Introducción al desarrollo multiplataforma con Qt® 4

Lisandro Damián Nicanor Pérez Meyer – Septiembre 2007

Introducción al desarrollo multiplataforma con Qt 4

Lisandro Damián Nicanor Pérez Meyer
perezmeyer en/at gmail.com
http://perezmeyer.com.ar/

Los ejemplos utilizados en éstas transparencias pertenecen al Tutorial de Qt 4.3 [0], parte de la documentación oficial de Qt [1].

[0] http://doc.trolltech.com/4.3/tutorial.html
[1] http://doc.trolltech.com/

Ésta presentación se encuentra disponible en
http://perezmeyer.com.ar/files/introduccionAQt/

Introducción al desarrollo multiplataforma con Qt 4

Bahía Blanca Linux Users Group
http://bblug.usla.org.ar/

Rama Estudiantil IEEE UNS
http://www.ieee.uns.edu.ar/

Departamento de Ingeniería
Eléctrica y Computadoras – UNS
http://www.ingelec.uns.edu.ar/

Grupo de Usuarios de
Software Libre de Córdoba
http://www.grulic.org.ar/

Grupo de Usuarios de software Libre
de la Facultad de Ingeniería
http://lug.fi.uba.ar/

Lisandro Damián Nicanor Pérez Meyer – Septiembre 2007

Introducción al desarrollo multiplataforma con Qt 4

¿Qué es Qt?

Qt es un framework para el desarrollo de aplicaciones multiplataforma.
Algunas de sus características son:

- Compatibilidad multiplataforma con un sólo código fuente
- Performance de C++
- Disponibilidad del código fuente
- Excelente documentación
- Fácilmente internacionabilizable
- Arquitectura lista para plugins

Pero lo mas importante de todo es que...

Lisandro Damián Nicanor Pérez Meyer - Septiembre 2007

Introducción al desarrollo multiplataforma con Qt 4

A los programadores les gusta :-)

Lisandro Damián Nicanor Pérez Meyer – Septiembre 2007

Disponibilidad de Qt

- Qt commercial edition
- Qt Open Source edition

Adivinen cuál vamos a ver... :-)

Componentes del framework Qt

- **Las librerías Qt** (clases en C++)
- **Qt Designer**, para crear formularios visualmente
- **Qt Assistant**, acceso rápido a la documentación
- **Qt Linguist**, traducción rápida de programas
- **qmake**, simplifica el proceso de construcción de proyectos en las diferentes plataformas soportadas
- Mas herramientas como moc, uic, rcc, qtconfig, ...

Introducción al desarrollo multiplataforma con Qt 4

Qué vamos a ver

Esta charla pretende ser una introducción a la programación de Interfaces Gráficas de Usuario (GUIs) usando Qt 4.
Iremos introduciendo las características de Qt a medida que sea necesario.
Crearemos todas las interfaces a través de código, sin utilizar Qt Designer, para poder ir descubriendo la filosofía detrás de Qt.

Lisandro Damián Nicanor Pérez Meyer – Septiembre 2007

Introducción al desarrollo multiplataforma con Qt 4

¡Hola mundo!

Infaltable en el botiquín del programador, nuestro primer encuentro será creando una aplicación que muestre el ya conocido ¡Hola mundo!
Éste ejemplo contiene lo mínimo necesario para obtener una aplicación hecha en Qt andando.

Introducción al desarrollo multiplataforma con Qt 4

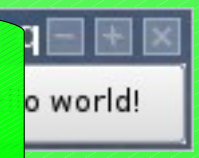
```
#include <QApplication>
#include <QPushButton>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QPushButton hello("Hello world!");
    hello.show();
    return app.exec();
}
```

Incorporamos la definición de la clase *QApplication*. Tiene que existir exactamente una *QApplication* por cada aplicación GUI que utilice Qt. *QApplication* es un ejemplo de patrón Singleton. *QApplication* maneja varios recursos globales de la aplicación, como el cursor y la fuente por defecto.

Introducción al desarrollo multiplataforma con Qt 4

```
#include <QApplication>
#include <QPushButton>

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

    QPushButton hello("¡Hola mundo!");
    hello.resize(100, 30);

    hello.show();
    return app.exec();
}
```

Para cada clase que sea parte pública de la API de Qt, existe un encabezado del mismo nombre que contiene su definición



Lisandro Damián Nicanor Pérez Meyer - Septiembre 2007

Introducción al desarrollo multiplataforma con Qt 4

```
#include <QApplication>
#include <QPushButton>

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

    QPushButton hello("Hello world!");
    hello.show();

    return app.exec();
}
```

La función *main* es el punto de entrada al programa.
El parámetro *argc* es el número de argumentos de la linea de comandos, y *argv[]* es un arreglo de argumentos de linea de comandos.

Lisandro Damián Nicanor Pérez Meyer - Septiembre 2007

Introducción al desarrollo multiplataforma con Qt 4

```
#include <QApplication>
#include <QPushButton>

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

    QPushButton hello("¡Hola mundo!");
    hello.resize(100, 30);

}
```

El objeto *app* es la instancia de *QApplication* de nuestro programa. Notar que se le pasa *argc* y *argv* como argumento para que pueda procesar argumentos de lineas de comando como *-display* en X11

Lisandro Damián Nicanor Pérez Meyer - Septiembre 2007

Introducción al desarrollo multiplataforma con Qt 4

Creamos un botón, y le asignamos el texto *“¡Hola mundo!”*. Debido a que no le especificamos una ventana padre (el segundo argumento de *QPushButton*), el mismo será una ventana por si mismo, con su marco y barra de título.

```
{
    QAppli...  app(argc, argv);

    QPushButton hello("¡Hola mundo!");
    hello.resize(100, 30);

    hello.sho...
    return ar...();
}
```

Le damos un tamaño inicial en pixeles, excluyendo el marco y la barra de título de la ventana.

Introducción al desarrollo multiplataforma con Qt 4

```
#include <QApplication>
#include <QPushButton>

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

    QPushButton hello("¡Hola mundo!");
    hello.resize(100, 30);

    hello.show();
    return app.exec();
}
```

Un widget (así llamamos a todos los componentes gráficos de Qt) **nunca** se muestra cuando se crea, para eso hay que llamar a la función *show()*

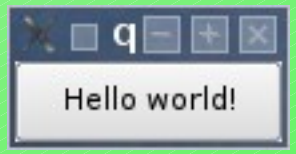
Introducción al desarrollo multiplataforma con Qt 4

Aquí es donde *main()* le pasa el control a Qt. *QCoreApplication::exec()* va a retornar cuando la aplicación termine.
QCoreApplication es la clase base de *QApplication*. Implementa el núcleo de la funcionalidad no-GUI de *QApplication*, y puede ser utilizado cuando se desarrollan aplicaciones **no GUI.**
En *QCoreApplication::exec()*, Qt recibe y procesa los eventos de usuario y sistema y se los pasa al widget apropiado.

```
    QPushB                ("¡Hola mundo!");
    hello.                0, 30);

    hello.s               w();
    return app.exec();
}
```

Introducción al desarrollo multiplataforma con Qt 4

¡A cocinar la torta!



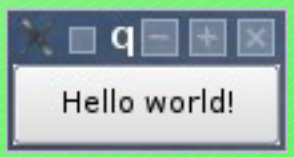
Ya tenemos nuestro código fuente listo para llevar al horno. Pero, para compilar una aplicación C++, normalmente se utiliza un archivo Makefile. Ésto suele ser un poco engorroso, mas si estamos pensando en que debería ser multiplataforma. Para eso existe la herramienta qmake:
Guardamos nuestro código como main.cpp y luego hacemos:

```
qmake -project
qmake
```

El primer comando le dice a qmake que cree un archivo de proyecto de extensión .pro. El segundo le dice que, basándose en el .pro existente, cree un archivo Makefile acorde a nuestra plataforma.
Sólo nos queda ejecutar make y ¡listo!, ya podemos ejecutar nuestro primer programa en Qt.

Lisandro Damián Nicanor Pérez Meyer - Septiembre 2007

Introducción al desarrollo multiplataforma con Qt 4



El proyecto (qt.pro)

```
TEMPLATE = app
TARGET = Ej1
DEPENDPATH += .
INCLUDEPATH += .

# Input
SOURCES += main.cpp
```

Introducción al desarrollo multiplataforma con Qt 4

Agreguemos funcionalidad

```
#include <QApplication>
#include <QFont>
#include <QPushButton>

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

  QPushButton quit("Salir");
  quit.resize(75, 30);
  quit.setFont(QFont("Times", 18, QFont::Bold));

  QObject::connect(&quit, SIGNAL(clicked()),
                   &app, SLOT(quit()));

  quit.show();
  return app.exec();
}
```

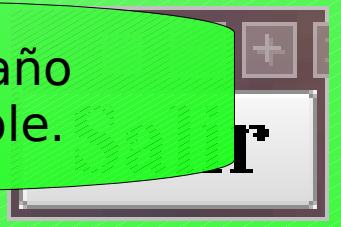
Ahora usaremos fuentes a través de la clase *QFont*

Introducción al desarrollo multiplataforma con Qt 4

Agreguemos funcionalidad

```
#include <QApplication>
#include <QFont>
#include <QPushButton>

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

    QPushButton quit("Salir");
    quit.resize(75, 30);
    quit.setFont(QFont("Times", 18, QFont::Bold));

    QObject::connect(&quit, SIGNAL(clicked()),
                     &app, SLOT(quit()));

    quit.show();
    return app.exec();
}
```

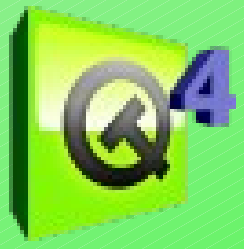
Podríamos haber usado *QFontMetrics* para ajustar el tamaño correcto o dejar que *QPushButton* elija un tamaño razonable.



Elegimos una fuente de 18 puntos para el botón, de la familia Times. También es posible cambiar la fuente por defecto para toda la aplicación utilizando *QApplication::setFont()*.

QObject::connect() es quizás la característica mas importante de Qt. Notar que *connect()* es una función estática de *QObject*.
La llamada a *connect()* establece una conexión de un sólo sentido entre dos objetos de Qt (objetos que heredan de *QObject*, directa u indirectamente). Cada objeto Qt puede tener tanto *signals* (señales) para enviar mensajes como slots (puertos) para recibirlos. Todos los widgets son objetos Qt, debido a que heredan de *QWidget*, que a su vez hereda de *QObject*.
Aquí la señal *clicked()* del botón *quit* está conectada al slot *quit()* de la aplicación, de manera de que cuando el botón sea apretado, la aplicación se cierre.

```
  QPushBut                    lir");
  quit.resiz                  );
  quit.setFon                 nt("Times", 18, QFont::Bold));

  QObject::connect(&quit, SIGNAL(clicked()),
                   &app, SLOT(quit()));

  quit.show();
  return app.exec();
}
```



Como padre e hijo

En nuestro próximo ejemplo crearemos nuestro propio widget y veremos como Qt se encarga de las relaciones entre ellos.

Introducción al desarrollo multiplataforma con Qt 4

```
#include <QApplication>
#include <QFont>
#include <QPushButton>
#include <QWidget>

class MyWidget : public QWidget
{
public:
  MyWidget(QWidget *parent = 0);
};
```

Aquí creamos una nueva clase. Como la misma hereda de *QWidget*, la nueva clasé será un widget y podrá ser tanto una ventana como un widget hijo (como el *QPushButton* del capítulo anterior).
Sólo definimos un constructor. Los demás métodos se heredan de *QWidget.*
El constructor es un constructor estándar de un widget Qt. Siempre se debe crear un constructor similar cuando se crean widgets.
El argumento es el widget padre. Para crear una ventana se debe especificar un puntero nulo (como es el caso por defecto de éste ejemplo).

```
}
```

Introducción al desarrollo multiplataforma con Qt 4

```
#include <QApplication>
#include <QFont>
#include <QPushButton>

public:
  MyWidget(QWidget *parent = 0);
};

MyWidget::MyWidget(QWidget *parent)
                  : QWidget(parent)
{
  setFixedSize(200, 120);

  QPushButton *quit = new QPushButton(tr("Salir"), this);
  quit->setGeometry(62, 40, 75, 30);
  quit->setFont(QFont("Times", 18, QFont::Bold));

  connect(quit, SIGNAL(clicked()), qApp, SLOT(quit()));
}
```

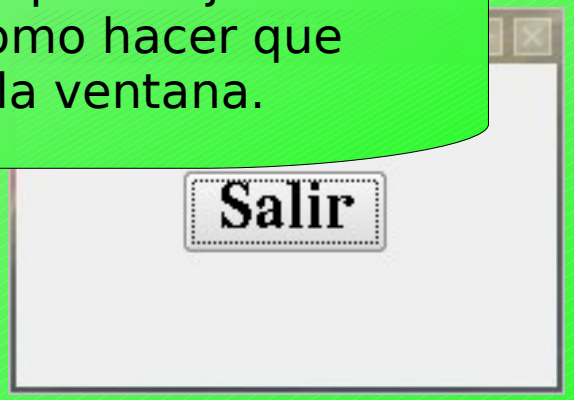
La implementación del constructor comienza aquí. Como la mayoría de los widgets, sólo pasa el padre al constructor de *QWidget*.

Lisandro Damián Nicanor Pérez Meyer - Septiembre 2007

Introducción al desarrollo multiplataforma con Qt 4

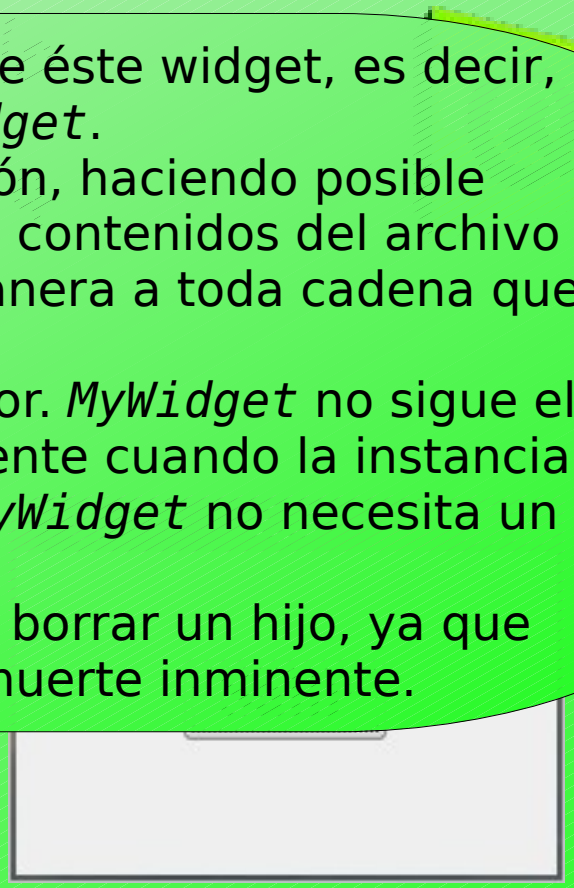
```
#include <QApplication>
#include <QFont>
#include <QPushButton>
#i

cl
{
pub
  MyWidget(QWi                    0);
};

MyWidget::MyWid                 idget *parent)
                  : QWidget(parent)
{
  setFixedSize(200, 120);

  QPushButton *quit = new QPushButton(tr("Salir"), this);
  quit->setGeometry(62, 40, 75, 30);
  quit->setFont(QFont("Times", 18, QFont::Bold));

  connect(quit, SIGNAL(clicked()), qApp, SLOT(quit()));
}
```

El widget no sabe como redimensionarse, por lo que le fijaremos su tamaño. En el próximo ejemplo veremos como hacer que responda ante un redimensionamiento de la ventana.



Lisandro Damián Nicanor Pérez Meyer - Septiembre 2007

Aquí creamos y damos parámetros un widget hijo de éste widget, es decir, el padre es la instancia de *MyWidget*.
La función tr() marca la cadena para su traducción, haciendo posible cambiarla en tiempo de ejecución basándose en los contenidos del archivo de traducción. Es una buena idea marcar de ésta manera a toda cadena que vé el usuario.
Notar que *quit* es una variable local en el constructor. *MyWidget* no sigue el rastro de ella, Qt lo hace, y la borrará automáticamente cuando la instancia del objeto *MyWidget* sea borrada. Por ésto es que *MyWidget* no necesita un destructor.
Por otro lado, no ocurre daño alguno si uno decide borrar un hijo, ya que éste le avisará automáticamente a Qt de su muerte inminente.



```
};

MyWidget::My                 t *parent)
                           dget(parent)
{
  setFixedSize      , 120);

  QPushButton *quit = new QPushButton(tr("Salir"), this);
  quit->setGeometry(62, 40, 75, 30);
  quit->setFont(QFont("Times", 18, QFont::Bold));

  connect(quit, SIGNAL(clicked()), qApp, SLOT(quit()));
}
```

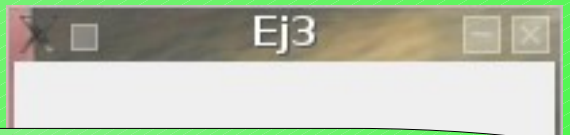
Introducción al desarrollo multiplataforma con Qt 4

```
#include <QApplication>
#include <QFont>
#include <QPushButton>
#include <QWidget>

class MyWidget : public QWidget
{
public:
  MyWidget(QWidget *parent = 0);
};

MyWidget::MyWidget(QWidget *parent)
                  : QWidget(parent)
{
  setFixedSize(200, 120);

  QPushButton *quit = new QPushButton(tr("Salir"), this);
  quit->setGeometry(62, 40, 75, 30);
  quit->setFont(QFont("Times", 18, QFont::Bold));
```

La llamada a *QWidget::setGeometry()* setea tanto la posición del widget en la pantalla como el tamaño. Es equivalente a llamar a *QWidget::move()* seguido de *QWidget::resize()*.

Introducción al desarrollo multiplataforma con Qt 4

```
#include <QApplication>
#include <QFont>
#include <QPushButton>
#include <QWidget>

class MyWidget : public QWidget
{
public:

};

My

{
  setFixedSize(200,

  QPushButton *quit = new                on(tr("Salir"), this);
  quit->setGeometry(62, 40, 7
  quit->setFont(QFont("Times", 18,  Font::Bold));

  connect(quit, SIGNAL(clicked()), qApp, SLOT(quit()));
}
```



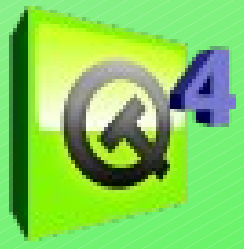
El puntero *qApp* es una variable local declarada en el encabezado de *<QApplication>*. Apunta a la única instancia de *QApplication* en el programa.

Lisandro Damián Nicanor Pérez Meyer - Septiembre 2007

Introducción al desarrollo multiplataforma con Qt 4

```
int main(int argc, char *argv[])
{
  QApplication app(argc, argv);
  MyWidget widget;
  widget.show();
  return app.exec();
}
```

Aquí instanciamos a *MyWidget*, lo mostramos e iniciamos la aplicación.

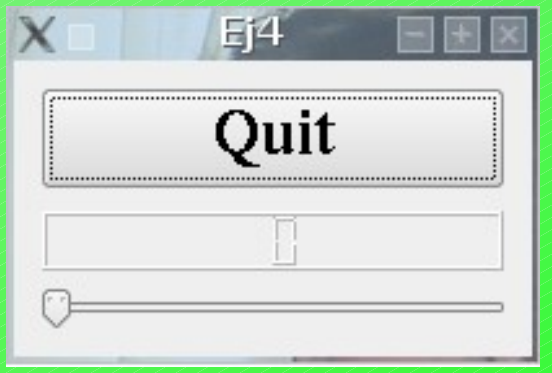


Acomodando las cosas dinámicamente

En el siguiente ejemplo veremos el uso de layouts y continuaremos viendo el concepto de signals y slots.

Introducción al desarrollo multiplataforma con Qt 4

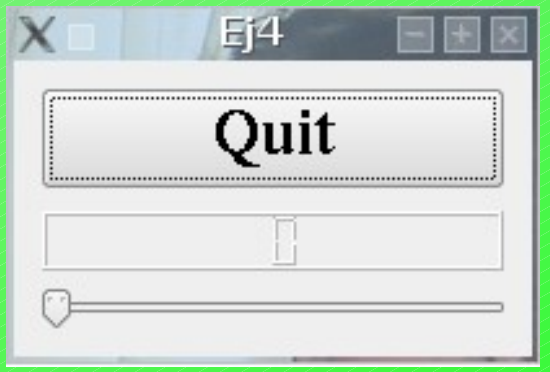
```
#include <QPushButton>
#include <QSlider>
#include <QVBoxLayout>
#include <QWidget>
#include <QLCDNumber>

class MyWidget : public QWidget
{
public:
  MyWidget(QWidget *parent = 0);
};
```



Definimos nuestra clase como hemos visto antes, y agregamos unos nuevos encabezados.

Introducción al desarrollo multiplataforma con Qt 4

```
MyWidget::MyWidget(QWidget *parent)
                  : QWidget(parent)
{
  QPushButton *quit = new QPushButton(tr("Quit"));
  quit->setFont(QFont("Times", 18, QFont::Bold));

  QLCDNumber *lcd = new QLCDNumber(2);
  lcd->setSegmen Style(QLCDNumber::Filled);

  QSlider *sl          QSlider(Qt::Horizontal);
  slider->se
  slider->
```

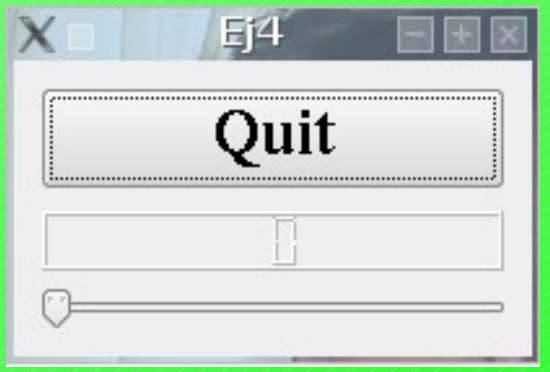


lcd es un *QLCDNumber*, un widget que muestra números emulando a un display LCD. Ésta instancia se configuró para que muestre dos dígitos. Configuramos la propiedad *QLCDNumber::segmentStyle* a *QLCDNumber::Filled* para hacer fácil la legibilidad del widget.

```
  layout->addWidget(quit);
  layout->addWidget(lcd);
  layout->addWidget(slider);
  setLayout(layout);
}
```

Introducción al desarrollo multiplataforma con Qt 4

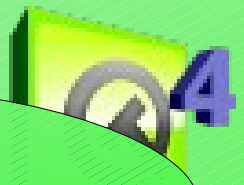
```
MyWidget::MyWidget(QWidget *parent)
                  : QWidget(parent)
{
  QPushButton *quit = new QPushButton(tr("Quit"));
  quit->setFont(QFont("Times", 18, QFont::Bold));

  QLCDNumber *lcd = new QLCDNumber(2);
  lcd->setSegmentStyle(QLCDNumber::Filled);

  QSlider *slider = new QSlider(Qt::Horizontal);
  slider->setRange(0, 99);
  slider->setValue(0);

  connect(quit      NAL(clicked()), qApp, SLOT(quit()));
  connect(sli       NAL(valueChanged(int)),
          l         lay(int)));
}
```



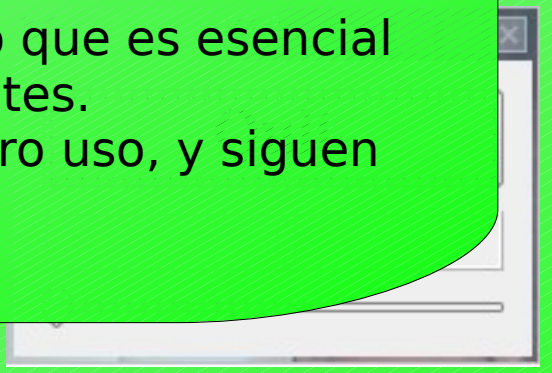
El usuario puede usar el widget *QSlider* para ajustar un valor entero dentro de un rango, en éste caso decidimos que fuese de 0 a 99, con valor inicial 0.

Lisandro Damián Nicanor Pérez Meyer - Septiembre 2007

Introducción al desarrollo



Aquí usamos el mecanismo de signals y slots para conectar la señal *valueChanged()* del slider al slot *display()* del LCD.
Cada vez que el slider cambie de valor, comunica el nuevo valor emitiendo la señal *valueChanged()*. Debido a que dicha señal está conectada a al slot *display()*, el mismo es llamado cuando se emite la señal. Ninguno de los dos objetos sabe sobre el otro, lo que es esencial en la programación orientada a componentes.
Los slots son funciones normales de C++ para todo otro uso, y siguen las normas de acceso normal de C++.



```
    QSli
    slider->setR
    slider->setVal

    connect(quit, SIGNAL(clicked()), qApp, SLOT(quit()));
    connect(slider, SIGNAL(valueChanged(int)),
            lcd, SLOT(display(int)));

    QVBoxLayout *layout = new QVBoxLayout;
    layout->addWidget(quit);
    layout->addWidget(lcd);
    layout->addWidget(slider);
    setLayout(layout);
}
```

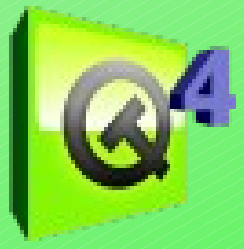
Introducción al desarrollo multiplataforma con Qt 4

MyWidget ahora usa *QVBoxLayout* para manejar la geometría de sus widgets hijos. Ésto también logra que los widgets hijos sean redimensionados cuando la ventana es redimensionada. Luego agregamos los widgets *quit*, *lcd* y *slider* al layout usando *QBoxLayout::addWidget()*.

La función *QWidget::setLayout()* instala el layout en *MyWidget*, convirtiendo al layout y a todos sus componentes en sus hijos, por lo que no debemos preocuparnos por destruirlos. Es por ésto que no especificamos el padre a *quit*, *lcd* y *slider*.

```
MyWid

    connect(qui                    cked()), qApp, SLOT(quit()));
    connect(sli                   AL(valueChanged(int)),
            lcd,                  (display(int)));

    QVBoxLayout *layout = new QVBoxLayout;
    layout->addWidget(quit);
    layout->addWidget(lcd);
    layout->addWidget(slider);
    setLayout(layout);
}
```

Introducción al desarrollo multiplataforma con Qt 4



Hasta el momento estuvimos viendo aplicaciones sencillas, pero que cubren parte de las características mas importantes de Qt.
Veamos otros ejemplos...

Introducción al desarrollo multiplataforma con Qt 4

Un ejemplo mas del tutorial, ésta vez haciendo mas hincapié en la “magia” de los signals y slots.

Lisandro Damián Nicanor Pérez Meyer - Septiembre 2007

Introducción al desarrollo multiplataforma con Qt 4

Un ejemplo mas del tutorial, ésta vez haciendo mas hincapié en la “magia” de los signals y slots.

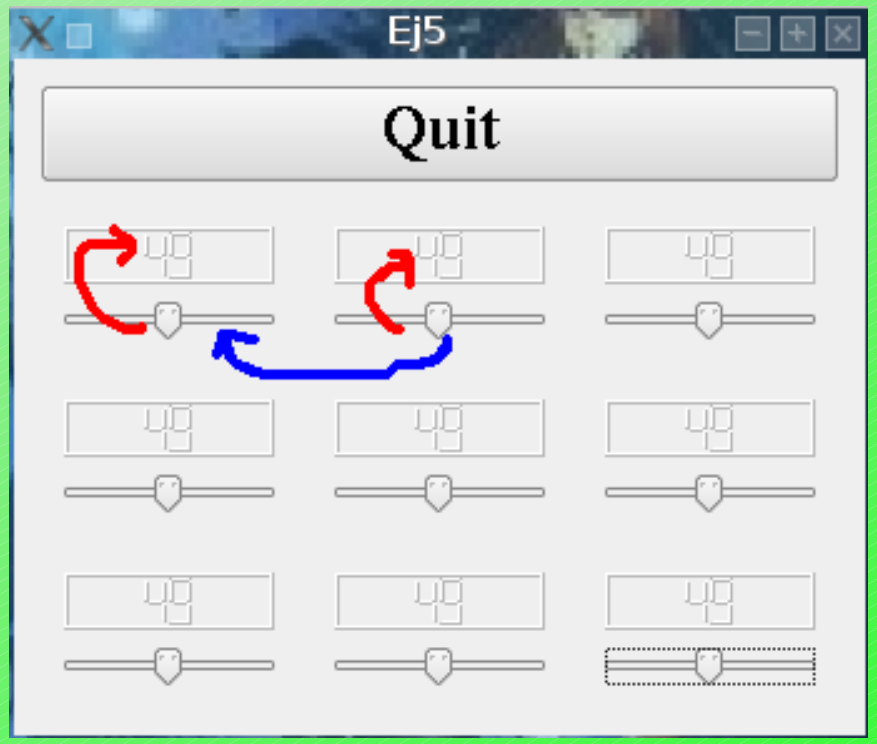
Lisandro Damián Nicanor Pérez Meyer - Septiembre 2007

Introducción al desarrollo multiplataforma con Qt 4

Un ejemplo mas del tutorial, ésta vez haciendo mas hincapié en la “magia” de los signals y slots.

Lisandro Damián Nicanor Pérez Meyer - Septiembre 2007

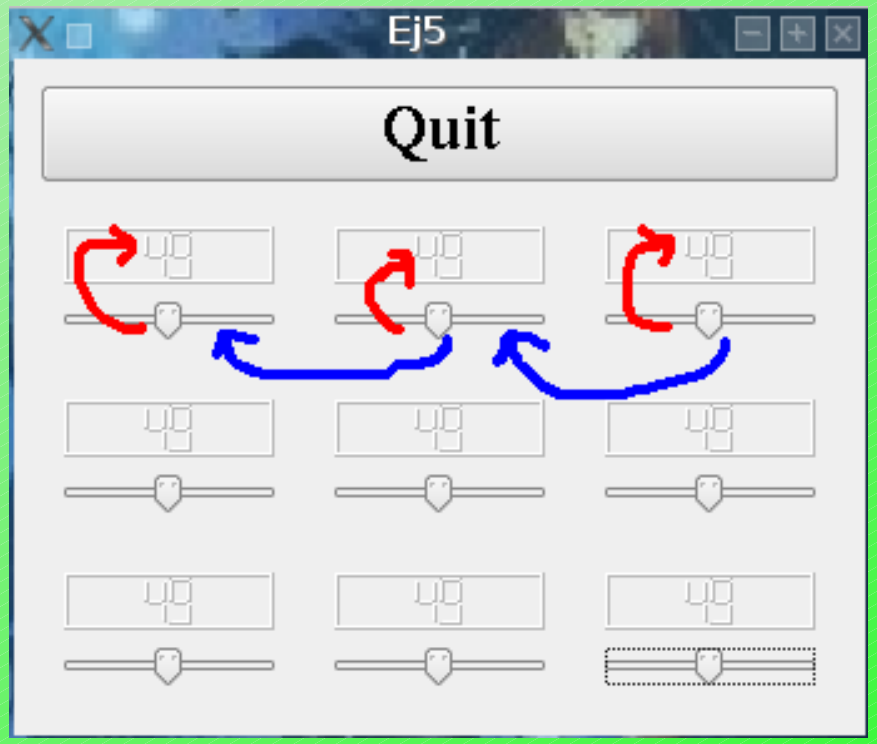
Introducción al desarrollo multiplataforma con Qt 4



Un ejemplo mas del tutorial, ésta vez haciendo mas hincapié en la “magia” de los signals y slots.

Lisandro Damián Nicanor Pérez Meyer - Septiembre 2007

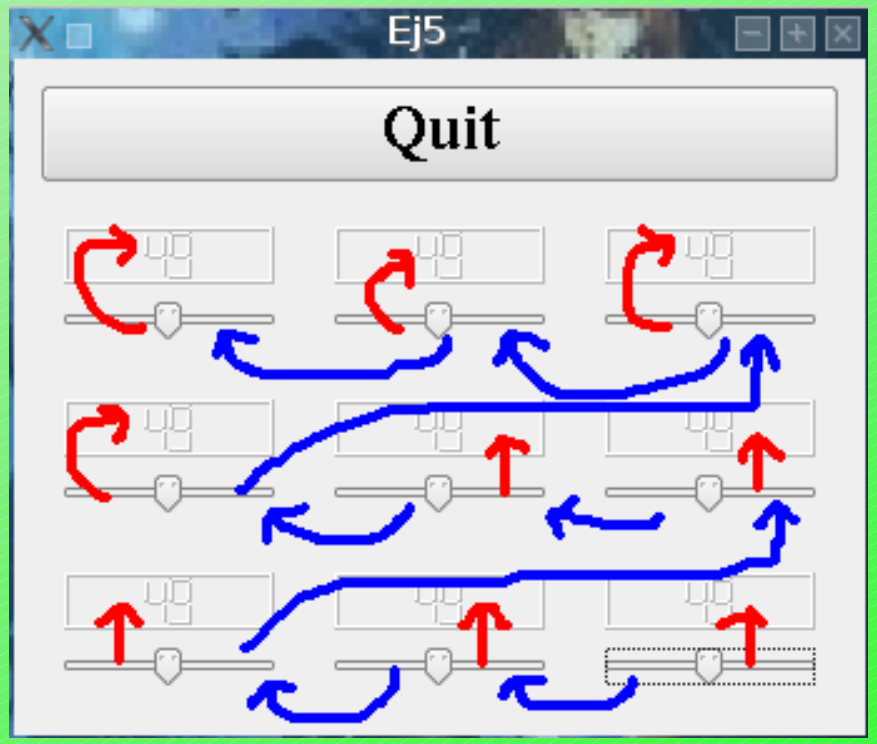
Introducción al desarrollo multiplataforma con Qt 4



Un ejemplo mas del tutorial, ésta vez haciendo mas hincapié en la “magia” de los signals y slots.

Lisandro Damián Nicanor Pérez Meyer - Septiembre 2007

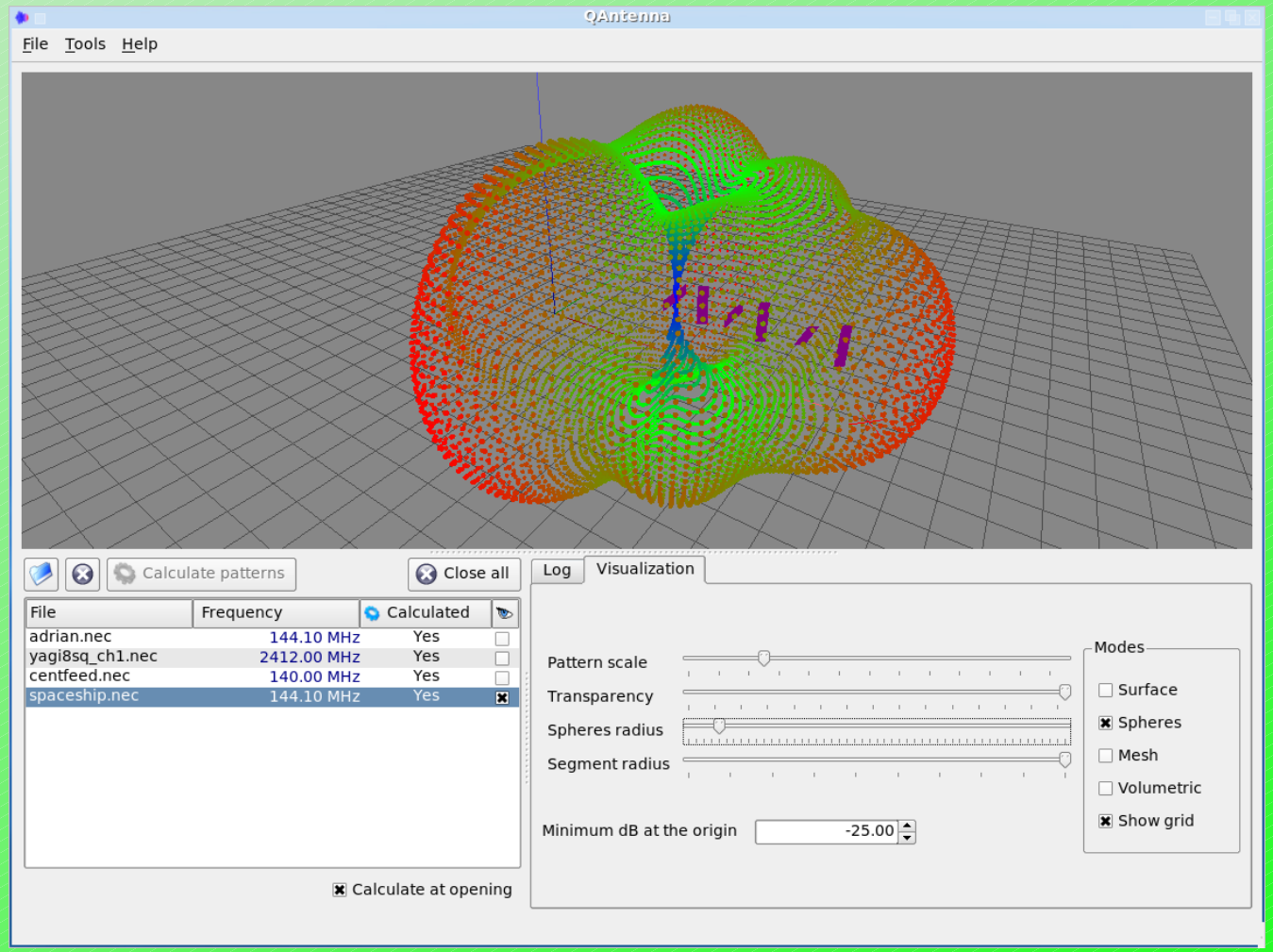
Introducción al desarrollo multiplataforma con Qt 4



Un ejemplo mas del tutorial, ésta vez haciendo mas hincapié en la “magia” de los signals y slots.

Lisandro Damián Nicanor Pérez Meyer - Septiembre 2007

Introducción al desarrollo multiplataforma con Qt 4



Un ejemplito que hace uso de OpenGL, cortesía de la gente de QAntenna (entre las que se encuentra vuestro servidor ;-))

Lisandro Damián Nicanor Pérez Meyer - Septiembre 2007

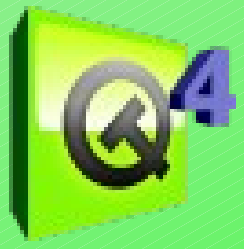
Y no nos podemos olvidar de...

Introducción al desarrollo multiplataforma con Qt 4

Lisandro Damián Nicanor Pérez Meyer - Septiembre 2007

¿Preguntas?

Introducción al desarrollo multiplataforma con Qt 4



¡Gracias!

Lisandro Damián Nicanor Pérez Meyer – Septiembre 2007