable Valide DEPART Moteur 0 : Devalide ; 1 Valide !! ATTENTION
R : Run DEPART Moteur SI Enable VALIDE AUSSI ! 0 : ARRET ; 1 DEMARRAGE !! ATTENTIO
N
NE PAS OUBLIER d'ARRETER le moteur SI DEMARRE !!
1
lecture
BT
3
modbus: send 03 00 24 00 02 ( R: tension_batt3 )
modbus: recv 03 04 CC CD 41 5C
modbus: send 03 00 2C 00 02 ( R: courant_charge_batt3 )
modbus: recv 03 04 99 9A 3E 99
tension 13.800000 v courant 0.300000A tapez votre choix : ou q pour quitter
```

Figure 1 : menu utilisateur du client fourni

Table	Address	Mask	Name	Comment	Hex	Display	Value
Coils							
Discrete Inputs							
Input Registers							
▼ Holding Registers							
▶ 040001	0000	----	local_distant	mode local ou distant	0000	Float	0
▶ 040003	0002	----	on_off_run	start stop	0000	Float	0
▶ 040005	0004	----	on_off_enable	Enable Variateur Parvex	0000	Float	0
▶ 040015	000E	----	vit_simu	vitesse moteur (tr/mn)	0000	Float	0
▶ 040017	0010	----	couple_simu	couple moteur Nm	0000	Float	0
▶ 040019	0012	----	masse_pilote	masse du pilote (kg)	0000	Float	80
▶ 040021	0014	----	temps_parcours	temps absolu du parcours	0000	Float	0
▶ 040023	0016	----	cons_pente	consigne de pente piste (%)	0000	Float	0
▶ 040025	0018	----	courant_batt1	courant reel dans batterie 1 ?	0000	Float	0
▶ 040027	001A	----	puissance_totale_batt	puissance totale 4 batteries en Watts	0000	Float	0
▶ 040029	001C	----	charge_elect_totale	charge Q (en coulomb:C ) des 4 batteries	0000	Float	0
▶ 040033	0020	----	tension_batt1	tension reel dans batterie 1	0000	Float	12
▶ 040035	0022	----	tension_batt2	tension reel dans batterie 2	0000	Float	14
▼ 040037	0024	----	tension_batt3	tension reel dans batterie 3	CCCD	Float	13.8
040038	0025				415C		
▶ 040039	0026	----	tension_batt4	tension reel dans batterie 4	3333	Float	13.2
▶ 040041	0028	----	courant_charge_batt1	courant reel dans batterie 1	0000	Float	1
▶ 040043	002A	----	courant_charge_batt2	courant reel dans batterie 2 ?	CCCD	Float	0.2
▼ 040045	002C	----	courant_charge_batt3	courant reel dans batterie 3 ?	999A	Float	0.3
040046	002D				3E99		
▶ 040047	002E	----	courant_charge_batt4	courant reel dans batterie 4 ?	CCCD	Float	0.4

**Figure 2 : Le serveur graphique ModipButler (simule l'automate)**

Vous noterez que les trames modbus (émises/reçues) s'affichent dans la fenêtre supérieure, avec une interprétation des codes de fonctions modbus.

## 2. Test avec le serveur (simulé par slave\_test ou Modipbutler ) et le client graphique ModipAsker

L'intérêt de ce client graphique baptisé *ModipAsker* (non demandé en développement dans ce TP) ; est de montrer une fenêtre de « debug » où l'on visualise les trames Modbus réelles !! Vous en profiterez pour parfaire votre connaissance des tables modbus. (Cf figure 3).

Internet Protocol

Server : 127.0.0.1 Port : 502 ☒ TCP ☐ UDP Close

Modbus Protocol Data Unit (PDU)

Modbus Application Protocol (MBAP)

Transaction Ident. (dec) : 6 ☒ Auto

Protocol Identifier (hex) : 0000

Length (dec) : 6 ☒ Auto

Unit Identifier (hex) : FF

Function : 3 - Read Holding Registers

Address : 19 ☒ dec ☐ hex

Number : 2 ☒ dec ☐ hex

Value : 0 ☐ dec ☒ hex

Data (hex) :

info: connection 127.0.0.1:502 closed  
 info: connected to 127.0.0.1:502  
 send: 00 05 00 00 00 06 FF 03 00 13 00 02  
 recv: 00 05 00 00 00 07 FF 03 04 42 A0 00 00

Send

**Figure 3 : Client graphique ModipAsker**

Vous noterez les champs de la valeur lue : ici 80 kg mis en format ieee732 float (42A0) 16

ModipButler

Load conf. Port : 502 Stop

tcp/ip: Client 16 connected  
 tcp/ip: Request from client 16  
 modbus: recv 03 00 13 00 02  
 modbus: FC 3, Address 0013, Number 2  
 modbus: send 03 04 42 A0 00 00

Table	Address	Mask	Name	Comment	Hex	Display	Value
Coils							
Discrete Inputs							
Input Registers							
Holding Registers							
040001	0000	----	local_distant	mode local ou distant	0000	Float	0
040003	0002	----	on_off_run	start stop	0000	Float	0
040005	0004	----	on_off_enable	Enable Variateur Parvex	0000	Float	0
040015	000E	----	vit_simu	vitesse moteur (tr/mn)	0000	Float	0
040017	0010	----	couple_simu	couple moteur Nm	0000	Float	0
040019	0012	----	masse_pilote	masse du pilote (kg)	0000	Float	80
040020	0013	----			42A0		

**Figure 4 : Réponse du serveur ModipButler**

### 3. Création du projet Qt du client

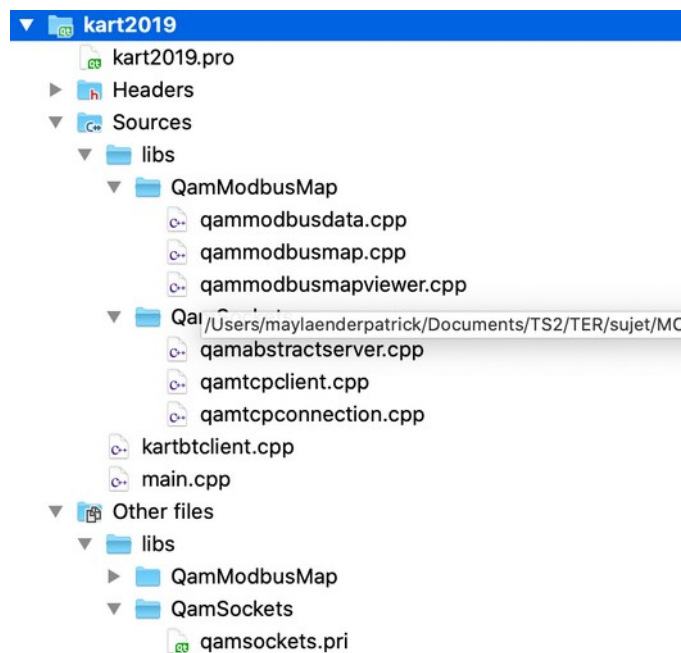
Decompresser le packet sourcesPourEleve.zip.

Vous allez partir du projet fourni **kart2019.pro**.

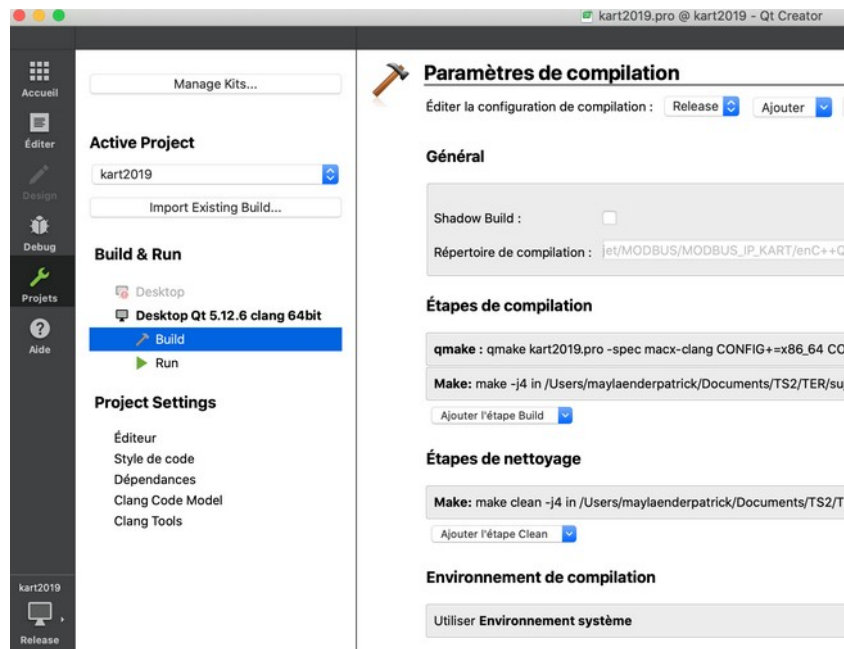
**L'ouvrir avec Qt par double clic, mais ne rien modifier dedans.**

**Il vous faudra aussi créer un kit (dans Qt) pour avoir une compilation adaptée à votre machine (type PC-Desktop64 bits en général [Cf figure 6](#) : installation d'un kit)**

Assurez-vous d'avoir les bonnes librairies *QamModbusMap* et *QamSockets* dans le sous répertoire librairie *lib*, relatif à votre projet. (CF figure 5: arborescence du projet)

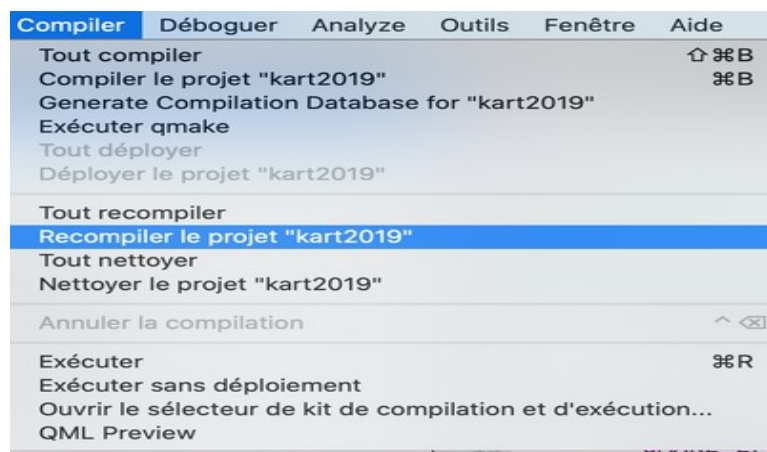


**Figure 5 : Arborescence du projet**



**Figure 6 : Installation d'un kit (compilateur, lieu) pour l'environnement**

A partir du code C++ fourni : classe ***kartbtclient.cpp*** / ***kartbtclient.h***, vous allez déjà compiler le programme (Cf figure 7) et vous inspirez de certaines méthodes déjà disponibles : lecture / écriture Mode Local/distant.



**Figure 7 : compilation**

- Etudier très sérieusement la documentation Doxygen (sous-répertoire Doc) fournie de QamModbus, fournie avec des exemples précis. **Concentrez vous notamment sur la classe *QamModbusMap* et *QamTcpClient***

**Ne rien modifier dans le main(). Vous ne coderez que dans le fichier *kartbtclient.cpp*** . Pour votre information, une instance de la classe *KartBtClient* est créée dans le main(), et tout le code va se « dérouler » dans la boucle infinie *app.exec()* qui vous fait basculer dans le monde de la programmation événementielle. Cf Module RAD en janvier...

#### 4. Implémentation de l'interface

Vous allez utiliser les fonctions modbus de la librairie ModbusQt développée par Alain Menu.

Le constructeur :

Quelques explications s'imposent...

Les étapes importantes sont d'abord la création d'une instance *m\_map* de *QamModbusMap* qui charge pour un client, une cartographie mémoire des registres modbus.

- Compléter dans *kartbtclient.cpp*, dans le constructeur , la création de *m\_map*.

```
m_map = new QamModbusMap( ..... , this ) ;
```

Vient ensuite une connexion de type signal/slot (technique de programmation propre à Qt Cf cours RAD à venir...) ; entre l'instance de la cartographie *map* et un signal *info()*

Cela permet un retour d'informations, par exemple pour savoir si la cartographie s'est correctement chargée ou non.

Ensuite on crée une instance *m\_client* de classe *QamSocket* afin de créer le client Modbus proprement dit.

- Compléter dans *kartbtclient.cpp*, dans le constructeur , la création de *m\_client*.

```
m_client = new QamTcpClient( ..... ) ;
```

Reste à faire la demande de connexion.

- Compléter dans *kartbtclient.cpp*, dans le constructeur , l'appel à la connexion socket TCP/IP *m\_client->sockConnect( ..... )* ;
- Compléter l'affichage du numéro de port , et adresse ip machine, ainsi que l'état de la connexion.





Vient ensuite la mise en place signal/slot entre la lecture au clavier d'un choix utilisateur, et la méthode (slot) *readConsole()* qui va devoir être compléter par vos soins.

Cette méthode permet d'implémenter le menu utilisateur demandé (CF figure 1)

Vous allez devoir coder une quinzaines de méthodes, dont les prototypes sont tous fournis dans l'interface *kartbtclient.h*.

- Compléter au fur et à mesure le menu. Dès que vous proposez une option ; codez en conséquence la méthode associée. Regardez bien le fichier interface *kartbtclient.h* (Cf Annexe 3)
- Compléter les méthodes privées d'interface permettant de lire / écrire les différents paramètres du Kart : Tension, courant Masse pilote etc..

**Conseil : utilisez l'accès aux méthodes de la classe QamModbusmap via, le membre agrégé *m\_map* !**

Pour bien situer votre classe, regarder les diagramme des classes fourni en annexe2.



## Annexe 1 : Fichier kart\_banc.csv

### (contenant la description de la cartographie)

**HOST;192.168.20.205 # adresse réelle de l'automate à coté du kart réel en bas de l'atelier**

#HOST;127.0.0.1

PORT;502

INFO;Kart Banc Test 2018 et Surveillance Batteries !

# Mise à jour PM 21/3/18

# holding registers

# commandes du Enable et Run + Mode local ou distant

40001;2; FFFF ;local\_dist; prise en main local = 0 - distant = 1; Float ;0.0

40003;2; FFFF ;on\_off\_run ; start stop; Float ;0.0

40005;2; FFFF ;on\_off\_enable ;Enable Variateur Parvex; Float ;0.0

#40011 et 40013 sont interdits car utilisés dans Asservissement Couple : Gain , Lag

# mesures des parametres dynamiques

40015;2; FFFF ;vit\_simu;vitesse moteur (tr/mn); Float ;0.0

40017;2; FFFF ;couple\_simu;couple moteur Nm; Float ;0.0

# parametres simulateur conduite + essai banc

40019;2; FFFF ;masse\_pilote;masse du pilote (kg);Float ;80.0

# temps du parcours

40021;2; FFFF ;temps\_parcours;temps absolu du parcours ; Float ;0.0

40023;2;FFFF ;cons\_pente;consigne de pente piste (%) ;Float ;0.0

# mesures courants de "decharges"

40025;2;FFFF ;courant\_batt1;courant reel dans batterie 1 ? ;Float ;0.0

40027;2;FFFF ;puissance\_totale\_batt;puissance totale 4 batteries en Watts ;Float ;0.0

40029;2;FFFF ;charge\_elect\_totale;charge Q (en coulomb:C ) des 4 batteries ;Float ;0.0

# mesures tensions batteries en charge et/ou utilisation !!

40033;2;FFFF ;tension\_batt1;tension reelle dans batterie 1 ;Float ;0.0

40035;2;FFFF ;tension\_batt2;tension reelle dans batterie 2 ;Float ;0.0

40037;2;FFFF ;tension\_batt3;tension reelle dans batterie 3 ;Float ;0.0

40039;2;FFFF ;tension\_batt4;tension reelle dans batterie 4 ;Float ;0.0

# mesures courants de charges !!

40041;2;FFFF ;courant\_charge\_batt1;courant reel dans batterie 1 ;Float ;0.0

40043;2;FFFF ;courant\_charge\_batt2;courant reel dans batterie 2 ? ;Float ;0.0

40045;2;FFFF ;courant\_charge\_batt3;courant reel dans batterie 3 ? ;Float ;0.0

40047;2;FFFF ;courant\_charge\_batt4;courant reel dans batterie 4 ? ;Float ;0.0



## Annexe 2 : Diagramme de classes de l'application



### Annexe 3 : interface kartbtclient.h

```
#ifndef KARTBTCLIENT_H
#define KARTBTCLIENT_H

#include <QObject>
#include <qamtcpclient.h>
#include <qammodbusmap.h>

#include <QSocketNotifier>
#include <QTimer>
#include <iostream>
using namespace std ;

class KartBtClient:public QObject
{
    Q_OBJECT
public :
    //init classe metier avec nom de fichier csv en arg

explicit KartBtClient(const QString& configFile, QObject *parent = nullptr);

//accesseurs

float get_cons_pente( ) { return cons_pente ; }
float get_masse_pilote() { return masse_pilote; }

//recupere tension batterie numero 1,4
float get_tension(int n){ return tension[n];}

//recupere courant charge batterie numero 1,4
float get_courant(int n){ return courant[n];}

//recupere le mode courant
int get_mode() { return mode_local_distant; }

//recupere la vitesse arbre roue tr/mn
float get_vitesse_simu() { return vitesse_simu; }
//mets consigne de pente
void set_cons_pente(float pente) { cons_pente = pente; }

//mets une masse
void set_masse_pilote(float m) { masse_pilote = m ; }

//mets un mode
void set_mode(int mode) { mode_local_distant = mode ;}

//Enable
void setEnable(int en) { enable = en;}
int getEnable(){ return enable;}

//Run
void setRun(int en) { enable = en;}
```



```

    int getRun(){ return enable;}

public slots:
    void info(const QString& src, const QString& msg ) ;
    void readConsole();

signals:
    void quit() ;    // signal à émettre pour terminer l'application...

private:
    void qSleep(int ms);
    QamModbusMap*          m_map ;
    QamModbusMap::PrimaryTable m_table ;

    QamTcpClient*          m_client ;
    QSocketNotifier*       m_notifier ;

    //données du kart
    float couple_simu, cons_pente, vitesse_simu, masse_pilote;
    float tension[4], courant[4], courant_total_decharge;

    int mode_local_distant;// 0 : Local ; 1 : Distant
    int enable; //Enable : 0 non valide; 1 valide
    int run; //Run : 0 non valide; 1 valide
    /*****
     * moteur OPERATIONNEL SSI E = 1 et Run = 1
     * *****/

    //réception données de MESURES   Read ONLY

    float getVitSimu();    //tr/mn-1

    float getCoupleSimu();

    float getCourantBatt(); // courant de decharge totale (batteries en series
                          // donc peu importe le numero de batterie //utile pour
                          // simulateur
    /* lecture des tensions des 4 batteries */
    float gettens1();
    float gettens2();
    float gettens3();
    float gettens4();

    /* lecture des 4 courants de charge des 4 batteries */
    float getcour1();
    float getcour2();
    float getcour3();
    float getcour4();

    //envoi / lecture donnees de CONSIGNE   R/W
    void setMasse_Pilote(QString weight);
    float getMasse_Pilote();

    void setConsignePente(QString pente);
    float getConsignePente();

    void setLocal_Distant(QString local_dist);

```



```
int getLocal_Distant();

// envoi / lecture Consigne commande broche Enable
void setConsigneEnable(QString valide);
int getConsigneEnable();

// envoi / lecture Consigne commande broche Run
void setConsigneRun(QString valide);
int getConsigneRun();
};
#endif // KARTBTCLIENT_H
```

