UNIVERSIDAD DE OVIEDO
Departamento de Ingeniería Eléctrica, Electrónica, de Computadores y Sistemas

Instalación y uso de QwtPlot

Ignacio Alvarez García – Octubre 2021

Indice

UNIVERSIDAD DE OVIEDO
Departamento de Ingeniería Eléctrica, Electrónica, de Computadores y Sistemas

1. Instalación Windows

Ver detalles en https://qwt.sourceforge.io/qwtinstall.html para otros sistemas operativos.

1.a) Descargar de https://sourceforge.net/projects/qwt/files/latest/download

1.b) Descomprimir en un directorio (se toma como ejemplo D:\Qt\QwtPlot-6-1-4)

1.c) Compilar en Qt-Creator.

1.c.1) Abrir proyecto D:\Qt\QwtPlot-6-1-4\qwt.pro

1.c.2) Seleccionar kit(s) para la compilación

1.c.3) En el proyecto, abrir qwt\qwtconfig\qwtconfig.pri y modificar directorio destino si se desea:

```
win32 {
    QWT_INSTALL_PREFIX = D:/Qt/Qwt-$$QWT_VERSION
}
```

1.c.4) Eliminar información de depuración QML:

Projects → Build → Build Settings → General → deshabilitar "Qml debugging and profiling"

1.c.5) Añadir paso de instalación:

Projects → Build → Build Settings → Build Steps → Add Build Step → make → Make arguments: install

1.c.6) Compilar. Ignorar algunos warnings que se obtienen, tipo [install headers] Error 3 (ignored).

1.c.7) Cerrar proyecto qwt

2. Uso básico en programas

2.a) Crear proyecto tipo aplicación – widgets

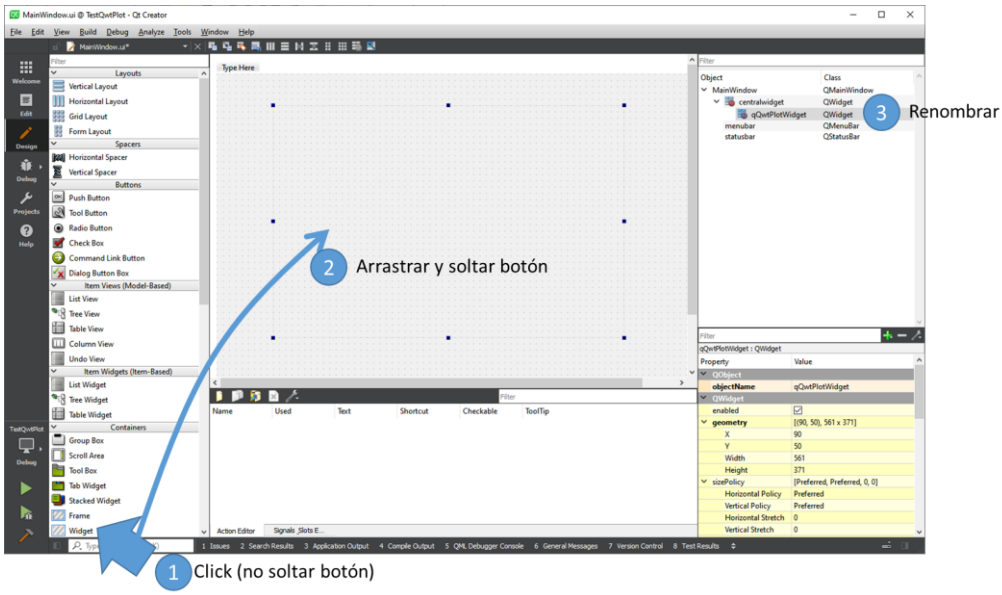
2.b) En archivo .pro, añadir la línea:

```
include(D:/Qt/Qwt-6.1.4/features/qwt.prf)
```

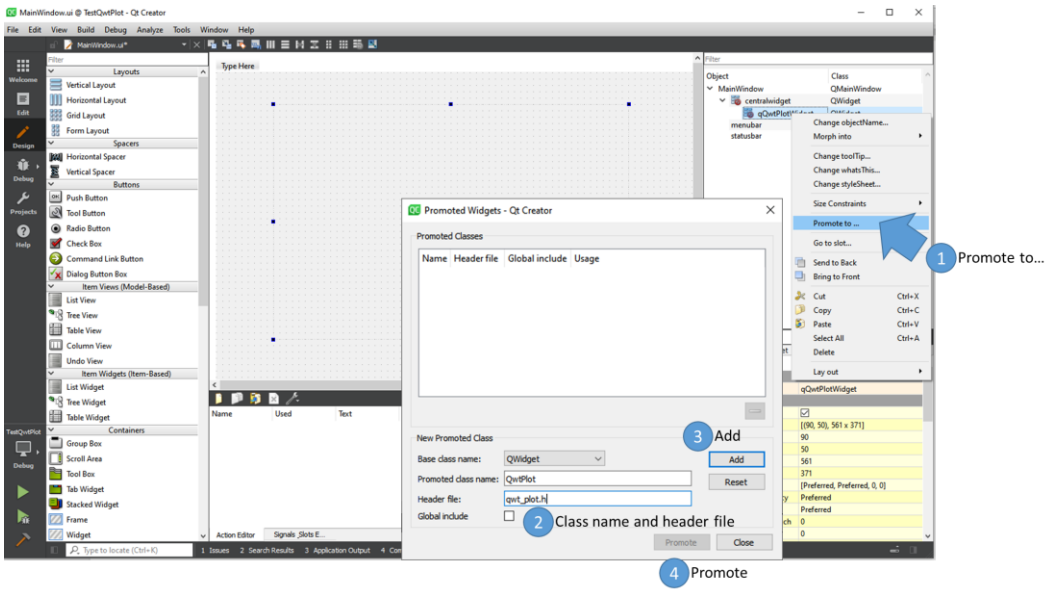
(donde la parte sombreada-cursiva debe sustituirse por el directorio de instalación que se estableció en el apartado 1.c.3)

2.c) Ejecutar Build → Run qmake

2.d) Añadir un widget genérico en el form que se desee. Ejemplo MainWindow.ui → Arrastrar Containers → Widget al lugar deseado. Renombrar el objeto del widget → ejemplo qQwtPlotWidget:



2.e) Promocionar widget a QwtPlot:



2.f) El elemento ui->qQwtPlotWidget ya es un QwtPlot (comprobar en la ventana superior derecha que se ha promocionado a ese tipo), y ya se pueden usar todas sus funcionalidades. Lo normal será:

Para cada curva a dibujar, crear un objeto de tipo QwtPlotCurve, otro de tipo QwtPointSeriesData y otro de tipo QVector<QPointF> en la clase (en este ejemplo, MainWindow):

MainWindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "qwt_plot_curve.h"
...
class MainWindow : public QMainWindow
```

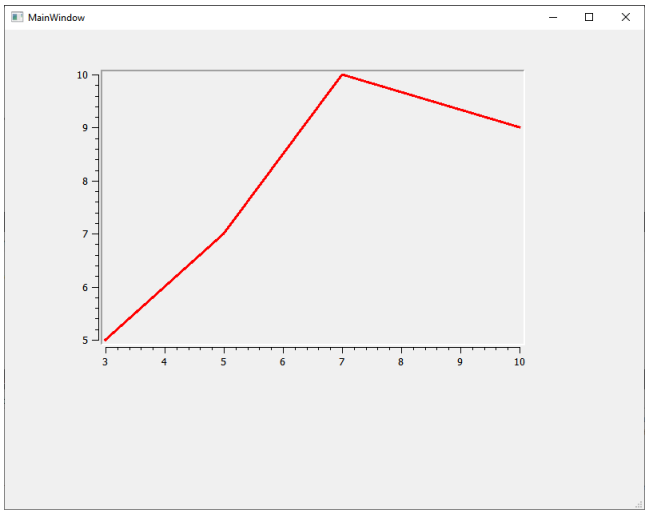
```
{
    Q_OBJECT
private:
    QwtPlotCurve curve1,curve2;
    QwtPointSeriesData series1,series2;
    QVector<QPointF> data1,data2;

    ...
};
#endif // MAINWINDOW_H
```

Inicializar todos ellos en el constructor, así como ui-> qQwtPlotWidget (ver ayuda de todos ellos para opciones disponibles). Por ejemplo:

MainWindow.cpp

```
#include "MainWindow.h"
#include "ui_MainWindow.h"

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

    // Curva 1: título, tipo de trazo, añadir al plot widget,
    // y establecer serie asociada
    curve1.setTitle("Datos");
    curve1.setPen(QColor(255,0,0),3,Qt::SolidLine);
    curve1.attach(ui->qQwtPlotWidget);
    curve1.setData(&series1);

     // Datos 1: se conocen en el contructor
    data1 << QPointF(3,5) << QPointF(5,7) << QPointF(7,10) <<
QPointF(10,9);
    series1.setSamples(data1);

    // Curva 2: título, tipo de trazo, añadir al plot widget,
    // y establecer serie asociada
    curve2.setTitle("Otro");
    curve2.setPen(QColor(50,50,100),3,Qt::DashLine);
    curve2.attach(ui->qQwtPlotWidget);
    curve2.setData(&series2);

    // Curva 2: no hay datos iniciales
}
```

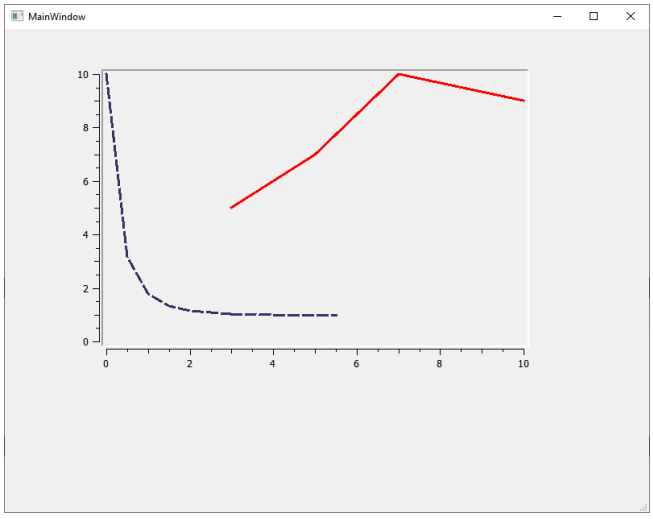
Si se compila y ejecuta, se obtendrá:



2.g) En los slots adecuados de MainWindow, modificar los datos deseados y ordenar redibujar. Por ejemplo, cada evento del timer se calcula un nuevo punto para la serie 2, se añade y se ordena el redibujo:

MainWindow.cpp

```
void MainWindow::OnTimer()
{
    if (data2.length()==0)
        data2 << QPointF(0,10);
    else
    {
        QPointF last=data2.last();
        data2 << QPointF(last.x()+0.5,sqrt(last.y()));
    }
    series2.setSamples(data2);
    ui->qQwtPlotWidget->replot();
}
```



3. Asignación dinámica de memoria

En la mayoría de los casos, se puede comprobar que los ejemplos y tutoriales utilizan **new** para crear objetos. El uso de **new** permite una creación dinámica (en tiempo de

ejecución) en el heap (similar a malloc), de forma que el objeto no desaparece hasta que se libere con **delete**.

A modo de ejemplo, para crear una serie que asociar a nuestro plot, hay 2 opciones correctas y 1 opción incorrecta. Supongamos que ante un determinado evento queremos crear una nueva serie (sería lo mismo para una nueva leyenda, ejes, etc.):

Opción 1 (incorrecta): los nuevos elementos son variables locales de la función

MainWindow.cpp

```
void MainWindow::OnNewSeries()
{
    QwtPointSeriesData s; // Se crean en el stack, así que se
    QwtPlotCurve c;       // destruyen al terminar la función,
    QVector<QPointF> d;   // por tanto no sirven

    d << ... << ... ;
    s.setSamples(d);
    c.setData(&s);
    c.attach(ui->qQwtPlotWidget);

    // A partir de aquí ya no existen s ni c
}
```

Opción 2 (correcta, pero "incómoda"): los nuevos elementos están previstos como elementos de MainWindow, y por tanto existen y usan recursos de forma permanente (se usen o no)

MainWindow.h

```
...
class MainWindow : public QMainWindow
{
    Q_OBJECT
private:
    ...
    QwtPointSeriesData s_new;  // Existe mientras haya MainWindow
    QwtPlotCurve c_new;        // Existe mientras haya MainWindow
    QVector<QPointF> d_new;    // Existe mientras haya MainWindow
};
#endif // MAINWINDOW_H
```

MainWindow.cpp

```
void MainWindow::OnNewSeries()
{

    d_new << ... << ... ;
    s_new.setSamples(d_new);
    c_new.setData(&s_new);
    c_new.attach(ui->qQwtPlotWidget);
}
```

Opción 3 (la más usada): los nuevos elementos se crean en el heap cuando se necesitan, y se borran cuando ya no se quieren

MainWindow.h

```
...
```

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
private:
    ...
    QwtPointSeriesData* s_new; // Existe mientras haya MainWindow
    QwtPlotCurve* c_new;       // Existe mientras haya MainWindow
    QVector<QPointF>* d_new;   // Existe mientras haya MainWindow
};
#endif // MAINWINDOW_H
```

MainWindow.cpp

```
MainWindow::MainWindow()
{
    s_new=nullptr;
    c_new=nullptr;
    d_new=nullptr;
}

void MainWindow::OnNewSeries()
{
    s_new=new QwtPointSeriesData(argumentos del constructor);
    c_new=new QwtPlotCurve(argumentos del constructor);
    d_new=new QVector<QPointF>(argumentos del constructor);

    // Ahora se tratan como punteros
    *d_new << ... << ... ;
    s_new->setSamples(*d_new);
    c_new->setData(s_new);
    c_new->attach(ui->qQwtPlotWidget);
}

void MainWindow::OnRemoveSeries()
{
    c_new->detach(ui->qQwtPlotWidget);
    delete c_new; c_new=nullptr;
    delete s_new; s_new=nullptr;
    delete d_new; d_new=nullptr;
}
```

Opción 3b: algunos de los nuevos elementos no requieren un puntero específico ya que son alcanzables desde algún otro objeto (en este caso la serie).

MainWindow.h

```
...
class MainWindow : public QMainWindow
{
    Q_OBJECT
private:
    ...
    QwtPlotCurve* c_new;       // Existe mientras haya MainWindow
    QVector<QPointF>* d_new;   // Existe mientras haya MainWindow
};
#endif // MAINWINDOW_H
```

MainWindow.cpp

```
void MainWindow::OnNewSeries()
{
    c_new=new QwtPlotCurve (argumentos del constructor);
    c_new->setData( new QwtPointSeriesData(args constructor) );
```

```
    // la serie se puede recuperar con c_new->data()
    c_new->attach(ui->qQwtPlotWidget);

}
```

4. Uso avanzado

La documentación de la librería es muy parca, no existen muchos tutoriales, así que es complicado alcanzar las máximas posibilidades de la misma sin un estudio exhaustivo. Se puede utilizar a modo de referencia:

https://ghorwin-github-io.translate.goog/qwtbook/basics/?_x_tr_sl=auto&_x_tr_tl=es&_x_tr_hl=en&_x_tr_pto=nui

Existen múltiples posibilidades en los gráficos, entre las cuales destacan:

- Tipos de gráficos: XY, de barras, …
- Widgets específicos: dial, barra, …
- Gestión de ejes
- Adición de leyendas e imágenes
- Gestión de la interacción del usuario a través de slots
- …