Introducción a Qt.

Programación gráfica en C++ con Qt4

Salvador Alemany Garrido

21 de noviembre de 2009

Contenido

Contenido

Prerequisitos de la charla y objetivos

Requisitos

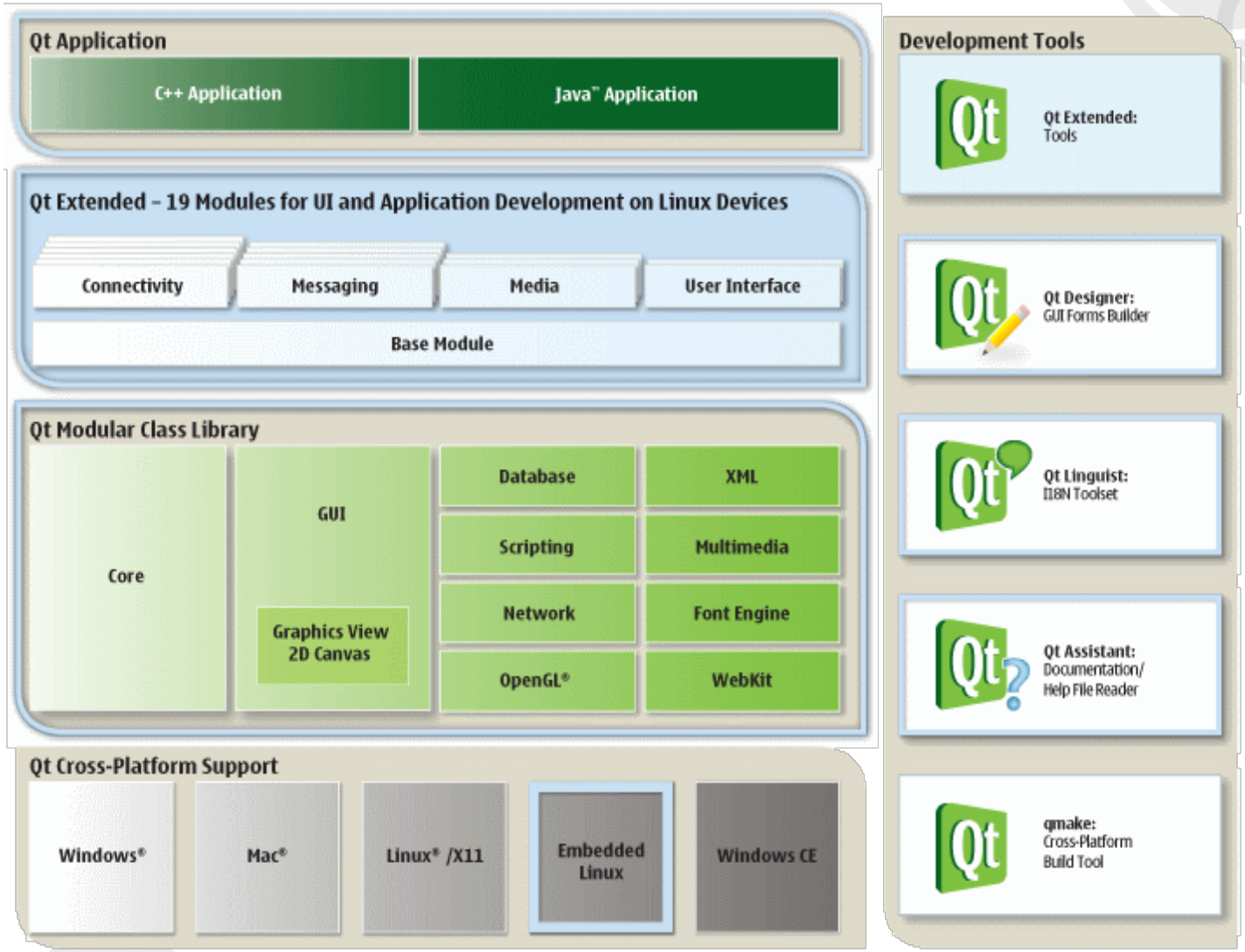
- Imprescidible: conocimientos básicos de C++, o bien C y Java.
- Importante:
  - Nociones de depuradores e IDEs
  - Nivel de inglés suficiente para la documentación oficial

Objetivos

- Conocer los componentes y las posibilidades de Qt
- Saber crear una aplicación, y opciones básicas
- Uso de widges básicos conjuntamente
- Definición de widgets personalizados.

¿Qué es Qt?

- Qt es un framework de desarrollo de aplicaciones multiplataforma.
- Viene acompañado de un conjunto de herramientas para facilitar su uso.


Qt Application
C++ Application
Java™ Application
Qt Extended – 19 Modules for UI and Application Development on Linux Devices
Connectivity
Messaging
Media
User Interface
Base Module
Qt Modular Class Library
Core
GUI
Graphics View
2D Canvas
Database
XML
Scripting
Multimedia
Network
Font Engine
OpenGL®
WebKit
Qt Cross-Platform Support
Windows®
Mac®
Linux® /X11
Embedded Linux
Windows CE
Development Tools
Qt Extended:
Tools
Qt Designer:
GUI Forms Builder
Qt Linguist:
I18N Toolset
Qt Assistant:
Documentation/
Help File Reader
qmake:
Cross-Platform
Build Tool

Qt, los módulos I

QtCore	contiene el núcleo no gráfico de Qt
QtGui	La colección básica de componentes gráficos
QtNetwork	Clases para escribir clientes y servidores TCP/IP
QtOpenGL	Para facilitar el uso de openGL
QtScript	Expone las aplicaciones a scripting con un lenguaje ECMAscript
QtScriptTools	un depurador de QtScript
QtSQL	integración de bases de datos
QtSVG	Soporte SVG

Qt, los Módulos II

QtWebKit	el popular motor web, con Qt
QtXml	soporte básico de Xml
QtXmlPatterns	un motor de XQuery 1.0 y XPath 2.0 y parcialmente Xslt
Phonon	El framework multimedia
Qt3Support	Compatibilidad con Qt3
Otros	QtDessigner, QtUiTools, QtHelp, QtAssistant, QtTest, QtDBus(solo Unix), y a partir de Qt 4.6 QtOpenVG y QtMultimedia

Herramientas

Charla

- Editor de textos y/o IDE. Kate y Qt Creator
- qmake, gestión de proyectos
- Qt designer,diseñador de interfaces
- Qt assistant, visor de documentación

Otras Herramientas

- Qt linguist, traducción
- Cmake, alternativa más flexible y potente a qmake
- Otros IDEs: Netbeans, Eclipse... ejem Visual Studio

Contenido

Tipos I

QtGlobal: tipos básicos

Tipos globales

Tenemos los típicos alias para los tipos básicos:

Tipo	Descripción
qintSIZE	Entero con los tamaños: 8, 16, 32, 64
quintSIZE	Entero sin signo de tamaños: 8, 16, 32, 64
qreal	double, excepto en ARM que es float
quintptr	Entero sin signo de ancho de palabra

Más en la documentación(QtGlobal)

Tipos II

Contenedores genéricos

Qt incluye una colección de contenedores genéricos:

- **QList** es el contenedor genérico más usado, **no** es una lista enlazada como std::list (el equivalente sería QLinkedList) sino que internamente usa un indice en array, es la clase padre de QStringList clave para gestionar colecciones de QString.
- **Otros:** QLinkedList, QVector, QStack, QQueue, QSet, QMap, QMultiMap, QHash, QMultiHash.
- Lo contenedores de Qt disponen de 2 tipos de iteradores: estilo STL, y estilo Java, que aputan entre los elementos y no a los elementos.

Más información en `Container Classes`

Tipos II

Las clases básicas del sistema de objetos

QObject

La base del sistema de objetos de Qt. Todo Widget y la mayoría de las otras clases de Qt heredan de él.

QApplication o QCoreApplication

La clase `QApplication` efectúa el control de flujo y las opciones principales de la aplicación. Solo una instancia por aplicación. Debe crearse antes de cualquier objeto relacionado con la GUI.

QWidget

La base de todos los objetos de interfaz de usuario. Los más comunes son `QMainWindow` y `QDialog` y sus subclases.

1^{er} Programa

¡Hola mundo!, el código

ejemplo

```
#include <QApplication>
#include <QLabel>

int main(int argc, char *argv[]) {
    QApplication app(argc, argv); // 1 Crear una app
        // y solo una,
    QLabel label("Hello Qt!"); // 2 Crear un widget

    label.show(); // 3 Mostramos el widget

    return app.exec(); // 4 Ejecutamos
}
```

Compilación

Procedimiento

Si vas a trabajar sin soporte de un IDE, como en este ejemplo para compilar debes hacer lo siguiente, desde la carpeta del proyecto.

1. `qmake -project # Genera proyecto`
   `# independiente de SO y compilador`
2. `qmake # Genera Makefiles`
3. `make # Compila`
4. `??? # como se llame tu programa`
5. `profit`

Contenido

Contenido

Extensiones a C++ I

Porque C++ no es suficiente complejo

Qt no es exactamente C++

Qt incluye una serie de funcionalidades que no están normalmente en C++. Para ello usa su propio preprocesador llamado MOC(Meta-Object Compiler)

Los Metaobjetos

Qt usa en gran parte de su API un tipo especial de objetos no nativo de C++, los metabjetos, los cuales se caracterizan por.

- *signals* y *slots*
- Propiedades,
- Arboles de objetos.
- Eventos
- *Timers:* Véase `Timers`

QObject

Uso

Para crear y usar un metaobjeto tenemos que hacer 3 cosas:

1. Heredar de QObject.
2. Incluir `Q_OBJECT` en la primera línea de la declaración.
3. Usar el Meta-object compiler, gestionado por qmake.

QObject II

Ejemplo

```
class MyDialog : public QDialog
{
    Q_OBJECT

public:
    MyDialog(QWidget *parent = 0); // Atentos a
    ~MyDialog();                   // este argumento opcional
private:
    QPushButton *button;
};
```

Signals y Slots I

Son un mecanismo de comunicación entre objetos que heredan de QObject. **Características:**

- Son *type-safe,* la firma de la señal debe encajar con la firma del slot
- Los slots son funciones que pueden ser llamados normalmente.
- Los slots privados, pueden ser invocados por una señal cualquiera independientemente del nivel de acceso.
- Las clases que emiten señales no necesitan preocuparse de que slot recibe la señal.

Signals y slots II

Declaración de Slots

Slots

Simplemente métodos en una clase que herede de QObject con la macro Q_OBJECT, declarados después de una de las siguientes líneas:

- `public slots:`
- `protected slots:`
- `private slots:`

Signals y slots III

Emisión de señales

signals

Se declaran como si fueran funciones, en una sección propia, pero no se implementan. El moc genera el código necesario.

uso

```
signals: // Declarar
exampleSignal(int emitValue);
```

```
// ejemplo de emitir
emit exampleSignal(4);
```

Signals y slots IV: Connect

La API de la conexión de un signal y un slot es:

```
connect(const QObject * sender, const char* signal,
        const QObject* receiver, const char* slot,
        Qt::ConnectionType type = QtAutoConnection)
```

Ejemplo

```
connect(button, SIGNAL(clicked()), qApp, SLOT(quit()));
// qApp es para acceder a la QApplicaction global
```

Las conexiones tienen más posibilidades, pueden ser automatizadas,canceladas, las señales bloqueadas y más, pero escapa al propósito de la charla.

Propiedades: Q_PROPERTY

Qt incluye un sofisticado sistema de propiedades, similar al que proporcionan algunos compiladores.

```
Q_PROPERTY(type name
           READ getFunction
           [WRITE setFunction]
           [RESET resetFunction]
           [DESIGNABLE bool]
           [SCRIPTABLE bool]
           [STORED bool]
           [USER bool])
```

Propiedades II

Quien es quien:

1. READ: una función de lectura, obligatoria.
2. WRITE: una función para modificar el valor de la propiedad, opcional.
3. RESET: función opcional, para que la propiedad vuelva al valor por defecto.
4. NOTIFY: opcional, emite una señal cuando el valor de la propiedad cambia.

Propiedades III

1. DESIGNABLE: indica si estará disponible en el diseñador, por defecto true.
2. SCRIPTABLE: indica si estará disponible para QtScript, por defecto true.
3. STORED: indica si la propiedad es de un valor por si mismo o dependiente de otros y si debe ser salvado cuando se guarda el estado del objeto, por defecto true.
4. USER: editable por el usuario.

Propiedades IV

Uso y justificación

Se puede acceder a una propiedad usando `QObject::setProperty(const char *, QVariant &)` y `QObject::property(const char *)` sin saber nada sobre la clase excepto el nombre de la propiedad. Es preferible usar directamente las funciones READ y WRITE, ya que son más rápidas y proporcionan mejores diagnósticos en tiempo de compilación. Pero requieren conocer la clase en tiempo de compilación, y los genéricos no.

ejemplo

```
QPushButton *button = new QPushButton;
QObject *object = button;
button->setDown(true);
object->setProperty("down", true);
```

El árbol de objetos

Utilidad

Los *QObjects* se organizan a si mismos en arboles de objetos. Cuando creas un `QObject` con otro como padre, es añadido a la lista de hijos(accesible con children()) y será borrado cuando el padre lo sea.

```
button = new QPushButton("Salir", this);
```

Eventos

Que hacer con ellos

Los eventos en Qt son objetos que derivan de la clase QEvent. A efectos prácticos típicamente para manejar eventos de forma personalizada reimplementamos funciones virtuales.

Contenido

QWidget

Hereda de QObject y QPaintDevice La base de todos los widgets, y componente atómico de la interfaz.
Un Widget sin un padre es una ventana independiente.
setWindowTitle() y setWindowIcon() establecen la barra de titulo y el icono respectivamente.
El constructor de todo Widget debe aceptar uno o 2 de los argumentos siguientes:

- QWidget *parent = 0: Si es 0 será una ventana principal.
- Qt::WindowFlags f = 0: establecer parámetros especiales para la ventana.

QWidget

- QWidget(e hijos) usa doble-buffer automáticamente
- Es posible crear ventanas con transparencias. (en X11: solo con Kwin o Compiz)
- Aparte de los estilos estándar podemos aplicar estilos usando CSS.
- Funcionalidades básicas: cursores, eventos, detección de pantalla completa, tooltips...

Diálogos I

QDialog e hijos

QDialog es la base de las ventanas de diálogos.
Disponemos básicamente de:

- QColorDialog
- QErrorMessage
- QFileDialog
- QFontDialog
- QInputDialog
- QMessageBox
- QPageSetupDialog
- QPrintDialog
- QPrintPreviewDialog
- QProgressDialog
- QWizard

Diálogos II

Descripción detallada

Un dialogo es una ventana de primer nivel siempre, usada para tareas y comunicaciones breves con el usuario.

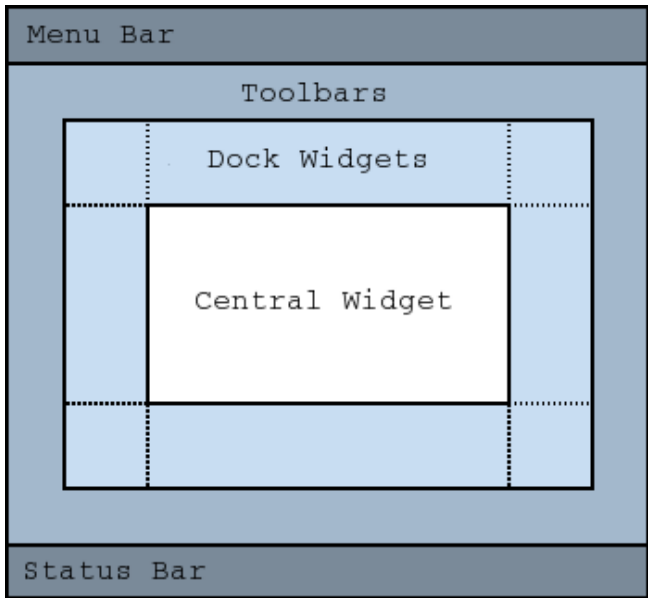
Posibilidades

- Diálogos modales: se aplica con exec()
- Diálogos no modales: usando el método show()
- Botones por defecto:
- Se cierran con ESC: ejecutando reject() en la salida
- Extensibilidad: véase *Extension example*

Qt MainWindow Framework

QMainWindow y amigos

Podemos entender que `QMainWindow` nos proporciona un marco de trabajo, para construir una GUI. QMainWindow tiene su propio layout al que se puede añadir `QToolBars`, `QDockWidgets`, un `QMenu`, y una `QStatusBar`.



QMainWindow

El widget central

El widget central

puede ser cualquier widged de Qt como `QTextEdit` o `QGraphicsView` o uno personalizado. Establecemos el widget central con setCentralWidget()
Una ventana principal puede tener una único(SDI) o múltiple(MDI) interfaz de documentos. Para hacer una aplicación MDI establece una QMDIArea como widget principal.

Podemos almacenar y restaurar el layout con:

- `saveState();`
- `restoreState();`

QMainWindow

Menús

Creación de menús

Los menús se implementan con QMenu y se mantienen organizados en una QMenubar. Cada entrada es una QAction. Añadimos menús con addMenu(QMenu), y acciones a los menús con con addAcction(QAction)

Ejemplo

```
void MainWindow::createMenus() {
    fileMenu = menuBar()->addMenu(tr("&File"));
    fileMenu->addAction(newAct);
    fileMenu->addAction(openAct);
    fileMenu->addAction(saveAct);
```

QMainWindow

Toolbars: barras de herramientas

Creación de toolbars

creamos una instancia de `QToolBar` y la añadimos con `addToolbar`, los elementos también son `QActions`.

Ejemplo

```
void MainWindow::createToolBars()
{
    fileToolBar = addToolBar(tr("File"));
    fileToolBar->addAction(newAct);
```

QString I

- Clase de manejo de texto Unicode. Almacena una cadena de `QChar`s, siendo cada QChar de 16 bits
- Usa compartición implícita de memoria, con copia en escritura.
- Dispone de funciones para convertir de y a números
- Es muy recomendable manejar las listas de QStrings usando QStringList

QString II

Dispone de 4 conversiones a const char *

- toAscii(): Autoexplicativo
- toLatin(): Convierte a una cadena de la codificación latin-1(ISO 8859-1
- toUtf8(): Convierte a UTF-8
- toLocal8Bit(): A la codificación local

QString III

Gracias al sistema de tipos de C++ y al hecho de que QString es implícitamente compartida, pueden ser tratadas como tipos básicos como int.

Inicialización

Bastante típica, diversas alternativas:

```
QString str1("Hello");
QString str2 = "Hello";
static const QChar data[4] = {0x0055, 0x006e,
0x10e3, 0x03a3};   QString str3(data, 4);
```

También carácter a carácter.

O en retornos de funciones.

```
return str1;
```

Otros widgets básicos

Botones

QCheckBox, QPushButton, QRadioButton, QToolButton y QCommandLinkButton.

Texto

QLineEdit, QPlainTextEdit, QTextEdit, QTextBrowser

Otros contenedores

QMDIArea, QTabWidget, QStackedWidget

Layouts

Basicamente se usan las siguientes posibilidades:

QHBoxLayout organiza los widgets en una fila horizontal

QVBoxLayout alínea los widgets en una columna vertical

QGridBoxLayout organiza los widgets en una rejilla, un widget puede ocupar varias casillas

QFormLayout Alternativa de alto nive a QGridBoxLayout para layouts de 2 columnas.

Contenido

Qt Designer I

Simplemente

Escribir código de interfaces tiene mucho de aburrido, repetitivo y rutinario. Es posible, en bastantes casos y recomendable usar un diseñador de interfaces. Aparte de eso las vemos tal cual las hacemos.

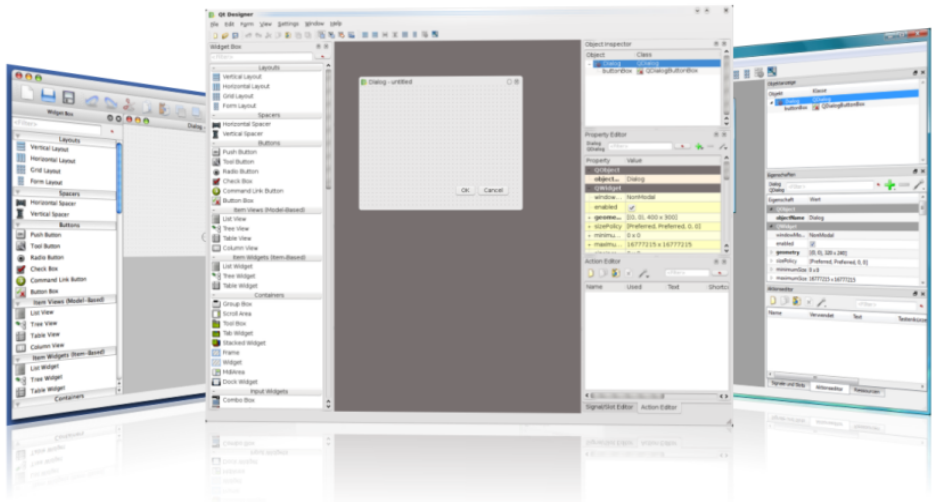
designer

Para, diseñar y crear interfaces de forma visual y cómoda con Qt disponemos de Qt Designer, bien *standalone*, bien integrado en Eclipse o Qt Creator

Qt Designer

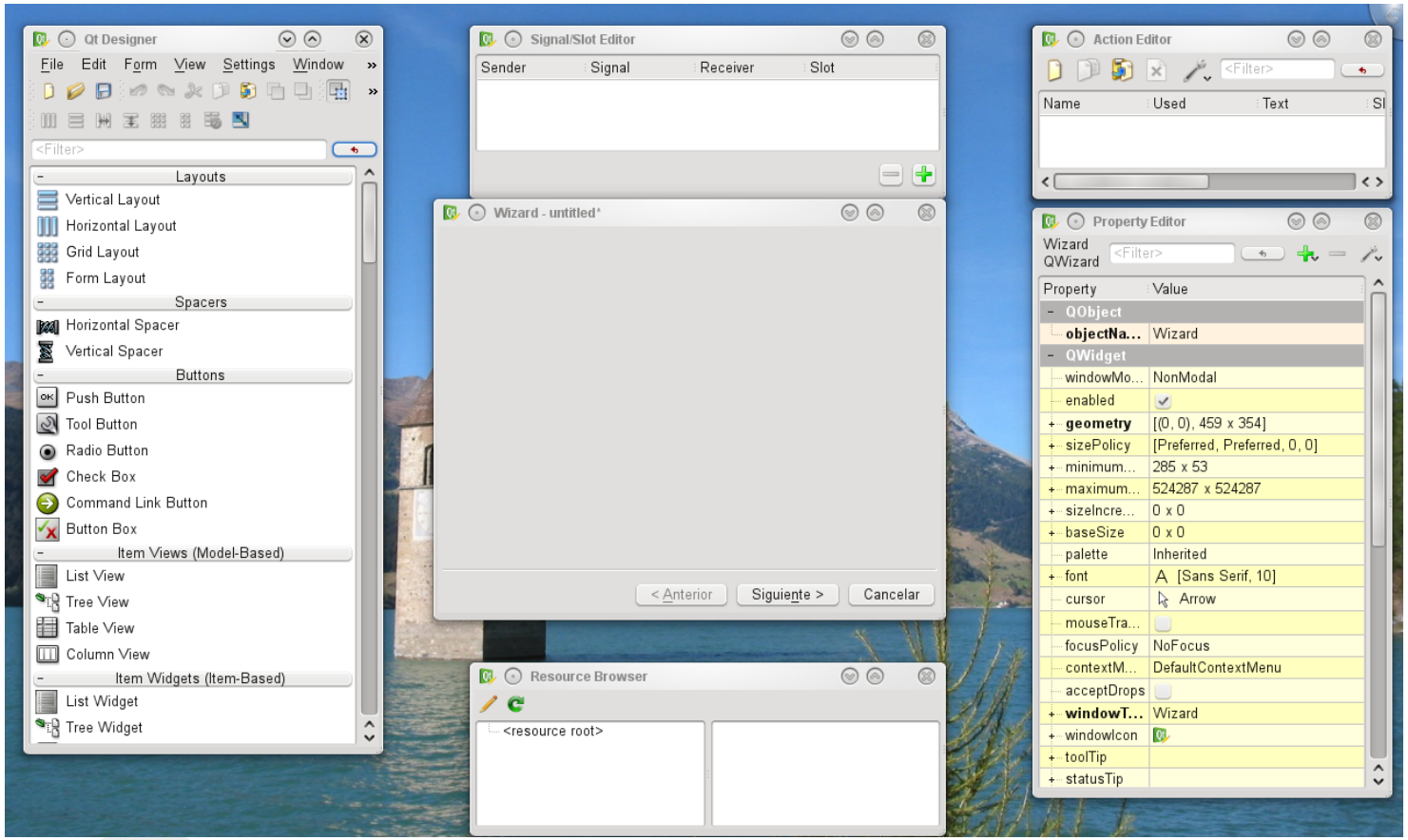
En imágenes I

La interfaz puede ser configurada para funcionar con una sola o con varias ventanas.



Qt Designer

En imágenes II



Qt Designer

La mejor manera de aprender a usar un diseñador de interfaces es usarlo.
En caso de duda consulta primero la documentación.

qmake I

Qmake es la herramienta de generación de Makefiles oficial de Qt4.
Características:

- Optimizado para Qt4
- Automáticamente genera reglas de compilación para moc y uic.
- Genera proyectos para el Visual Studio.

qmake II

Variables I

Modificamos valores de una variable con la sintaxis: `VARIABLE` $\pm =$ `valor` SOURCES Ficheros de código fuente del proyecto.
Algunas variables de interés:

- SOURCES: los ficheros de código fuente. SOURCES += file.cpp
- HEADERS: cabeceras del proyecto. HEADERS += file.h
- FORMS: Archivos de interfaces(.ui) FORMS += dialog.ui
- TARGET: nombre de la aplicación o librería; por defecto el nombre del proyecto
- CONFIG: opciones, merece un vistazo en profundidad

qmake III

- RESOURCES: una lista de ficheros de recursos a incluir
- TEMPLATE: la plantilla a usar para el proyecto disponemos de app, lib, subdirs(reglas para subdirectorios), vcapp, vclib.
- DEFINES: para incluir definiciones con el preprocesador
- QT: indica que componentes de Qt incluye el proyecto, por defecto core y gui

qmake IV

Los módulos de Qt

Opción	Modulo
core	QtCore
gui	QtGui
network	QtNetwork
opengl	QtOpenGL
sql	QtSql
svg	QtSvg
xml	QtXml
xmlpatterns	QtXmplPatterns
webkit	QtWebKit
qt3support	Qt3Support

qmake V
CONFIG

CONFIG es el cajón de sastre de las opciones.

Opción	Descripción
release	Crea el proyecto en modo release
debug	Crea el proyecto en modo debug
debug_and_release	Crea el proyecto en ambos modos
debug_and_release_target	ídem y en subdirectorios
build_all	compila todos los modos
warn_on	Habilita todos los warnings posibles
warn_off	Deshabilita bastantes warnings

qmake

Consejos y recordatorios

Comandos

- Crear proyecto con: `qmake -project`
- Generar Makefiles simplemente: `qmake`

Trucos

- Usa `QMAKE_LFLAGS += -s` en modo release para generar ejecutables más pequeños
- Puedes dar opciones dependientes de plataforma simplemente poniendo el nombre de la plataforma seguido de unas llaves y las opciones entre ellas
- Lee la documentación, hay para hacer 40 transparencias más, aparte de funcionalidades no documentadas.

Qt Assistant

La aplicación

Es un sencillo y potente visor de documentación que proporciona múltiples maneras de acceder a la información. De forma jerárquica en contenidos, búsquedas sobre el índice o todo el contenido y marcadores.
Permite añadir nuestra propia documentación, si esta en el formato qch(Qt compressed help).

¿Que información contiene?

Por defecto incluye la documentación de Qt, qmake, Qt Assistant, Qt Dessigner, Qt Linguist, y en ocasiones Qt Creator.

Contenido

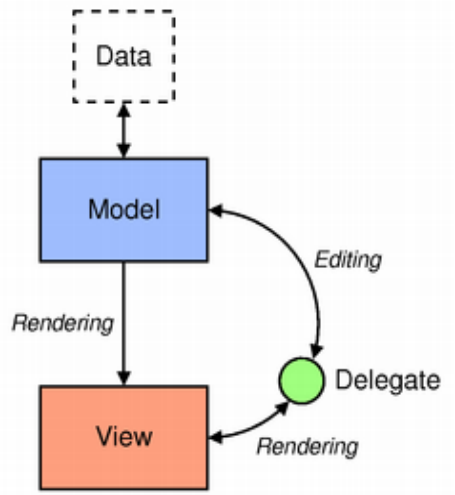
Contenido

Componentes del modelo-vista

models Proporcionan una interfaz estándar que vistas y delegados usan para acceder a los datos. Los datos en si no están almacenados en el modelo.

Views las vistas obtienen datos del modelo y los muestran al usuario.

Delegates Obtienen datos de los usuarios.



Las Graphics view

Las Graphics View proporcionan una superficie para gestionar e interactuar con un gran número de componentes 2D personalizados.

Como aproximación al Model-View

Es un modelo basado en componentes. Varias vistas pueden observar una misma escena y una escena puede contener items de variados shapes geométricos.

La escena: Proporcionada por QGraphicsScene

La vista: Proporcionada por QGraphicsView

El item: proporcionado por QGraphicsItem, y QGraphicsWidget(**No** hereda de QWidget)

Otros puntos de interes I

Dentro de QtGui

- Qt Style Sheets:[1] hojas de estilo con una sintaxis similar a la del CSS web.
- Drag and Drop: arrastrar y soltar
- La página *All Overviews and HOWTOs* puede ser un buen punto de inicio para buscar en caso de buscar información sobre temas de interés como la internacionalización, intercomunicación de procesos, teclados, plugins, integración con el escritorio...

[1] Consultar las páginas con estos nombres

Otros puntos de interes II

General

- QtConcurrent, primitivas orientadas a objetos para programación multihilo, así como apis de alto nivel.
- Otros cosas a tener en cuenta al desarrollar aplicaciones como manejo de eventos(más general), y recursos(qrc)

Fuentes

Recursos útiles si quieres profundizar

- Web de la documentación de Qt
- C++ Gui Programming with Qt4, 2^{nd} Edition
  Jasmin Blanchette y Mark Summerfield
- Edición vieja disponible para descarga.
- An introduction to design patterns in C++ with Qt 4
  Ezust, Alan, Upper Saddle River [etc.]

¿Preguntas o sugerencias?

Salvador Alemany Garrido
salalgar@fiv.upv.es