UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Mi primera aplicación Qt.
Entendiendo la filosofía

Apellidos, nombre	Perles Ivars, Àngel (aperles@disca.upv.es)
Departamento	Informática de Sistemas y Computadores
Centro	Universitat Politècnica de València

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

1 Resumen de las ideas clave

En este artículo se van a presentar los fundamentos sobre los que se apoya el desarrollo de aplicaciones multiplataforma en Qt. A continuación se resumen las ideas clave que se van a introducir:

- Los elementos que componen una aplicación Qt.
- La creación de una clase diálogo mediante derivación.
- El programa principal que contendrá la aplicación Qt.
- El archivo de proyecto con las pautas para construir la aplicación.
- La construcción de la aplicación.

2 Introducción

Es muy posible que hayas utilizado aplicaciones populares de ordenador como Skype, Google Earth, VLC, Calibre, y, si has llegado hasta aquí, quizá sea porque sabes que se han desarrollado usando Qt [1] y quieres aprender.

Pues, posiblemente, Qt es el mejor conjunto de herramientas para el desarrollo de aplicaciones multiplataforma en C++. Es, por tanto, un buen candidato para poder desarrollar aplicaciones de cualquier naturaleza con "moderada" sencillez.

Respecto a introducirse en su uso ... libros hay varios, pero quizá se pueda simplificar su introducción con la propuesta que se da aquí.

Esto no es un sustituto de un buen libro de Qt. Es una primera inmersión a pulmón libre en Qt. Al mismo tiempo, se recomienda ir usando el excelente libro de Molkentin [2].

El propósito de este tutorial es intentar transmitir la filosofía correcta sobre Qt/C++ de una manera lo más inmediata y limpia posible.

3 Objetivos

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

- Comprender los elementos básicos que componen una aplicación Qt.

4 Desarrollo

4.1 Requisitos previos

Este artículo solo se puede seguir sí:

- Tienes conocimientos del lenguaje de programación C, en especial los fundamentos de punteros.
- Tienes unas nociones básicas de programación orientada a objetos en C++.

4.2 Que se va a lograr

El producto final será una pequeñísima aplicación Qt, pero creada desde 0 para que comprendas todas piezas. En la figura 1 tienes la aplicación ejecutándose en varias plataformas.

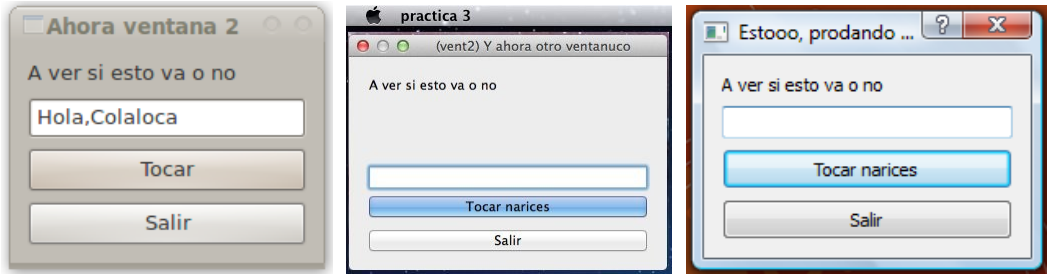

Figura 1.De izquierda a derecha. Ejecutándome en Linux, Mac OS X y Windows

4.3 Consiguiendo las herramientas

Necesitas instalarte Qt. Puedes desarrollar el trabajo utilizando las plataformas Linux, Microsoft Windows o Apple Mac OS X.

Este artículo se ha preparado usando Windows con la configuración que emplea solo herramientas libres, es decir Qt 5.0.2 32 bits para Windows con compilador mingw [3].

4.4 La primera aplicación

Vamos allá. Crearemos todos los elementos que van a formar la aplicación, que serán: una ventana de diálogo, un programa principal y un archivo de proyecto. Después le pasaremos las herramientas de construcción.

4.4.1 Definiendo una ventana de diálogo

Empecemos por una ventana. Para ello se propone crear una clase C++ que derivará de otras clases que proporciona Qt. No importa si no se comprenden ahora los significados de los nombres de las clases.

Como siempre en C++, primero hay que definir la clase. Ésta es la que se propone:

```
01.   /**
02.    * @file tventanuco.h
03.    * @brief Primer programa Qt para SII
04.   */
05.
06.   #ifndef TVENTANUCO_H
```

```
07.  #define TVENTANUCO_H
08.
09.  // Elementos que voy a necesitar
10.  #include <QDialog>  // voy a crear un dialogo!
11.  #include <QPushButton>
12.  #include <QVBoxLayout>
13.  #include <QLabel>
14.  #include <QLineEdit>
15.
16.  // definicion de la nueva clase TVentanuco que vamos
17.  // a derivar de QDialog
18.  class TVentanuco : public QDialog
19.  {
20.      Q_OBJECT    // <-- ponlo para que funcione signal/slot
21.
22.  public:
23.      TVentanuco(const char *mensaje);   // el constructor
24.
25.  private:
26.      // vamos a meter unas cuantas cosas
27.      double valor;
28.      QVBoxLayout *lyDistribuidor;
29.      QLabel *lbEtiqueta;
30.      QLineEdit *leEdicion;
31.      QPushButton *btToquetear;
32.      QPushButton *btCerrar;
33.  };
34.
35.  #endif // TVENTANUCO_H
```

La nueva clase ***TVentanuco*** va a ser una derivación de la clase ***QDialog*** de Qt. ***QDialog*** permite la construcción de ventanas de diálogo. Si se mira el manual de Qt, se hubiese podido derivar de otras cosas, por ejemplo, un ***QWidget***, una ***QMainWindow***, un ***QSplash***, etc.

En la línea 10 del código se tiene una inclusión de la cabecera ***QDialog*** sin el .h, que es necesaria porque se va a derivar de ***QDialog***. En Qt, para simplificar el tener que recordar nombres de cosas, las cabeceras y los tipos de clase tienen el mismo nombre. Así, si se necesita un ***QPushButton***, simplemente se tendrá que hacer ***#include <QPushButton>*** en el archivo que lo necesite.

Destacar en la línea 26 la inclusión de la etiqueta especial ***Q_OBJECT***. Hay que ponerla siempre.

En la zona privada del objeto, entre las líneas 31-38, se declaran varios punteros a clases de Qt. En el manual de Qt se puede mirar para qué sirven estas clases.

Ahora se pasa a la implementación. Se propone el siguiente listado:

```
01.  /**
02.   * @file tventanuco.h
03.   * @brief Primer programa Qt para SII
04.  */
05.
06.  // hacen las cabeceras de las clases Qt a usar
07.  #include <QPushButton>
08.  #include <QVBoxLayout>
09.  #include <QLabel>
10.  #include <QLineEdit>
11.
12.  #include "tventanuco.h"
13.
14.
15.  TVentanuco::TVentanuco(const char *mensaje)
16.      : QDialog()  // llamamos al constructor de la base
17.  {
18.      // ponemos el mensaje en el titulo del dialog
19.      setWindowTitle(mensaje);
20.
21.      //crear el distribuidor de objetos
22.      lyDistribuidor = new QVBoxLayout(this);
23.
24.      // creamos una etiqueta y la metemos en el "layout"
25.      lbEtiqueta = new QLabel(tr("A ver si esto va o no"));
26.      lyDistribuidor->addWidget(lbEtiqueta);
27.
28.      // parar jugar con los "layout", meto un "muelle"
29.      lyDistribuidor->addStretch();
30.
```

```
31.         // ahora una linea de edicion
32.         leEdicion = new QLineEdit;
33.         lyDistribuidor->addWidget(leEdicion);
34.
35.         // ahora un boton
36.         btToquetear = new QPushButton(tr("Tocar narices"));
37.         lyDistribuidor->addWidget(btToquetear);
38.
39.         // ahora el boton de cerrar
40.         btCerrar = new QPushButton(tr("Salir"));
41.         lyDistribuidor->addWidget(btCerrar);
42.     }
```

Lo primero es incluir la cabecera de las clases de Qt que necesitemos y que no estén ya en la cabecera de ***TVentanuco***. En este ejemplo se podrían eliminar las que se han puesto.

Lo único que se ha implementado es el constructor, que es bastante intuitivo. Simplemente se van creando dinámicamente los objetos y se van introduciendo en un "distribuidor" (Layout).

Observar que todos los textos que vayan a salir en pantalla van dentro de la función ***tr()*** (en realidad es ***QObject::tr()***) que, a partir de ahora, se recomienda usar, pues sirve para facilitar el desarrollo de versiones multilingües. Se podría eliminar ***tr()*** sin problema.

4.4.2 El programa para usar el diálogo

A continuación tenemos un programa principal muy sencillo para poder usar el diálogo antes propuesto.

```
01.   /**
02.    * @file main.cpp
03.    * @brief Primer programa Qt para SII
04.    *
05.    * Probando el TVentanuco desde la linea de ordenes de
   Windows
06.   */
07.
08.
09.   #include <QApplication>
10.   #include "tventanuco.h"
11.
```

```
12.  int main(int argc, char *argv[])
13.  {
14.          QApplication app(argc, argv);
15.          int resultado;
16.
17.          // un ventanuco creado dinamicamente
18.          TVentanuco *vent1;
19.          vent1 = new TVentanuco("(vent1) Estooo, probando");
20.          vent1->show();
21.
22.          // ventanuco creado estaticamente como variable
   automatica
23.          TVentanuco vent2("(vent2) Y ahora otro ventanuco");
24.          vent2.show();
25.
26.          // cuando termine la aplicacion, no se habra hecho
   "delete" de vent1,
27.          // pero si que se habra llamado automaticamente al
   destructor de vent2.
28.          // recordadme que os explique lo que pasa
29.
30.          resultado = app.exec(); // a funcionar! empieza el
   bucle de eventos.
31.
32.          return resultado;
33.  }
```

Como se quiere hacer una aplicación gráfica Qt, lo primero es crear una instancia de la clase ***QApplication***. En la línea 14 se crea ***app***, que es esa instancia, y a la que se le pasan los argumentos ***argc*** y ***argv*** que el S.O. le pasa a la aplicación. Esto permite, por ejemplo, que se pueda configurar la apariencia de nuestro programa.

Destacar que la construcción de ***app*** se hace usando variables automáticas. El beneficio de hacerlo así es que, al terminar el programa, C++ llamará automáticamente al destructor de ***app***.

De la línea 17 a la 24 se crean un par de objetos de la clase ***TVentanuco*** y se llama al método publico ***show()***.

Finalmente, se llama al método ***exec()*** de ***QApplication*** para pasar el control al bucle de eventos de Qt. En este punto, el programa se quede a la espera de que usemos los diálogos como se hace en cualquier aplicación de ventana.

4.4.3 El archivo de proyecto (.pro)

Se tienen ya todas las piezas de código C/C++ para poder construir la aplicación. El siguiente paso es proporcionar a Qt un archivo con la información de qué se quiere hacer con esas piezas.

El siguiente listado es un típico archivo de proyecto Qt para dar dicha información:

```
01.  # File: qt_per_tots.pro
02.  # Configuracion del proyecto para el primer ejemplo de SII
03.
04.  # Modulo Qt a emplear. Nos aseguramos que funciona tambien
   en Qt 5
05.  QT       += core gui
06.  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
07.
08.  # Tipo de destino deseado. En este caso una aplicacion
09.  TEMPLATES = app
10.
11.  # Nombre de la aplicacion destino
12.  TARGET = qt_per_tots
13.
14.  # con += anyades y con -= quitas
15.
16.  # Indicar la lista de modulos cpp que forman la aplicacion
17.  SOURCES += tventanuco.cpp
18.  SOURCES += main.cpp
19.
20.  # Indicar la lista de cabeceras propias que se van a usar
21.  HEADERS += tventanuco.h
```

La sintaxis de este archivo es parecida a un archivo de guión de Bourne shell para Unix. En este ejemplo básico, se establece el valor de ciertas variables que, en este caso, son:

- ***QT***: Para indicar los módulos de Qt a emplear. Destacar en este caso una condición para permitir que la aplicación se pueda construir en Qt 4 y en Qt 5.
- ***TEMPLATES***: Para indicar que tipo de salida queremos: un programa, una biblioteca, etc. En este caso, app significa que queremos un programa ejecutable.
- ***TARGET***: Nombre que se le dará al objeto destino.

- ***SOURCES***: Para indicar la lista de los módulos C/C++ que forman mi aplicación.
- ***HEADERS***: Para indicar los archivos de cabecera "míos" que son parte de la aplicación y quiero tener a mano.

Y ya está todo. Así de fácil, ¿no?.

4.4.4 Obteniendo el ejecutable a "pelo"

Ahora vamos a intentar obtener el ejecutable de la aplicación.

Los entornos de desarrollo integrados (IDE) son muy monos y útiles, pero muchísimas veces no se sabe lo que se está haciendo. El propósito de este apartado es ilustrar lo que hará el IDE que se usará más adelante y, así, comprender mejor la filosofía Qt.
Para ello se va utilizar la línea de órdenes de Microsoft Windows (probado en XP, Vista y 7). Para abrir una consola de Windows correctamente configurada, haremos: ***Inicio->Programas->Qt SDK *... ->Qt Command prompt***. La figura 2 muestra la consola abierta.

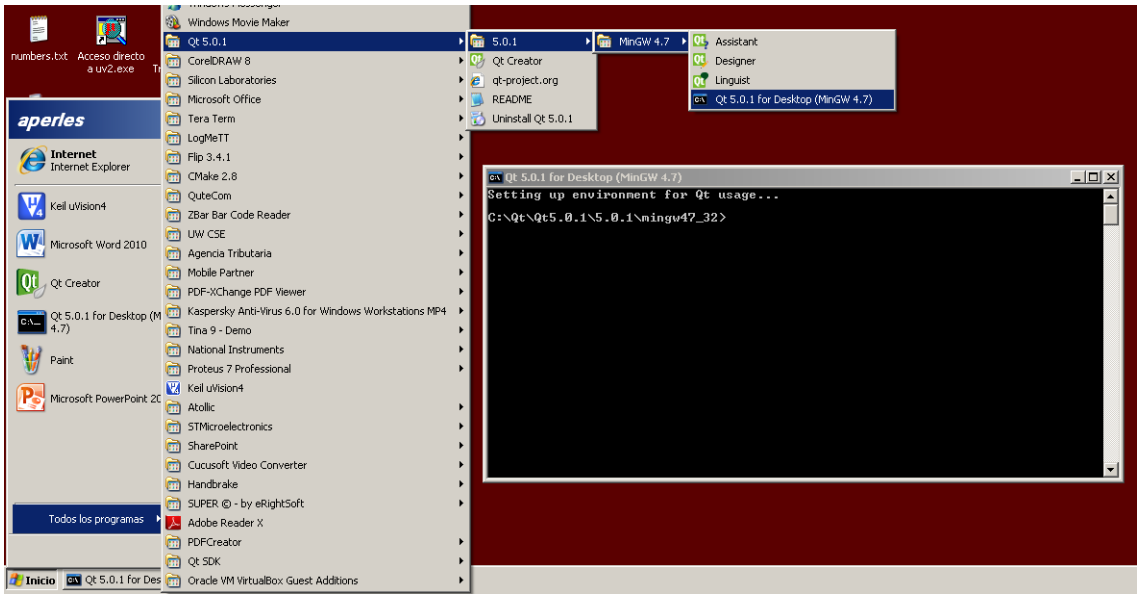

Figura 2. Llegando a la consola de Windows

Antes de continuar, hay que crear un directorio y depositar en él los archivos que hemos creado. Por ejemplo, podemos crear ***c:\qt_per_tots*** y movernos a él de la siguiente manera:

c:\ ...> md c:\qt_per_tots

c:\ ...> c:

c:\ ...> cd \qt_per_tots

Con el explorador de Windows, colocaremos los archivos en el directorio y usaremos la orden ***dir*** para comprobar el contenido.

Desde dentro del directorio, se usará la orden de Qt ***qmake*** con el nombre del archivo de proyecto como argumento.

c:\ ...> qmake qt_per_a_tots.pro

La orden ***qmake*** procesa el archivo de proyecto y genera archivos de guión (***makefiles***) y directorios que incluyen toda la información para que el compilador C/C++ configurado sea capaz de construir el proyecto.

Recuérdese que Qt es un conjunto de herramientas para desarrollo de aplicaciones en C/C++, pero no es un compilador de C/C++. Hay que emplear un conjunto de herramientas C/C++ externo para conseguir la aplicación. En función del entorno podríamos tener:

- Si se usa Linux o MacOSX, se deberá ejecutar ***make*** en el símbolo del sistema, y se procesarán, compilarán y enlazarán los archivos hasta obtener el ejecutable.
- Si se usa Windows y la versión LGPL de Qt , se deberá ejecutar ***mingw32-make*** porque se emplea el compilador GNU portado a Windows.
- Si se quiere desarrollar para Windows Mobile, Windows Phone o Windows con el compilador de Microsoft Visual C++, se deberá ejecutar ***nmake***.
- Etc. etc. etc. porque esta etapa puede llegar a ser bastante compleja. Téngase en cuenta que es posible desarrollar para iOS, Android, Blackberry 10, Jolla Sailfish, Linux empotrado (Raspberry Pi, Amazon Kindle, ...).

En este tutorial se sugiere empezar con la versión LGPL de Qt para Windows, por tanto, por debajo está el compilador de GNU ***gcc*** portado a Windows.

Ahora ya se pueden utilizar las herramientas externas que se haya configurado para procesar los archivos de guión. Estos archivos de guión tienen el nombre ***Makefile.**** y son de texto, así que estaría bien echarles un vistazo por dentro para entender un poquito más la idea (y asustarse).

Para procesar el archivo de guión se hará:

c:\...> mingw32-make

Y saldrán un montón de líneas y, al rato, dirá que ya está todo hecho (con suerte). La figura 3 muestra el aspecto del terminal una vez construido el proyecto.

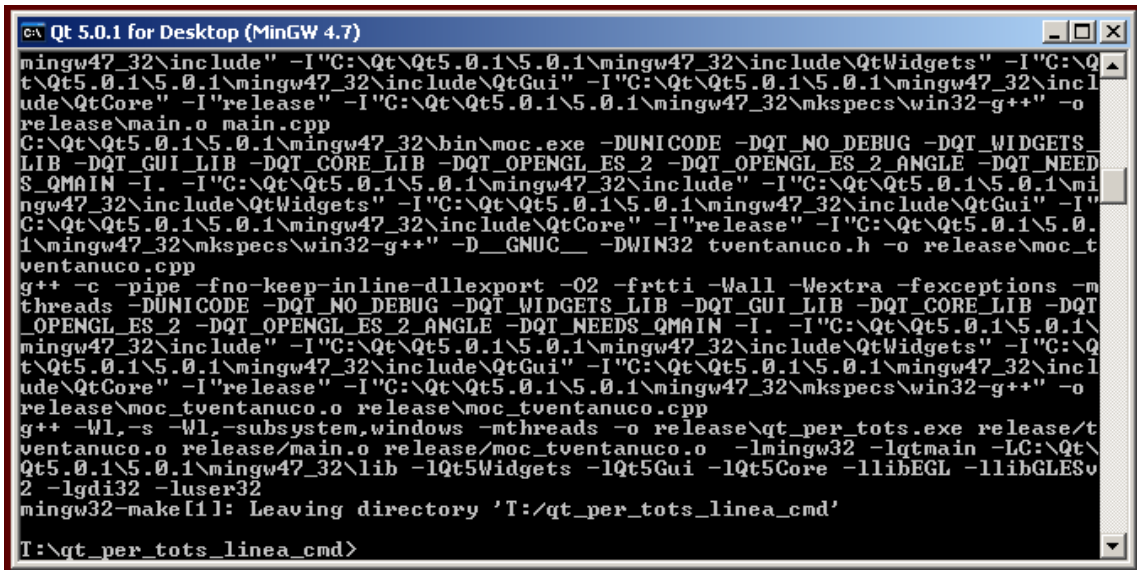

Figura 3. Lo que debería salir al hacer el "make" del proyecto

Si se ha copiado el programa a mano, seguro que salen un montón de errores y, para corregirlos, habrá que esforzarse en entender qué falla. Vale la pena intentarlo así esta vez. Se aprende mucho.

Ahora hay que comprobar si todo ha ido bien, para ello nos moveremos al directorio ***release*** y ejecutar el programa siguiendo estos pasos:

C:\...> cd release3C:\...> qt_per_tots.exe

Y, tachaaaaaannnnnnnnnnnnn. ¡¡¡¡¡¡Funcionannndooooo!!!!!!.

No funcionará si se hace clic desde el explorador de Windows. Eso será otro día.

5 Cierre

A lo largo de este objeto de aprendizaje hemos visto qué piezas básicas forman una aplicación Qt y cómo construirlas para lograr un ejecutable.

Debe quedar claro que la pretensión es tener claro el concepto de aplicación. A la hora de ser productivos, se recomienda emplear plantillas preconfiguradas y entornos de desarrollo integrados.

6 Bibliografía

[1] The Qt project. The Qt project foundation official page. 2013. Disponible en: URL: http://qt-project.org/.

[2] Molkentin, Daniel. The Book of Qt 4: The Art of Building Qt Applications. No Starch Press, 2007. ISBN: 1593271476 / 978-1593271473.

[3] Qt 5.0.2 for Windows 32-bit (MinGW 4.7, 650 MB). 2013. Disponible en: http://qt-project.org/downloads