TP03 : création d'un terminal pour une interface série

Objectif :

L'application permet à l'utilisateur de se connecter à travers une communication série. Une fois les paramètres de la communication sont mis. L'interface permet d'afficher le contenu émis/reçu en utilisant cette communication.

Travail demandé :

- Suivre les étapes du TP pour construire l'application.
- Documentation de l'application

1- La classe QSerialPort :

L'application utilise principalement la classe QSerialPort.Vous pouvez obtenir des informations sur les ports série disponibles à l'aide de la classe d'assistance QSerialPortInfo, qui permet d'énumérer tous les ports série du système. Ceci est utile pour obtenir le nom correct du port série que vous souhaitez utiliser. Vous pouvez transmettre un objet de la classe d'assistance en tant qu'argument aux méthodes setPort () ou setPortName () pour affecter le périphérique série souhaité.

Après avoir défini le port, vous pouvez l'ouvrir en mode lecture seule (read-only r/o), écriture seule (write-only w/o) ou lecture-écriture (read-write r/w) à l'aide de la méthode open ().

Utilisez la méthode close () pour fermer le port et annuler les opérations d'E/S.

Après l'ouverture du port, QSerialPort essaie de déterminer la configuration actuelle du port et de l'initialiser. Vous pouvez reconfigurer le port sur le paramètre souhaité à l'aide des méthodes setBaudRate (), setDataBits (), setParity (), setStopBits () et setFlowControl ().

Il existe quelques propriétés pour travailler avec les signaux de brochage, à savoir: QSerialPort :: dataTerminalReady, QSerialPort :: requestToSend. Il est également possible d'utiliser la méthode pinoutSignals () pour interroger le jeu de signaux de brochage actuel.

Une fois que vous savez que les ports sont prêts à lire ou à écrire, vous pouvez utiliser les méthodes read () ou write (). Alternativement, les méthodes pratiques readLine () et readAll () peuvent également être appelées. Si toutes les données ne sont pas lues en même temps, les données restantes seront disponibles pour plus tard, car de nouvelles données entrantes sont ajoutées au tampon de lecture interne de QSerialPort. Vous pouvez limiter la taille du tampon de lecture à l'aide de setReadBufferSize ().

QSerialPort fournit un ensemble de fonctions qui suspendent le thread appelant jusqu'à ce que certains signaux soient émis. Ces fonctions peuvent être utilisées pour implémenter des ports série bloquants:

- waitForReadyRead () bloque les appels jusqu'à ce que de nouvelles données soient disponibles pour la lecture.
- waitForBytesWritten () bloque les appels jusqu'à ce qu'une charge utile de données ait été écrite sur le port série.

Exemple :

```
int numRead = 0, numReadTotal = 0;
char buffer[50];

for (;;) {
    numRead  = serial.read(buffer, 50);

    // Do whatever with the array

    numReadTotal += numRead;
    if (numRead == 0 && !serial.waitForReadyRead())
        break;
}
```

Si waitForReadyRead () renvoie false, la connexion a été fermée ou une erreur s'est produite.

Si une erreur se produit à un moment donné, QSerialPort émettra le signal errorOccurs (). Vous pouvez également appeler error () pour rechercher le type d'erreur survenu en dernier.

La programmation avec un port série bloquant est radicalement différente de la programmation avec un port série non bloquant. Un port série bloquant ne nécessite pas de boucle d'événement et conduit généralement à un code plus simple. Toutefois, dans une application graphique, le port série bloquant ne doit être utilisé que dans les unités d'exécution non graphiques, afin d'éviter de geler l'interface utilisateur.

enum QSerialPort::BaudRate

Cette énumération décrit le débit en bauds avec lequel le dispositif de communication fonctionne.

Constant	Value	Description
QSerialPort::Baud1200	1200	1200 baud.
QSerialPort::Baud2400	2400	2400 baud.
QSerialPort::Baud4800	4800	4800 baud.
QSerialPort::Baud9600	9600	9600 baud.
QSerialPort::Baud19200	19200	19200 baud.
QSerialPort::Baud38400	38400	38400 baud.
QSerialPort::Baud57600	57600	57600 baud.
QSerialPort::Baud115200	115200	115200 baud.
QSerialPort::UnknownBaud	-1	Unknown baud. This value is obsolete. It is provided to keep old source code working. We strongly advise against using it in new code.

enum QSerialPort::DataBits

Cette énumération décrit le nombre de bits de données utilisés.

Constant	Value	Description
QSerialPort::Data5	5	The number of data bits in each character is 5. It is used for Baudot code. It generally only makes sense with older equipment such as teleprinters.
QSerialPort::Data6	6	The number of data bits in each character is 6. It is rarely used.
QSerialPort::Data7	7	The number of data bits in each character is 7. It is used for true ASCII. It generally only makes sense with older equipment such as teleprinters.
QSerialPort::Data8	8	The number of data bits in each character is 8. It is used for most kinds of data, as this size matches the size of a byte. It is almost universally used in newer applications.
QSerialPort::UnknownDataBits	-1	Unknown number of bits. This value is obsolete. It is provided to keep old source code working. We strongly advise against using it in new code.

enum QSerialPort::Direction flags QSerialPort::Directions

Cette énumération décrit les directions possibles de la transmission de données.

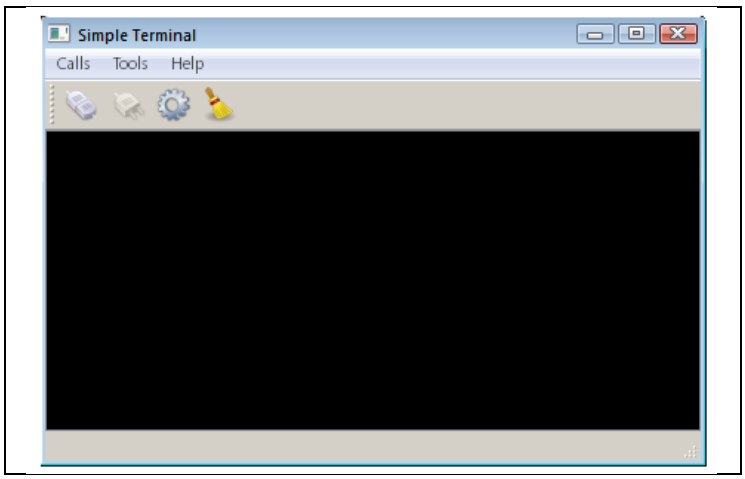
Constant	Value	Description
QSerialPort::Input	1	Input direction.
QSerialPort::Output	2	Output direction.
QSerialPort::AllDirections	Input | Output	Simultaneously in two directions.

enum QSerialPort :: Parity

Cette énumération décrit le schéma de parité utilisé.

Constant	Value	Description
QSerialPort::NoParity	0	No parity bit it sent. This is the most common parity setting. Error detection is handled by the communication protocol.
QSerialPort::EvenParity	2	The number of 1 bits in each character, including the parity bit, is always even.
QSerialPort::OddParity	3	The number of 1 bits in each character, including the parity bit, is always odd. It ensures that at least one state transition occurs in each character.
QSerialPort::SpaceParity	4	Space parity. The parity bit is sent in the space signal condition. It does not provide error detection information.
QSerialPort::MarkParity	5	Mark parity. The parity bit is always set to the mark signal condition (logical 1). It does not provide error detection information.
QSerialPort::UnknownParity	-1	Unknown parity. This value is obsolete. It is provided to keep old source code working. We strongly advise against using it in new code.

2- Terminal d'une communication série



Cet exemple montre les principales fonctionnalités de la classe QSerialPort, telles que la configuration, la mise en œuvre d'E/S, etc. En outre, la classe QSerialPortInfo est appelée pour afficher des informations sur les ports série disponibles sur le système.

Comme mentionner avant, QSerialPort prend en charge deux approches de programmation générales:

- L'approche asynchrone (non bloquante). Les opérations sont planifiées et effectuées lorsque le contrôle revient à la boucle d'événements de Qt. QSerialPort émet un signal lorsque l'opération est terminée. Par exemple, QSerialPort :: write () est renvoyé immédiatement. Lorsque les données sont envoyées au port série, QSerialPort émet bytesWritten ().
- L'approche synchrone (bloquante). Dans les applications non graphiques et multithread, les fonctions waitFor ... () peuvent être appelées (c'est-à-dire QSerialPort :: waitForReadyRead ()) pour suspendre le thread appelant jusqu'à la fin de l'opération.

Dans cet exemple, on utilisera l'approche asynchrone. L'exemple de l'esclave bloquant illustre l'approche synchrone.

Notre exemple contient des widgets d'interface graphique GUI:

- MainWindow (terminal/mainwindow.cpp) - est la fenêtre principale de l'application qui contient toute la logique de travail pour la programmation du port série, y compris la configuration, le traitement des E/S, etc., tout en héritant de QMainWindow.
- Console (terminal/console.cpp) - est le widget central de la fenêtre principale, affichant les données transmises ou reçues. Le widget est dérivé de la classe QPlainTextEdit.
- SettingsDialog (terminal/settingsdialog.cpp) - est une boîte de dialogue permettant de configurer le port série, ainsi que d'afficher les ports série disponibles et des informations les concernant.

Le port série est instancié dans le constructeur de MainWindow. Le widget principal est passé en tant que parent. La suppression d'objet a donc lieu automatiquement en fonction du mécanisme parent et enfant dans Qt:

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    m_ui(new Ui::MainWindow),
    m_status(new QLabel),
    m_console(new Console),
    m_settings(new SettingsDialog),
    m_serial(new QSerialPort(this))
{
    ...
    m_serial(new QSerialPort(this))
```

Le seul signal QSerialPort appelé dans cet exemple est le signal readyRead(), qui indique que de nouvelles données ont été reçues et donc disponibles:

```
...
    connect(m_serial, &QSerialPort::readyRead, this, &MainWindow::readData);
    ...
}
```

Cliquez sur le bouton Connecter pour appeler le slot openSerialPort ():

```
void MainWindow::openSerialPort()
{
    const SettingsDialog::Settings p = m_settings->settings();
    m_serial->setPortName(p.name);
    m_serial->setBaudRate(p.baudRate);
    m_serial->setDataBits(p.dataBits);
    m_serial->setParity(p.parity);
    m_serial->setStopBits(p.stopBits);
    m_serial->setFlowControl(p.flowControl);
    if (m_serial->open(QIODevice::ReadWrite)) {
        m_console->setEnabled(true);
        m_console->setLocalEchoEnabled(p.localEchoEnabled);
        m_ui->actionConnect->setEnabled(false);
        m_ui->actionDisconnect->setEnabled(true);
        m_ui->actionConfigure->setEnabled(false);
        showStatusMessage(tr("Connected to %1 : %2, %3, %4, %5, %6")
                          .arg(p.name).arg(p.stringBaudRate).arg(p.stringDataBits)
                          .arg(p.stringParity).arg(p.stringStopBits).arg(p.stringFlowControl));
    } else {
        QMessageBox::critical(this, tr("Error"), m_serial->errorString());

        showStatusMessage(tr("Open error"));
    }
}
```

Dans cet emplacement, les paramètres sont lus à partir de SettingsDialog et une tentative d'ouverture et d'initialisation du port série est effectuée en conséquence. En cas de succès, la barre d'état affiche un message indiquant que l'ouverture a réussi avec la configuration donnée. Sinon, une boîte de message s'affiche avec le code d'erreur et le message appropriés. Si les paramètres du port série n'ont jamais été appelés, le terminal tente d'ouvrir le port avec les paramètres par défaut: 9600 8N1.

Cliquez sur le bouton Déconnecter pour appeler le slot closeSerialPort ():

```
void MainWindow::closeSerialPort()
{
    if (m_serial->isOpen())
        m_serial->close();
    m_console->setEnabled(false);
    m_ui->actionConnect->setEnabled(true);
    m_ui->actionDisconnect->setEnabled(false);
    m_ui->actionConfigure->setEnabled(true);
    showStatusMessage(tr("Disconnected"));
}
```

Taper des caractères dans la console appelle l'emplacement writeData ():

```
void MainWindow::writeData(const QByteArray &data)
{
    m_serial->write(data);
}
```

Ce slot envoie les caractères saisis dans le widget de console donné au port série.

Lorsque le port série reçoit de nouvelles données, le signal readyRead () est émis et ce signal est connecté au slot MainWindow::readData():

```
void MainWindow::readData()
{
    const QByteArray data = m_serial->readAll();
    m_console->putData(data);
}
```

Ce slot lit les données du port série et les affiche dans le widget Console.

En cliquant sur le bouton Configurer, on appelle le slot show() qui appartient au widget SettingsDialog.

Cette méthode (terminal/settingsdialog.cpp) affiche le SettingsDialog dans lequel l'utilisateur peut choisir le port série souhaité, voir les informations sur le port sélectionné et définir les paramètres souhaités du port série donné.

3- Enumération des ports série

Ajouter l'option énumération des ports existant dans votre application terminal.

Montre comment afficher des informations sur les périphériques série d'un système.
L'énumérateur montre comment utiliser la classe QSerialPortInfo pour obtenir des informations sur les périphériques série présents dans le système.
Le fichier enumerator.pro contient :

```
QT += widgets serialport

TARGET = enumerator
TEMPLATE = app

SOURCES += \
    main.cpp

target.path = $$[QT_INSTALL_EXAMPLES]/serialport/enumerator
INSTALLS += target
```

Le fichier main.cpp contient :

```
#include <QApplication>
#include <QLabel>
#include <QScrollArea>
#include <QSerialPortInfo>
#include <QVBoxLayout>
#include <QWidget>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    auto layout = new QVBoxLayout;
```

```
    const auto infos = QSerialPortInfo::availablePorts();
    for (const QSerialPortInfo &info : infos) {
        QString s = QObject::tr("Port: ") + info.portName() + "\n"
                    + QObject::tr("Location: ") + info.systemLocation()
+ "\n"
                    + QObject::tr("Description: ") + info.description()
+ "\n"
                    + QObject::tr("Manufacturer: ") +
info.manufacturer() + "\n"
                    + QObject::tr("Serial number: ") +
info.serialNumber() + "\n"
                    + QObject::tr("Vendor Identifier: ") +
(info.hasVendorIdentifier() ? QString::number(info.vendorIdentifier(),
16) : QString()) + "\n"
                    + QObject::tr("Product Identifier: ") +
(info.hasProductIdentifier() ? QString::number(info.productIdentifier(),
16) : QString()) + "\n"
                    + QObject::tr("Busy: ") + (info.isBusy() ?
QObject::tr("Yes") : QObject::tr("No")) + "\n";

        auto label = new QLabel(s);
        layout->addWidget(label);
    }

    auto workPage = new QWidget;
    workPage->setLayout(layout);

    QScrollArea area;
    area.setWindowTitle(QObject::tr("Info about all available serial
ports."));
    area.setWidget(workPage);
    area.show();

    return a.exec();
}
```

Classe Mainwindow :

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QSerialPort>

QT_BEGIN_NAMESPACE

class QLabel;

namespace Ui {
class MainWindow;
}

QT_END_NAMESPACE

class Console;
class SettingsDialog;

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void openSerialPort();
    void closeSerialPort();
    void about();
    void writeData(const QByteArray &data);
    void readData();

    void handleError(QSerialPort::SerialPortError error);

private:
    void initActionsConnections();

private:
    void showStatusMessage(const QString &message);

    Ui::MainWindow *m_ui = nullptr;
    QLabel *m_status = nullptr;
    Console *m_console = nullptr;
    SettingsDialog *m_settings = nullptr;
    QSerialPort *m_serial = nullptr;
};

#endif // MAINWINDOW_H
```

Classe Console.h

```
#ifndef CONSOLE_H
#define CONSOLE_H

#include <QPlainTextEdit>

class Console : public QPlainTextEdit
{
    Q_OBJECT

signals:
    void getData(const QByteArray &data);

public:
    explicit Console(QWidget *parent = nullptr);

    void putData(const QByteArray &data);
    void setLocalEchoEnabled(bool set);

protected:
    void keyPressEvent(QKeyEvent *e) override;
    void mousePressEvent(QMouseEvent *e) override;
    void mouseDoubleClickEvent(QMouseEvent *e) override;
    void contextMenuEvent(QContextMenuEvent *e) override;

private:
    bool m_localEchoEnabled = false;
};

#endif // CONSOLE_H
```

Classe SettingsDialog.h:

```
#ifndef SETTINGSDIALOG_H
#define SETTINGSDIALOG_H

#include <QDialog>
#include <QSerialPort>

QT_BEGIN_NAMESPACE

namespace Ui {
class SettingsDialog;
}

class QIntValidator;

QT_END_NAMESPACE

class SettingsDialog : public QDialog
{
    Q_OBJECT

public:
    struct Settings {
        QString name;
        qint32 baudRate;
```

```
        QString stringBaudRate;
        QSerialPort::DataBits dataBits;
        QString stringDataBits;
        QSerialPort::Parity parity;
        QString stringParity;
        QSerialPort::StopBits stopBits;
        QString stringStopBits;
        QSerialPort::FlowControl flowControl;
        QString stringFlowControl;
        bool localEchoEnabled;
    };

    explicit SettingsDialog(QWidget *parent = nullptr);
    ~SettingsDialog();

    Settings settings() const;

private slots:
    void showPortInfo(int idx);
    void apply();
    void checkCustomBaudRatePolicy(int idx);
    void checkCustomDevicePathPolicy(int idx);

private:
    void fillPortsParameters();
    void fillPortsInfo();
    void updateSettings();

private:
    Ui::SettingsDialog *m_ui = nullptr;
    Settings m_currentSettings;
    QIntValidator *m_intValidator = nullptr;
};

#endif // SETTINGSDIALOG_H
```