Tutorial de Qt4 Designer y QDevelop

David González Gutiérrez

CONTENIDO

1 Introducción

1.1 Descripción

Con el presente proyecto se pretende unificar y actualizar a la versión 4.0 de Qt, dos tutoriales existentes en línea, que son:

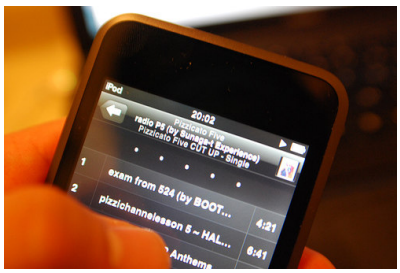
- Clive Cooper: How to Create a Linux Desktop App in 14 Minutes for Beginners.
- Jean Pierre Charalambos: Tutorial de Qt4 Designer.

El tutorial deberá estar disponible mediante soporte online, por lo que a modo de ampliación se ha optado por el desarrollo de una aplicación web. De este modo un usuario podrá interactuar con cada capítulo del tutorial, valorándolo y escribiendo comentarios que considere oportunos.

1.2 Motivación

La interfaz gráfica de usuario (desde ahora la llamaremos GUI) es aquello que permite a un usuario interactuar con un programa. Debido a que constituye la primera toma de contacto de un sistema por parte del usuario, es necesario que sea agradable y con gran facilidad de uso para que éste se lleve una primera impresión positiva del software.

La creación de GUIs siempre ha sido y sigue siendo un tema clave, que puede incluso llegar a tener un impacto mayor que otros factores a la hora de definir el éxito de un producto en el mercado. Interfaces innovadoras, interactivas y creativas, auguran un gran éxito para el producto, tal y como hemos podido comprobar con diferentes gadgets electrónicos, como por ejemplo la interfaz táctil de Ipod (Apple), o el *boom* de la web 2.0 en internet.



Ipod Touch (Interfaz táctil)

Buscar información sobre lo que realmente determina una buena interfaz puede ser algo muy complicado. Sin embargo, todos los desarrolladores están de acuerdo en que una interfaz sencilla, intuitiva y portable catapulta el éxito de un programa; sin embargo, una interfaz compleja y con un alto grado de aprendizaje influye muy negativamente sobre el software, pudiendo hacer incluso que fracase a pesar de ser enormemente eficiente y eficaz.

Qt es una tecnología en auge que nos proporciona un juego de herramientas y elementos gráficos para la creación de interfaces y aplicaciones multiplataforma. Actualmente cuenta con un gran éxito y una gran implantación en diferentes ámbitos, que van desde las aplicaciones de escritorio hasta los sistemas electrónicos industriales y empotrados. La mayoría de sitios de referencia y ayuda on-line sobre Qt se centran en foros de ayuda, donde, si bien podemos acceder para solucionar dudas puntuales, no podremos encontrar una guía detallada para introducirnos y empezar a crear aplicaciones "desde cero".

Asimismo, un tutorial on-line es una gran oportunidad para poner en práctica conocimientos del ámbito de las aplicaciones web y bases de datos adquiridos en diferentes asignaturas, mediante el uso de diversos lenguajes (como por ejemplo HTML, PHP y SQL); así como de diversas tecnologías aplicadas al mundo de la web, como es el caso de Ajax (javaScript asíncrono).

1.3 Objetivo

El objetivo principal de este proyecto es conseguir un tutorial unificado, actualizado y ampliado de Qt y QDevelop tomando como referencia dos tutoriales existentes, a fin de conseguir un tutorial que pueda servir de referencia para aquellos usuarios que se quieran introducir en el mundo de la creación de GUIs y aplicaciones mediante Qt.

Secundariamente y a modo de ampliación, se pretende realizar una aplicación web que contenga dicho tutorial, y que pueda constituir un punto de intercambio de conocimientos e ideas entre los diferentes usuarios que visiten y sigan el tutorial, con el objetivo de conseguir un *feedback* directo de los usuarios.

1.4 Organización de la memoria

La memoria estará organizada en diversos apartados. La primera parte de la memoria se centrará en un breve análisis de Qt, tratando aspectos generales como su historia, sus componentes y los IDEs que actualmente lo soportan.

Ello servirá como prefacio para la segunda parte de la memoria, el tutorial en sí, en el que introduciremos y explicaremos todos los conceptos que se tratarán en cada capítulo de manera detallada.

La tercera parte tratará, desde un punto de vista basado en la ingeniería del software aplicada a sistemas web, y abarcando fases como la especificación y el diseño, cómo los usuarios podrán interaccionar con el tutorial.

Finalmente, la cuarta parte de la memoria tratará aspectos técnicos relacionados con el desarrollo, instalación y configuración del sistema, así como aspectos de coste económico y de planificación.

Con la presente organización se pretende dar una versión unificada de todas las áreas que se pretenden abarcar con este proyecto, de un modo organizado y con un orden temporal lógico.

2 Análisis previo

2.1 El paquete Qt

2.1.1 Breve historia de Qt

El framework Qt vio la luz de forma pública por primera vez en el año 1995. Fue desarrollado por dos ingenieros noruegos, Haavard Nord (futuro CEO de Trolltech) y Eirik Chanble-Eng (futuro presidente de Trolltech), como respuesta a la necesidad de disponer de un GUI para una aplicación C++ multiplataforma orientado a objetos. Dentro de la palabra "Qt", la letra "Q" fue escogida debido a su belleza en el editor Emacs de Haavard, y la letra "t" se añadió para significar "toolkit" (de manera similar a Xt, XToolkit). La compañía se fundó en 1994 como Quasar Technologies, y fue evolucionando hasta convertirse en Trolltech.

Evolución del logotipo Qt a lo largo del tiempo

Trolltech empezó a ofrecer Qt con la licencia GPL a partir del año 2000, siendo ésta gratuita para el desarrollo de software libre, pero de pago para el desarrollo de software privativo (constituía una fuente de ingresos para Trolltech). Actualmente, y a raíz de la compra de Trolltech por parte de Nokia a principios del año 2009, se ha anunciado que Qt se ofrecerá con licencia LGPL. Con ello se pretende popularizar y extender Qt a base de permitir su uso en programas privativos, ya que no será necesario pagar ninguna licencia por su uso.

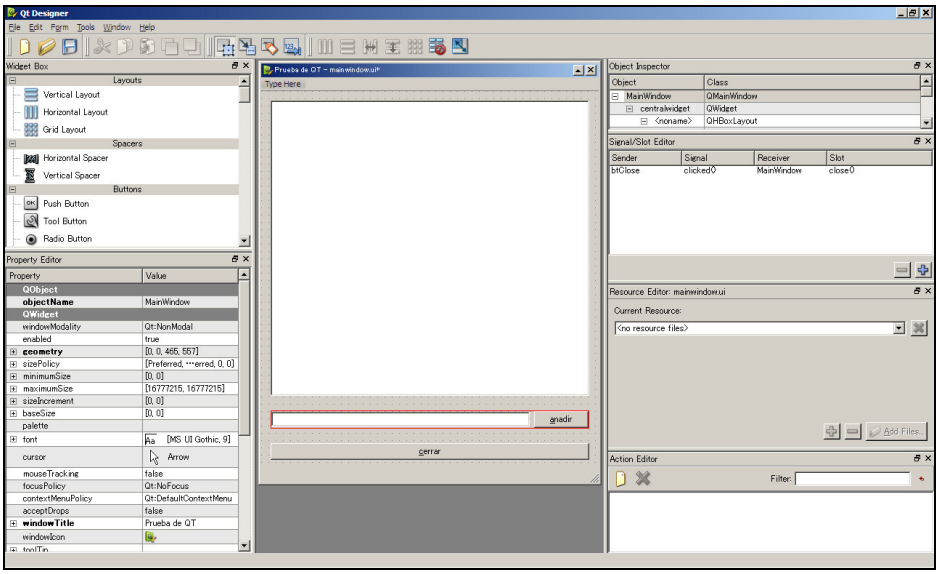
En la última década, Qt ha pasado de ser un producto usado por unos pocos desarrolladores especializados, a un producto usado por miles de desarrolladores open source en todo el mundo, por lo que el futuro de esta tecnología es hoy día muy prometedor.

2.1.2 Qt y sus componentes principales

Qt es un framework open source con licencia GPL concebido para el desarrollo de aplicaciones e interfaces multiplataforma. El paquete Qt integra herramientas de desarrollo y soporte tales como QtDesigner y QtAssistant, así como librerías de clases auxiliares a Qt. También incluye el compilador Gcc MinGW.

2.1.2.1 QtDesigner

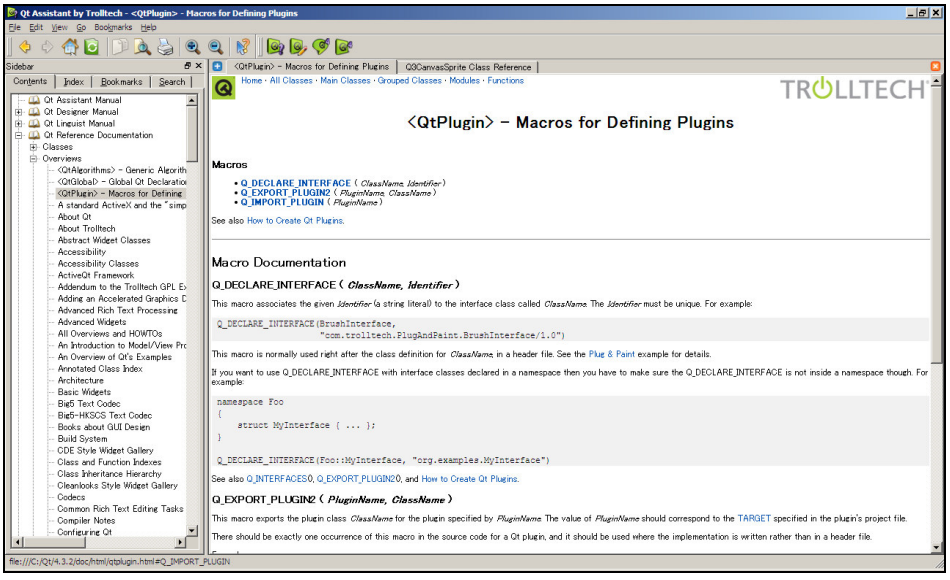
QtDesigner es una herramienta de desarrollo que nos permite crear interfaces graficas de usuario. Nos proporciona un conjunto de componentes estándar y un mecanismo de conexión llamado signal-slot, explicado en el capítulo 3 del tutorial, con el que conectaremos eventos de la interfaz con la lógica de programa que han de soportar.



Pantalla principal de QtDesigner para el diseño de GUI

2.1.2.2 QtAssistant

QtAssistant es un componente de Qt que nos permite navegar por la documentación, en forma de páginas HTML, e implementa opciones de búsqueda y de extensión.



Pantalla principal de QtAssistant

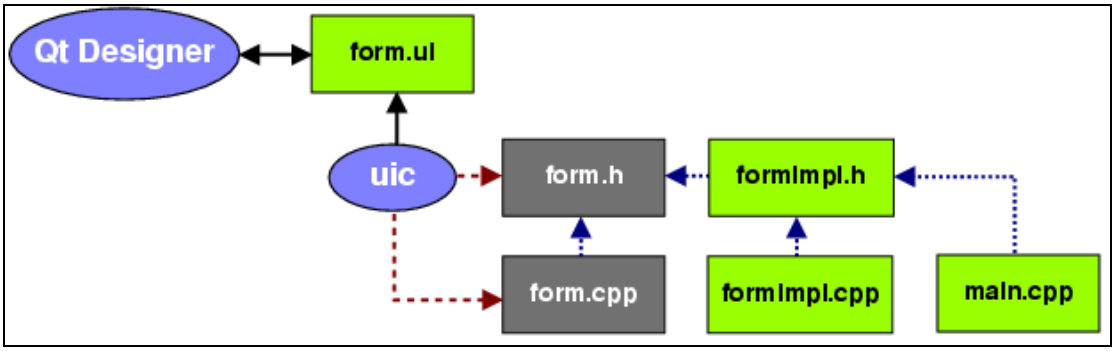
2.1.2.3 MinGW

MinGW es una implementación del compilador Gcc para la plataforma Windows, que además incluye un conjunto de la API de Win32, permitiendo un desarrollo de aplicaciones nativas para esa plataforma y pudiendo generar tanto ejecutables como librerías usando la API de Windows.

2.1.3 La evolución de QtDesigner

Qt ha superado actualmente su cuarta versión desde sus inicios. A continuación mostraremos brevemente cómo ha ido evolucionando el patrón de desarrollo de aplicaciones hasta la versión actual, y qué cambios lo han hecho posible.

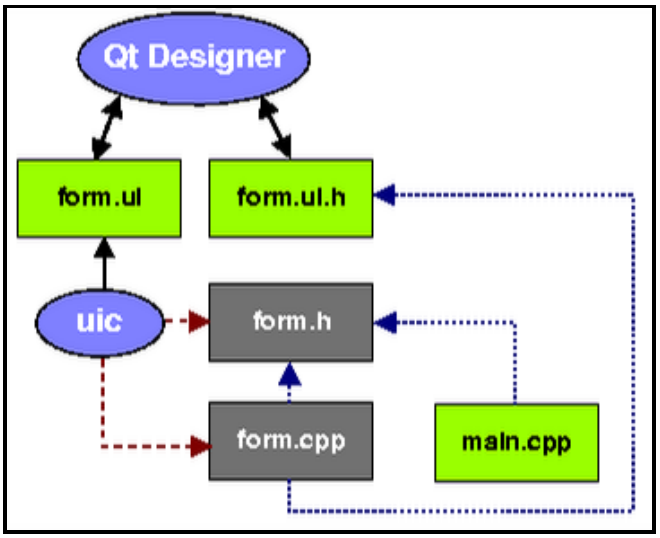
En la versión 1.0 y 2.2 de Qt, se creaban los diálogos (diseños de las ventanas) mediante QtDesigner, y éstos que se convertían a código mediante la utilidad UIC (User Interface Compiler). En esta versión ya se daba soporte al mecanismo estrella de Qt (signals y slots): el UIC generaba los slots (virtuales) y estos se implementaban con código mediante herencia.



Esquema de desarrollo en Qt 1.0

En la versión 3.0 de Qt, se incorporó el mecanismo del "archivo ui.h", que permitía implementar slots sin necesidad de herencia, y que era incluido automáticamente por el código fuente generado por UIC.

QtDesigner tenía un editor de código integrado para poder editar el archivo ui.h, y una funcionalidad (llamada *source*) para añadir variables e includes. También disponía de capacidad de carga de *plugins* para dar soporte a Custom Widgets (haciendo que fueran visibles en QtDesigner) e incluso un *parser* de archivos de proyecto .pro. Adicionalmente, se podían realizar conexiones a Bases de Datos y ver el contenido de sus tablas. Por ello, era posible construir una aplicación completa usando únicamente QtDesigner.



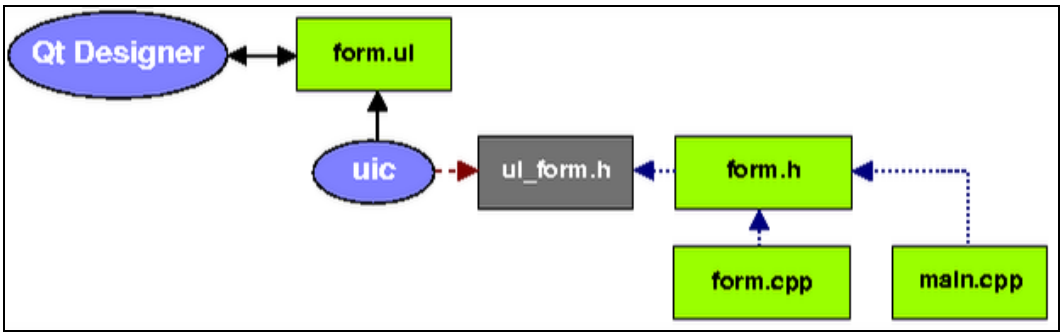
Esquema de desarrollo en Qt 3.0

Sin embargo, esta aproximación de edición "centralizada" mediante QtDesigner tenía una serie de carencias, y eran los siguientes:

- El editor de código de QtDesigner no disponía de funciones básicas (como por ejemplo, ir a una determinada línea).
- Existían problemas de sincronización al editar el archivo ui.h externamente a QtDesigner.
- No era posible su integración con IDEs existentes, por ello desarrolladores acostumbrados a otros IDEs debían utilizar a la fuerza QtDesigner para programar en Qt.

En la versión 4.0 de Qt se han corregido todas estas carencias. De hecho, el cambio más importante ha sido la posibilidad de integración de QtDesigner con IDEs existentes (por ejemplo, Eclipse) lo que derivó en la desaparición del editor de texto básico del que hacía gala QtDesigner en su versión 3.0.

Además, QtDesigner genera el archivo .ui con la interfaz, y el UIC genera el código para ésta (similar al comportamiento de las versiones 1.0 y 2.2, pero generando únicamente un archivo .h). Ahora, la herencia se realiza de la clase principal de la interfaz y de la interfaz de objetos de usuario (clases QObject y QDialog), siendo todo este código generado automáticamente.



Esquema de desarrollo en Qt 4.0

2.1.4 Qt y sus clientes en la actualidad

Qt es una tecnología muy presente en la actualidad: muchísimas aplicaciones open source han apostado por Qt a la hora de desarrollar sus GUIs. Sin embargo, muchas compañías privadas también han decidido adoptar Qt en productos comerciales, especialmente por su actual licencia LGPL. A continuación, se citan algunas de sus aplicaciones más famosas:

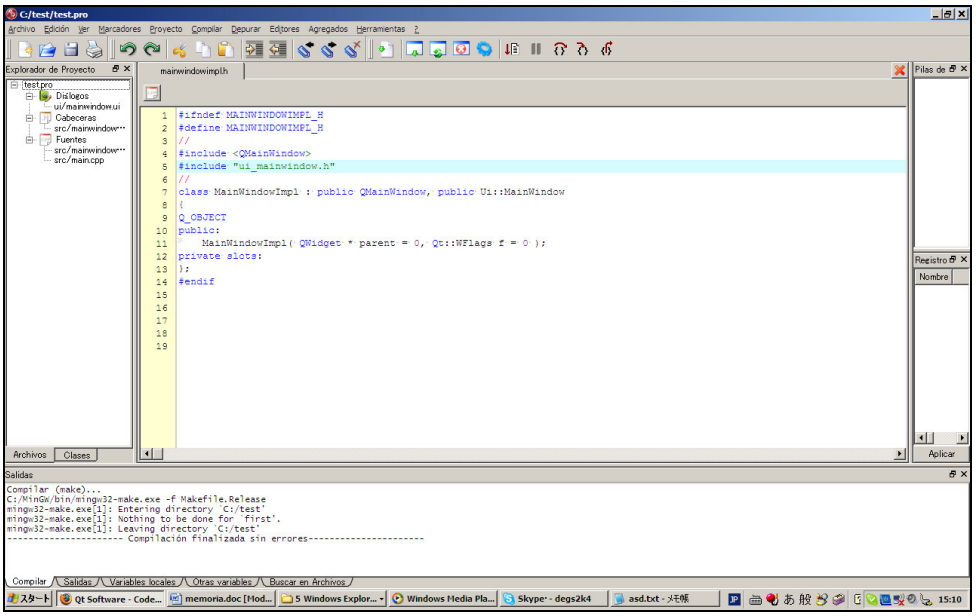
- Google Earth, VLC, etc.…: Muchas aplicaciones de escritorio se han desarrollado gracias a Qt.
- Teléfonos Skype: Los terminales Wi-fi Skype de Netgear han contado con Qt para sus interfaces de usuario.
- Volvo: El fabricante de automóviles ha optado por Qt para el GUI del ordenador de a bordo de sus coches.

Volvo, Netgear y Google han confiado en Qt para sus productos

2.2 IDEs con soporte a Qt

2.2.1 QDevelop

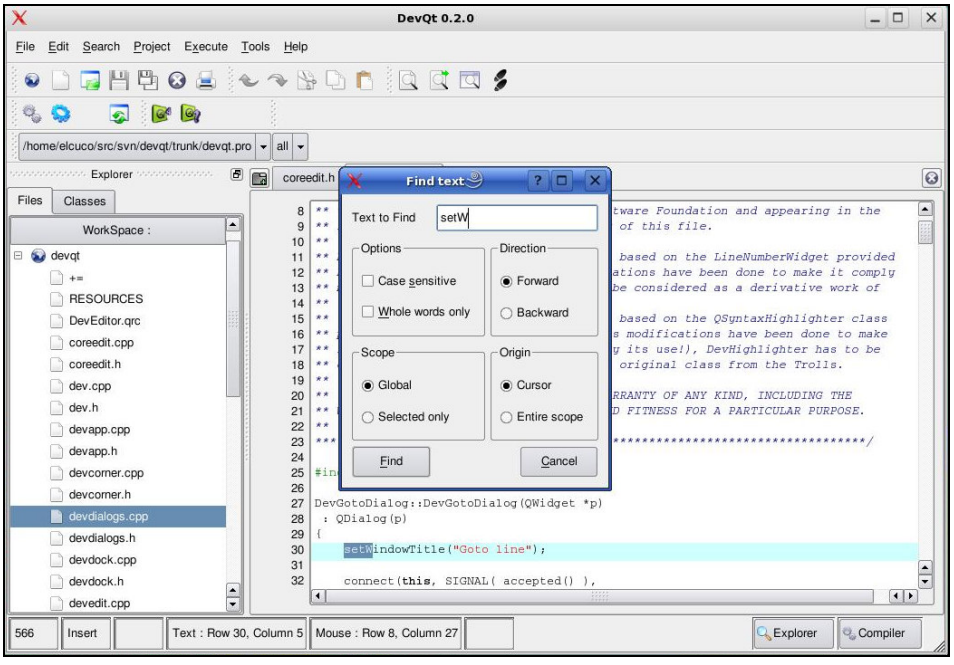
QDevelop es un entorno de desarrollo con licencia GPL desarrollado por Jean Luc Biord (un usuario de la comunidad Qtfr.org). Disponible para Windows y Linux, una de las características más interesantes de QDevelop es el hecho de admitir *plugins* externos, como por ejemplo el depurador GDB y la sugerencia automática vía CTags. QDevelop no integra Designer dentro de su interfaz, pero puede invocarlo externamente.



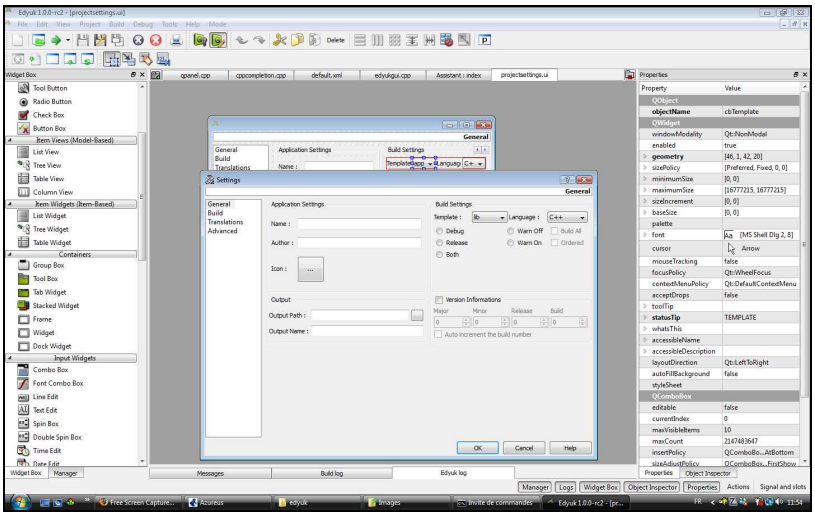
Pantalla principal de QDevelop

2.2.2 QDev / Edyuk

QDev es un editor nacido de la mano de un conocido usuario del foro de QtCentre (http://Qtcentre.org), una de las mayores comunidades online de Qt. El proyecto estuvo activo hasta Febrero de 2006, a partir de entonces el creador decidió continuar su desarrollo bajo el nombre de Edyuk. Este último tiene características muy atractivas, como por ejemplo capacidades de integración para Designer.



Pantalla principal de QDev



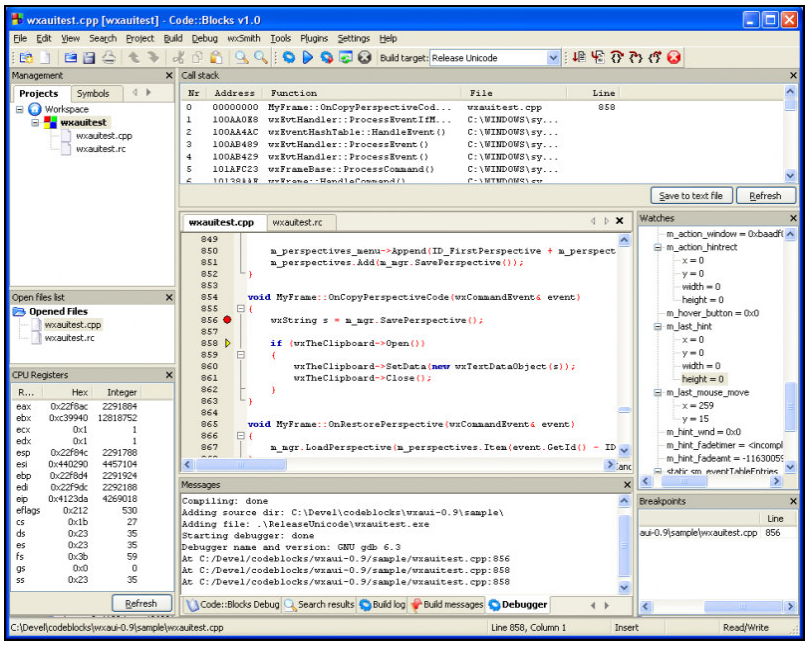
Designer integrado en Edyuk

2.2.3 Cobras

Cobras es un IDE para Qt simple y prematuro, capaz de interactuar con archivos .pro de qmake y activar las librerías resultantes. A pesar de ser exclusivo en Windows y todavía tener un explorador de clases no funcional, los usuarios coinciden en que se trata de un IDE muy estable.

2.2.4 Code::Blocks

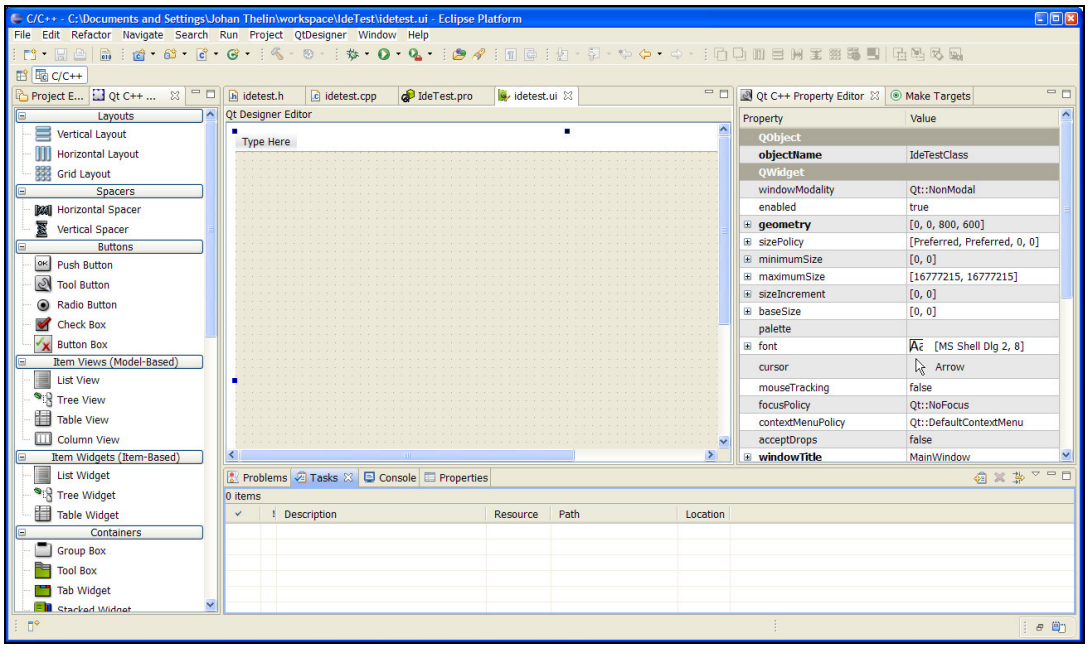
Code::Blocks (CB) es un IDE multiplataforma open-source para C++ desarrollado mediante WxWorks. Se puede integrar con Qt mediante un *plugin*.



Pantalla de depuración de CB

2.2.5 Eclipse

Eclipse es quizás uno de los IDEs más famosos y conocidos con soporte para múltiples plataformas y lenguajes, por ello Qt ha lanzado recientemente un módulo con el que es posible integrar Qt en Eclipse. Incluye opciones integradas de depuración, integración con Designer, editor de archivos de proyecto e incluso un editor de recursos. La integración de Qt con este IDE funciona muy bien, sin embargo el gran consumo de recursos necesarios para su correcto funcionamiento es su mayor punto negativo.



Pantalla de Designer en Eclipse

2.3 Herramientas externas: Ctags y GDB

Existen herramientas potencialmente útiles que se pueden manejar de forma centralizada desde un IDE en Qt: dos de las más conocidas son CTags y GDB.

La primera, Ctags, es una sencilla herramienta que nos permite generar un *tagfile*, un diccionario de todos los objetos encontrados en un código fuente para una rápida posterior localización de éstos mediante un editor textual (en nuestro caso QDevelop). Dentro de QDevelop, también nos permitirá activar funciones tan interesantes como el autocompletado de código.

La segunda, GDB, es la versión para Windows del depurador estándar de sistemas operativos GNU. Viene distribuido bajo licencia GPL, y funciona para varios lenguajes de programación tales como C, C++ y Fortran.

2.4 Tutoriales a unificar

En este apartado comentaremos brevemente los dos tutoriales que se pretenden unificar y actualizar en el presente proyecto.

2.4.1 Tutorial (Clive Cooper)

El primer tutorial que trataremos será el de Clive Cooper, titulado "*How To Create a Linux Desktop App In 14 Minutes For Beginners*". Se introducen aspectos básicos de Qt y se explica el uso conjunto de QDevelop y Qt con el objetivo de realizar una simple aplicación de prueba de Qt.

Página principal del tutorial de Clive Cooper

2.4.2 Tutorial (Jean Pierre)

El segundo tutorial que trataremos será el de Pierre Charalambos. Dentro de este tutorial se presentan aspectos más avanzados de Qt y se utiliza el editor gráfico de QtDevelop y un Custom Widget para crear una aplicación, que nos permite dibujar formas y texto en pantalla.

KDE - The K Desktop Environment

Tutorial para Qt-Designer

Siguiente

Tutorial para Qt-Designer

Jean Pierre Charalambos

Revisión 1.00.00 (2004-30-06)

El objetivo de este taller es el de introducir el uso de la herramienta *designer* para hacer el desarrollo de una aplicacion de "modo visual". La aplicacion que vamos a desarrollar en las proximas casi tres horas es la Canvas Demo. Se trata de una aplicacion que pinta objetos en un widget tipo canvas.

Tabla de contenidos

Página principal del tutorial de Jean Pierre

3 Contenido del tutorial

3.1 Capítulo 0: Instalación y configuración

3.1.1 Instalación y configuración de Qt y MinGW

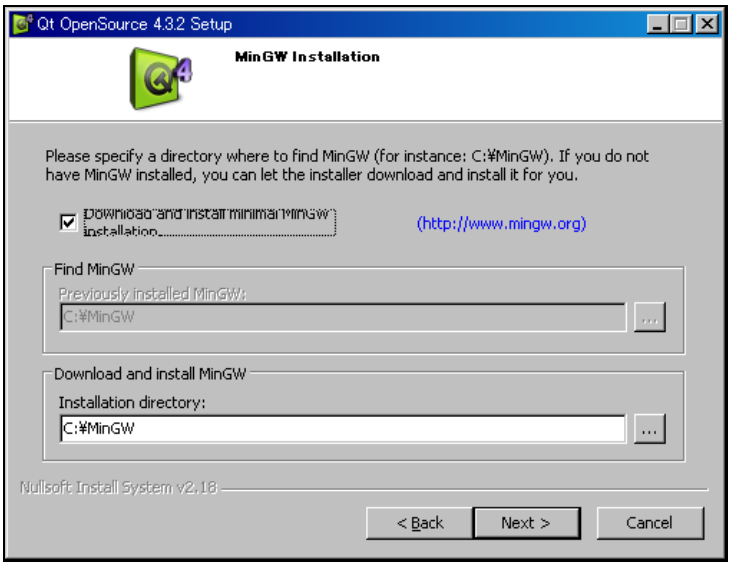
En este manual instalaremos la versión 4.3.2 de Qt[1], disponible tanto desde la sección de descargas para plataformas Windows de la página web oficial como desde el ftp de Trolltech:
(HTTP) http://trolltech.com/developer/downloads/Qt/windows
(FTP) ftp://ftp.trolltech.com/Qt/source/

Página web official de Trolltech

Hay dos versiones disponibles para descargar, la versión que incluye el compilador Gcc MinGW (la instalación de MinGW se llevara a cabo conjuntamente con la instalación de Qt4) y la versión sin el compilador (tendremos que descargar externamente MinGW). En este manual, para una mayor simplicidad, hemos elegido la versión con MinGW. Una vez descargado Qt4, simplemente haremos doble clic sobre el archivo para instalarlo.

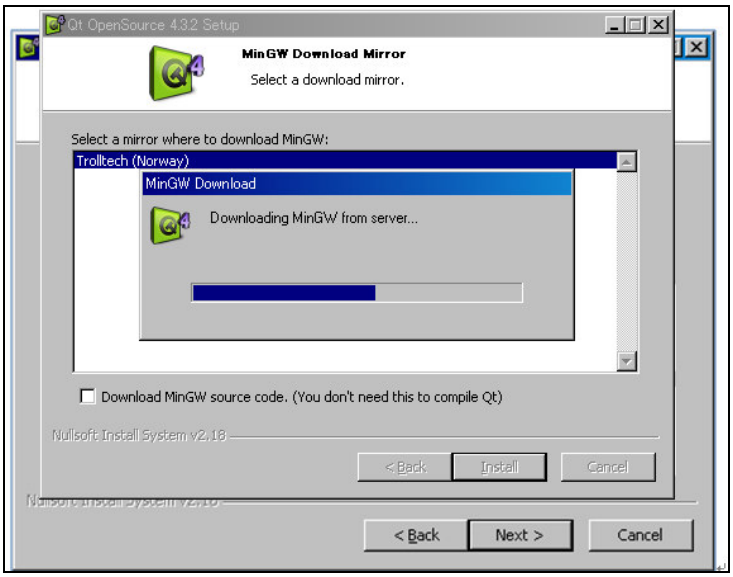
Antes de empezar la instalación de Qt4, podremos elegir dos opciones para instalar MinGW: o bien usar la ruta hacia MinGW en caso de tenerlo ya instalado, o bien descargarlo e instalarlo.

[1] Nota: En lo que a versiones de Qt se refiere, la versión 4.3.3 tenía un bug que generaba el error (ui_*.h" file not generated"). Este bug ya está arreglado y disponible en la versión 4.3.4 como patch pero solo para usuarios comerciales, por lo que nosotros hemos usado la versión 4.3.2 en este manual.



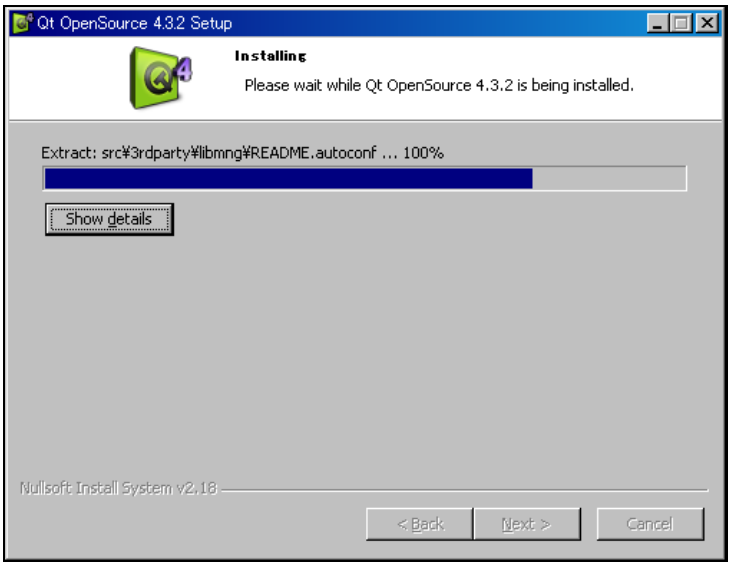
Instalacion de Qt: MinGW

La instalación de MinGW se iniciará antes de la de Qt, teniendo que aceptar los acuerdos de licencia y seleccionar un mirror para la descarga:



Instalacion de Qt: Descarga de MinGW

A continuación, se iniciará la instalación de Qt4:



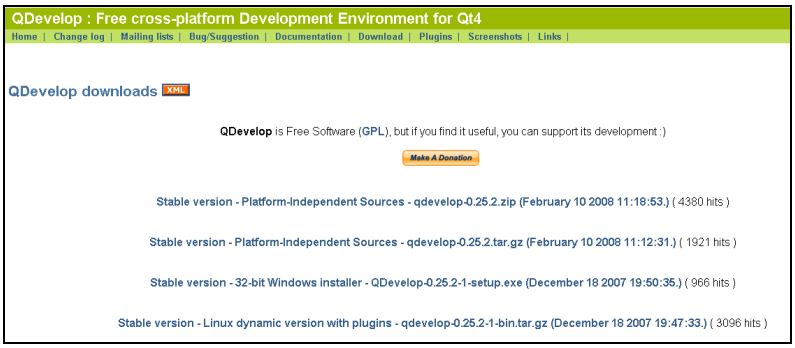
Instalacion de Qt

Al concluir la instalación de Qt4, simplemente tenemos que añadir a la variable de entorno PATH de Windows las rutas correspondientes a las librerías de Qt y ejecutables de Qt y MinGW (suponiendo que hayamos realizado la instalación de MinGW y Qt en x:\MinGW y en X: \Qt, respectivamente):

- X:\MinGW\bin;
- X:\Qt\4.3.2\bin;
- X:\Qt\4.3.2\lib;

3.1.2 Instalación y configuración de QDevelop

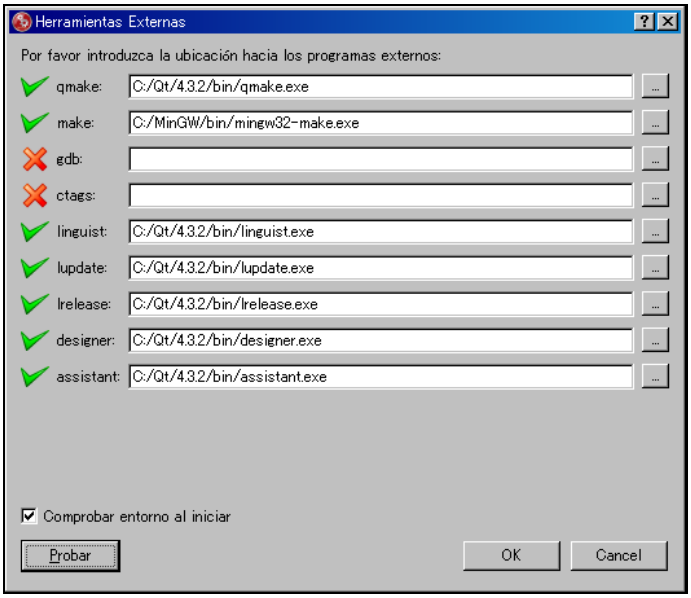
En este manual instalaremos la versión 0.25.2 de QDevelop (para sistemas Windows de 32 bits), disponible desde la sección de descargas de su página web oficial (http://Qdevelop.free.fr/)



Página web oficial de QDevelop

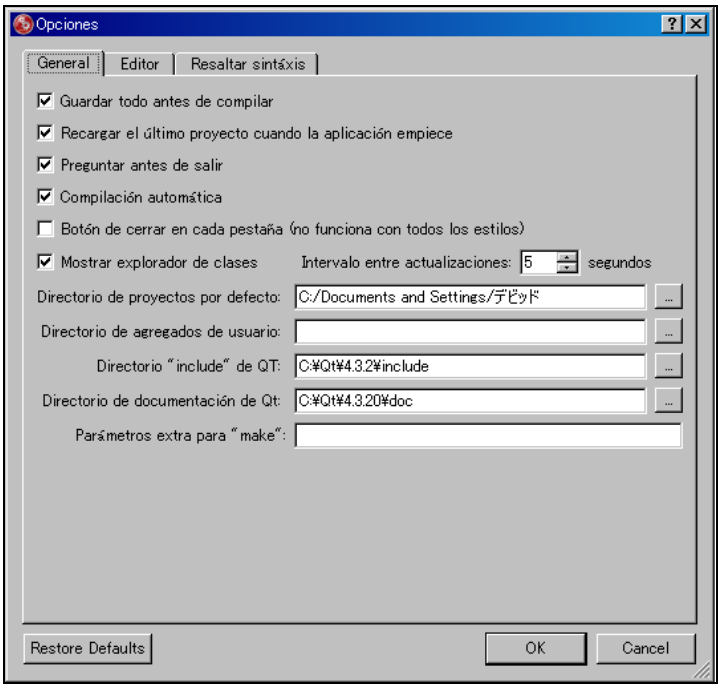
Una vez descargado, haremos doble clic sobre el ejecutable para instalarlo, y una vez concluida la instalación, iniciaremos QDevelop. Al hacerlo, se nos mostrará en primer lugar una ventana de configuración de herramientas externas (Qt, MinGW, CTags y GDB), en la que deberemos indicar la ruta hacia cada uno de los archivos binarios solicitados. Hasta ahora solo hemos instalado Qt y MinGW, por lo que de momento únicamente rellenaremos los paths correspondientes a estos.

Bastará con buscar dichos paths y pulsar el botón “Probar” para que QDevelop valide las rutas, poniendo tics verdes en las rutas correctas y cruces rojas en las incorrectas, tal y como muestra la siguiente imagen:



Ventana de herramientas externas de QDevelop

Finalmente será necesario comprobar que los directorios de includes y documentación de Qt están correctamente indicados en QDevelop, mediante el menú Herramientas > Opciones:



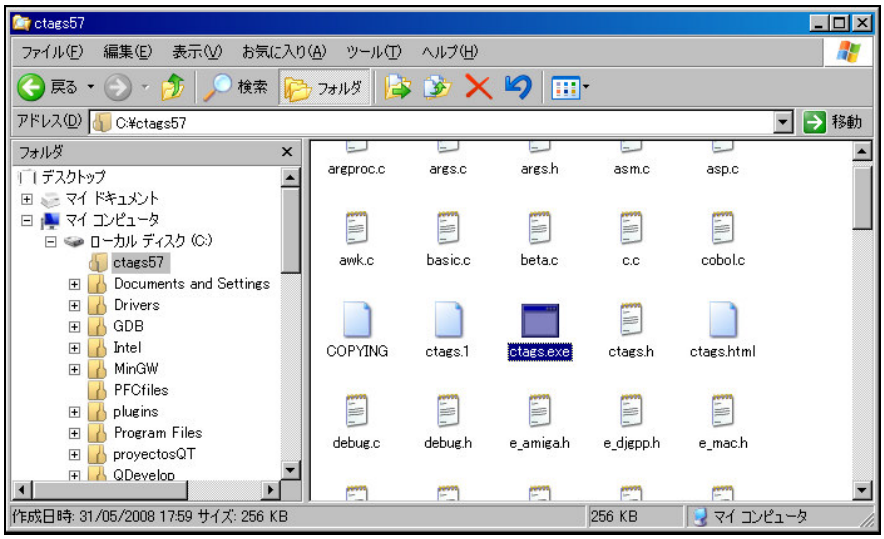
Ventana de opciones de herramientas de QDevelop

3.1.3 Instalación y configuración de CTAGS

CTags es un programa que genera un índice (conocido como *tagfile*) de objetos encontrados en archivos de código fuente para que éstos puedan ser accesibles rápida y fácilmente mediante un editor de texto o cualquier otra utilidad. CTags soporta alrededor de 25 lenguajes de programación, y en el caso de C/C++ en particular, permite generar tags de:

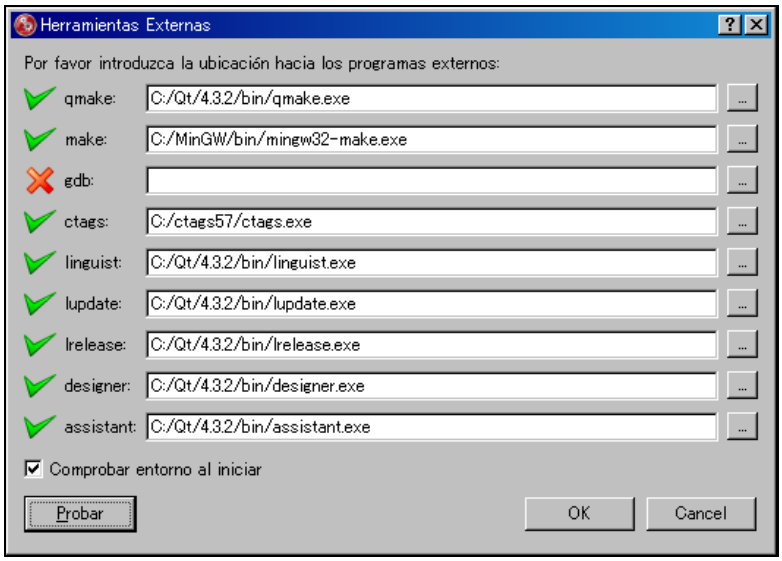
- Clases, interfaces, estructuras y otros tipos de datos.
- Definiciones de macros, funciones, variables y tipos.
- Enumeraciones.
- Prototipos/declaraciones de funciones.

En primer lugar descargaremos el paquete de CTags desde el siguiente enlace:
(Link directo: http://prdownloads.sourceforge.net/CTags/ec554w32.zip?download)
Al descargar CTags y descomprimirlo, veremos que contiene una serie de archivos fuentes y un ejecutable, CTags.exe. Lo único que nos va a interesar es precisamente dicho ejecutable:



Ejecutable ctags.exe

A continuación volveremos a configurar QDevelop para poder usar CTags: dentro de la opción Herramientas > Herramientas Externas indicaremos la ruta al ejecutable de CTags:

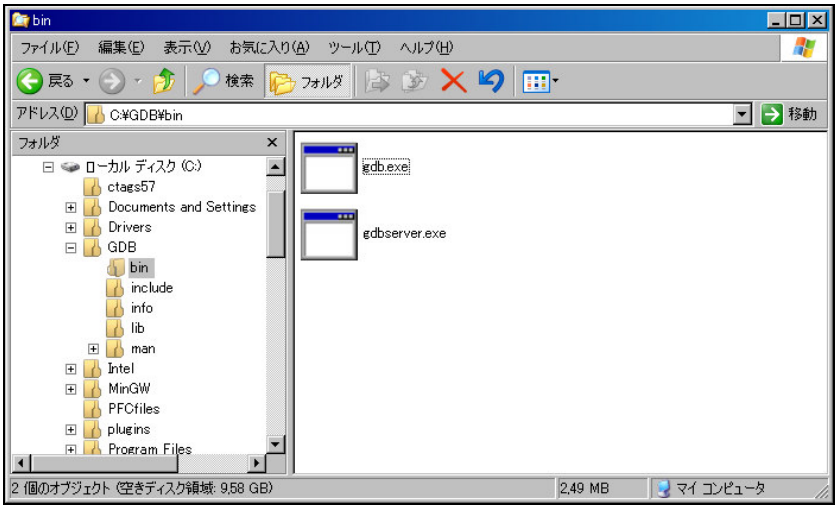


Ventana de configuración de herramientas externas de QDevelop

En estos momentos QDevelop ya podrá acceder a CTags y generarnos un tag file con toda la información de las clases de Qt4 [2].

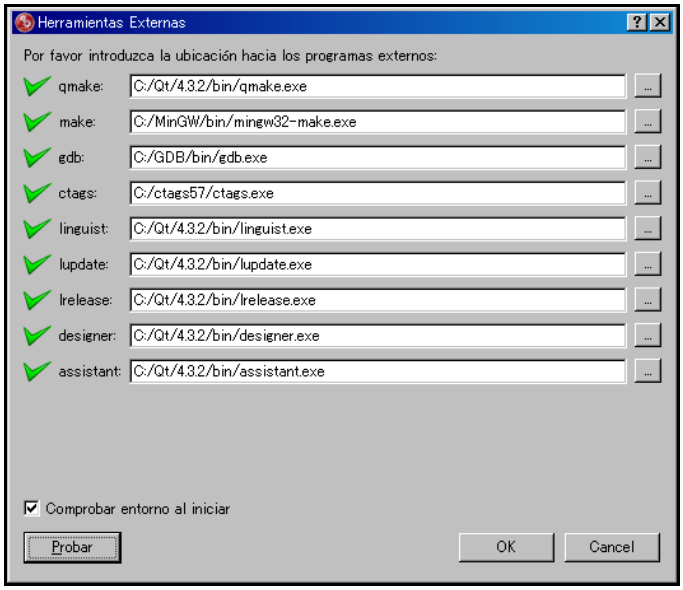
[2] La versión actual (0.25) de QDevelop presenta un *bug* que impide el correcto uso con CTags. Este bug ha sido corregido en la versión 0.26 de QDevelop, disponible en http://code.google.com/p/QDevelop. Instalación y configuración de GDB

La instalación del depurador GDB sigue los mismos pasos que la de CTags. En primer lugar nos bajaremos el paquete de GDB para Mingw y lo descomprimiremos en nuestro disco: (http://downloads.sourceforge.net/mingw/gdb-6.8-mingw-3.tar.bz2?modtime=1208982015&big_mirror=1)



Archivos pertenecientes al depurador GDB

Finalmente, y tal y como hicimos con CTags, configuraremos de nuevo las herramientas externas de QDevelop, indicando la ruta hasta el ejecutable gdb.exe.

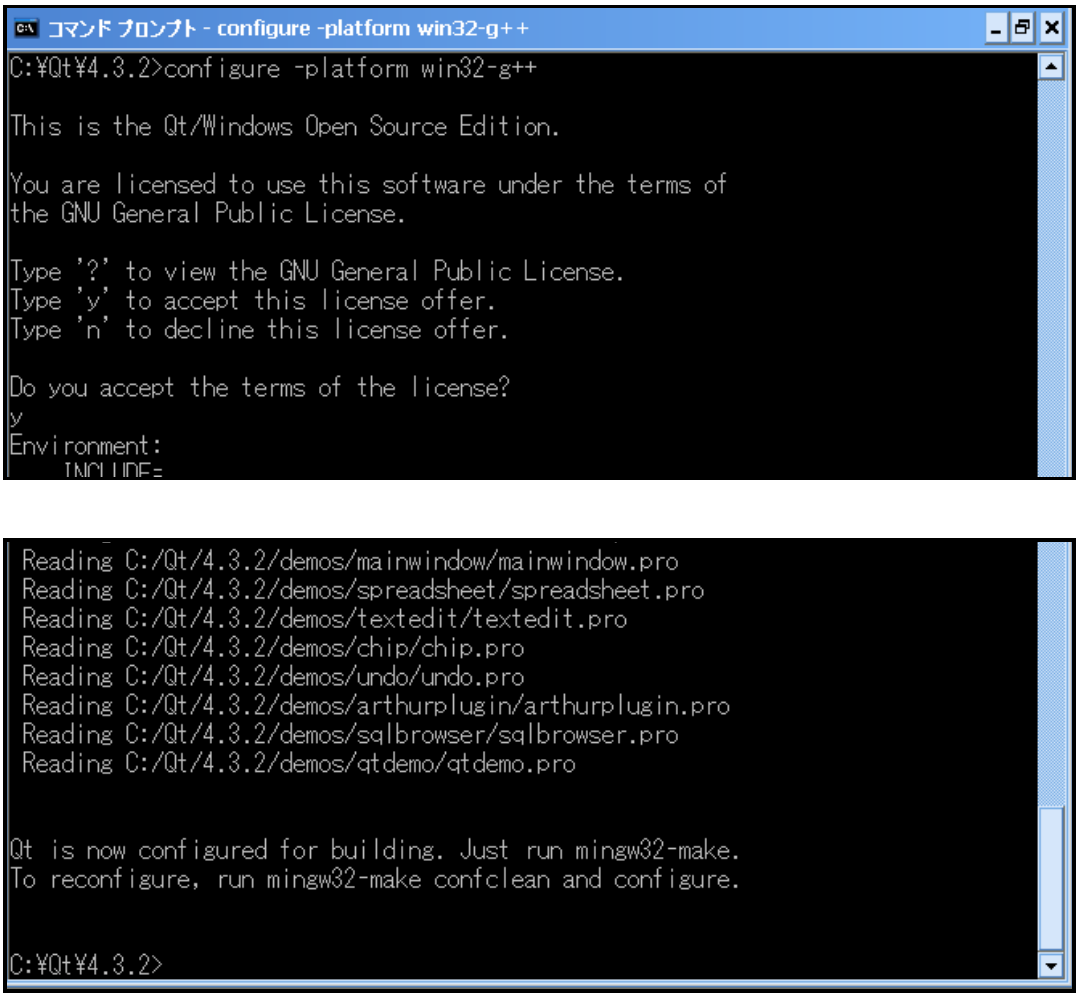


Ventana de configuración de herramientas externas de QDevelop

3.1.4 Configuración post-instalación

En este punto ya tenemos configuradas todas nuestras herramientas externas. Pero para poder empezar a construir y compilar proyectos con Qt, será necesario configurar el compilador de Qt para nuestro sistema. Para ello bastará con ejecutar los dos siguientes comandos desde el directorio raíz de Qt (Qt/4.3.2):

- configure –platform win32–g++



Configuración de la plataforma para Qt

En este comando hemos usado la plataforma win32-g++ debido a que la opción win32-msvc (MS Visual Studio) no está soportada por la versión Open Source de Qt. Podemos consultar la lista de plataformas disponibles dentro del archivo readme.txt del directorio de MinGW.

Si se intenta ejecutar el anterior comando con configure -platform win32-msvc, se acabaran aceptando los acuerdos de licencia de la instalación pero el programa de configuración finalizará directamente sin dar ninguna explicación de lo sucedido.

Seguidamente ejecutaremos el siguiente comando para crear el make de MinGW:

- mingw32-make

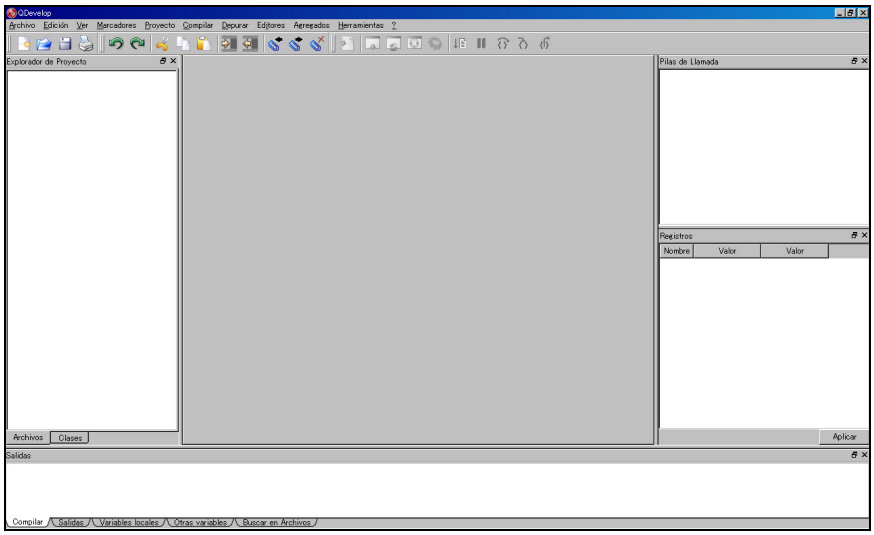
```
C:¥Qt¥4.3.2>mingw32-make
cd src¥winmain¥ && mingw32-make -f Makefile
mingw32-make[1]: Entering directory `C:/Qt/4.3.2/src/winmain'
mingw32-make -f Makefile.Debug all
mingw32-make[2]: Entering directory `C:/Qt/4.3.2/src/winmain'
g++ -c -g -Wall -frtti -fexceptions -mthreads -DQT_THREAD_SUPPORT -DUNICODE -DQT
_LARGEFILE_SUPPORT -DQT_NEEDS_QMAIN -DQT_NO_CAST_TO_ASCII -DQT_ASCII_CAST_WARNIN
GS -DQT_44_API_QSQLQUERY_FINISH -DQT3_SUPPORT -DQT_MOC_COMPAT -D_USE_MATH_DEFINE
S -I"..¥..¥include" -I"tmp" -I"..¥..¥include¥QtCore" -I"c:¥Qt¥4.3.2¥include¥qtma
in" -I"tmp" -I"c:¥Qt¥4.3.2¥include¥ActiveQt" -I"tmp¥moc¥debug_shared" -I"." -I".
.¥..¥mkspecs¥win32-g++" -o tmp¥obj¥debug_shared¥qtmain_win.o qtmain_win.cpp
```

Configuración del make de minGW

Hay que tener en cuenta que el tiempo de proceso de este segundo comando es superior a 2 horas (probado sobre un Intel Centrino 1.70GHZ) y son necesarios más de 2GB de espacio libre en el disco duro para los archivos generados. Con este paso ya habremos configurado el make de MinGW y ya podremos compilar nuestros proyectos.

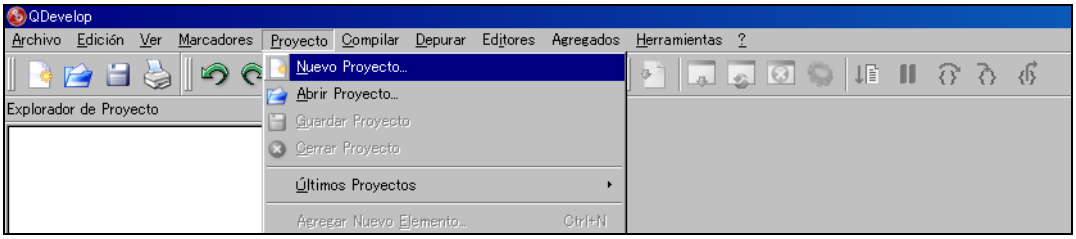
3.2 Capítulo 1: Creación de la aplicación inicial (I)

En primer lugar, y a modo de práctica, vamos a crear una pequeña aplicación muy sencilla con la que podremos entender y empezar a manejar Qt. Iniciamos QDevelop:



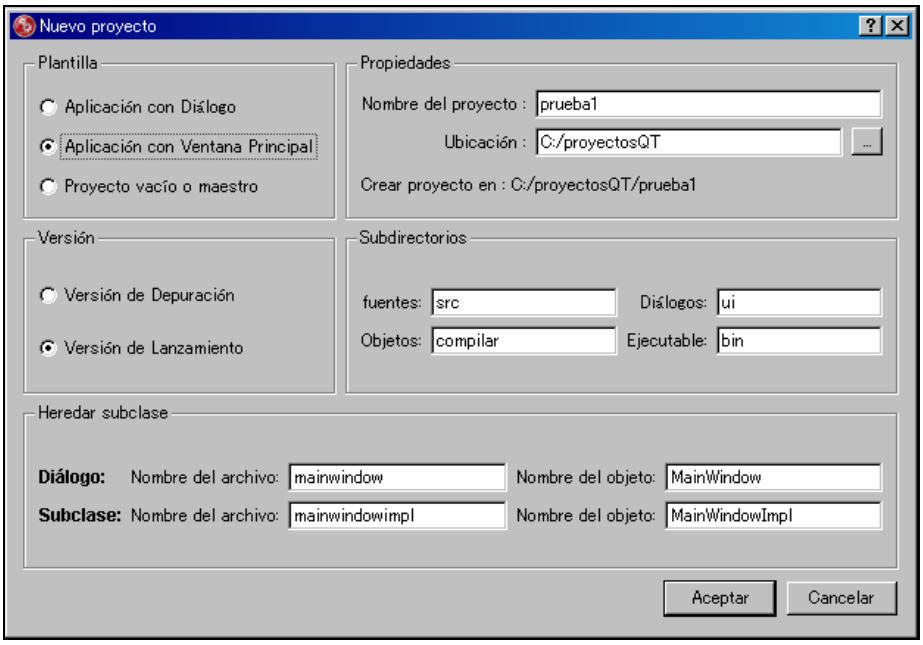
Ventana inicial de QDevelop

A continuación creamos un nuevo proyecto mediante la opción de menú Proyecto > Nuevo proyecto:



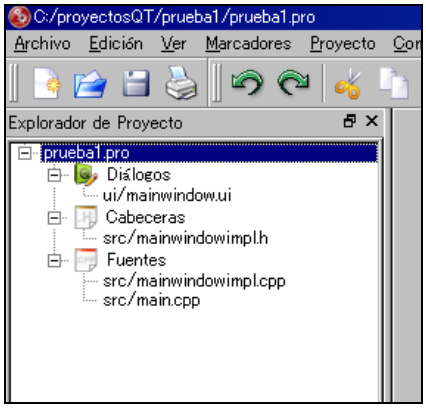
Creación de un nuevo proyecto

Seguidamente se nos aparecerá una ventana desde la que básicamente seleccionaremos el tipo de aplicación a crear (en esta primera aplicación será “Aplicación con ventana principal”) y el nombre del proyecto (“prueba1”) tal y como muestra la siguiente imagen, y pulsamos sobre el botón de aceptar:



Ventana de nuevo proyecto en QDevelop

A continuación volvemos a la pantalla principal de QDevelop, en la que podremos observar, a mano derecha en el Explorador del Proyecto, el árbol de archivos que corresponde a nuestro proyecto:

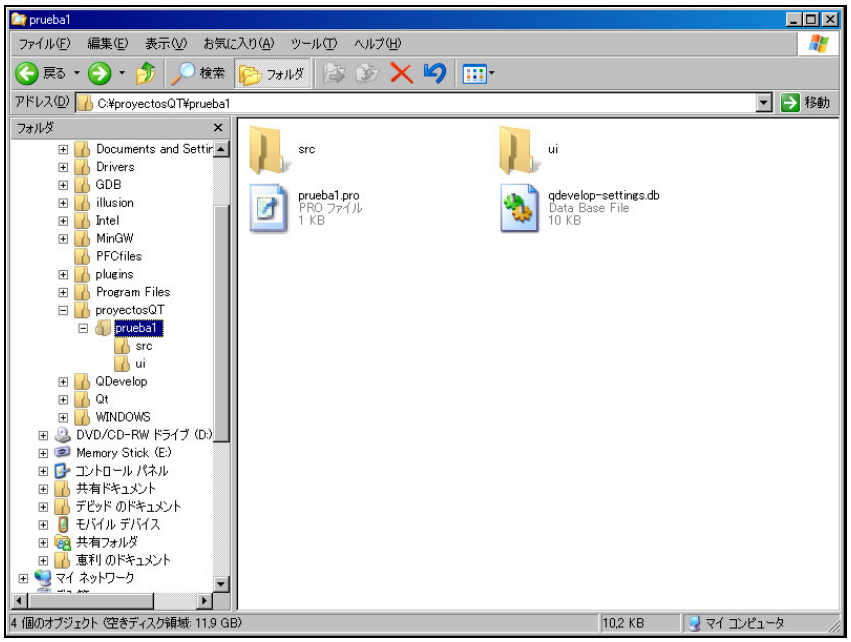


Jerarquía de archivos en QDevelop

Este proyecto, generado automáticamente, se compone de 3 grupos de archivos:

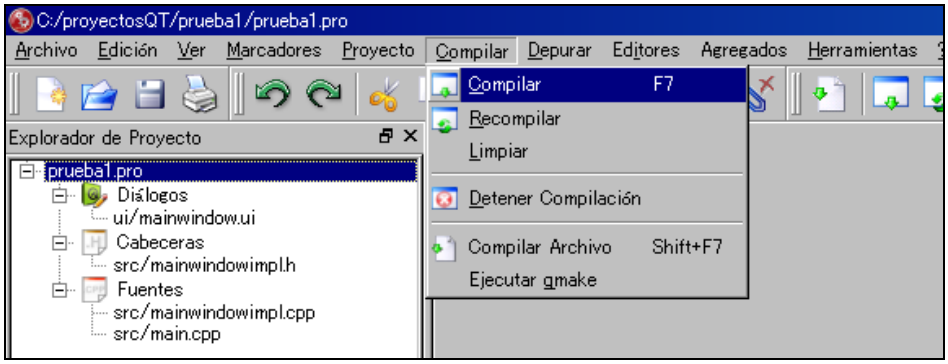
- Dialogo de la aplicación: viene a ser como la interfaz o “ventana” de la aplicación, que es lo que realmente verá el usuario al ejecutar la aplicación.
- Cabeceras: Ficheros de la aplicación que contienen las declaraciones del proyecto.
- Fuentes: Ficheros que contienen el código fuente del proyecto.

Físicamente, en la carpeta del proyecto, nos encontramos con lo siguiente: en la raíz del proyecto figuran los archivos de configuración de este, y los directorios *src* y *ui* que contienen, respectivamente, el código fuente y el archivo de interfaz de la aplicación.



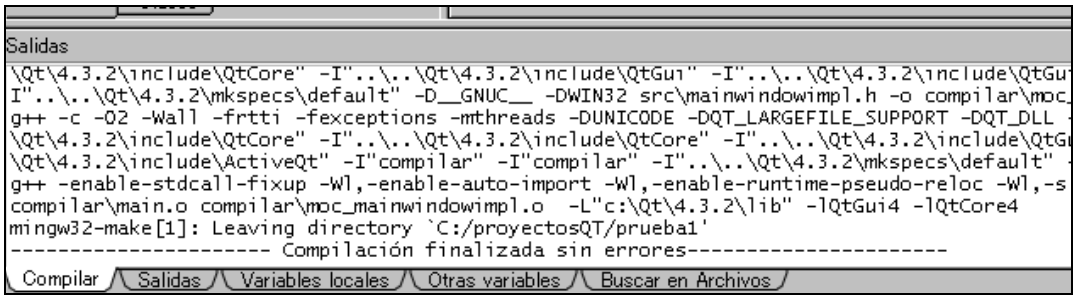
Organización del directorio de trabajo en QDevelop

Una vez creado el proyecto y tras haber comprobado la estructura de éste, vamos a intentar compilarlo y ejecutarlo. Para compilarlo bastará con seleccionar la opción Compilar > Compilar del menú (o bien pulsando F7):



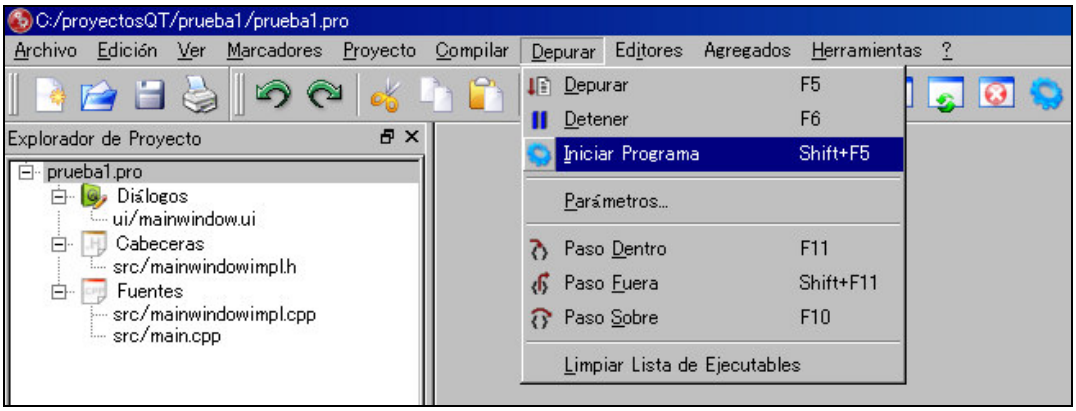
Opción de compilar en el menú de QDevelop

Podremos ver el resultado de la compilación en la zona de “Salidas” que se encuentra en la parte inferior de la ventana de la aplicación. Si tenemos seleccionada la pestaña de Compilar, se nos mostrarán los resultados de la compilación.



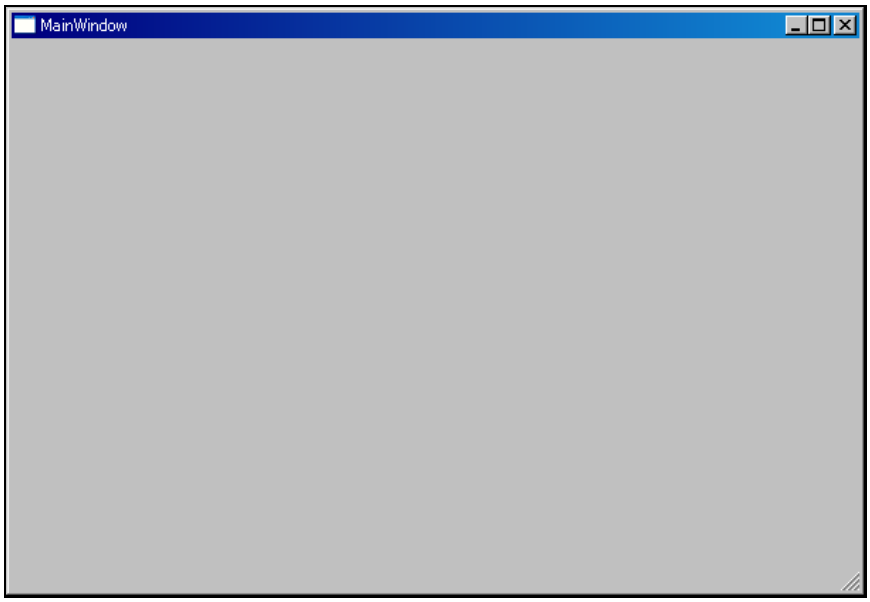
Pestaña compilar en la zona de salidas de QDevelop

Una vez compilado sin errores, bastará con la opción Depurar > Iniciar Programa (o bien pulsando F5) para ejecutarlo:



Opción de ejecutar programa en QDevelop

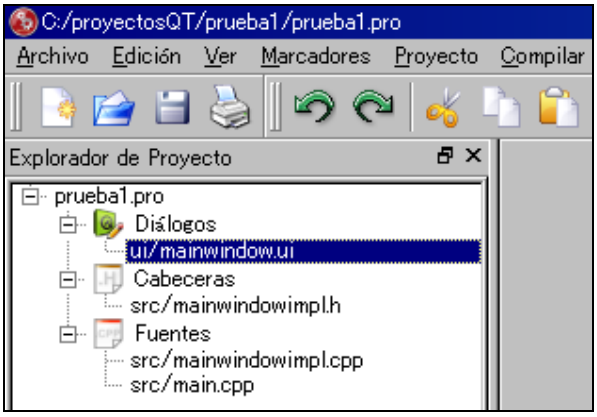
Si hemos seguido todos los pasos correctamente, el resultado será una ventana Win32 vacía, en la que no hay implementada ninguna funcionalidad pero que será el punto inicial del proyecto:



Ventana inicial, por defecto, de nuestro primer proyecto

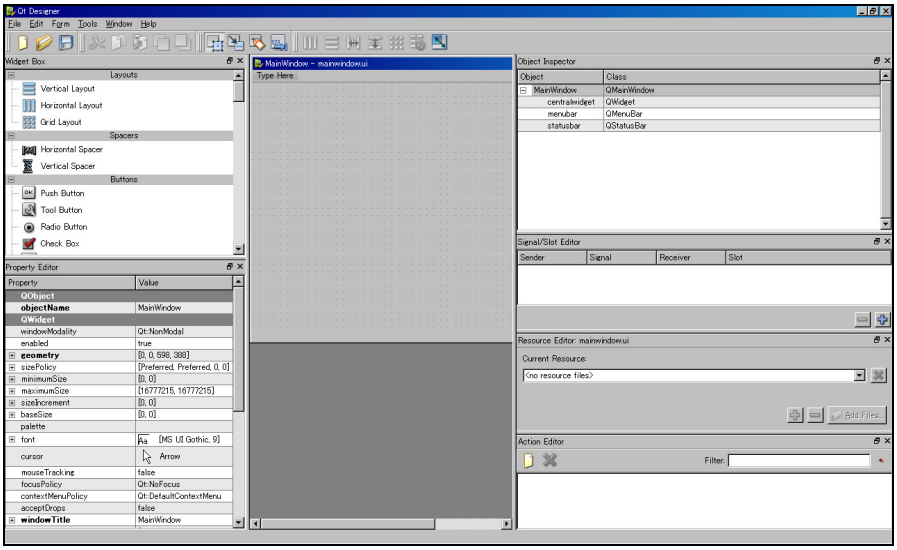
3.3 Capítulo 2: Creación de la aplicación inicial (II): (Modificando la interfaz)

En los siguientes pasos extenderemos la aplicación creada en el paso anterior con el objetivo de implementar alguna funcionalidad sencilla, como por ejemplo rellenar una lista con el input recogido del usuario mediante un campo de texto. En primer lugar extenderemos la interfaz de usuario para dar soporte a las nuevas funcionalidades que contendrá la aplicación mediante QtDesigner. Para ello modificaremos el archivo mainwindow.ui del proyecto. Este archivo, mainwindow.ui, representa la ventana principal del proyecto (por defecto es una ventana vacía), y ha sido creado por QDevelop. Haciendo doble clic sobre éste…



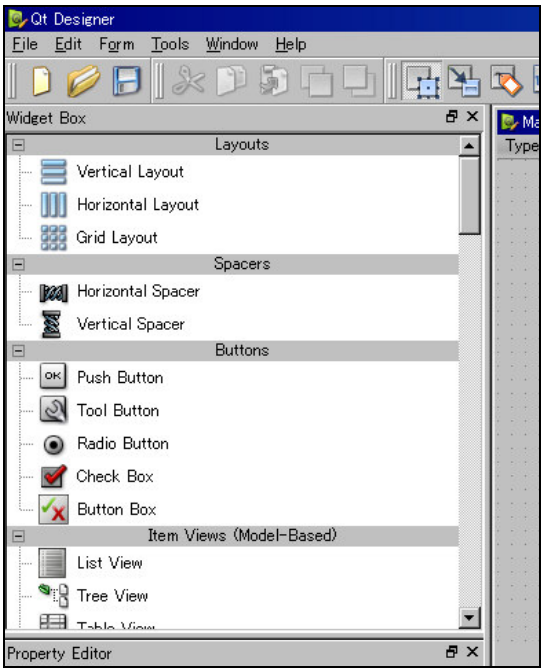
Archivo mainwindow.ui bajo "Diálogos"

…se abrirá el editor grafico QtDesigner y podremos modificarla:



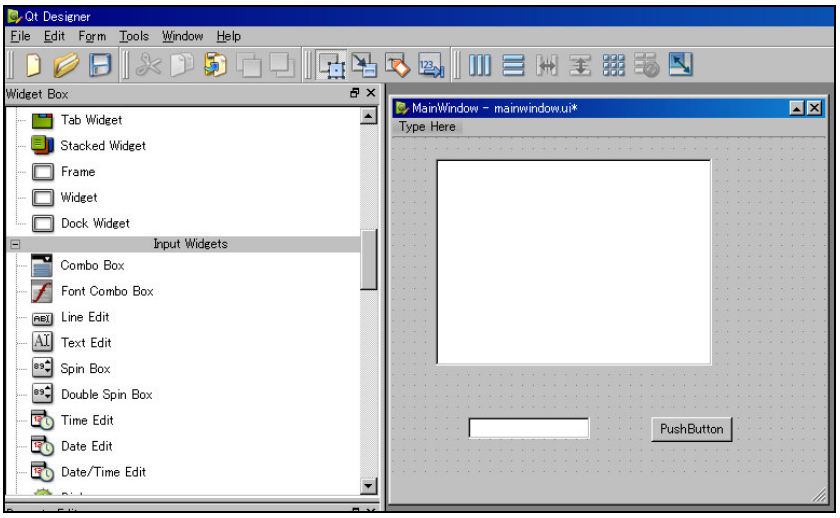
Ventana principal de QtDesigner

El manejo de QtDesigner es bastante sencillo, simplemente hemos de arrastrar los componentes que deseemos (llamados widgets) desde una barra de herramientas de la izquierda (llamada Widget Box) hasta la ventana de nuestro proyecto, donde podremos recolocarlos.



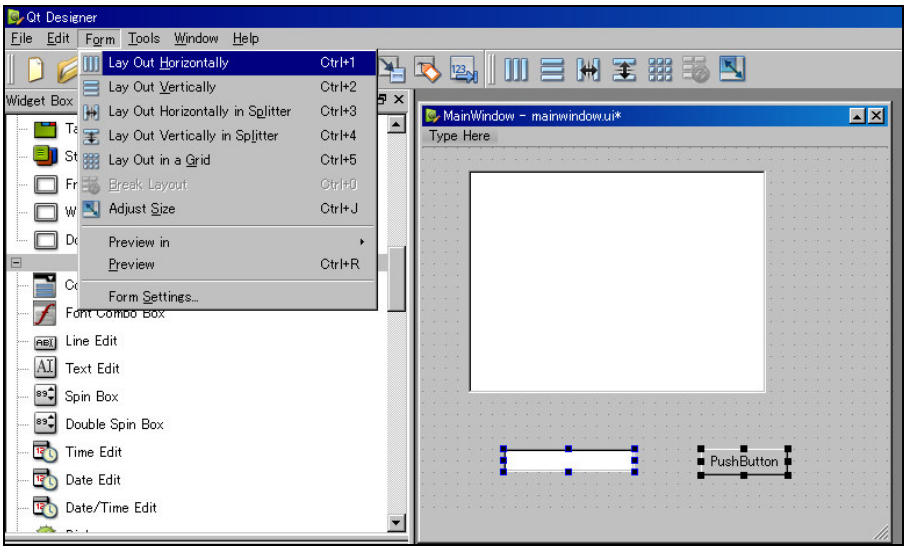
WidgetBox de QtDesigner

A continuación vamos a proceder a colocar los elementos de interfaz necesarios, que son un List View (subcategoría Item Views), un Line Edit (subcategoría Input Widgets) y un Push Button (subcategoría Buttons), arrastrándolos desde la Widget box hasta nuestra ventana, quedando esta del modo siguiente:



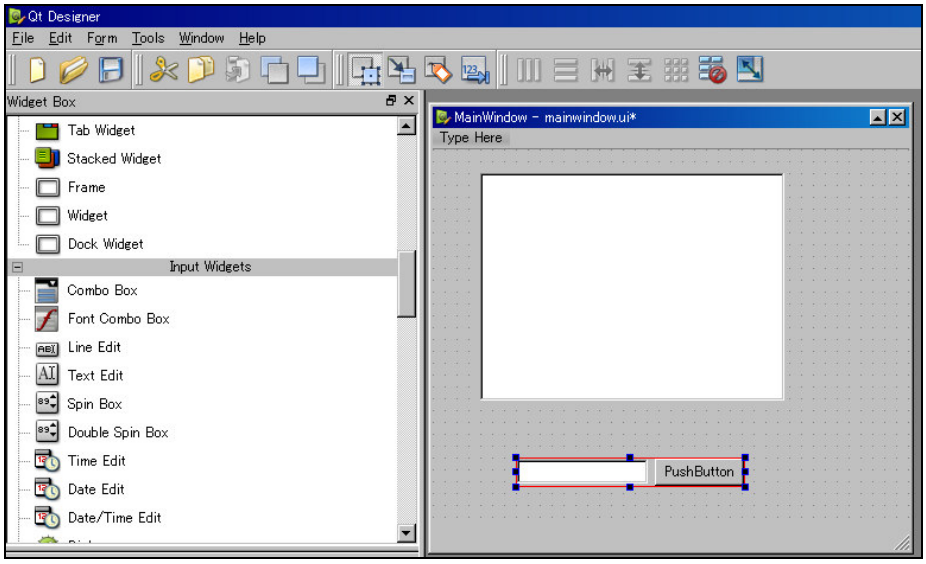
Aspecto del formulario tras colocar los elementos

Es necesario configurar el *layout* de los elementos dentro de la ventana y alinear los elementos. Para ello hemos de seleccionar los elementos que deseamos alinear y elegir el botón de *layout* apropiado. Podemos alinear un elemento con otro elemento, un elemento con grupos alineados de elementos, grupos alineados de elementos, e incluso todos los elementos. El nuestro caso, seleccionaremos en primer lugar el botón y el Line Edit, y haremos clic sobre el botón *layout* horizontal, una de las opciones de layout presentes en la barra de iconos (o bien en la opción de menú Form > Layout Horizontally)



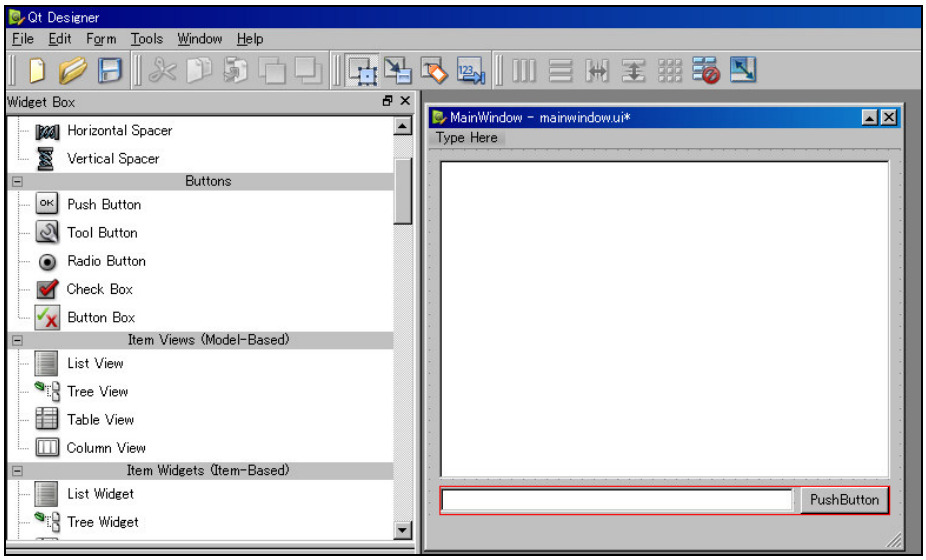
Opción de menú Lay Out Horizontally en QtDesigner

Una vez hecho esto podemos ver cómo estos dos elementos han cambiado de tamaño y se han redistribuido en la ventana de la aplicación de manera idéntica:



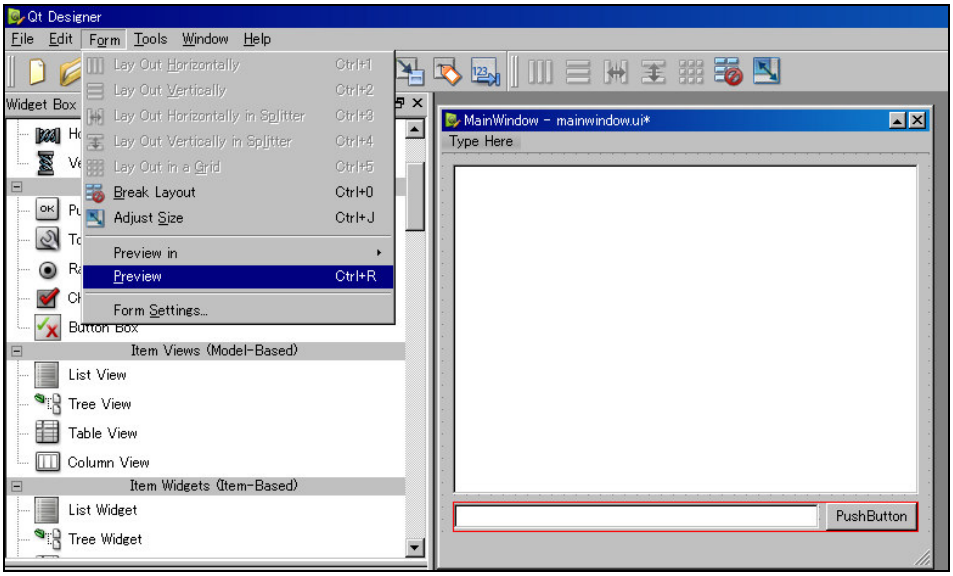
Aspectos de los elementos tras aplicar el layout

A continuación fijaremos el elemento restante, y haremos que el tamaño de los elementos sea proporcional y uniforme al tamaño de la ventana: clicaremos en un área vacía de la ventana y seleccionamos la opción Layout in a Grid, quedando la ventana de la manera siguiente:



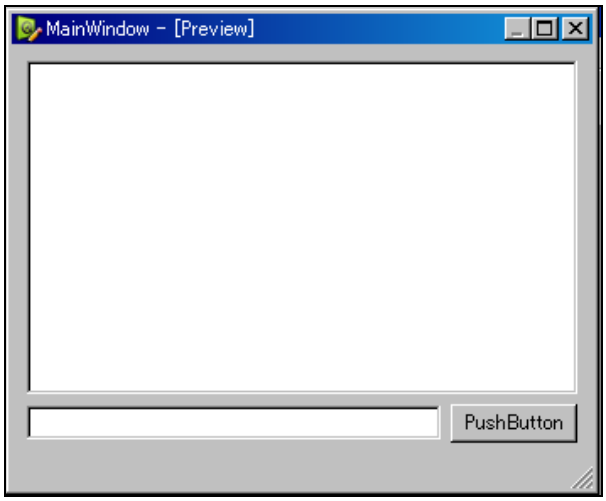
Aspecto del formulario tras aplicar "Layout in a Grid"

Para ver una previsualización de la ventana podemos seleccionar la opción Form > Preview o bien mediante el atajo Ctrl + R:



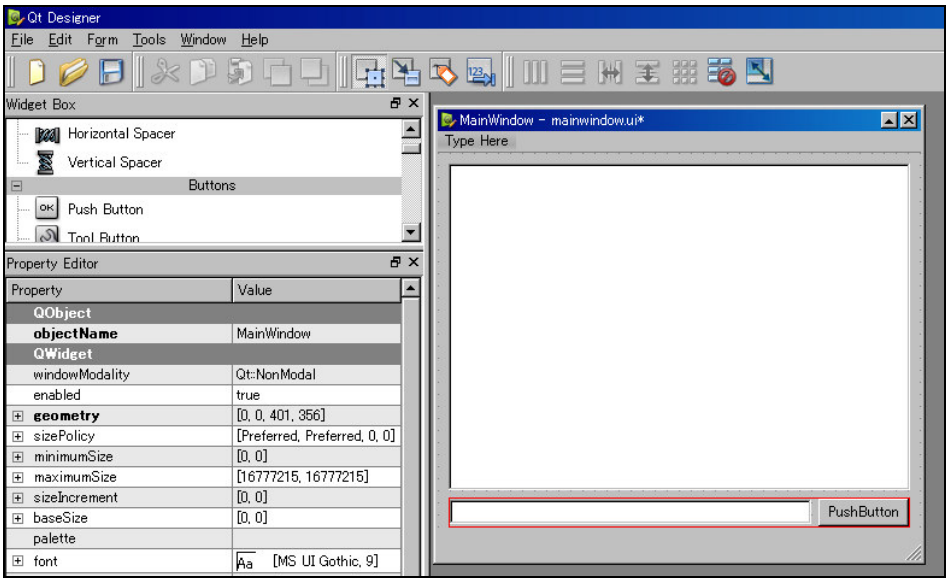
Opcion de previsualización en QtDesigner

La ventana aparecerá y podremos ver que al cambiar el tamaño de ésta, los elementos contenidos cambian de tamaño de manera proporcional. Para cerrar la ventana pulsaremos el botón X de la barra de titulo de la ventana.



Aspecto de la previsualización de la ventana de nuestro proyecto

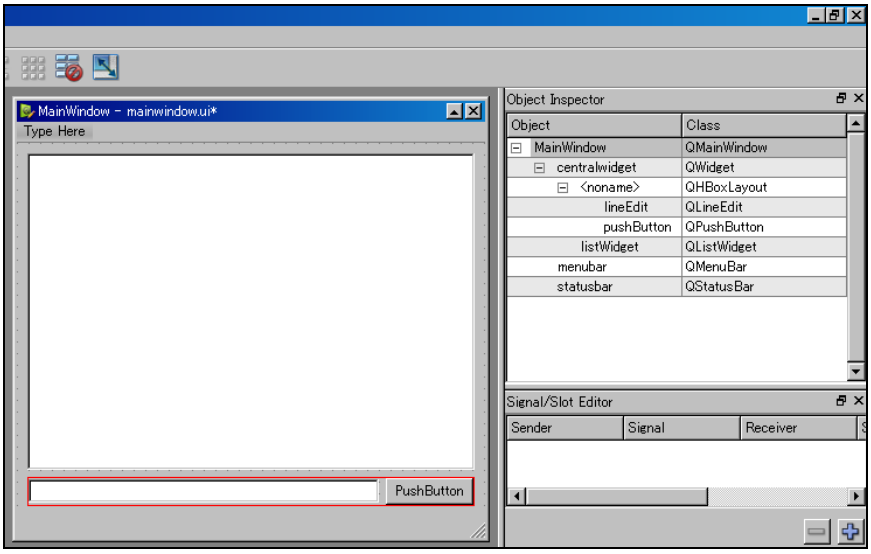
Los elementos que hemos usado para crear la ventana se llaman widgets, y tienen una serie de propiedades. Ahora que ya tenemos el diseño básico de la ventana, pasaremos a configurar algunas propiedades de los widgets implicados. Para modificar las propiedades de los widgets usaremos el editor de propiedades (Property Editor) que se encuentra en la parte inferior izquierda de la ventana principal de QtDesigner. Éste contiene los pares propiedad-valor para los widgets del diseño que estén seleccionados actualmente:



Property editor de QtDesigner

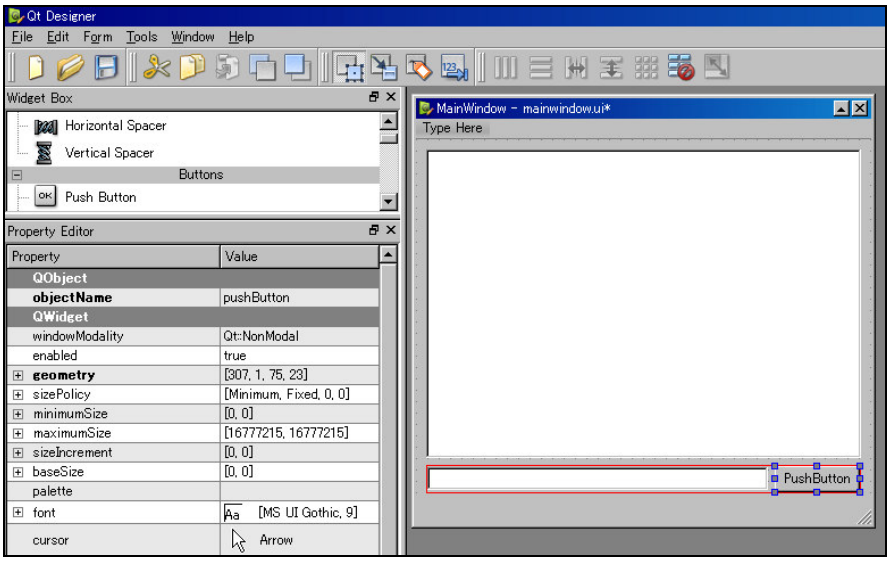
En primer lugar hemos de seleccionar de qué widget queremos modificar propiedades. Existen dos maneras para seleccionarlo:

- Seleccionando el widget desde la ventana principal.
- Seleccionando el widget desde la barra de herramientas Object Inspector (parte superior derecha de la ventana principal de QtDesigner). En esta barra aparece una jerarquía de todos los objetos que forman parte de la ventana principal.



Object Inspector de QtDesigner

Por ejemplo, para cambiar las propiedades del botón lo que haremos será seleccionar el botón de la ventana principal, y seguidamente el Property Editor nos mostrará todas las propiedades disponibles para el botón.

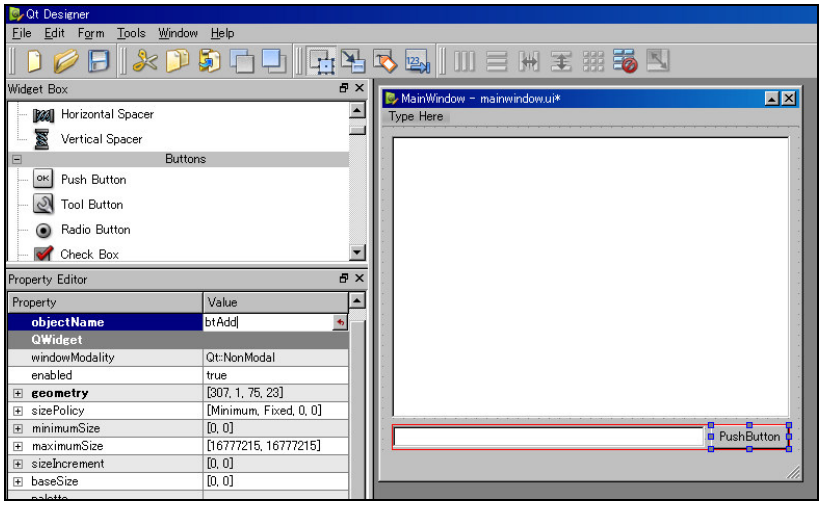


Propiedades del elemento seleccionado en Property Editor de QtDesigner

De estas propiedades, vamos a cambiar:

- ObjectName->btAdd (nombre para el widget)
- TextProperty->&Anadir (texto que mostrara el botón)

(Nótese que el & delante del String “Anadir” hará referencia al shortcut para el botón)



Cambiando la propiedad ObjectName del elemento

Haremos lo mismo para el widget Line Edit, las propiedades que cambiaremos serán:

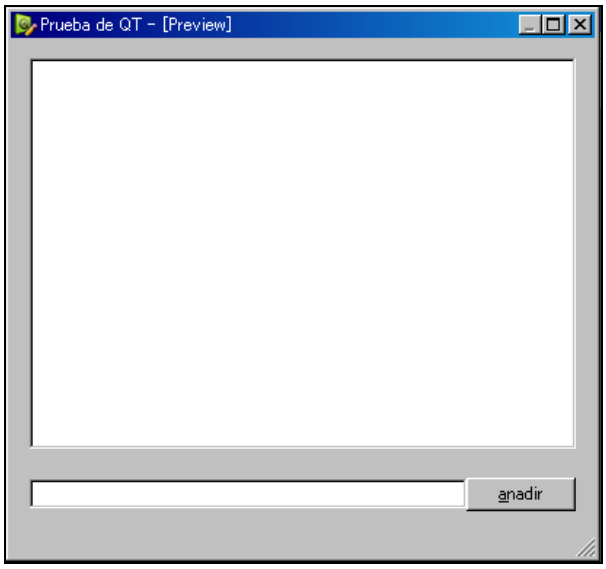
- ObjectName->edText (nombre para el widget)

Y finalmente también haremos lo mismo para el elemento List Widget:

- ObjectName->lbMyList (nombre para el widget)

Desde el Property Editor no sólo podemos configurar las propiedades de los widgets de la ventana, también las propiedades de la ventana en sí. Por ejemplo, podemos ajustar el diseño (layout, márgenes, etc.…) mediante el objeto centralwidget (debajo de MainWindow en el Object Inspector). Al hacer clic sobre éste, Property Editor nos mostrará sus propiedades, que podremos modificar.

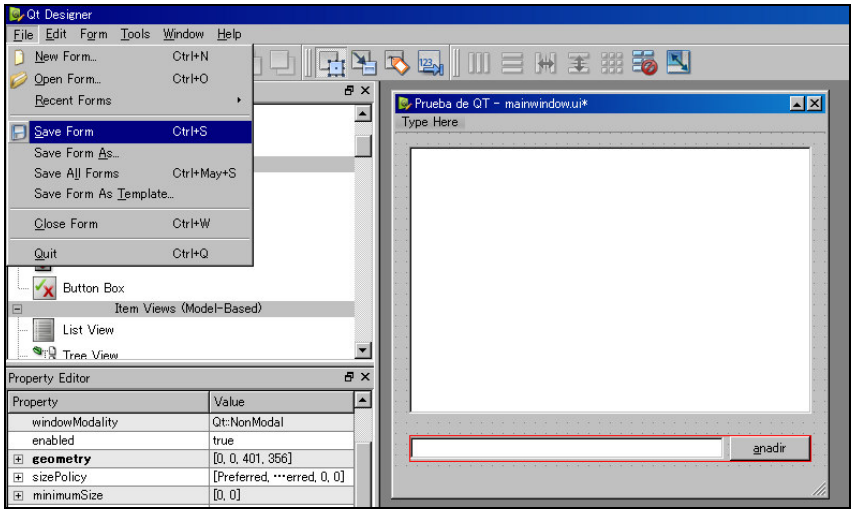
También podemos probar a cambiar el título de la ventana principal. Para ello, seleccionaremos el objeto MainWindow desde el Object Editor (o bien haciendo clic sobre un área vacía de la ventana) y modificaremos su propiedad “windowTitle” con el String que deseemos, quedando la ventana finalmente de la siguiente manera:



Aspecto de nuestra ventana tras cambiarle el título

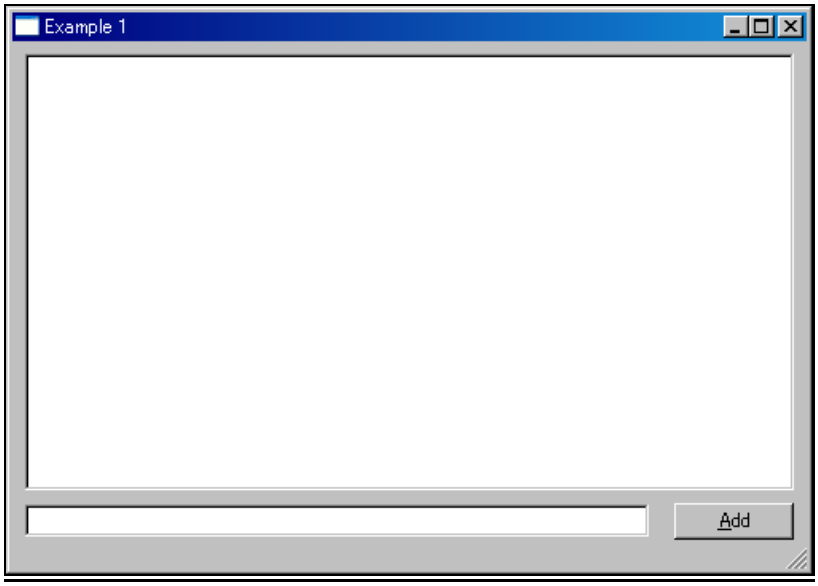
Hay multitud de propiedades que permiten configurar cada Widget en particular, es recomendable navegar un poco por la estructura de objetos e ir mirando las propiedades de cada uno de ellos para acabar de entender el funcionamiento de QtDesigner.

A continuación, guardamos la ventana mediante la opción de menú File > Save Form, o haciendo clic sobre el icono del diskette en la barra de iconos:



Opción de guardado de QtDesigner

Seguidamente cerramos QtDesigner, volviendo a QDevelop. Una vez en QDevelop, recompilamos y ejecutamos el proyecto para ver si funciona, obteniendo una ventana idéntica a la que previamente habíamos previsualizado en QtDesigner:



Ventana principal de nuestro proyecto

3.4 Capítulo 3: Creación de la aplicación inicial (III): (Modificando el comportamiento)

A partir de ahora vamos a incorporar el código necesario para que nuestra aplicación ejecute alguna tarea útil, como puede ser recoger todos los inputs del usuario y mostrarlos en una lista, mediante el mecanismo estrella de Qt, Signals y Slots.

3.4.1 El mecanismo Signal-Slot

En la programación de Interfaces Graficas de Usuario (GUI) se busca que los cambios o eventos producidos sobre un objeto sean comunicados a la resta de objetos. El mecanismo de Signals y Slots es un mecanismo de comunicación entre objetos seguro, flexible y totalmente orientado a objetos, en comparación con otros métodos como por ejemplo los callbacks.

Los objetos de Qt pueden contener Signals y Slots. Básicamente, al producirse un evento sobre un objeto este emite un Signal, que hará que se ejecute un determinado Slot (una función en respuesta al Signal emitido) de otro objeto. Para que este sistema funcione es necesaria una conexión entre los Signals y Slots de los objetos implicados de antemano.

Podemos considerar este mecanismo como algo equiparable al sistema de interrupciones (Signals) y RSI -Rutinas de Servicio de Interrupciones- (Slots) de Arquitectura de Computadores. En nuestra aplicación, el botón emitirá un Signal (clicked) y queremos que cuando eso suceda se ejecute una determinada función de la ventana principal. Esa función es el Slot. Para conseguirlo, primero vamos a definir un Slot en el archivo de cabecera de la ventana principal, mainwindowimpl.h:

```
Q_OBJECT
public:
MainWindowImpl( QWidget * parent = 0, Qt::WFlags f = 0 );
private Slots:
void addClicked();
};
#endif
```

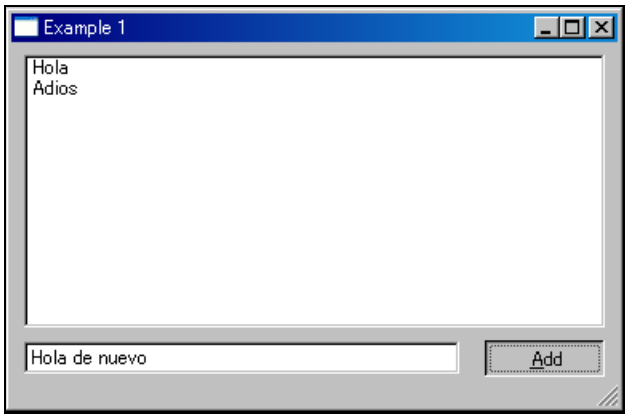
Ahora implementaremos el Slot (addClicked) que hemos definido. Queremos que al recibir el Signal clicked del botón, se lea y añada el contenido del campo de texto a una lista, y se acabe borrando el contenido de este campo de texto. El siguiente código, que deberá ir dentro del archivo fuente mainwindowimpl.cpp, responde a nuestras necesidades:

```
void MainWindowImpl::addClicked()
{
if(edText->text() == "") // If there is no text then exit this function
return;
lbMyList->addItem(edText->text()); // Add the text typed to the list
edText->setText(""); // erase the text from the box they typed in
edText->setFocus(); // set the focus back to the LineEdit from the button
}
```

Finalmente falta conectar el Signal del objeto emisor (Signal clicked del botón) con el Slot del objeto receptor (Slot addClicked() de la ventana principal). Este último paso lo haremos en la función constructora MainWindowImpl, dentro del archivo fuente mainwindowImpl.cpp, añadiendo la siguiente línea:

```
MainWindowImpl::MainWindowImpl( QWidget * parent, Qt::WFlags f)
: QMainWindow(parent, f)
{
setupUi(this);
connect(btAdd, SIGNAL(clicked()), this, SLOT(addClicked()));
}
```

Si ahora compilamos y volvemos a ejecutar la aplicación, esta debería mostrarnos el comportamiento que deseamos.



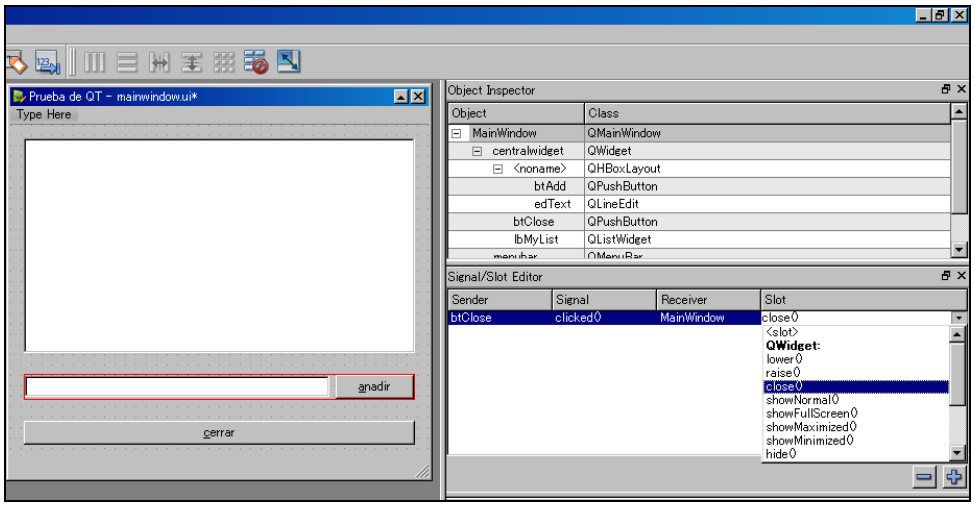
Comportamiento de nuestra primera aplicación de Qt

Hay que destacar que hay diversas maneras de realizar la conexión Signal-Slot en Qt4. Explicaremos estas maneras mediante un ejemplo muy sencillo, que implementará la funcionalidad de terminar (cerrar) la aplicación al pulsar un botón. Este ejemplo tiene dos pasos: el primer paso será añadir un nuevo botón (de nombre btClose) a nuestro proyecto, y el segundo paso será realizar la conexión de todas las maneras posibles.

Debido a que anteriormente ya explicamos cómo añadir y colocar widgets en QtDesigner, obviaremos el primer paso y nos centraremos en el segundo.

a) Conexión mediante QtDesigner:

Volvemos a QtDesigner haciendo doble clic sobre el archivo de dialogo (mainwindow.ui) del proyecto en QDevelop. En la barra de herramientas Signal-Slot Editor, pulsaremos el botón + para crear una nueva asociación. Hecho esto, aparece una nueva fila de asociación vacía, en la que deberemos seleccionar el emisor, receptor, Signal del emisor y Slot del receptor. Conectaremos el Signal clicked() del elemento btClose (se supone que lo hemos definido anteriormente) con el Slot close() de la ventana principal (elemento MainWindow) de la manera siguiente:



Conexión signal-slot en QtDesigner

Este es quizás el método más rápido e intuitivo, ya que no se requiere de código, pero es demasiado limitado. Además, hay que tener en cuenta que los Slots declarados por nosotros en QDevelop no aparecerán aquí; únicamente se nos mostraran los Signals y Slots por defecto de cada Widget en Qt. Una vez hecho esto podemos guardar, volver a QDevelop, compilar y ejecutar la aplicación para verificar el funcionamiento.

b) Conexión mediante código (QDevelop)

La manera más flexible es crear las conexiones es a base de código en QDevelop, aunque todos los Slots definidos serán invisibles en QtDesigner. Tenemos dos opciones:

Opción 1 (clásica)

La primera opción para conectar un Signal de un elemento emisor con un Slot de un receptor es mediante la siguiente línea de código, idéntica a la que vimos anteriormente y que deberá ir dentro de la función Mainwindowimpl en el archivo fuente MainWindowImpl.c:

```
connect(btClose, SIGNAL(clicked()), this, SLOT(close()));
```

Mediante esta línea estamos conectando el Signal (clicked) del emisor (botón btClose) con el Slot (addClicked) del receptor (this, MainWindow).

Opción 2 (conexión automática)

La segunda opción es usar la función de conexión automática de la que hace gala Qt4. Para ello únicamente se han de implementar los Slots con un nombre específico, dentro del archivo fuente MainWindowImpl.cpp:

```
void MainWindowImpl::on_(EMISOR)_(SEÑAL){ (código ) }
```

Con esto, Qt se encargará automáticamente de que cuando el elemento (EMISOR) envíe el Signal (SEÑAL) se ejecute el código (código). Se podría decir que la propia definición del Slot hace de conector con el Signal. En el ejemplo del botón del cierre de la aplicación, la siguiente función bastaría:

```
void MainWindowImpl::on_btClose_clicked(){ this(close); }
```

Finalmente, hay que tener en cuenta que muchos Widgets disponen de Signals sobrecargados (Signals con el mismo nombre pero diferentes argumentos) que se emiten de modo simultáneo ante un mismo evento: Por ejemplo, al hacer clic sobre un botón se envía el Signal clicked, pero también el Signal clicked(bool). Al hacer la conexión, únicamente se conecta un Signal específico. Si queremos que todos los Signals con un mismo nombre sean conectados al mismo Slot, usaremos la siguiente línea de código dentro de la función de MainWindowImpl:

```
QMetaObject::connectSlotsByName(this);
```

La función connectSlotsByName(Objeto) busca recursivamente en todos los hijos de Objeto (en nuestro caso “this”, MainWindow), conectando todos aquellos Signals que tengan un nombre idéntico (independientemente de los parámetros) con el mismo Slot; por ello al usar esta función todos los Signals sobrecargados se conectarían al mismo Slot.

Una vez visto como crear una sencilla aplicación con Qt4, los widgets y el mecanismo Signal-Slot de Qt4, nos será mucho más fácil entender la estructura interna y las clases de Qt4.

3.5 Capítulo 4: Creación de Custom Widgets (I)

En ocasiones los widgets/objetos que vienen con Qt “por defecto” no son suficientes, y no queda otro remedio que crear los nuestros propios. En este paso veremos dos métodos para hacerlo.

3.5.1 Herencia de un Widget existente

Muchas veces queremos implementar una funcionalidad muy similar, prácticamente exacta, a la que desempeña un Widget ya existente. Una solución a este problema es heredar de este Widget ya existente, y adaptarlo a nuestras necesidades.

Por ejemplo, el Widget QspinBox (un selector de valor) solo soporta valores decimales enteros, y nos encontramos en la situación en la que necesitamos un QspinBox que soporte valores Hexadecimales. Para realizar un Custom Widget mediante este método, hemos de seguir estos pasos:

1. Seleccionar un Widget existente
2. Crear un nuevo Widget que herede del existente
3. Reimplementar las funciones del nuevo Widget para dar soporte a nuestras necesidades.

3.5.2 Herencia directa de clase QWidget

En muchas ocasiones queremos implementar Custom Widgets que son combinaciones de Widgets convencionales, o incluso de otros Custom Widgets. Éstos también se pueden desarrollar en QtDesigner, mediante los siguientes pasos:

1. Crear un nuevo formulario usando la plantilla “Widget”
2. Colocar los diferentes Widgets necesarios, redimensionarlos y aplicarles un layout si es necesario.
3. Configurar Signals y Slots de éstos.

El hecho de combinar diferentes Widgets en uno también se puede realizar a base de código. En cualquier caso, la clase resultante heredará de QWidget.

Incluso cuando no podemos ni adaptar ni combinar ningún tipo de Widget existente, también podemos crear nuestro Custom Widget. Para ello heredaremos de la clase QWidget y reimplementaremos unos cuantos manejadores de eventos de clases para dibujar el Widget y responder a los clics del ratón. De este modo tenemos total libertad a la hora de crear un Custom Widget. Podemos implementar cualquier Custom Widget usando las funciones públicas de la clase QWidget, abstrayéndonos de la plataforma de desarrollo en cuestión. Existen

Widgets convencionales (por defecto) en Qt4 que han sido implementados mediante esta técnica, como por ejemplo Qlabel y QPushButton.

En este manual no crearemos ningún Custom Widget, usaremos un Custom Widget existente (MyCanvasView) y lo usaremos en un proyecto de prueba.

3.6 Capítulo 5: Creación de Custom Widgets (II) Integración

Ahora nos encontramos con el problema de cómo integrar un Custom Widget en Qt4. Existen dos métodos para que QtDesigner pueda reconocer un Custom Widget en Qt4.

3.6.1 Promoción

El método de promoción es quizás el más sencillo. Consiste en elegir un Widget existente, con una API similar a nuestro Custom Widget, y rellenar un diálogo de promoción con la información necesaria. Con esto, podremos usar nuestro Custom Widget dentro de QtDesigner, aunque en edición y previsualización se representará como el Widget del que hemos promocionado.

Por ejemplo, para el caso del ejemplo anterior, HexSpinBox, seguiríamos estos pasos:

1. Arrastrar y colocar el widget QSpinBox desde la Widget Toolbox hasta nuestra ventana.
2. Hacer clic con el botón derecho del ratón sobre el Widget y seleccionar la opción “promocionar a Custom Widget” del menú contextual.
3. Rellenar los campos de “nombre de la clase” y “archivo de cabecera” con HexSpinBox y HexSPinBox.h (una nueva definición para el Widget), respectivamente.

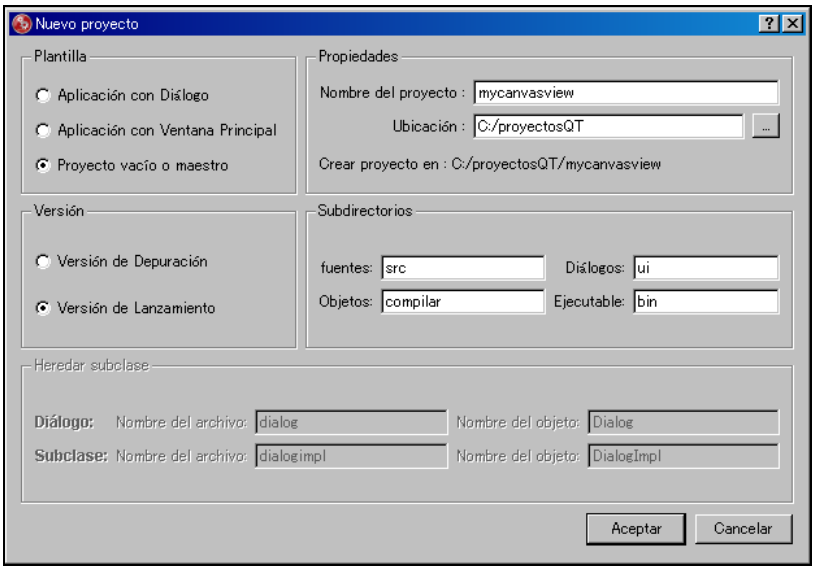
La ventaja de este método es, sin duda, su rapidez y simplicidad. Sus inconvenientes son básicamente dos: no podremos acceder a las propiedades definidas exclusivamente para el Custom Widget dentro de QtDesigner, y el Custom Widget aparecerá dentro de QtDesigner como un Widget convencional y no un Custom Widget. Estos problemas se verán subsanados por el segundo método, la creación de un plugin o librería para el Custom Widget.

3.6.2 Creación de plugin (librería)

En este segundo método, se crea un plugin (librería) para el Custom Widget, que será usado por QtDesigner a la hora de crear instancias del Custom Widget. De este modo, dentro de QtDesigner se está usando el Custom Widget real, y mediante Qt Meta-Object se pueden cargar las propiedades definidas para éste.

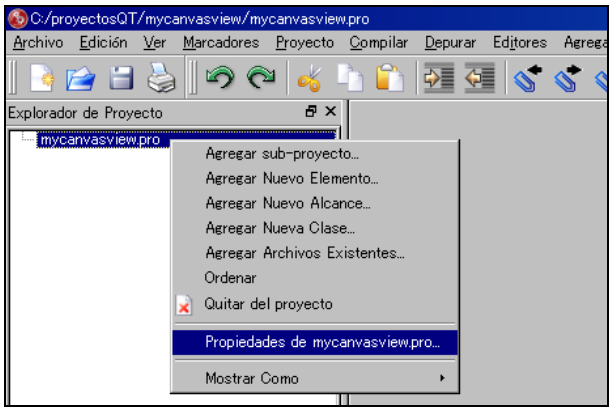
En este manual en concreto, nos encargaremos de crear un plugin para el Custom Widget MyCanvasView presentado en el capítulo anterior.

Para ello, crearemos un nuevo proyecto, que contendrá tanto la implementación del plugin como la implementación del Custom Widget.
Creamos un nuevo proyecto llamado mycanvasview, seleccionando la opción Proyecto > Nuevo Proyecto del menú. Dentro de la ventana que aparecerá, seleccionaremos la opción Proyecto Vacío o maestro en "Plantilla". Finalmente pasamos a rellenar las propiedades del proyecto (nombre, localización, etc.) tal y como se indica en la siguiente imagen.



Ventana de nuevo proyecto en QDevelop

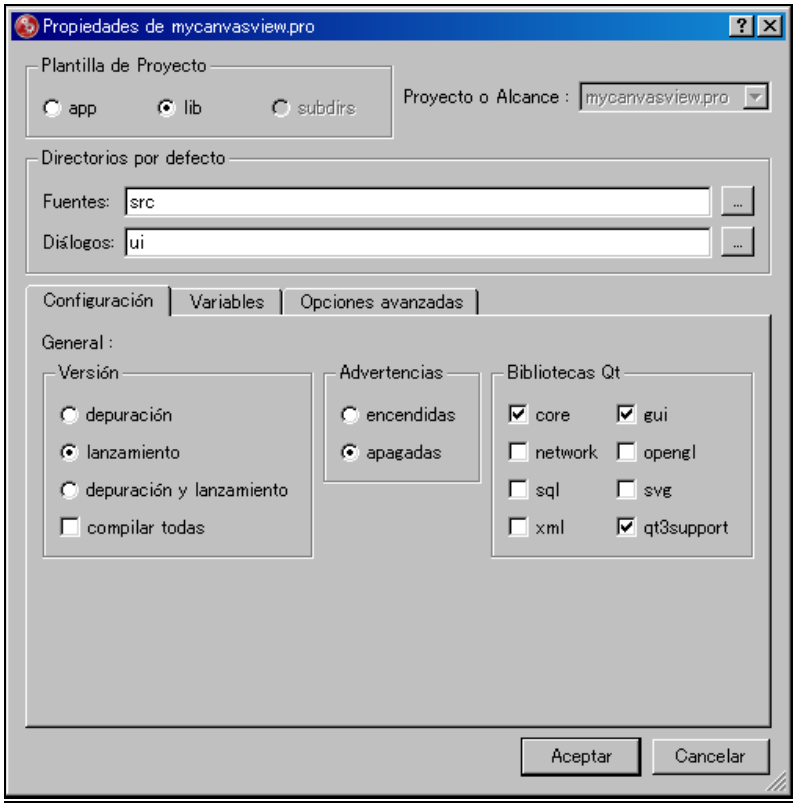
Una vez creado el proyecto vacío, hemos de cambiar algunas propiedades de éste para que constituya el plugin. Para ello seleccionamos el nombre del proyecto (mycanvasview.pro) en el Explorador del Proyecto, hacemos clic con el botón derecho para abrir el menú contextual, y hacemos clic sobre "Propiedades de mycanvasview.pro".



Opción "propiedades de mycanvasview.pro" en QDevelop

En la siguiente ventana debemos configurar el proyecto de la manera siguiente:

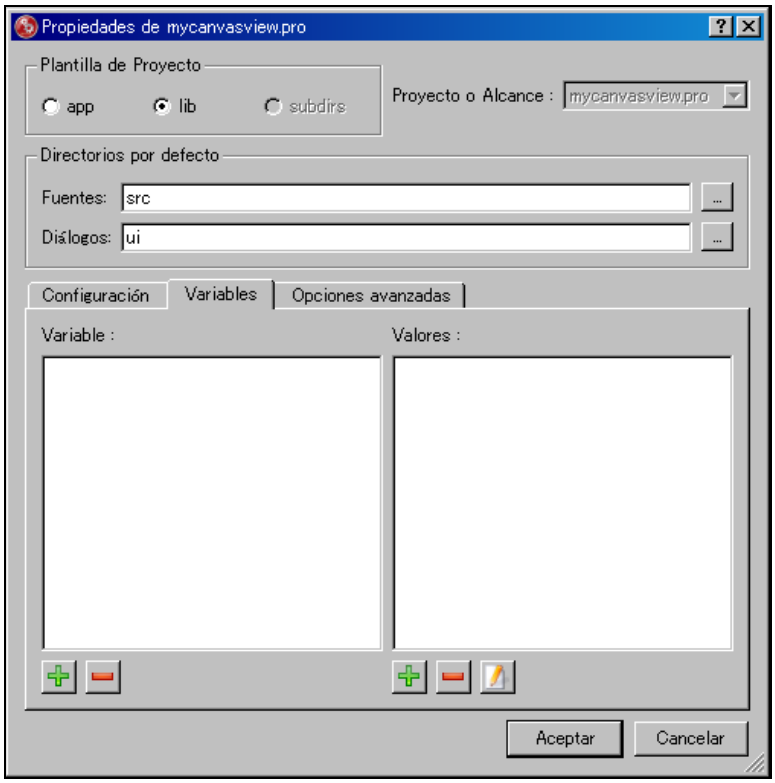
- Plantilla de Proyecto: "lib" (Librería, un plugin es una librería para QtDesigner).
- Pestaña de Configuración: activar versión de lanzamiento, advertencias apagadas, y las bibliotecas de Qt: core, gui y Qt3support (esta última únicamente si estamos creando un plugin para un Custom Widget de la versión Qt3, siendo el caso)



Ventana de propiedades de nuestro archivo .pro

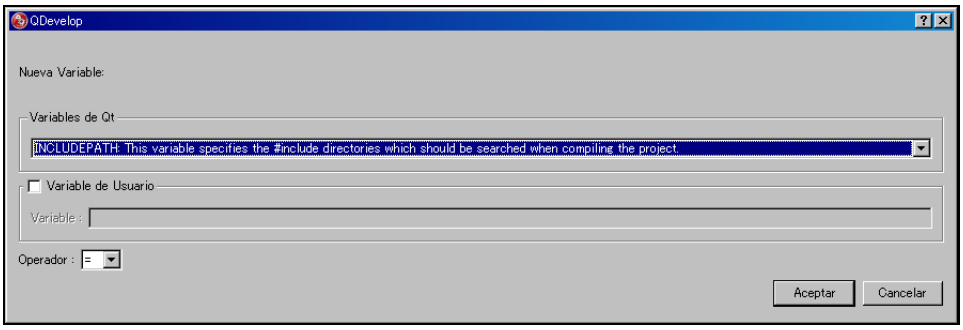
- Pestaña de variables: Deberemos especificar variables de entorno adicionales para el plugin. Estas serán INCLUDEPATH, (valor "C:/Qt/4.3.2/include/QtDesigner") y DESTDIR (valor "C:/Qt/4.3.2/plugins/designer"). Los archivos del plugin generados como resultado de la compilación se enviaran al directorio especificado en DESTDIR, desde donde serán leídos por QtDesigner.

En primer lugar añadiremos la variable de entorno INCLUDEPATH: pulsamos en el icono con forma de + debajo del recuadro de variables:



Pestaña de Variables en la ventana de propiedades

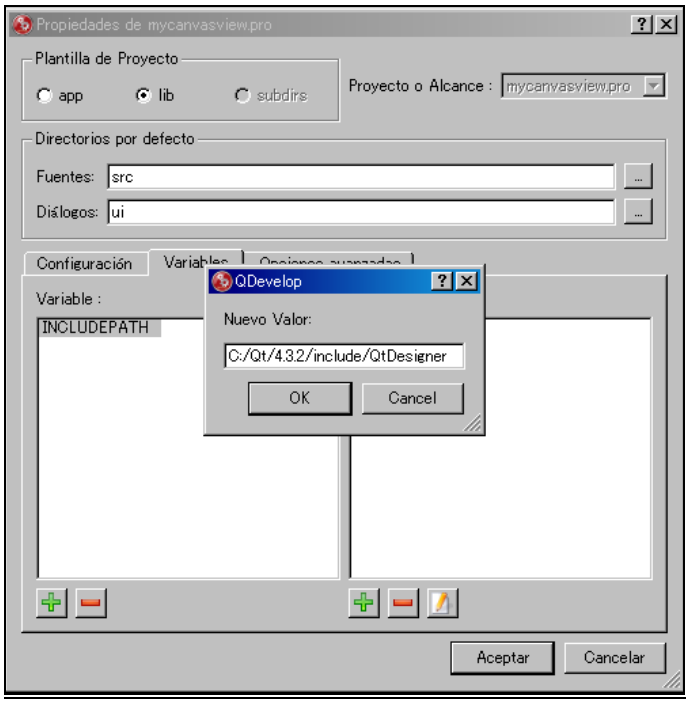
A continuación se nos mostrará una pantalla en la que podremos seleccionar de una lista la variable INCLUDEPATH (o bien introducirla mediante texto, si no aparece en la lista):



Selección/Inclusión de nuevas variables a nuestro proyecto

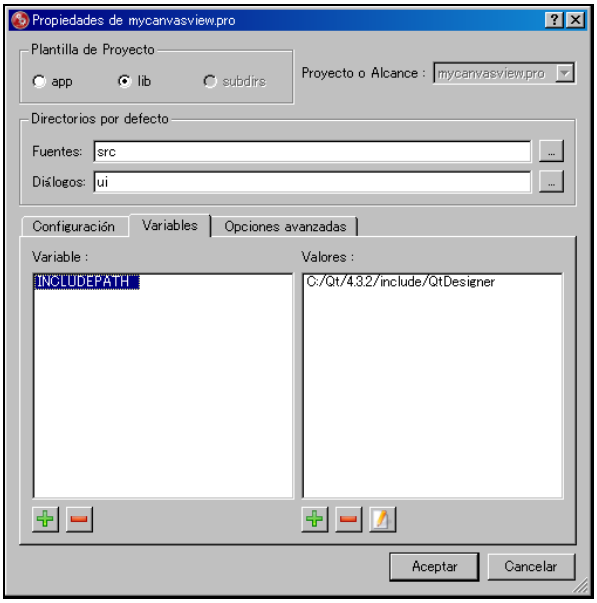
Una vez seleccionada/introducida, pulsamos Aceptar y volvemos a la pantalla de propiedades del proyecto, en la que el recuadro de variables se habrá actualizado mostrando la variable que hemos seleccionado.

Ahora le daremos el valor. Para darle un valor a esta variable, la seleccionamos en el primer recuadro (recuadro de variables) y pulsamos sobre el icono + que se encuentra debajo del segundo recuadro (de valores). Al hacerlo, se nos mostrara la siguiente pantalla:



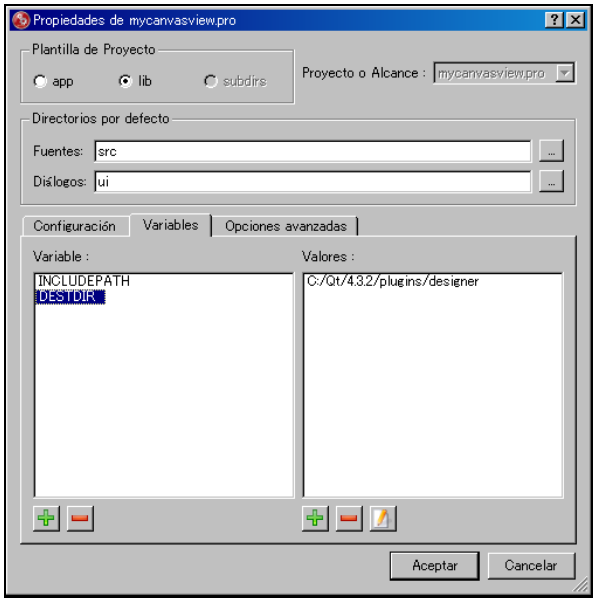
Asignación de valor a la variable

En ésta, introduciremos el valor (en este caso la ruta de includes de QtDesigner, C:/Qt/4.3.2/include/QtDesigner) y pulsamos Aceptar, volviendo a la pantalla de propiedades.



Ventana de propiedades con la variable y su valor

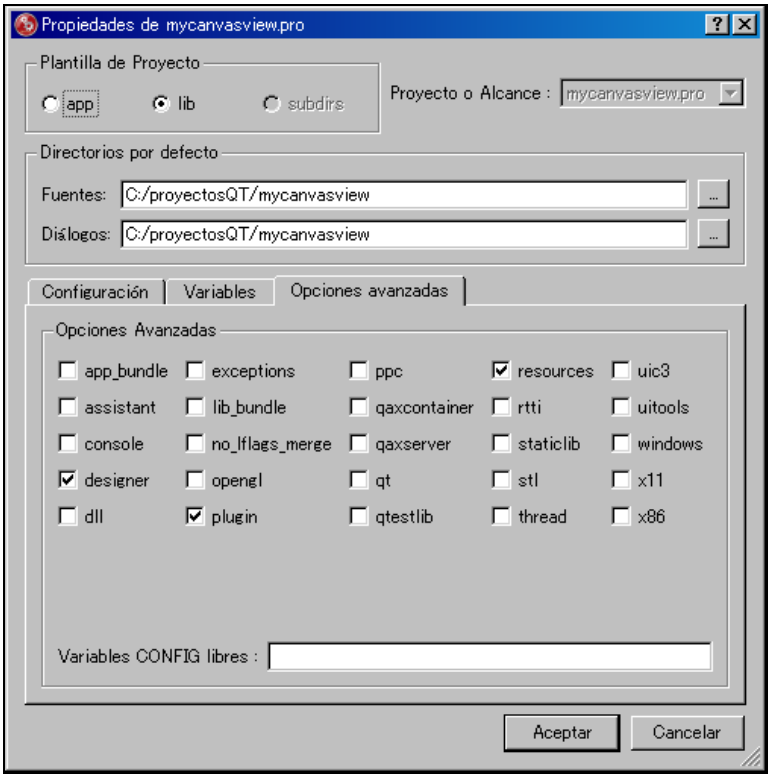
Repetiremos el mismo proceso para introducir otra variable necesaria, DESTDIR, con el valor "C:/Qt/4.3.2/plugins/designer", quedando finalmente de la manera siguiente (al pulsar sobre una variable aparecerán los valores de esta en el recuadro de valores)



Aspecto final de la pestaña Variables en la ventana de propiedades

- Pestaña de Opciones Avanzadas: Deberemos seleccionar las opciones “designer” (debido a que los Custom Widgets normalmente usan componentes proporcionados con QtDesigner), “plugin” (para que qmake considere la librería como un plugin) y “resources” (para indicar que el plugin requiere de recursos).

Gráficamente quedaría de la manera siguiente:



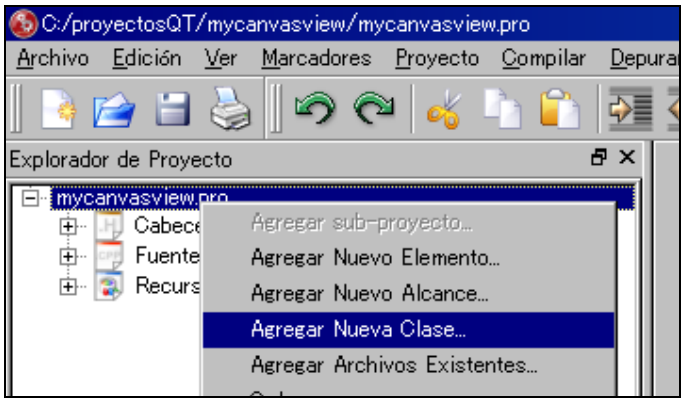
Aspecto final de la pestaña de Opciones Avanzadas

Seguidamente pulsamos el botón Aceptar y listo.

Los cambios de estos valores de configuración del proyecto se verán reflejados como directivas pre-procesador en el archivo .pro de nuestro proyecto (archivo que será usado por gmake a la hora de hacer el make).

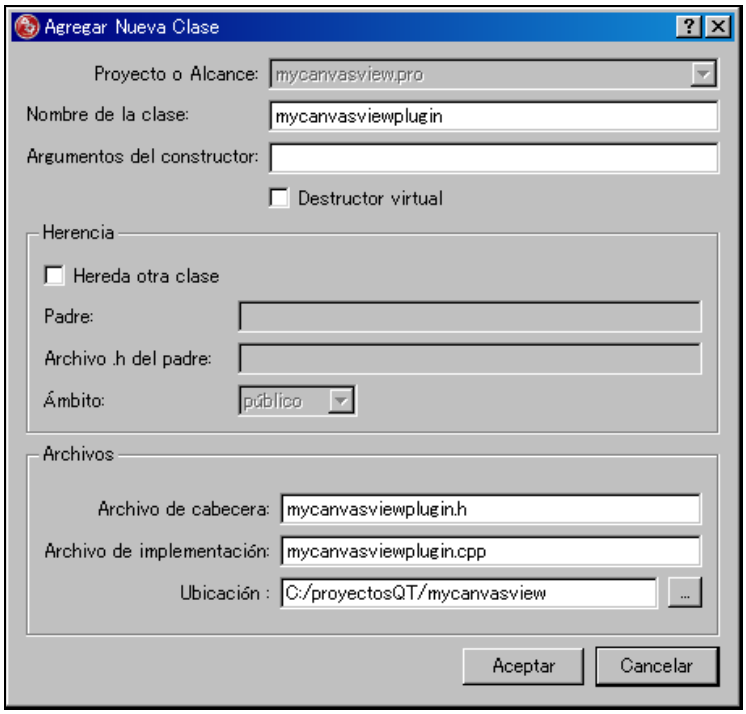
Una vez configurado el proyecto con los pasos anteriores, ya podemos empezar a escribir el código del plugin para nuestro Custom Widget. En concreto rellenaremos 2 archivos, los archivos de cabecera (.h) y de implementación (.cpp) del plugin.

Para añadir los 2 nuevos archivos correspondientes al plugin en el proyecto, haremos clic con el botón derecho del ratón sobre el proyecto, y seleccionamos la opción “agregar nueva clase” del menú contextual:



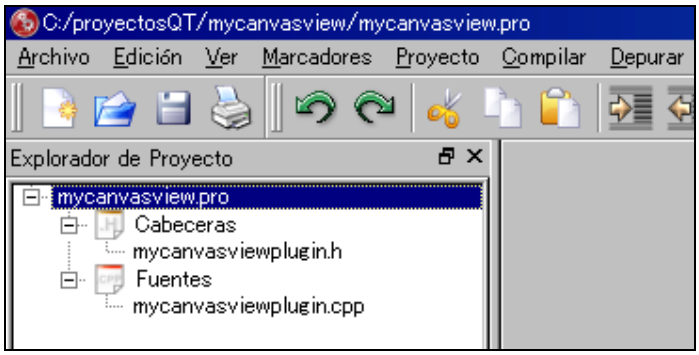
Opción para agregar una nueva clase en QDevelop

En la siguiente ventana rellenamos el nombre de la clase (por ejemplo, “mycanvasviewplugin”). Cambiamos la ubicación de estos ficheros seleccionando la ruta correcta a nuestro proyecto, y finalmente pulsamos sobre Aceptar.



Ventana de agregación de nueva clase en QDevelop

Hecho ésto, si volvemos al explorador de QDevelop veremos los archivos que se han creado:



Nuevos archivos en el Explorador de Proyecto de QDevelop

Ahora rellenaremos el contenido de estos dos archivos. Para editar un archivo, bastará con seleccionarlo para que se nos muestre el contenido de éste en QDevelop.

1) "mycanvasviewplugin.h"

Primero rellenaremos el archivo de cabecera (.h) del plugin, "mycanvasviewplugin.h". El plugin heredará de las clases QObject y de QdesignerCustomWidgetInterface, e implementará la interfaz definida por QdesignerCustomWidgetInterface mediante una serie de funciones preestablecidas, que iremos explicando a la hora de rellenar la implementación del plugin. El contenido de este archivo .h será el siguiente:

```
#ifndef MYCANVASVIEWPLUGIN_H
#define MYCANVASVIEWPLUGIN_H
#include <QObject>
#include "QDesignerCustomWidgetInterface"
class MyCanvasViewPlugin : public QObject,public QDesignerCustomWidgetInterface
{
Q_OBJECT
Q_INTERFACES(QDesignerCustomWidgetInterface)
public:
MyCanvasViewPlugin(QObject *parent = 0);
QString name() const;
QString includeFile() const;
QString group() const;
QIcon icon() const;
QString toolTip() const;
QString whatsThis() const;
bool isContainer() const;
QWidget *createWidget(QWidget *parent);
};
#endif
```

2) mycanvasviewplugin.cpp

Seguidamente pasaremos a rellenar el archivo .cpp del plugin función por función.

En primer lugar, el plugin deberá contener un constructor válido por defecto (trivial):

```
MyCanvasViewPlugin::MyCanvasViewPlugin(QObject *parent)
: QObject(parent)
{
}
```

En segundo lugar, el plugin deberá de proporcionar la información que permita representar a un Custom Widget dentro de QtDesigner. Esta información se proporciona mediante una serie de funciones preestablecidas (las que hemos definido anteriormente en el archivo .h). Implementaremos cada una de estas funciones de la manera siguiente:

1) **Name():** Devolverá el nombre del Custom Widget.

```
QString MyCanvasViewPlugin::name() const
{ return "MyCanvasView"; }
```

2) **IncludeFile():** Devolverá el nombre del archivo de cabecera (.h) del Custom Widget que encapsula el plugin. Este archivo se incluirá dentro del código mediante la herramienta UIC.

```
QString MyCanvasViewPlugin::name() const
{ return "MyCanvasView";}
```

3) **Group():** Devolverá el nombre del grupo de widgets bajo el que aparecerá en QtDesigner. Si no existe ningún grupo con ese nombre, QtDesigner lo creará.

```
QString MyCanvasViewPlugin::group() const
{return tr("JeanPierre Tutorial Qt3");}
```

4) **Icon():** Devolverá la ruta al archivo de icono que representara al widget dentro de la barra de selección de widgets en QtDesigner. El icono debe corresponder a un archivo resource válido del proyecto.

```
QIcon MyCanvasViewPlugin::icon() const
{ return QIcon(":/images/Qtlogo.png");}
```

5) **Tooltip()**: Devolverá la información que muestra QtDesigner al poner el cursor del ratón encima del widget.

```
QString MyCanvasViewPlugin::toolTip() const
{return tr("Jean Pierre My Canvas View");}
```

6) **WhatsThis():** Devolverá el texto what's this del Custom widget, que también aparecerá en QtDesigner.

```
QString MyCanvasViewPlugin::whatsThis() const
{return tr("This widget is presented in Chapter 6 of Jean Pierre Tutorial");}
```

7) **isContainer():** Devolverá un booleano que especifica si el Custom Widget puede contener otros objetos (true) o no (false).

```
bool MyCanvasViewPlugin::isContainer() const
{return false;}
```

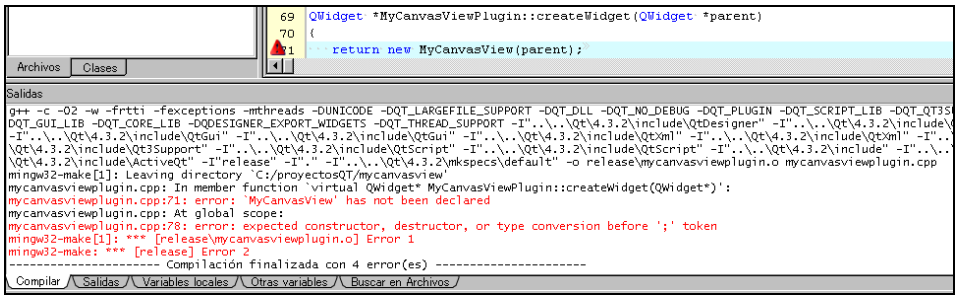
8) **createWidget(parent):** función invocada por QtDesigner para crear una instancia del nuevo Widget. Dentro de esta función simplemente deberemos incorporar el código necesario para crear un nuevo widget (usando la constructora por defecto).

```
QWidget *MyCanvasViewPlugin::createWidget(QWidget *parent)
{return new MyCanvasView(parent); }
```

En tercer y último lugar, el plugin también deberá contener exactamente una ocurrencia de la macro Q_EXPORT_PLUGIN2(pluginName, className) que exportará la clase plugin className con el nombre pluginName:

```
Q_EXPORT_PLUGIN2(MyCanvasView, MyCanvasViewPlugin);
```

Al compilar el proyecto en este punto, debería dar un error por no encontrar MyCanvasView, ya que todavía no lo hemos introducido el código del Custom Widget en nuestro proyecto.



Error de compilación en QDevelop

En el siguiente paso del manual precisamente haremos eso, incluiremos el código del Custom Widget MyCanvasView, portándolo de Qt3 a Qt4.

3.7 Capítulo 5: Creación de Custom Widgets (III) Portabilidad

Antes de portar una aplicación hecha en Qt3 al nuevo Qt4, es aconsejable echar un vistazo a la lista de características que nos ofrece la nueva versión de Qt4, disponible en:
http://doc.trolltech.com/4.2/Qt4-intro.html

Hay que destacar que los binarios de Qt3 no son compatibles con Qt4, así que será necesario recompilar **toda** la aplicación en Qt4, para lo que se necesitan una serie de cambios.

Si queremos portar la aplicación conservando nuestro antiguo código Qt3, Qt4 nos proporciona unas clases y herramientas de soporte para tal efecto. En cambio, si lo que queremos es evitar usar las estas clases de soporte (hasta un cierto punto) y aprovechar las posibilidades que nos ofrecen nuevas clases de Qt4, no queda otro remedio que buscar clase por clase, analizando los cambios que ha habido en la nueva versión y de qué forma ello afecta en lo que queremos implementar. Podemos encontrar una lista de las clases y sus cambios con respecto a Qt3 aquí:
http://doc.trolltech.com/4.2/porting4.html

En este manual en concreto, nos encargaremos de portar el Custom Widget MyCanvasView de Qt3 a Qt4, conservando el código Qt3 de éste y utilizando las clases de soporte que nos proporciona Qt4 para ello.

En primer lugar comprobemos los archivos de código necesarios para el Custom Widget. Éstos son seis:

- mycanvasview.cpp / mycanvasview.h
- bouncylogo.cpp / bouncylogo.h
- imageitem.cpp / imageitem.h

En segundo lugar comprobemos los archivos de recursos auxiliares necesarios. Éstos son tres:

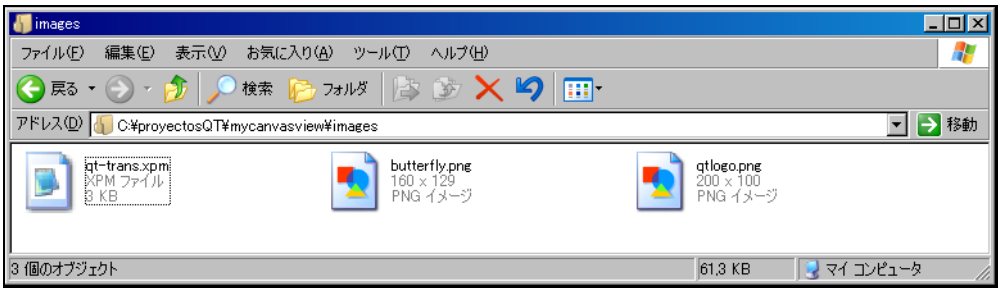
- butterfly.png / Qtlogo.png / Qt-trans.xpm

3.7.1 Portando recursos: Archivos de recursos (.qrc)

El sistema de recursos de Qt es un mecanismo independiente de la plataforma que permite almacenar archivos binarios en un ejecutable. El concepto de recurso es muy amplio y generalizable tanto a aplicaciones como a Custom Widgets. Es muy probable que un Custom Widget / aplicación requiera de algún archivo externo (por ejemplo, un bitmap) para su funcionamiento: estos archivos externos utilizados por el Custom Widget / aplicación serán los recursos (*resources*) de éste.

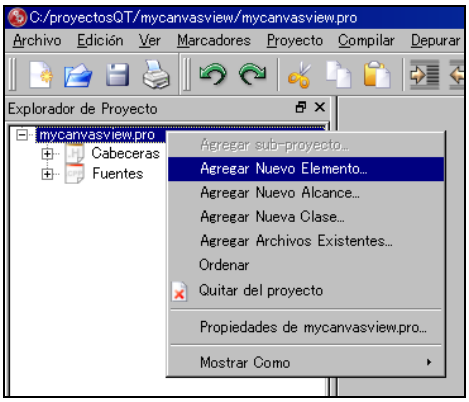
En el caso del Custom Widget MyCanvasView, tres archivos de recursos son necesarios, que copiaremos a una carpeta nueva, llamada images, en la raíz de nuestro proyecto:

- butterfly.png / Qtlogo.png / Qt-trans.xpm



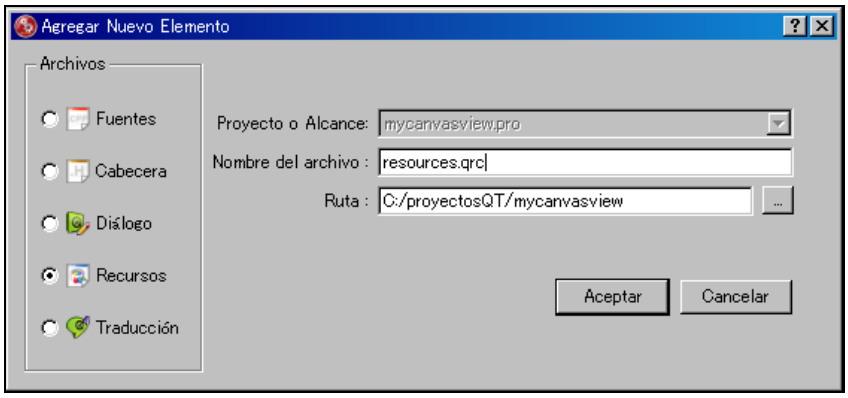
Archivos de recursos necesarios

Será necesario registrar la localización de estos recursos necesarios en un “archivo de recursos”. Para hacer esto, simplemente vamos al explorador del proyecto, hacemos clic en el proyecto con el botón derecho y seleccionamos la opción “agregar nuevo elemento” del menú contextual.



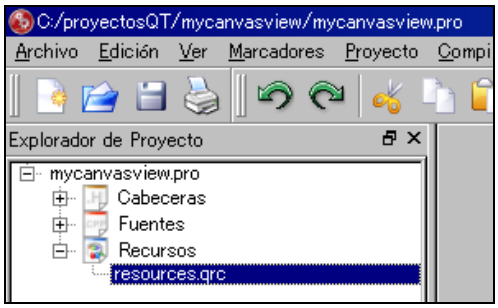
Menú contextual para agregar un nuevo elemento

En la siguiente pantalla seleccionaremos el tipo de elemento a añadir (archivo de recursos), pondremos el nombre a este “archivo de recursos” (por ejemplo, “resources.qrc”, nótese la extensión .qrc) y listo:



Creando el nuevo archivo de recursos

A continuación volveremos al explorador del proyecto y veremos cómo se nos ha creado un nuevo grupo, Recursos, con el archivo solicitado:



Explorador del proyecto y archivos actualizados

Ahora sólo queda editar este archivo de recursos, haciendo clic sobre él. Simplemente tendremos que definir en XML la ruta hacia los recursos necesarios, de la manera siguiente:

```
<!DOCTYPE RCC><RCC version="1.0">
<qresource prefix="">
<file>images/Qt-trans.xpm</file>
<file>images/butterfly.png</file>
<file>images/Qtlogo.png</file>
</qresource>
</RCC>
```

Opcionalmente también podemos definir algún *resource name* que pueda ser usado por la aplicación para acceder a estos recursos, e incluso un alias para poder acceder al recurso dentro del código sin necesidad de saber la ruta. Nótese también que se está utilizando el RCC (Qt ResourCe Compile) en la definición para la posterior compilación.

Este archivo de recursos también es necesario a la hora de hacer el make, por lo que deberemos incluirlo en el archivo .pro (archivo de propiedades de nuestro proyecto). Para ello, añadiremos una nueva variable de entorno, RESOURCES, tomando como valor el nombre del archivo de recursos (resources.qrc):

Una vez hechos todos los pasos, la aplicación/Custom Widget podrá acceder, dentro del código, a estos recursos mediante el path definido (empezando el path por ":/",) o bien por el alias. Por ejemplo, si tenemos el siguiente recurso definido:

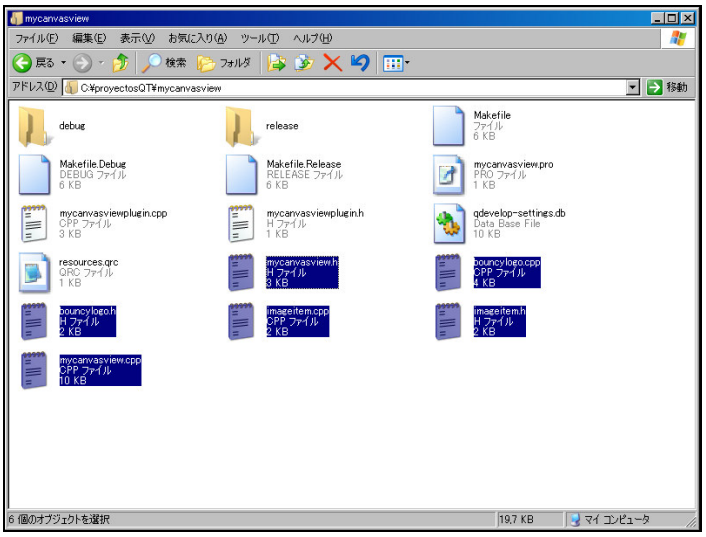
```
<file alias="butterfly.png">images/butterfly.png</file>
```

Podremos acceder a éste mediante la ruta ":/images/butterfly.png" o bien directamente mediante el alias, "butterfly.png".

3.7.2 Portando archivos de código (.h/.cpp)

En el caso del Custom Widget MyCanvasView son necesarios seis archivos de código, que copiaremos a la raíz de nuestra carpeta del proyecto:

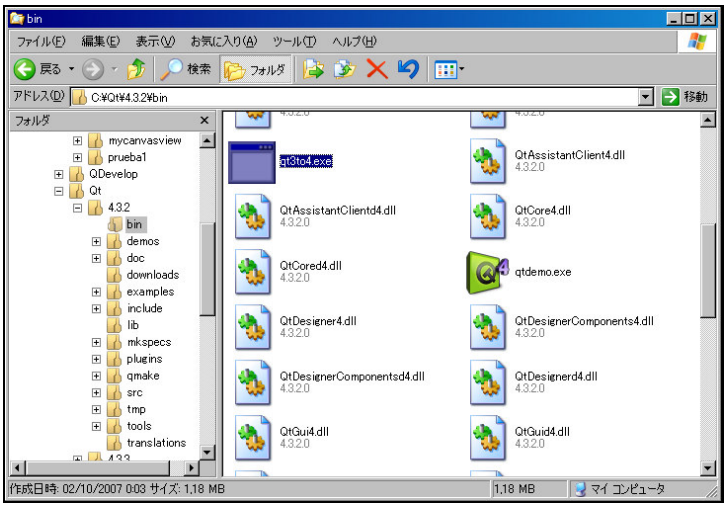
- mycanvasview.cpp / mycanvasview.h
- bouncylogo.cpp / bouncylogo.h
- imageitem.cpp / imageitem.h



Archivos de código necesarios

El primer paso será *linkear* nuestro proyecto con la librería Qt3Support, para activar el soporte de compatibilidad a Qt3 (ya se hizo en el paso anterior).

El segundo paso será ejecutar la utilidad Qt3to4. Qt3to4 es un ejecutable que recibe el nombre de un archivo fuente como parámetro, busca dentro de éste todas las clases obsoletas de Qt3 y las reemplaza por las nuevas clases de soporte que ofrece Qt4. Normalmente estas clases de soporte se diferencian por tener el nombre Q3X: por ejemplo, la antigua clase de Qt3 **QlistBox** corresponderá a la nueva clase **Q3ListBox** en Qt4. Este paso también lo podemos hacer manualmente, aunque tardaremos mucho más tiempo. Empecemos por encontrar dicho binario, Qt3to4: se encuentra en el directorio de ejecutables de Qt4, (Qt3to4.exe en C:\Qt\4.3.2\bin).



Ejecutable Qt3to4.exe

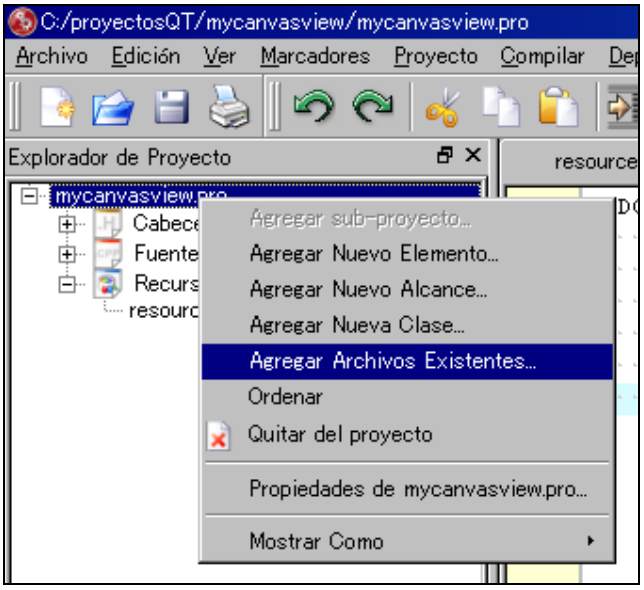
Ejecutaremos el binario desde la consola de Windows (cmd) con el comando…

gt3to4 (ruta y nombre de archivo)

…para cada uno de los seis archivos, tal y como muestra la siguiente imagen:

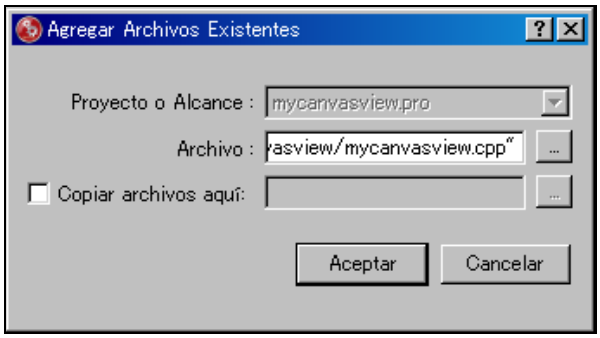
Uso del archivo qt3to4 para convertir a las nuevas clases de Qt4

Una vez hecho esto, ya podemos añadir los seis archivos a nuestro proyecto. Para ello haremos clic con el botón derecho del ratón sobre el nombre del proyecto dentro del explorador de proyecto de QDevelop, y seleccionamos la opción agregar archivos existentes:



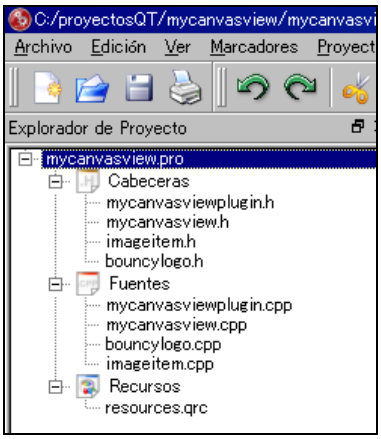
Opción para agregar archivos existentes a nuestro proyecto

A continuación se nos mostrara una pequeña ventana desde la que podremos buscar los archivos que queramos agregar (podremos agregar los 6 archivos a la vez), y acabaremos pulsando el botón Aceptar para confirmar y agregar.



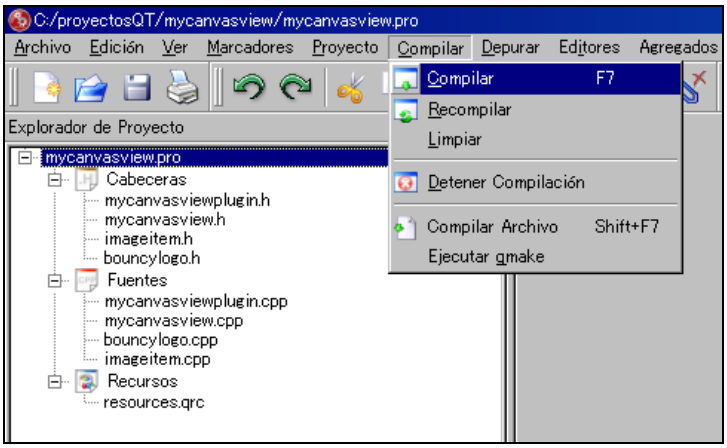
Ventana para agregar archivos al proyecto

Al volver al explorador del proyecto, este debería tener la siguiente estructura:



Explorador del proyecto actualizado

A continuación compilaremos el proyecto:



Opción Compilar de QDevelop

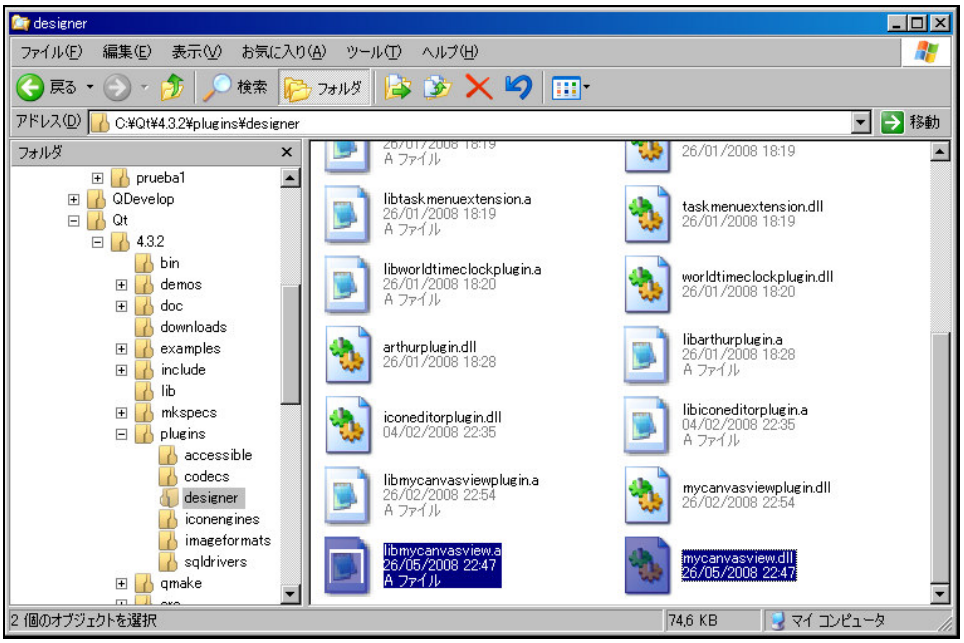
En este punto seguiremos recibiendo el error que nos notifica que el archivo cpp del plugin (mycanvasview.cpp) no encuentra el constructor de MyCanvasView, por lo que ahora añadiremos al archivo mycanvasplugin.cpp las siguientes líneas de inclusión:

```
#include "mycanvasview.h"
#include <QtCore/qplugin.h>
```

Ahora el resultado de la compilación será correcto. Se nos informara del éxito y de los archivos creados mediante la ventana de salidas de compilación:

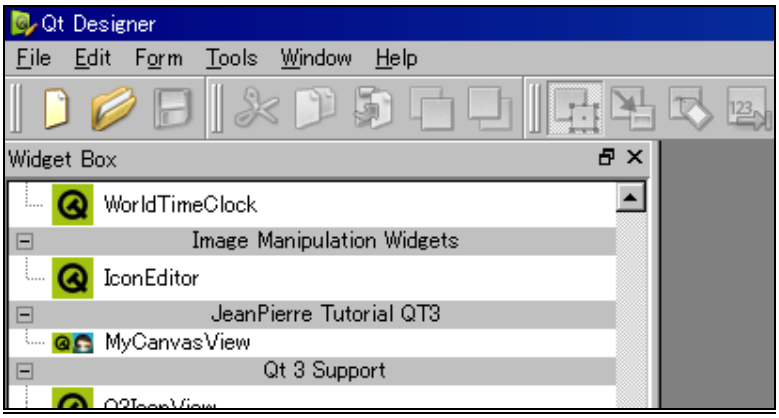
Zona de salidas de QDevelop: no hay errores

Podemos comprobar la librería yendo a la carpeta destino de los archivos generados:



En la ruta apuntada por la variable DESTDIR ha aparecido la librería que hemos generado

Finalmente, si ahora ejecutamos QtDesigner, y examinamos detenidamente la barra de herramientas que contiene los widgets (Widget Box) veremos que habrá aparecido nuestro Custom Widget, y que además contendrá tanto el texto como el icono definidos en el plugin:



La Widget Box ahora contiene nuestro plugin

3.7.3 Portando archivos de la interfaz (.uic)

Aunque no veremos cómo hacer esto con el ejemplo de este tutorial, explicaremos brevemente como se convertirían los archivos de interfaz Qt3 a Qt4.

Una de las cosas que más han cambiado en Qt4 ha sido el concepto de QtDesginer respecto a Qt3, que ha dejado de ser un IDE para convertirse en un editor gráfico de formularios compatible con otros IDEs más extendidos. Con ésto, Qt ha ganado tanto en modularidad como en compatibilidad.

Cambios más importantes:

- QtDesigner sólo puede leer y editar archivos .ui, desconoce los archivos de proyecto .pro.
- QtDesigner ya no es un IDE, por lo que no podremos editar código fuente con él.
- Qt4 no puede leer archivos creados por Qt3, pero disponemos de una herramienta (uic3, del estilo de Qt3to4) que nos permite convertir archivos .ui al nuevo formato de Qt4.
- Los iconos se almacenarán en archivos de recursos (.qrc).

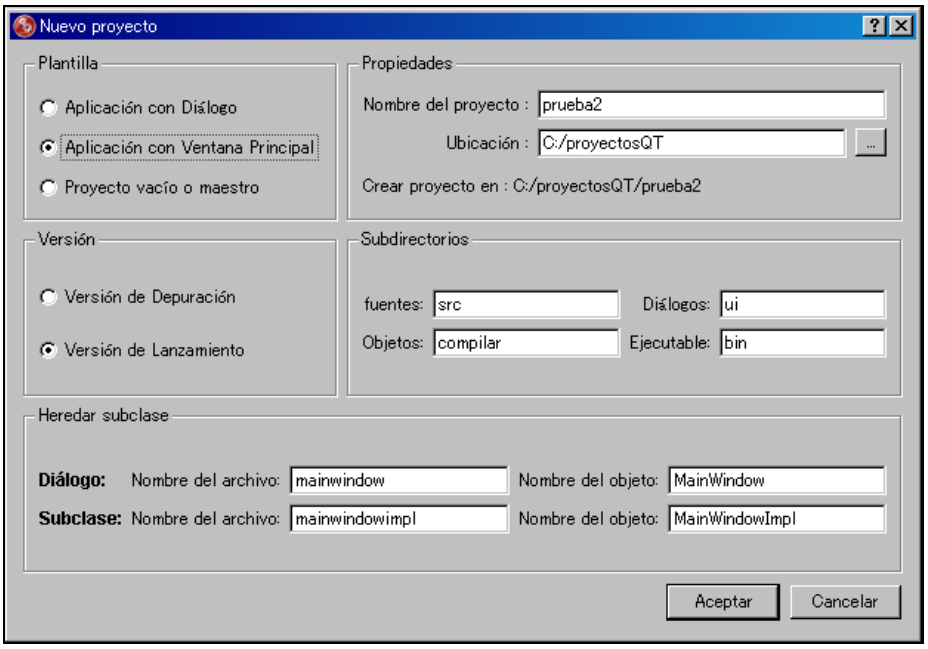
Más información en: http://doc.trolltech.com/4.2/porting4-designer.html

3.8 Capítulo 7: Creación de una aplicación para el Custom Widget

En este momento, ya tenemos el Custom Widget funcionando en QtDesigner, por lo que ahora intentaremos hacer una pequeña aplicación de prueba para mostrar el funcionamiento de este.

El Custom Widget MyCanvasView tiene una serie de Slots definidos que nos permiten dibujar aleatoriamente una serie de formas geométricas (rectángulos, hexágonos, líneas), texto, bitmaps, etc.; por ello la aplicación que crearemos constará de diversos botones que emitirán Signals hacia estos Slots para dibujar una forma u otra.

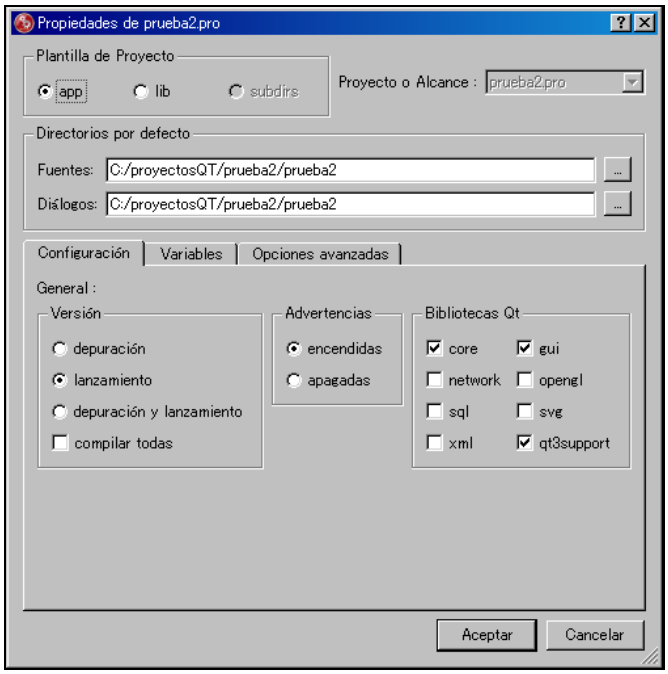
Empezaremos por crear un nuevo proyecto, mediante la plantilla “aplicación con ventana principal”, al que llamaremos, por ejemplo, “prueba2”.



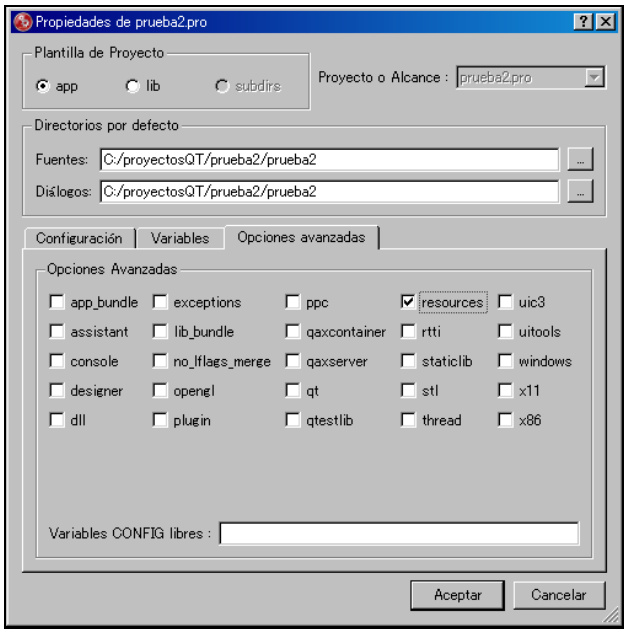
Ventana de nuevo proyecto en QDevelop

A continuación es necesario modificar algunas propiedades del proyecto (haciendo clic con el botón derecho sobre el proyecto y seleccionando la opción Propiedades del Proyecto):

Pestaña de Configuración: en bibliotecas Qt marcaremos “core”, “gui” y “Qt3support”.

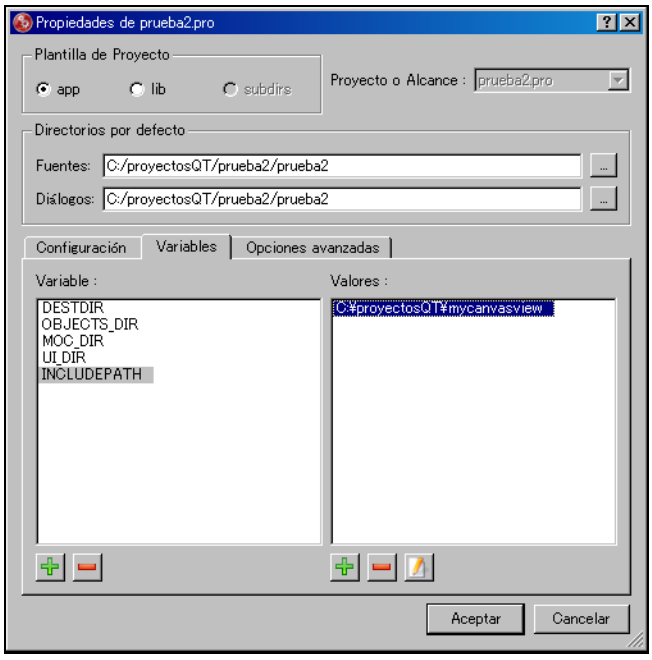


Pestaña de Opciones Avanzadas: marcaremos “recursos”.



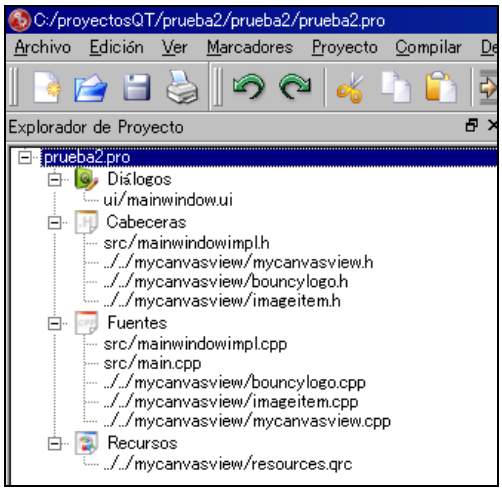
Ventana de propiedades del proyecto, Opciones avanzadas

Pestaña de Variables: añadiremos la variable de entorno “INCLUDEPATH” con el valor “C:\proyectosQt\mycanvasview” (el directorio donde tengamos los archivos del Custom Widget)



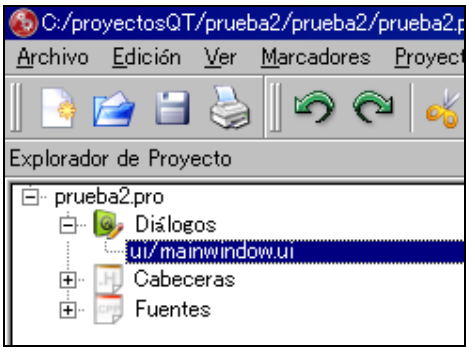
Ventana de propiedades del proyecto, Variables

También es necesario agregar una serie de archivos existentes necesarios en el proyecto, básicamente serán todos los del Custom Widget a excepción del plugin: mycanvasview.cpp/h, imageitem.c/h bouncylogo.c/h y resources.qrc. Ahora, la estructura del proyecto quedará así:



Explorador de Proyecto de QDevelop

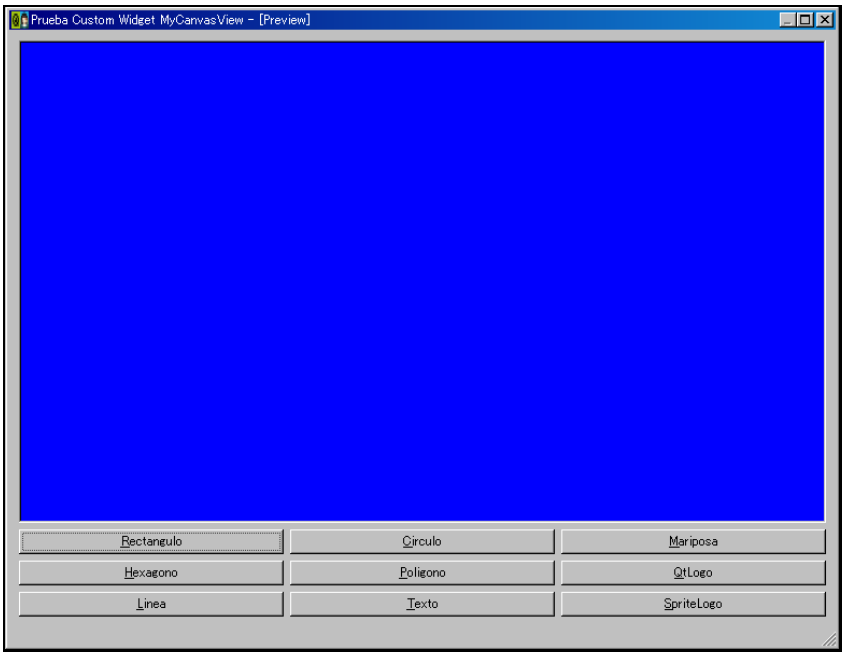
A continuación editaremos el archivo de diálogo (ventana) de la aplicación con QtDesigner, haciendo doble clic sobre el archivo mainwindow.ui del proyecto y abriendo así QtDesigner:



Archivo de diálogo, bajo la categoría Diálogos

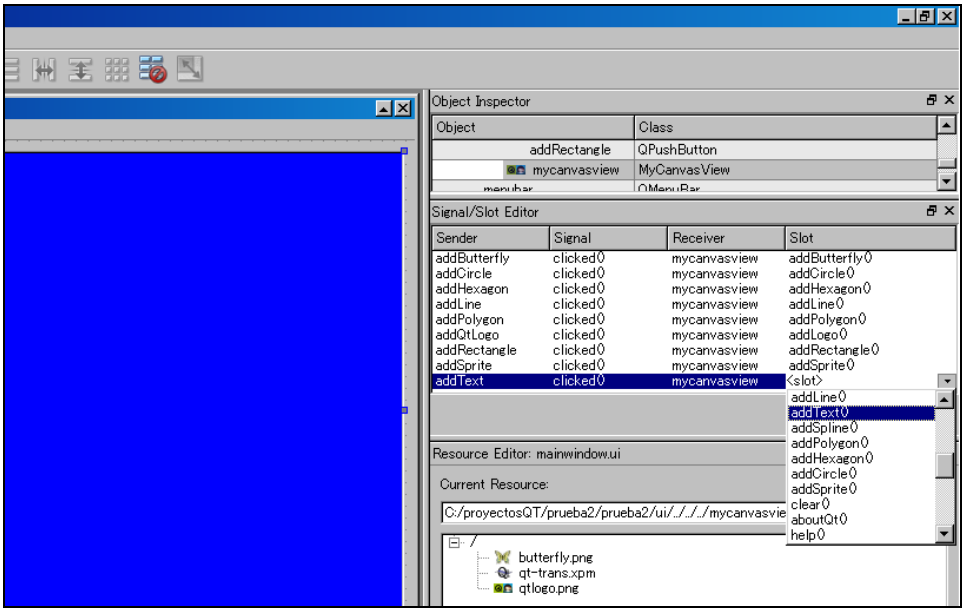
Una vez abierto QtDesigner, arrastramos el Widget MyCanvasView junto con nueve PushButtons hacia nuestra ventana del proyecto, para luego recolocarlos. Seguramente será necesario aplicar una serie de layouts tanto entre botones como en la ventana principal. Para más información sobre layouts se recomienda revisar los anteriores pasos.

Podremos ir modificando el aspecto de la ventana hasta tener algo como lo siguiente:



Aspecto final del diálogo del proyecto

Seguidamente implementaremos conexiones Signal-Slot mediante la barra de herramientas Signal-Slot Editor de QtDesigner, haciendo las asociaciones de la manera siguiente:



Conectando Signals y Slots mediante QDevelop

A continuación guardamos los cambios en la interfaz y cerramos QtDesigner, volviendo a QDevelop. Una vez en QDevelop, solo faltara compilar la aplicación y ejecutarla, para por fin obtener un resultado similar al que muestra la siguiente imagen:

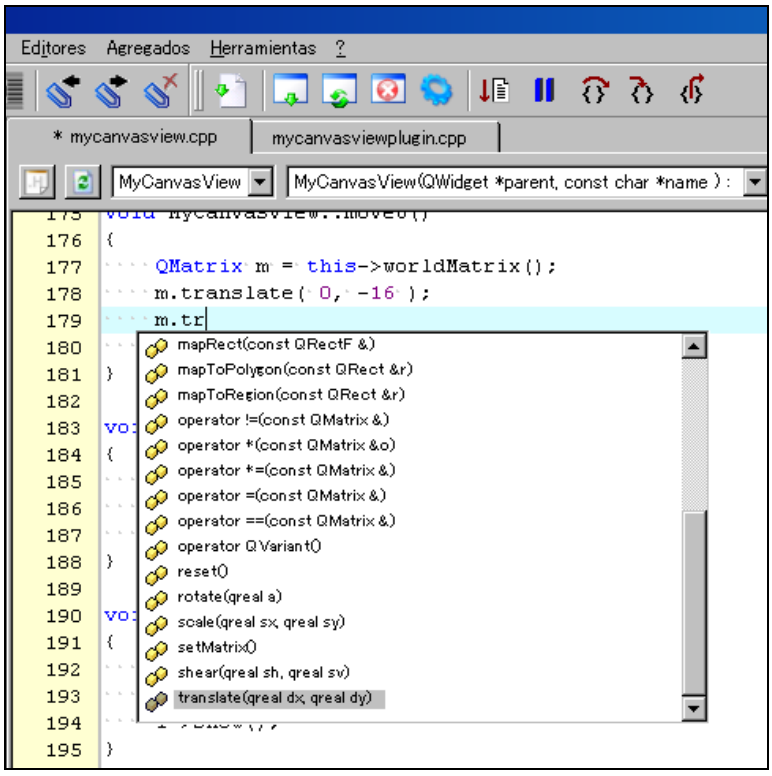
Ventana de la aplicación funcionando correctamente

3.9 Capítulo 8: Herramientas adicionales (Ctags y GDB)

3.9.1 Sugerencia automática con Ctags

No se requiere ningún tipo de configuración especial para usar CTags en QDevelop, simplemente podremos comprobar su funcionamiento al declarar cualquier variable de una clase de Qt e intentando acceder a algún atributo/método propio.

Con ello, automáticamente se nos aparecerá una sugerencia de las posibles opciones que se irá refinando a medida que introduzcamos más caracteres:



Sugerencia automática de texto con QDevelop y Ctags

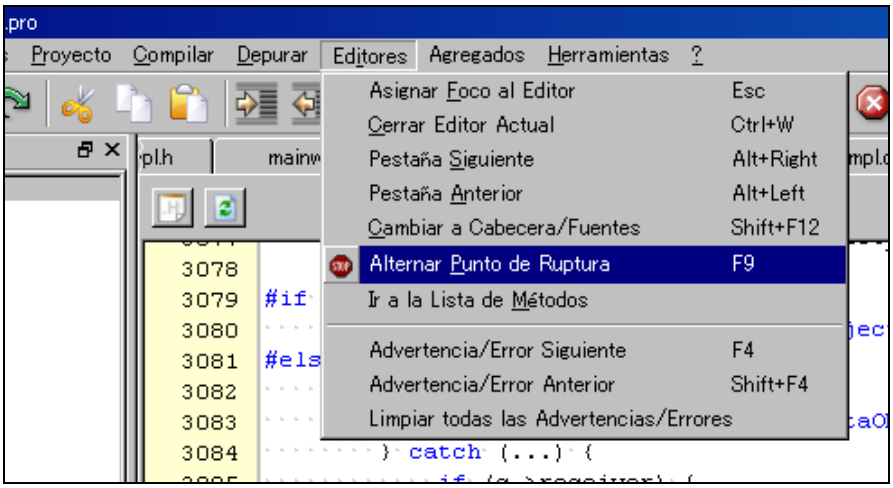
3.9.2 Depuración de programas con GDB

A continuación explicaremos brevemente cómo usar GDB para depurar programas en Qt.

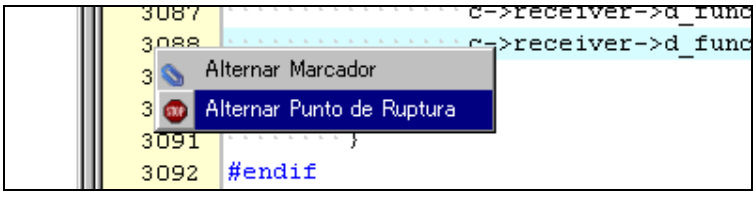
Para poder depurar un programa es necesario que éste se encuentre en modo de depuración. Podemos cambiar el modo/versión del programa mediante la pantalla de propiedades del programa, en el apartado de versión:

Ventana de propiedades del proyecto, configuración

Para poder depurar el programa necesitaremos pararlo en algún punto durante su ejecución, para lo que usaremos *Breakpoints* o "Puntos de ruptura". Para poner un punto de ruptura en una línea, hemos de seleccionar la opción del menú: Editores > Alternar Punto de Ruptura, o bien hacer clic sobre el numero de línea del editor y seleccionar la opción "Alternar Punto de Ruptura".

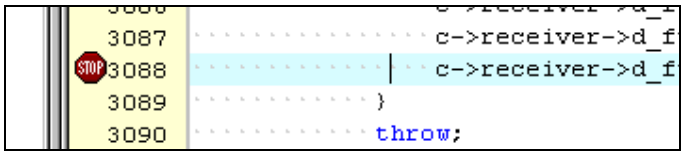


Opción para alternar puntos de ruptura en QDevelop



Hacer clic sobre el número de línea también nos permite alternar un punto de ruptura

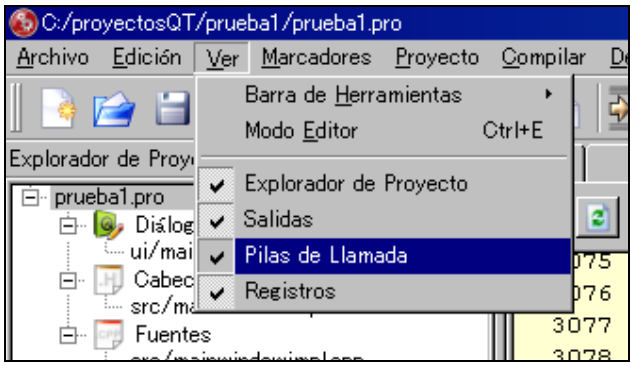
Con ello aparecerá una marca de "stop" en la línea actual, indicando que la ejecución del programa se detendrá en esa línea:



Marca de Stop, indicando que existe un breakpoint activo

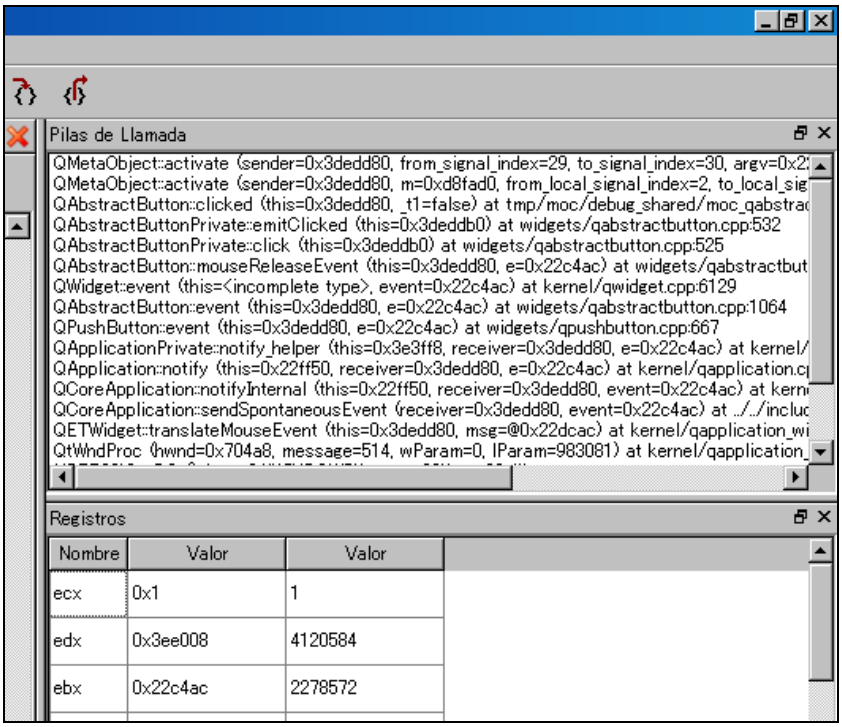
Al iniciar el programa en modo de depuración, la ejecución se detendrá en el Breakpoint.
A partir de aquí, podemos:

- Continuar la ejecución línea a línea pulsando F10.
- Continuar la ejecución normalmente pulsando F6.
- Continuar la ejecución al salir y entrar de funciones (mediante F11/Shift+F11)
- Mostrar la pila de llamadas y el valor de registros. Para ello hay que tener seleccionadas las opciones “Pilas de llamadas” y “Registros” desde el menú Ver:



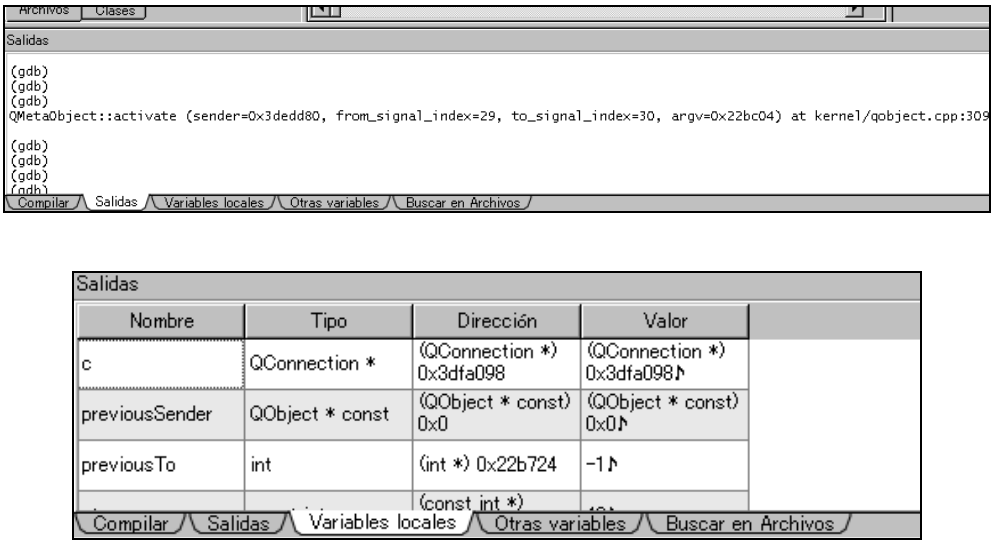
Opción para ver la Pila de Llamadas mediante el menú Ver

La información sobre las llamadas y los registros aparecerá como una barra de herramientas a mano derecha:



Pila de Llamadas y registros en QDevelop

Además, cuando la ejecución se encuentre detenida, ya sea por un Breakpoint o por seguir la ejecución línea a línea, también podremos ver en la parte inferior de QDevelop información proporcionada por el depurador sobre la ejecución y las variables del programa:



Información de depuración en la zona de salidas de QDevelop

4 Especificación

4.1 Requisitos

A continuación vamos a detallar los requisitos funcionales y no funcionales del proyecto, desde el punto de vista de los sistemas basados en la web.

4.1.1 Requisitos funcionales

4.1.1.1 Identificación de usuarios

La aplicación ha de dar soporte a un único tipo de usuario, que se corresponde con un usuario sin experiencia o con nociones muy básicas de Qt, y que busca un manual/tutorial de Qt para poder realizar aplicaciones en Qt de forma sencilla.

4.1.1.2 Casos de uso

Teniendo en cuenta que el sistema es un tutorial on-line, podemos pensar que la posible (y quizás única) interacción por parte de los usuarios sería hacer valoraciones y/o comentarios sobre un determinado capitulo del tutorial. En el siguiente diagrama de casos de uso se muestra de forma concisa la participación de los actores (usuarios) en la aplicación

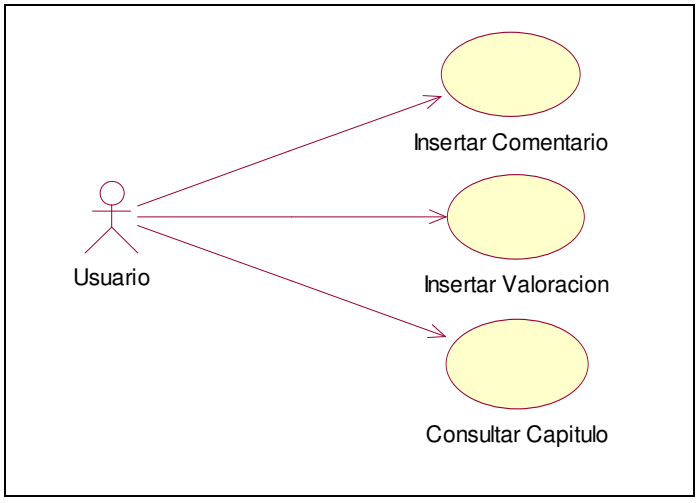


Diagrama básico de Casos de uso

Caso de uso: Consultar capítulo

Usuario	Sistema
1. El usuario pulsa el enlace a un determinado capitulo.	2. El sistema genera la página a la que el usuario quiere acceder

Caso de uso: Insertar comentario

Usuario	Sistema
1. El usuario rellena los campos solicitados y pulsa el botón enviar	2. El sistema procesa el comentario, lo almacena y lo muestra dinámicamente

Flujo Alternativo: en (2):

El sistema detecta algún error en los datos introducidos e informa del error dinámicamente.

Caso de uso: Insertar valoración

Usuario	Sistema
3. El usuario selecciona una determinada valoración.	4. El sistema procesa la valoración, re-calcula la valoración media del capítulo y la muestra dinámicamente.

4.1.2 Requisitos no funcionales

4.1.2.1 Mantenimiento

Una de las características más importantes de una aplicación web es su mantenimiento. El sistema ha de ser fácilmente ampliable y cambiable, no solo a nivel de contenido (capítulos) sino también a nivel gráfico. En el contexto de este proyecto se quiere dejar la aplicación preparada para mantener estas propiedades sin necesidad de efectuar grandes cambios, o sin requerir de cambios de gran complejidad.

4.1.2.2 Usabilidad

Los usuarios de un sistema basado en la web requieren de un sistema usable mediante el que se pueda acceder a la información rápidamente y de la manera más cómoda posible. Para ello, la interfaz del sistema será totalmente visible (sin información oculta) y dinámica, se hará una gestión coherente de los colores y fuentes para mejorar la legibilidad, y se dotará al sistema de una gran sencillez de uso de tal modo que el grado de aprendizaje sea sumamente bajo.

4.1.2.3 Accesibilidad

Otro de los objetivos primordiales de cualquier sistema basado en la Web respecto a otro sistema es su accesibilidad, es decir, que sea compatible con los principales navegadores usados en la actualidad, ya que las estadísticas de uso de navegadores por parte de los

usuarios son más bien dinámicas. En el contexto del proyecto, la aplicación ha de ser perfectamente accesible mediante Mozilla Firefox v2.0 y Internet Explorer v7.0 o superiores.

2008	IE7	Chrome	Firefox	Safari	Opera
Diciembre	26.1%	3.6%	44.4%	2.7%	2.4%
Noviembre	26.6%	3.1%	44.2%	2.7%	2.3%
Octubre	26.9%	3.0%	44.0%	2.8%	2.2%

Estadísticas de usuarios por navegador en el último trimestre de 2008

Fuente: http://www.w3schools.com/browsers/browsers_stats.asp

4.1.2.4 Seguridad

Aunque la seguridad no sea un aspecto clave en un tutorial on-line, dotaremos al sistema de la seguridad suficiente para evitar que un usuario pueda valorar más de una vez el mismo capítulo, evitando así valoraciones totalmente subjetivas y pertenecientes a un único usuario.

4.1.2.5 Rapidez y Estabilidad

El sistema ha de ser todo lo rápido posible, ya que un sistema lento provocaría que posibles visitantes al sistema no se conviertan en usuarios potenciales debido a los tiempos de carga. Por ello, no se utilizará demasiado material grafico, contenidos audiovisuales ni consultas complejas a la Base de Datos que pudieran hacer peligrar la rapidez y estabilidad del sistema.

4.2 Modelo conceptual del sistema

Teniendo en cuenta las características del sistema, el siguiente modelo conceptual sería necesario:

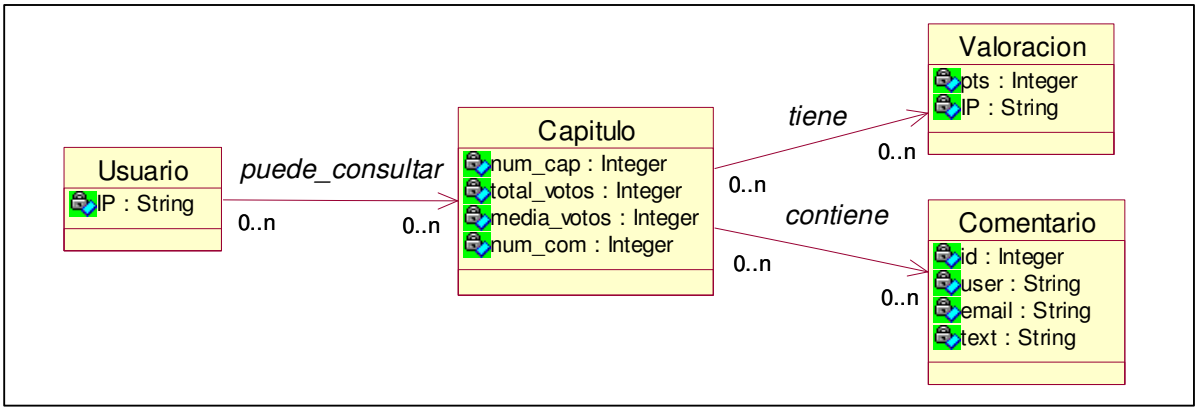


Diagrama de clases del sistema

Podemos diferenciar una única restricción de integridad en el sistema, y sería la siguiente: No puede existir más de una valoración con el mismo atributo IP. Esto es debido a que no queremos que un único usuario pueda realizar un número tan elevado de valoraciones que pueda afectar de manera desproporcional a la valoración media calculada hasta el momento.

5 Diseño

5.1 Patrones de diseño: MVC

La aplicación web en cuestión sólo ha de ofrecer funcionalidades básicas a los visitantes para poder valorar capítulos e insertar comentarios, por ello no es realmente necesario del uso de ningún patrón de diseño. Sin embargo, se ha optado por el uso del patrón de diseño MVC (Modelo – Vista - Controlador), debido a que nos permite estructurar la aplicación de forma más estructurada y facilita la programación en diferentes capas de manera independiente y paralela.

5.2 Modelo

El modelo se encarga de todo lo que tiene que ver con la persistencia de datos. Guarda y recupera la información del medio persistente que utilicemos, ya sea una base de datos, ficheros de texto, XML, etc…

5.3 Vista

Las vistas son las representaciones del modelo en forma grafica disponible para la interacción del usuario. En nuestro sistema tendremos, en primer lugar, una página principal o de inicio, que constituirá el punto de entrada en el sistema, y a partir de la cual, mediante enlaces, podremos acceder a las vistas generadas de forma dinámica.

5.4 Controladores

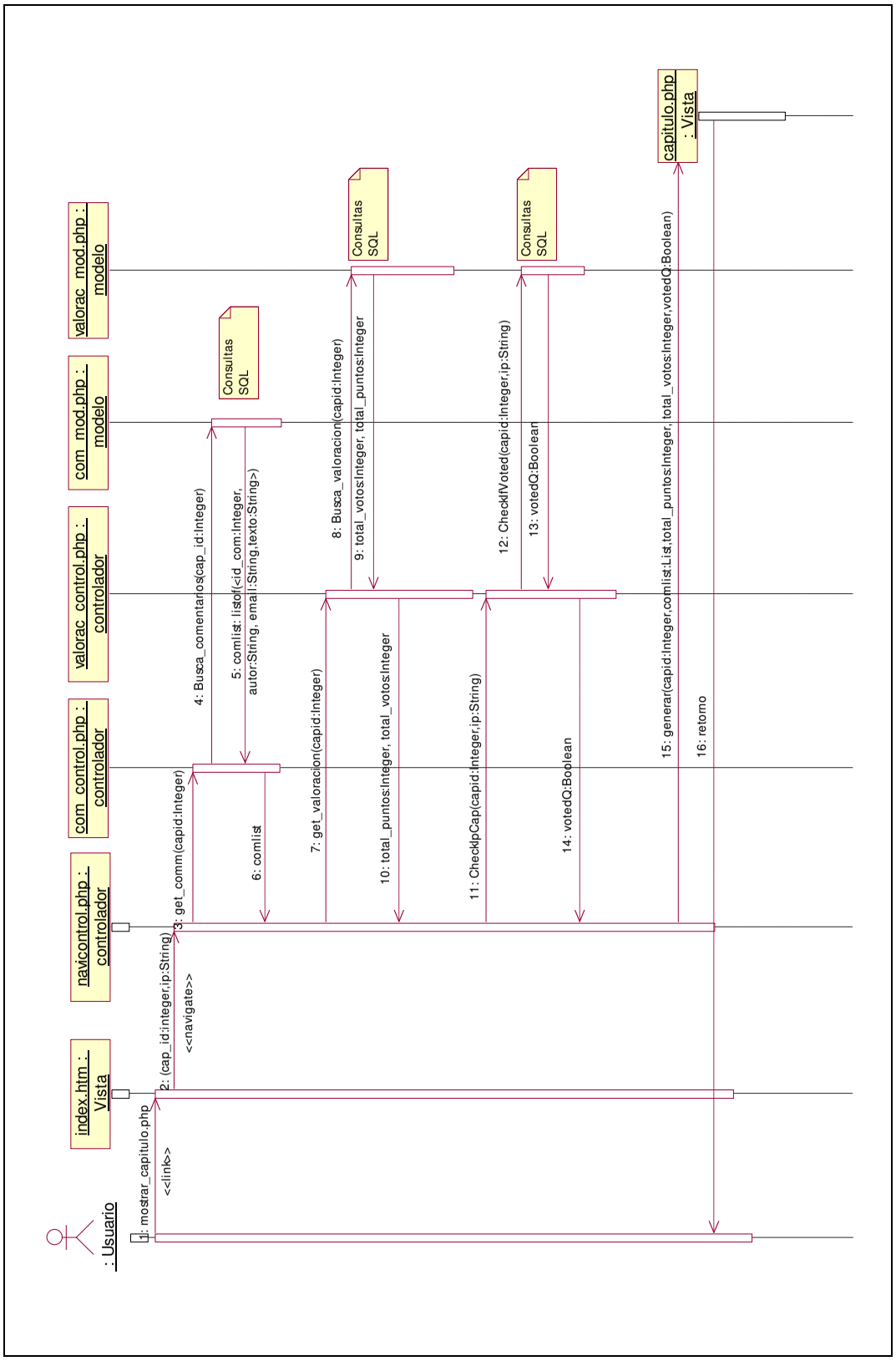
Los controladores se encargan de manejar y responder las peticiones del usuario, procesando y accediendo a la información del modelo. Para ello, piden al modelo la información necesaria e invocan/generan la vista que corresponda para que la información sea presentada.

En el sistema actual tendremos 3 controladores, el controlador de **comentarios** (responderá a las peticiones relacionadas con los comentarios), el de **valoraciones** (responderá a las peticiones relacionadas con las valoraciones) y finalmente el de **navegación** (responderá las peticiones de navegación del cliente y generará las paginas dinámicas, comunicándose con los anteriores controladores para obtener la información correspondiente)

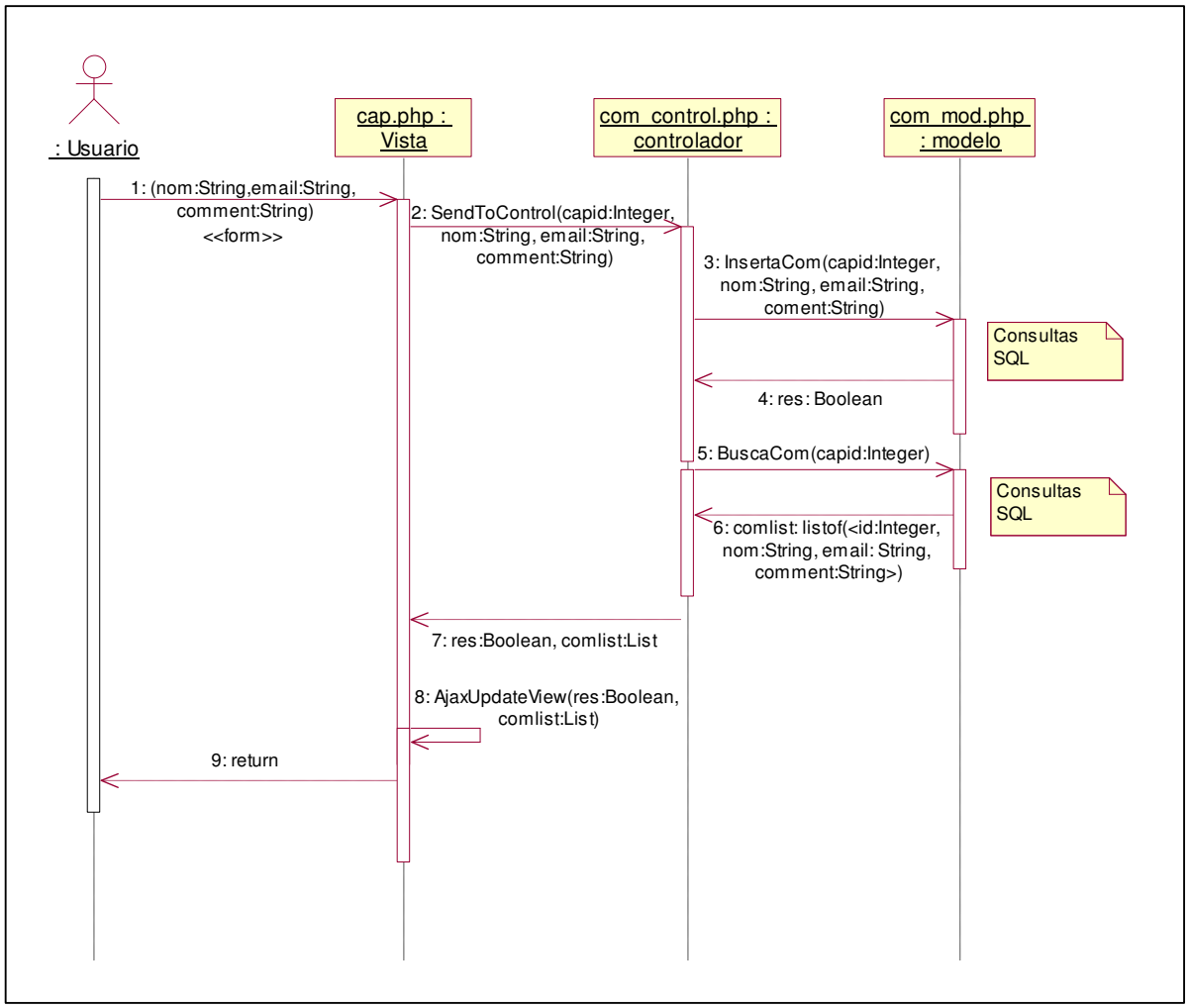
5.5 Diagramas de secuencia

A continuación se describe a modo informativo y mediante diagramas de secuencia, la interacción para cada caso de uso, teniendo en cuenta el patrón MVC:

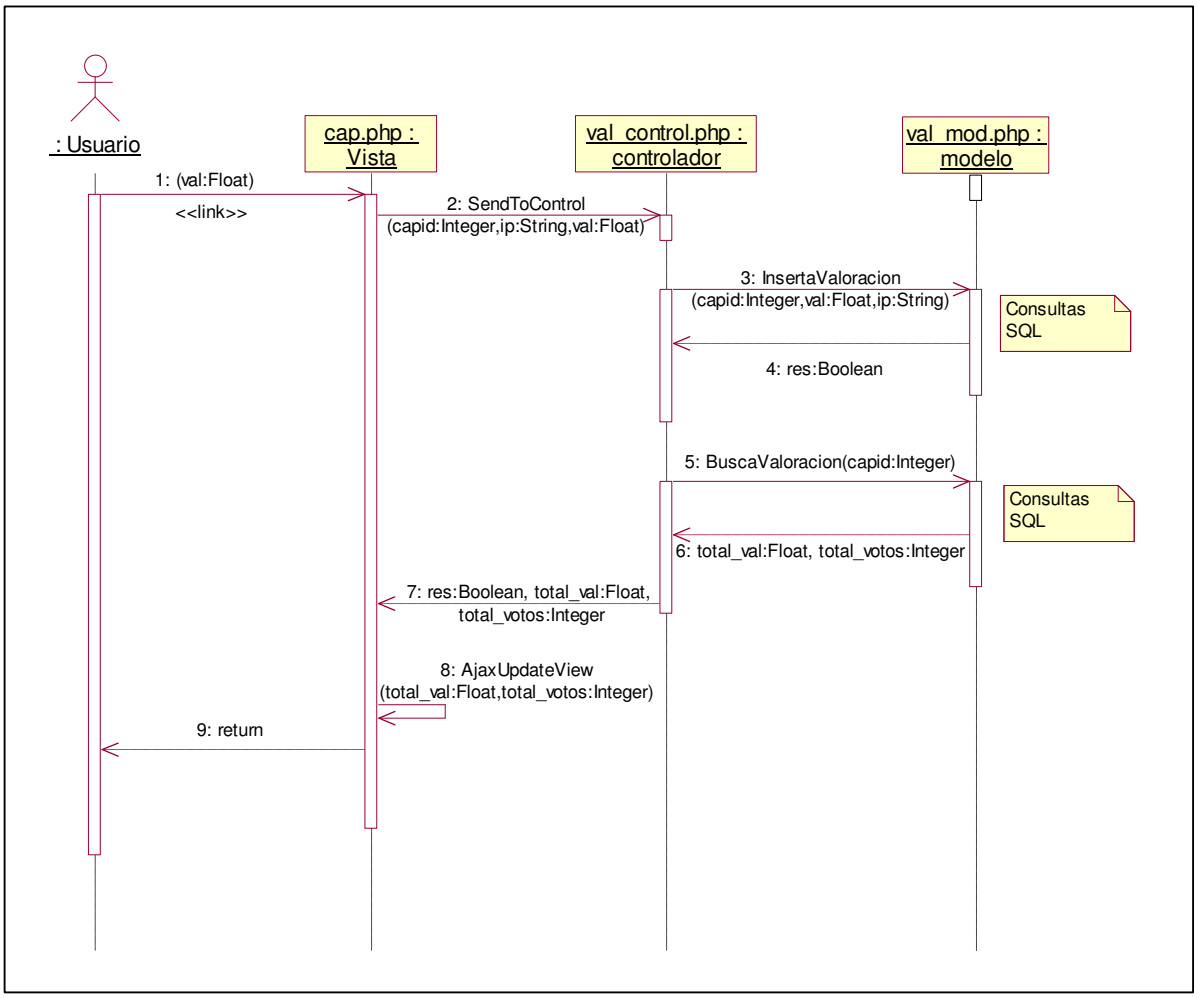
5.5.1 Consultar capitulo



5.5.2 Insertar comentario



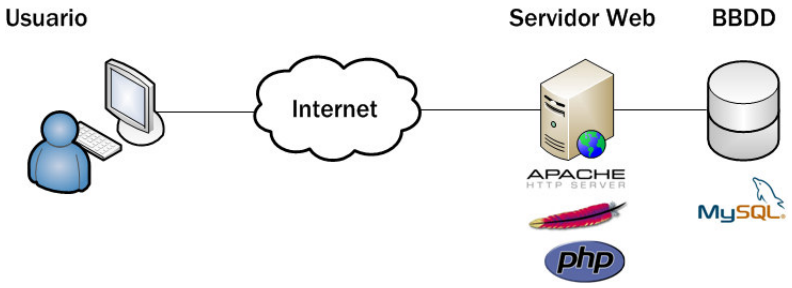
5.5.3 Insertar valoración



6 Desarrollo de la aplicación

6.1 Esquema general de la aplicación

La aplicación sigue un esquema básico de desarrollo de aplicaciones web, en la que un usuario se conecta a un servidor web y realiza peticiones que desencadenarán accesos a la BBDD. Las tecnologías que se han utilizado en la parte del servidor son Apache, PHP y MySQL.



6.2 Lenguajes y tecnologías utilizadas

Para el desarrollo de la aplicación web se ha optado por diferentes tecnologías / lenguajes:

- HTML: utilizado para el desarrollo de páginas web.
- CSS: utilizado para configurar los estilos gráficos comunes de las páginas web.
- Ajax: utilizado para realizar eventos javaScript asíncronos, dando lugar a paginas dinámicas sin necesidad de refrescar toda la pagina. Cabe destacar el uso conjunto con *Script.aculo.us*, un Framework ajax utilizado para realizar efectos gráficos (*Fade, Blur, Appear,* etc...)
- PHP: utilizado para la lógica de la aplicación, nos permite generar el contenido de las páginas web.
- SQL: utilizado para acceder a la información almacenada en la Base de Datos.

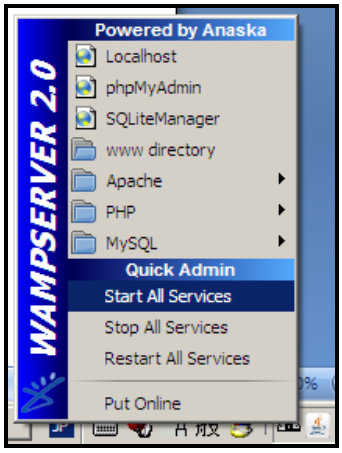
6.3 Instalación integrada del servidor y la BBDD

Para la instalación del sistema web se ha optado por software libre y gratuito. En este proyecto hemos decidido utilizar un servidor WAMP (Windows + Apache + MySQL + PHP) totalmente integrado, disponible en: http://www.wampserver.com. El paquete contiene el siguiente software:

- Apache 2.2.11
- MySQL 5.1.30
- PHP 5.2.8

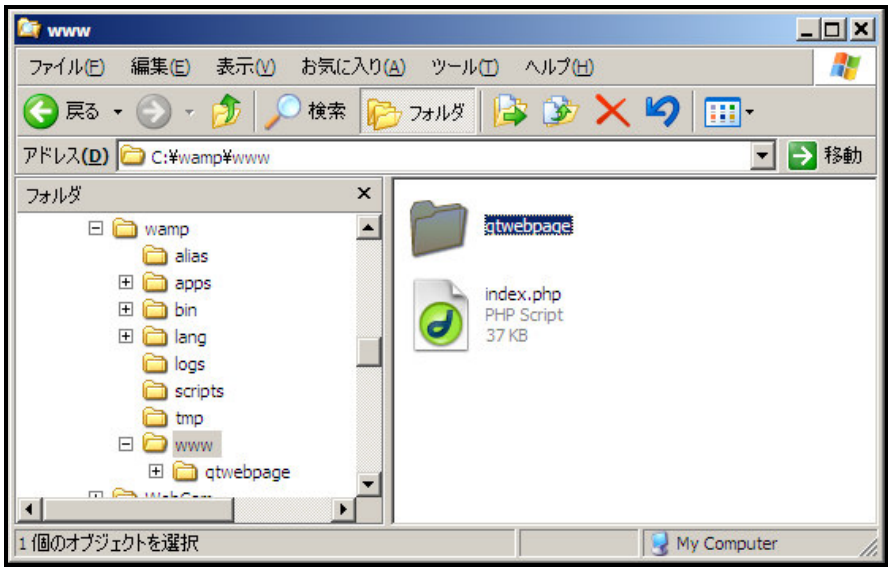
Página principal del sitio de descarga de wamp (http://www.wampserver.com)

La instalación del servidor no requiere de más pasos que la instalación guiada mediante el asistente del ejecutable. Una vez instalado, hemos de configurar el puerto de peticiones para el servidor (por defecto será el puerto 80, configurable en el fichero *httpd.conf*), y arrancar todos los servicios seleccionando la opción "*start all services*":



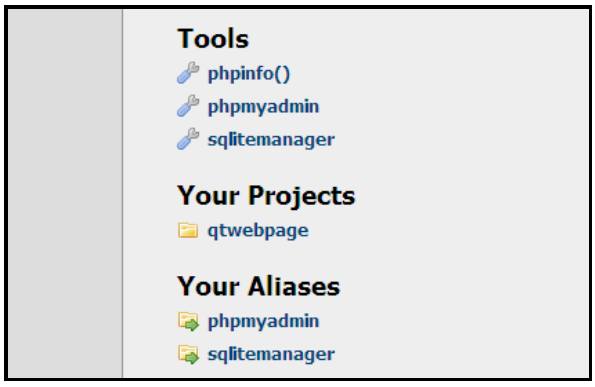
Menu para el control de los servicios del servidor

Para instalar el tutorial, copiaremos la carpeta del CD adjunto "Qtwebpage" a la carpeta "www" del directorio de instalación del servidor (por defecto será x:/wamp):



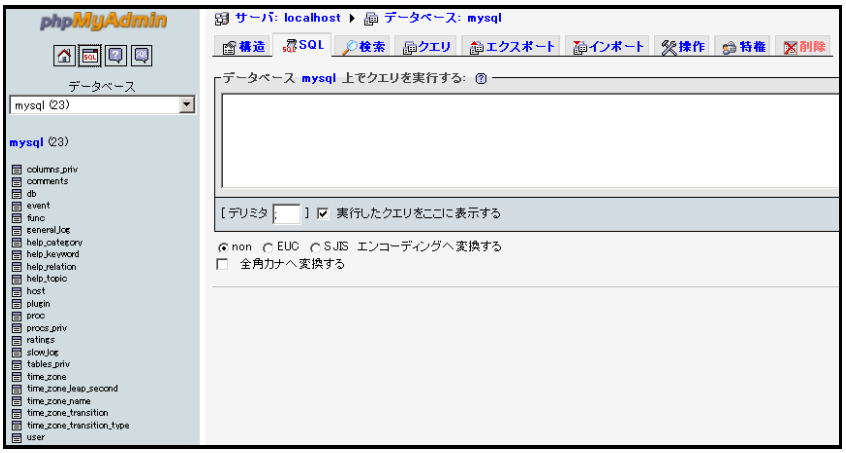
Estructura de directorios del servidor

A continuación, accediendo a la dirección http://localhost:80 (substituyendo el 80 por el puerto al que hayamos configurado para el servidor) podremos acceder a la página de inicio del servidor. Dentro de esta página, podemos entrar en el tutorial mediante el enlace a Qtwebpage (la carpeta que contiene el tutorial) desde la zona "your projects", o bien accediendo con la dirección http://localhost:80/Qtwebpage/.



Estructura de proyectos y enlace al tutorial

Sin embargo, antes debemos crear las tablas necesarias para el correcto funcionamiento del sistema de valoración y de comentarios. Para ello, dentro de la página principal, accederemos a phpmyadmin dentro de la zona "your aliases". Una vez direccionados, haremos clic sobre la base de datos (mysql) y crearemos las tablas necesarias mediante las siguientes sentencias SQL (sentencias incluidas también en el fichero dbconfig_readme.txt en la carpeta del tutorial):



Panel de control de phpMyAdmin, desde donde ejecutaremos sentencias SQL

```
CREATE TABLE `ratings` (
`id` varchar(11) NOT NULL,
`total_votes` int(11) NOT NULL default 0,
`total_value` int(11) NOT NULL default 0,
`used_ips` longtext,
PRIMARY KEY (`id`) )
```

```
CREATE TABLE `comments` (
`id` int(200) NOT NULL auto_increment,
`capid` int(200) NOT NULL,
`name` varchar(200) NOT NULL,
`email` varchar(200) NOT NULL,
`comment` varchar(200) NOT NULL,
PRIMARY KEY (`id`))
```

Sentencias SQL para crear las tablas que necesitamos

A continuación ya podremos acceder e interactuar con el tutorial sin problemas.

7 Análisis

7.1 Interfaz gráfica

Para el diseño de la interfaz del sistema, se ha optado por una de combinación de colores coherente y unificada en todas las páginas, donde predomina el verde como referencia al color de Qt. Las páginas se caracterizan por un estilo Web 2.0 y muestra fuentes con tamaño superior a 10px a fin de facilitar la lectura y evitar el cansancio en el lector.

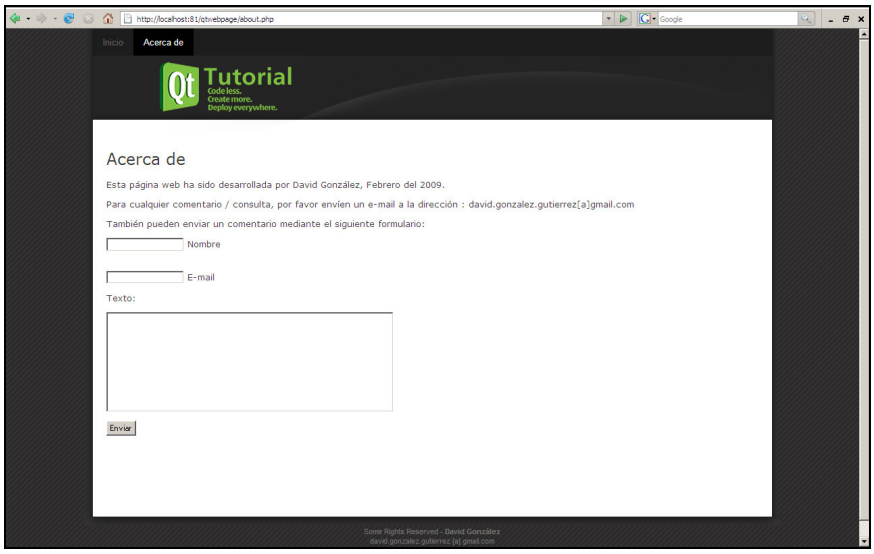
7.1.1 Página principal

Ésta es la página de entrada al sistema. En primer lugar, y en la parte superior, podemos diferenciar enlaces para poder acceder a la página de crédito del autor. Seguidamente, a mano izquierda podemos apreciar los diferentes capítulos de los que se compone el tutorial. Finalmente, a mano derecha y en la parte superior tenemos una breve introducción a Qt, mientras que en la parte inferior observamos enlaces relacionados con Qt.

Página principal del sistema

7.1.1.1 Página de contacto

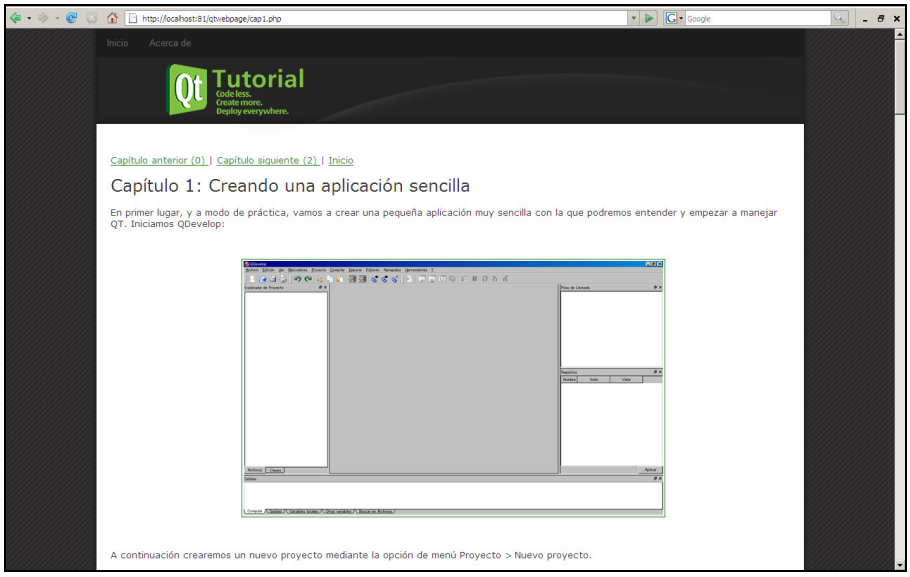
En la página de contacto podemos ver información relacionada con el autor del tutorial, y un formulario para ponernos en contacto con el autor para cualquier sugerencia, crítica, etc…



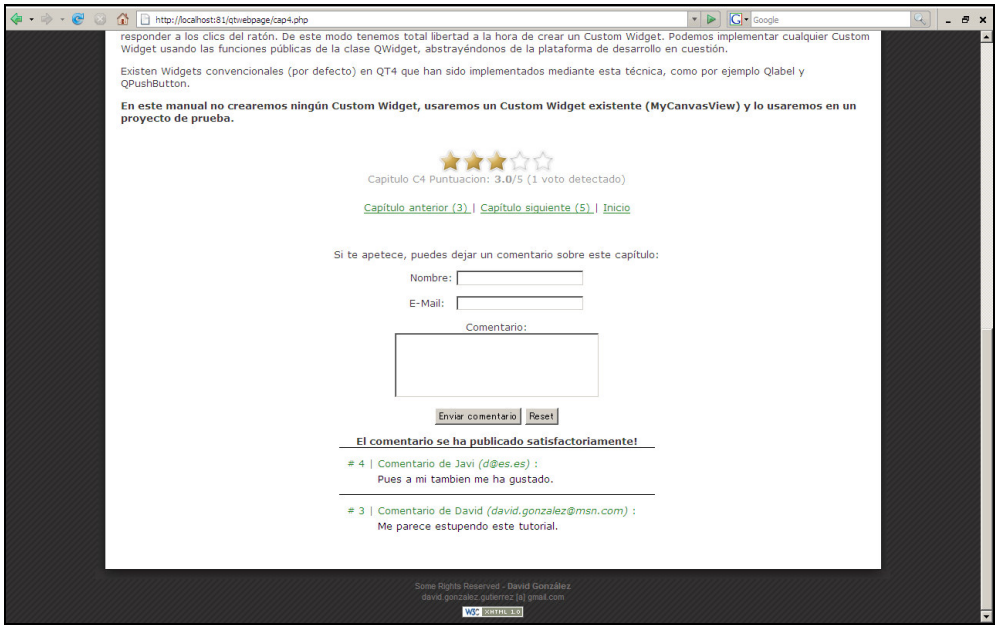
Página de contacto

7.1.1.2 Página de capítulo

Las páginas de capitulo se asemejan bastante a las de contacto, a excepción de tener una franja de navegación superior e inferior (con enlaces a los capítulos anterior, siguiente e índice), y una zona de valoración y comentarios, cuyo contenido será proporcionado por los visitantes.



Página de capítulo (parte superior)



Página de capítulo (parte inferior)

8 Planificación y estudio económico

8.1 Planificación temporal

En la siguiente imagen podemos apreciar la distribución del tiempo para el proyecto según cada una de las fases que lo integran:

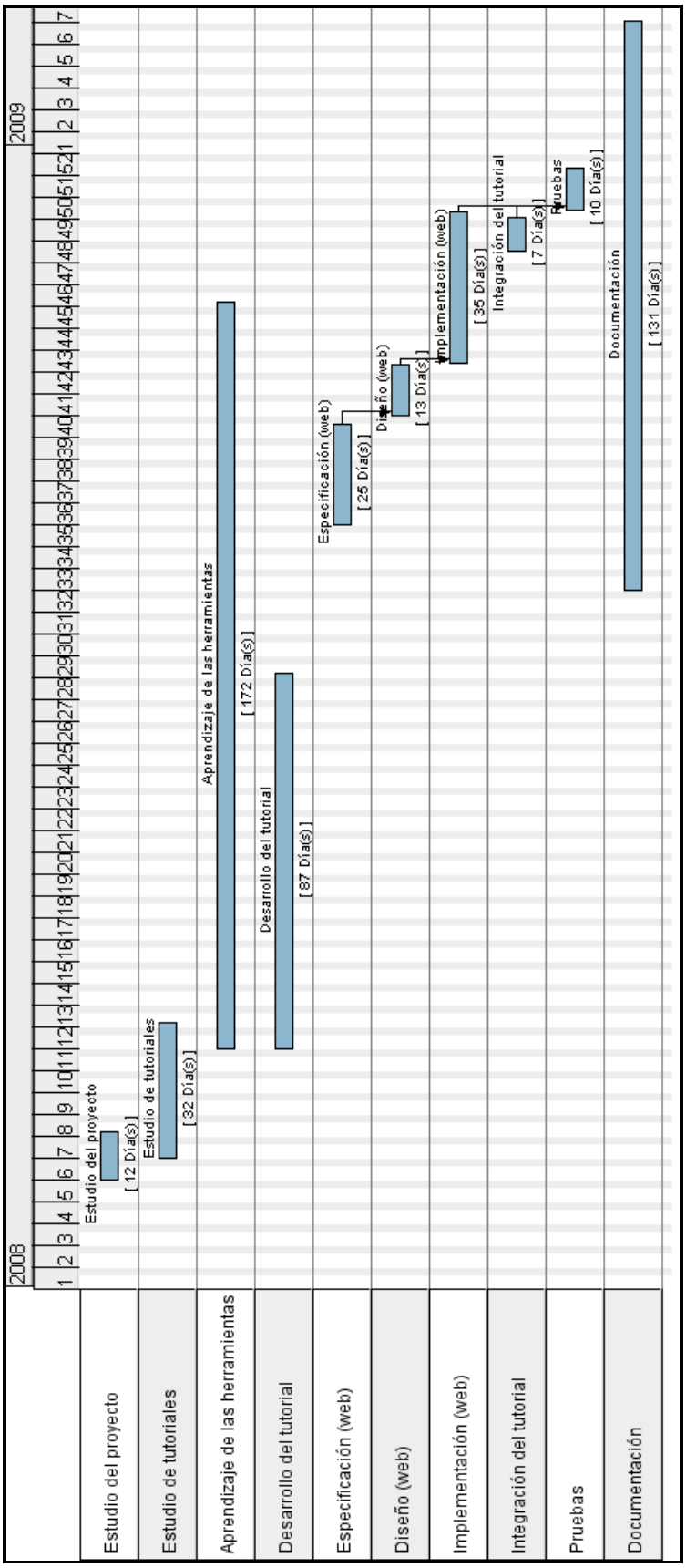
GANTT project

Nombre	Fecha de inicio	Fecha de fin	Duración
Estudio del proyecto	4/02/08	20/02/08	12
Estudio de tutoriales	11/02/08	26/03/08	32
Aprendizaje de las herramientas	17/03/08	12/11/08	172
Desarrollo del tutorial	17/03/08	16/07/08	87
Especificación (web)	1/09/08	4/10/08	25
Diseño (web)	6/10/08	23/10/08	13
Implementación (web)	23/10/08	11/12/08	35
Integración del tutorial	28/11/08	9/12/08	7
Pruebas	11/12/08	25/12/08	10
Documentación	11/08/08	10/02/09	131

Página de capítulo (parte inferior)

Cabe destacar que la prácticamente totalidad del proyecto se ha realizado paralelamente a la realización de prácticas laborales mediante una beca de movilidad en Japón (conocida como beca *Vulcanus*), por lo que el tiempo del proyecto se ha tenido que coordinar y distribuir con dichas prácticas en la empresa, dando lugar a una planificación más extensa de lo habitual.

A continuación se muestra el diagrama de Gantt, con el objetivo de ordenar gráficamente el origen y el final de las diferentes fases del proyecto, así como las dependencias y solapamientos entre ellas.


2008
2009
Estudio del proyecto
[12 Día(s)]
Estudio de tutoriales
[32 Día(s)]
Aprendizaje de las herramientas
[172 Día(s)]
Desarrollo del tutorial
[87 Día(s)]
Especificación (web)
[25 Día(s)]
Diseño (web)
[13 Día(s)]
Implementación (web)
[35 Día(s)]
Integración del tutorial
[7 Día(s)]
Pruebas
[10 Día(s)]
Documentación
[131 Día(s)]

8.2 Estudio económico

El coste total para el desarrollo del proyecto se puede desglosar de la manera siguiente:

Hardware	Precio estimado €
PC (desarrollo y edición) Sony VAIO VGN-FJ91PS	800
Contratación de Hosting 1 año (soporte a PHP y BBDD MySQL)	2Ex12 meses : 24E (http://www.webhostingpad.com)

Software	Versión	Licencia	Precio estimado €
Microsoft Windows XP	ProfesionalSP3	Estudiante	0
OpenOffice.org	3.0	LGPL	0
ArgoUML	0.26.2	BSD	0
WAMP Server	2.0f	BSD	0
Adobe Fireworks	CS3	Usuario	319
Super Edi (edición HTML)	4.2	Freeware	0
GANTT Project	2.0.9	GPL	0
Camtasia Studio	5.0	Usuario	299

Trabajo	Rol	Precio/hora	Horas	Coste total €
Implementación y pruebas	Programador PHP	10	60	600

Coste total estimado: **2042 EUR**

9 Conclusiones y líneas de futuro

Este tutorial ha evolucionado hasta convertirse en una pequeña aplicación web, con la que los usuarios que quieran iniciarse en Qt y QDevelop encontrarán una guía bastante detallada sobre el uso de éstos. De este modo, podemos verificar que el objetivo principal del proyecto se ha cumplido con éxito.

También se puede asegurar que se han cumplido todos los requisitos, tanto funcionales como no funcionales presentados durante la especificación del proyecto, por lo tanto podemos afirmar que se han cumplido todos los objetivos acordados desde el inicio del proyecto.

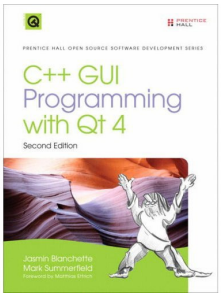
A nivel de planificación y costes, también podemos comprobar que se ha cumplido la planificación esperada bajo el presupuesto acordado sin ningún tipo de problemas.

Finalmente, y como posible trabajo de futuro, podemos diferenciar dos posibles líneas de trabajo. La primera, y a nivel de contenidos del tutorial, sería la de añadir más capítulos, haciendo referencia al uso e instalación de otros IDEs compatibles con Qt, así como de más tutoriales para la creación de aplicaciones más avanzadas. La segunda posible línea de trabajo estaría más enfocada a la aplicación web: pensamos que se podría implementar funciones de sesión para usuarios, a efectos de mejorar la seguridad en el sistema, junto con la integración de algún tipo de foro donde los usuarios puedan crear temas de consulta y opiniones a debatir.

10 Bibliografía

A continuación se detallan las fuentes bibliográficas que se han utilizado en el desarrollo del proyecto.

10.1 Bibliografía impresa



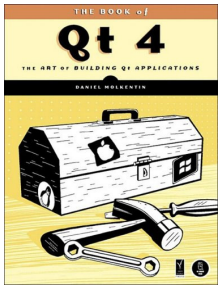
C++ GUI Programming with Qt 4

Autores: Jasmin Blanchette, Mark Summerfield

Prentice Hall, 2008

ISBN-10: 0132354160

ISBN-13: 978-0132354165



The Book of Qt4 Programming

Autores: Daniel Molkentin

No starch Press, 2007

ISBN-10: 1593271476

ISBN-13: 978-1593271473

10.2 Bibliografía digital

Clive Cooper: How To Create a Linux Desktop App In 14 Minutes For Beginners.

http://www.clivecooper.co.uk/tutorial/

Jean Pierre Charalambos: Tutorial de Qt4 Designer.

http://www.lsi.upc.es/~charalam/taller/index.html

Sitio oficial de Qt y Trolltech

http://www.Qtsoftware.com

Online Reference Documentation (Trolltech)

http://doc.trolltech.com/

Bibliotecnica UPC

http://bibliotecnica.upc.edu/

Cross Platform Qt4 Ides

http://www.thelins.se/johan/2006/10/cross-platform-Qt-4-ides.html

GNU operative system

http://www.gnu.org/

Qt Integrated Development Environments

http://www.thelins.se/index.php?title=Qt_IDEs

Enciclopedia en línea

http://es.wikipedia.org

Páginas oficiales de IDEs

http://qdevelop.org/

http://eclipse.org

http://www.codeblocks.org

http://www.beesoft.org/cobras.html

http://devQt.berlios.de/ , http://edyuk.berlios.de/

Buscadores

http://www.google.es

http://www.yahoo.es

Foro de programación de la comunidad QtCentre

http://www.Qtcentre.org/forum/f-Qt-programming-2.html

Principios generales de usabilidad en el web

http://www.desarrolloweb.com/articulos/1133.php

Diseñando aplicaciones con ArgoUML

http://trevinca.ei.uvigo.es/~jgarcia/TO/usoArgoUML/index.html

Principles of Good GUI design

http://www.iie.org.mx/Monitor/v01n03/ar_ihc2.htm

Referencia en línea de las bibliotecas Qt y KDE

http://docs.kde.org/kde3/es/kdevelop/kde_app_devel/chapter2.html

Licencia GPL vs LGPV

http://www.gnu.org/licenses/why-not-lgpl.es.html

11 Glosario

A continuación se recoge el significado de algunos términos que se han utilizado, y que se cree que puedan requerir de una breve explicación para mejorar su entendimiento.

GUI: Interfaz Gráfica de Usuario (*Graphical User Interface*) es un tipo de interfaz de usuario que utiliza un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz.

API: interfaz de programación de aplicaciones (*Application Programming Interface*), conjunto de funciones y procedimientos /métodos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

IDE: entorno de desarrollo integrado (Integrated Development Environment), programa compuesto por un conjunto de herramientas para un programador.

CSS (HTML): hojas de estilo en cascada (*Cascading Style Sheets*), lenguaje formal usado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML).

Web 2.0: segunda generación en la historia de la web basada en comunidades de usuarios, que fomentan la colaboración y el intercambio ágil de información.

Framework: estructuras de soporte definidas con la que otro proyecto de software puede ser organizado y desarrollado. Normalmente, incluye soporte de programas, bibliotecas y un lenguaje interpretado para ayudar a desarrollar y unir componentes de un proyecto.

Open Source / **Software Libre:** software de código abierto, que permite a los usuarios de éste, una vez obtenido, de usarlo, copiarlo, modificarlo y redistribuirlo libremente. Suele ser gratuito, aunque no siempre es así. Existen multitud de licencias: MSL, BSD, GPL, LPGL…

GNU: Siglas de *GNU is not Unix*, se trata de un proyecto iniciado por Richard Stallman con el objetivo de crear un sistema operativo completamente libre. Actualmente se utiliza el término GNU/Linux para referirnos a un sistema GNU con núcleo Linux (núcleo GNU no finalizado)

GPL: Licencia pública general (*General Public License*), creada por la Free Software Foundation en los años 80. Declara que el software cubierto por esta licencia es software libre y su autor conserva los derechos de copyright. Posibilita la modificación y redistribución bajo la misma licencia.

LGPL: Licencia pública general reducida (*Lesser General Public License*). Declara que el software cubierto por esta licencia es software libre, y se diferencia de la licencia GPL en que un software LGPL / biblioteca LPGL puede enlazarse con / ser utilizada por un programa no-GPL (que puede ser software libre o software no libre).

Plugin: aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy especifica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la API.

Parser: analizador sintáctico, constituye una de las partes de un compilador que convierte un texto de entrada en una representación basada en estructuras de datos jerárquicas.

Depurar/Depuración: proceso de identificar y corregir errores de programación. En inglés se le conoce como *debugging*, ya que se asemeja a la eliminación de bichos (*bugs*), manera en que se conoce informalmente a los errores de programación.