UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Módulo de extensión Qt Serial Port para comunicaciones serie en Qt

Apellidos, nombre	Perles Ivars, Àngel (aperles@disca.upv.es)
Departamento	Informática de Sistemas y Computadores
Centro	Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

1 Resumen de las ideas clave

Se introduce el uso del nuevo módulo de extensión "Qt Serial Port" [1] para Qt [2], que aporta la capacidad de gestionar comunicaciones serie. A continuación se resumen las ideas clave que se van a trabajar:

- Cómo se configuran los proyectos para poder usar el módulo.
- Cómo son las primitivas básicas de envío y recepción de datos.
- Cómo se desarrollan aplicaciones básicas de uso.

2 Introducción

Normas de comunicación serie como RS-232, RS-485 y variantes (incluyendo las virtuales) son de amplio uso en ambientes industriales.

La importancia de estas comunicaciones ha fomentado el desarrollo de un nuevo módulo oficial para Qt: el módulo "Qt Serial Port". Gracias este módulo, ya no es necesario incorporar bibliotecas de terceras partes como la que se presentó en un anterior artículo docente [3], simplificándose mucho su explotación.

Este artículo presenta, de manera básica, el uso de este módulo para desarrollar aplicaciones multiplataforma nativas en Microsoft Windows, Linux y Mac OS X.

3 Objetivos

Una vez leído y practicado con este documento, el lector será capaz de:

- Emplear comunicaciones serie en sus aplicaciones Qt.

4 Desarrollo

4.1 Requisitos previos

Este artículo solo se puede seguir sí:

- Se tienen nociones básicas de programación con Qt.
- Se tienen nociones básicas sobre interfaces serie tipo RS-232.

Para el desarrollo de la experiencia será necesario, como mínimo:

- Qt 5.3 para Windows, Linux o Mac OS X [4].
- Un serie en el computador de desarrollo, ya sea real o virtual (Bluetooth, USB, infrarrojos, …). Como opción se puede emplear un emulador de conexión null-modem, por ejemplo, com0com [5]

4.2 Incorporando el módulo a la aplicación Qt

El módulo Qt Serial Port es parte de Qt desde la versión 5.1, por lo que su incorporación es muy sencilla.

El primer paso es modificar el archivo de proyecto añadiendo la siguiente línea:

```
QT       += serialport
```

Para que los módulos de código sean capaces de usar las funciones del módulo, bastará entonces con añadir la siguiente cabecera:

```
#include <QSerialPort>
```

4.3 Usando el módulo

4.3.1 Creando una instancia

El módulo proporciona varias clases para el uso de la conexión serie. La que interesa para este caso sencillo es QSerialPort. La creación de instancias de esta clase sigue las mismas pautas que cualquier otra clase de Qt.

Aunque en el manual se indican distintos tipos de constructores, por claridad, se empleará el constructor que menos configuraciones iniciales hace. Su prototipo es:

```
QSerialPort(QObject * parent = 0);
```

Por ejemplo, se puede crear una instancia de la clase de la siguiente manera:

```
QSerialPort *port; // puntero al objeto que maneja el puerto

void funcion(void)
{
    QSerialPort *port = new QSerialPort();
}
```

4.3.2 Asociando el objeto a un puerto

Una vez creado el objeto, el siguiente paso es asociarlo a un determinado puerto serie, para ello se puede emplear el siguiente método:

```
void setPortName(const QString & name);
```

Con setPortName() se establece el nombre del dispositivo serie a utilizar.

Por ejemplo, se podría hacer:

```
port->setPortName("COM1");
```

4.3.1 "Abriendo" y "cerrando" la conexión

Para que una aplicación pueda utilizar un determinado puerto serie, primero debe solicitarlo al sistema operativo para que se lo asigne. A este procedimiento se le suele llamar "abrir" el puerto.

De la misma manera, cuando ya no se usa una conexión serie, se debe devolver al SO para que otra aplicación pueda disponer de ella. A esto se le llama "cerrar" el puerto.

Para gestionar estas actividades, se pueden utilizar los siguientes métodos:

```
bool open (OpenMode mode);
void close ();
```

La función open() permite "abrir" el puerto y saber si se ha logrado abrir. Como parámetro admite los valores QIODevice::ReadOnly (solo lectura/recibir), QIODevice::WriteOnly (solo escritura/enviar), QIODevice::ReadWrite (lectura/recibir y escritura/enviar).

Por ejemplo, se podría hacer:

```
printf("Abriendo el puerto ... ");
if (port->open(QIODevice::WriteOnly)) {
    printf("abierto!!!\n");
} else {
    printf("Vaya, esto FALLA\n");
}
// ahora se usa el puerto
...
printf("Cerrando el puerto\n");
port->close();
```

El prefijo QIODevice delata la naturaleza de QSerialPort, pues es una derivación de dicha clase. Hay que tenerlo en cuenta a la hora de consultar la documentación oficial para poder localizar aquellos médotos heredados que podría ser de utilidad en la aplicación.

4.3.2 Configurando la conexión

Antes de iniciar la transmisión de datos, es necesario configurar los parámetros del puerto a utilizar. Aunque hay muchas opciones, los siguientes métodos son los básicos:

```
bool    setBaudRate(qint32 baudRate);
bool    setDataBits(DataBits dataBits);
bool    setParity(Parity parity);
bool    setStopBits(StopBits stopBits);
bool    setFlowControl(FlowControl flow);
```

Come ejemplo, una típica configuración 9600-8N1 sin control de flujo se podría hacer:

```
port->setBaudRate(QSerialPort::Baud9600);
port->setDataBits(QSerialPort::Data8);
port->setParity(QSerialPort::NoParity);
port->setStopBits(QSerialPort::OneStop);
port->setFlowControl(QSerialPort::NoFlowControl);
```

4.3.3 Enviando datos

Antes de entrar a explotar el objeto, es importante resaltar que el módulo es una simple interfaz con los servicios que el sistema operativo proporciona para gestionar los puertos serie. El sistema operativo gestiona internamente una serie de buffers de transmisión/recepción que proporcionan una asincronía entre la petición de envío de datos y su envío efectivo al exterior. Así, los siguientes métodos pasarán inmediatamente todos los datos al SO y la aplicación podrá continuar inmediatamente su ejecución.

Una vez abierta la conexión, se pueden enviar datos empleando distintos métodos. Por ejemplo :

```
putChar (char c);
write (const QByteArray &byteArray);
write (const char *data, qint64 maxSize);
```

El método putChar() permite enviar un solo dato y el método write() permite enviar al buffer de salida una cadena o maxSize datos apuntados por el puntero data. Por ejemplo:

```
char *mensaje = "Bon dia pel mati";
char datos[] = {125,42,37,9};
//...
port->putChar(34);
port->putChar('A');
port->putChar(0xF3);
port->write(mensaje);
port->write(datos,4);
```

Destacar aquí que cada "dato" corresponderá a un dato asíncrono transferido por la conexión, no estando ligados datos subsecuentes. Aunque a nivel de la aplicación los datos son de 8 bits, en la transmisión se descartan los bits de mayor peso de cada dato si la configuración de datos es inferior a 8.

4.3.1 Recibiendo datos

De la misma manera que la transmisión, en la recepción es el SO el encargado de ir acumulando datos en buffers internos (hasta cierto límite de descarte). La aplicación desarrollada simplemente solicita al SO los datos contenidos en dichos buffers.

Para recibir datos se dispone también de varios métodos. Por ejemplo:

```
bool getChar (char *c);
qint64 read (char *data, qint64 maxSize);
```

El método getChar() permite recibir datos individualmente, y la función read() recoger hasta maxSize datos del buffer de entrada del SO. La función read() devolverá la cantidad de datos efectivamente recogidos y, como no es bloqueante, devolverá 0 en el caso de que no haya nada.

Como ejemplo, para recoger datos se podría hacer:

```
char buffer[100];
int num_datos;

num_datos = read (buffer, 100);
printf("Se han recibido %d datos", num_datos);
```

En algunas situaciones, es interesante saber la cantidad de datos que el SO tiene en el buffer de entrada sin recogerlos, para ello, se puede emplear la siguiente función:

```
qint64 bytesAvailable();
```

El método bytesAvailable(), devuelve el número de datos disponibles en el buffer de entrada.

Por ejemplo, se podría hacer:

```
int num_datos;
char dato;

num_datos = puerto->bytesAvailable();
if (num_datos > 0) {     // hay algo
    puerto->getChar(&dato);
    printf("Recibido el ascii %d (%c)\n",(int)dato, dato);
}
```

Para sacer el máximo partido al paradigma signal-slot, es interesante aprovechar las señales proporcionadas por QSerialPort, en especial para el caso de la recepción. Destacar la siguiente señal, que es emitida cuando hay datos en el buffer de entrada:

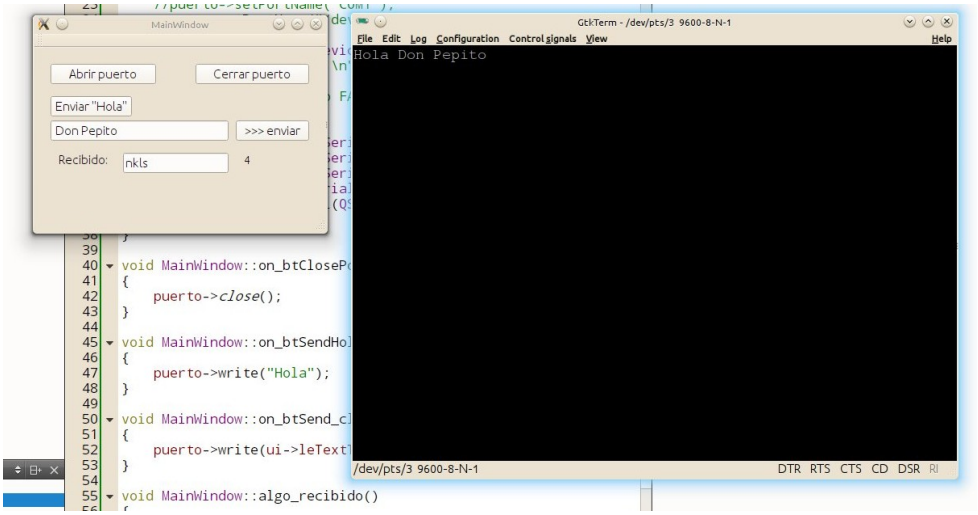
El método bytesAvailable() devuelve el número de datos disponibles en el buffer de entrada.

```
void readyRead()
```

4.4 Un ejemplo completo

La siguiente aplicación es una ejemplo completo que demuestra la aplicación de lo explicado. En el ejemplo se prima la simplicidad por encima de ofrecer una versión profesional.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA



Archivo mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QSerialPort>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
private slots:
    void on_btOpenPort_clicked();
    void on_btClosePort_clicked();
    void on_btSendHola_clicked();
    void on_btSend_clicked();

    void algo_recibido(void);
private:
    Ui::MainWindow *ui;
    QSerialPort *puerto;
```

```
};

#endif // MAINWINDOW_H
```

Archivo mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    puerto = new QSerialPort();
    connect(puerto, SIGNAL(readyRead()),this,SLOT(algo_recibido()));
}

MainWindow::~MainWindow()
{
    delete puerto;
    delete ui;
}

void MainWindow::on_btOpenPort_clicked()
{
    puerto->setPortName("COM1");
    if (puerto->open(QIODevice::ReadWrite)) {
        qDebug("Abierto!!!\n");
    } else {
        qDebug("Vaya, esto FALLA\n");
    }

    puerto->setBaudRate(QSerialPort::Baud9600);
    puerto->setDataBits(QSerialPort::Data8);
    puerto->setStopBits(QSerialPort::OneStop);
    puerto->setParity(QSerialPort::NoParity);
    puerto->setFlowControl(QSerialPort::NoFlowControl);
}

void MainWindow::on_btClosePort_clicked()
```

```
{
    puerto->close();
}

void MainWindow::on_btSendHola_clicked()
{
    puerto->write("Hola");
}

void MainWindow::on_btSend_clicked()
{
    puerto->write(ui->leTextToSend->text().toLatin1());
}

void MainWindow::algo_recibido()
{
    char buffer[101];
    int num_datos;

    num_datos = puerto->read(buffer,100);

    buffer[num_datos] = '\0';
    ui->leReceivedText->setText(buffer);
    ui->lbCountRead->setNum(num_datos);
}
```

5 Cierre

A lo largo de este objeto de aprendizaje se ha visto cómo proporcionar la capidad de emplear una conexión serie mediante Qt.

6 Bibliografía

[1] Qt Serial Port module Add-on documentation for Qt 5.2. Disponible en: http://qt-project.org/doc/qt-5/qtserialport-index.html

[2] The Qt project. The Qt project foundation official page. 2014. Disponible en: URL: http://qt-project.org/.

[3] Perles, A. "Comunicaciones serie en Qt usando QextSerialPort". 2013. http://hdl.handle.net/10251/30515

[4] Qt 5.3 Disponible en: http://qt-project.org/downloads

[5] com0com virtual null-modem emulator for Microsoft Windows. http://com0com.sourceforge.net/