Procesamiento de Imágenes
Máster NTI

Guión de prácticas

Sesión 1. Programación visual con Qt Creator

Descripción
Instalación
El entorno Qt
Primer programa
Estructura
¡Hola Mundo!
Contador
Depuración
Añadir icono
Formularios
Distribución
Seguir explorando

DESCRIPCIÓN

- **Qt Creator** es un Entorno Integrado de Desarrollo o IDE (editor + compilador + depurador) bastante completo, moderno, potente, fácil de manejar, eficiente, abierto y gratuito, que permite el desarrollo rápido de aplicaciones en entornos MS Windows, Mac OS y Linux. Algunos ejemplos de programas creados con las librerías Qt: Adobe Photoshop Album, Google Earth, KDE, Opera, Skype, VLC media player… Nosotros usaremos **Qt Creator 2.7.2** de julio de 2013.

- Características fundamentales de **Qt Creator**:
  - Utiliza el lenguaje de programación orientado a objetos C++.
  - Se basa en Qt, una librería multiplataforma y gratuita para la creación de interfaces gráficos, programación web, multihilo, bases de datos, etc.
  - Permite realizar programación visual y programación dirigida por eventos.
  - Características avanzadas de IDE: sintaxis coloreada, compleción automática de código, ayuda sensible al contexto, inspector de objetos, diseñador visual, compilador y depurador integrados, etc.

- **Programación visual**: el programador centra su atención en diseñar el aspecto gráfico de la aplicación, la distribución de los elementos visuales (llamados ***widgets***: formularios, botones, menús, cuadros de texto, etc.), la interacción entre los mismos, los distintos tipos de ventanas existentes, etc.
  - Un entorno de programación visual se asemeja a un programa de dibujo, donde la imagen es una **ventana** (o **formulario**), y los elementos para dibujar son botones, etiquetas de texto, menús, etc.
  - El programador diseña el aspecto gráfico que tendrá la aplicación (*WYSIWYG*, *What You See Is What You Get*).

- **Programación dirigida por eventos**: el programador escribe el código que se ejecutará en respuesta a determinados eventos (llamados ***slots***: pulsar un botón, elegir una opción del menú, abrir o cerrar una ventana, etc.).
  - No existe la idea de un control de flujo *secuencial* en el programa, sino que el programador toma el control cuando se dispara un evento.
  - La labor del programador es asociar a cada evento el comportamiento adecuado.

- Las ventanas son clases, los componentes (*widgets*) son clases, y los eventos (*slots*) son métodos de las ventanas. Nuestra ventana es una subclase de la clase ventana (QMainWindow, QDialog o QWidget).

- Componentes del entorno:
  - **Qt Creator** es el entorno de desarrollo. Usamos la versión 2.7.2.
  - Se basa en la **librería Qt**, un conjunto de funciones para la creación de entornos gráficos multiplataforma. En concreto, nuestra versión de Qt Creator usa la versión 5.1.0 de Qt.
  - Puesto que **Qt** es solo una librería, **Qt Creator** necesita el compilador **MinGW** (http://sourceforge.net/projects/mingw), que es un compilador GCC para Windows. Qt Creator 2.7.2 usa la última versión disponible, que era la MinGW 4.8.1 (se instala con Qt Creator).

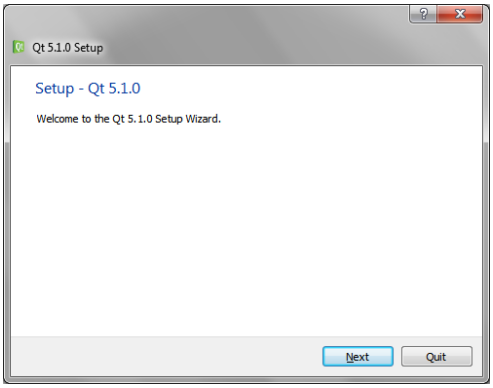
INSTALACIÓN DE QT CREATOR 2.7.2

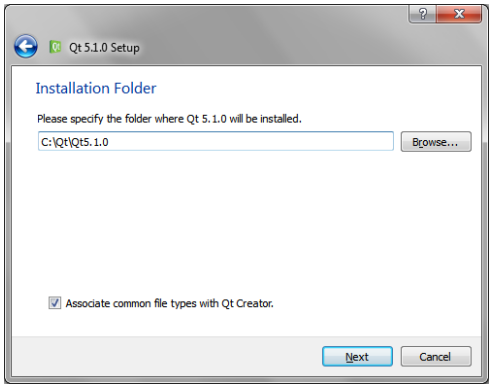
1. **Descargar** "Qt 5.1.0 for Windows 32-bit (MinGW 4.8, OpenGL, 666 MB)" desde:

   http://qt-project.org/downloads

   (Es posible que en este momento exista alguna versión más reciente de Qt Creator, puesto que salen revisiones cada cierto tiempo. Por compatibilidad, recomendamos que para las prácticas se use la versión 2.7.2.)

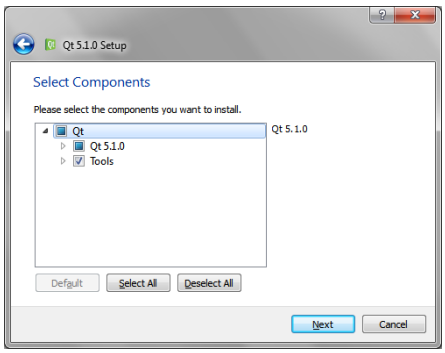
2. **Ejecutar** el fichero: qt-windows-opensource-5.1.0-mingw48_opengl-x86-offline.exe



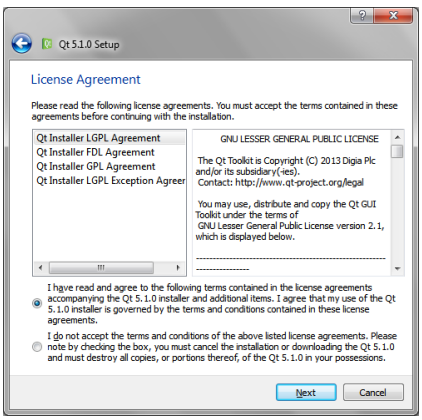


Instalamos en el directorio por defecto: C:\Qt\Qt5.1.0 y asociamos las extensiones de Qt.

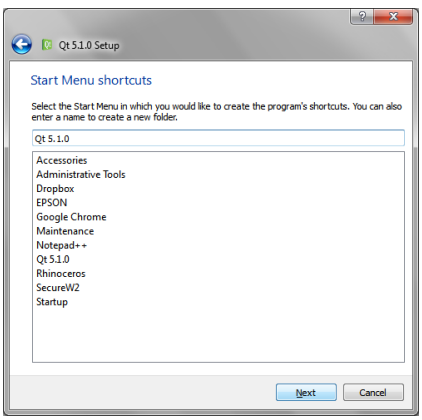
3. **IMPORTANTE:** si se instala en Windows XP, es recomendable que esté actualizado con el SP3. En un Windows XP sin SP3 puede instalarse pero *podría* dar problemas al ejecutarse… Tampoco funciona en Windows 2000.



Dejamos que se instalen los componentes por defecto.



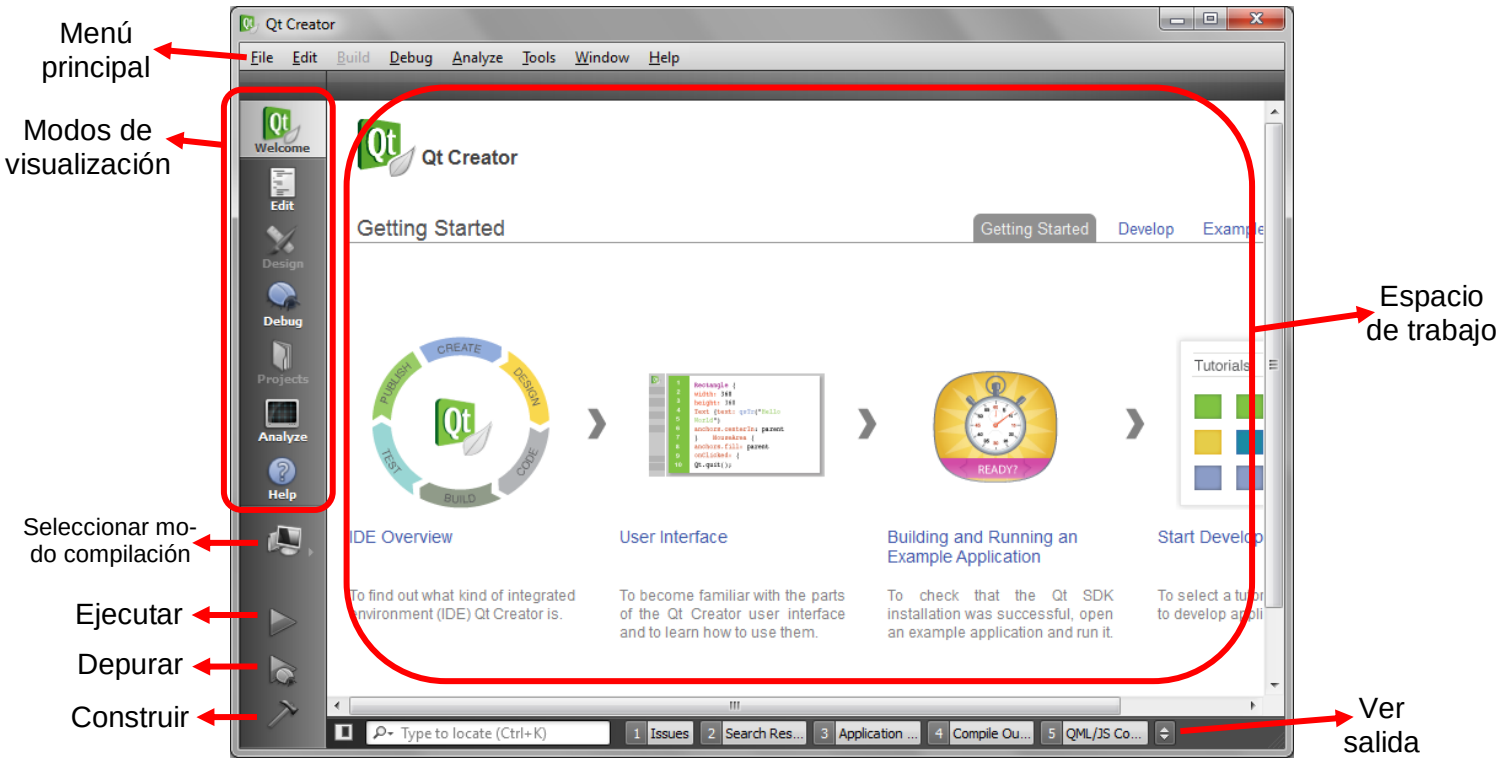
La licencia, también por defecto la LGPL. Seleccionar "I have read…"



Y por último la entrada del menú de inicio. La instalación puede tardar unos 6 minutos.

EL ENTORNO DE QT CREATOR

4. **Ejecutar** Qt Creator. Se abrirá una ventana como la mostrada abajo.



5. **Modos de visualización.** Existen siete modos de visualización diferentes, que nos permiten ver la información más adecuada en cada momento, según estemos editando, depurando, ejecutando, buscando ayuda, etc.
   5.1. **Modo Welcome (Bienvenida).** Aparece siempre al empezar. Contiene tutoriales de Qt, ejemplos, y nos permite abrir proyectos rápidamente.
   5.2. **Modo Edit (Edición).** Sirve para editar el código fuente de nuestra aplicación. Es el modo más habitual de visualización cuando estamos escribiendo el programa.
   5.3. **Modo Design (Diseño).** Nos permite ver y modificar el diseño de las ventanas y formularios de nuestra aplicación.
   5.4. **Modo Debug (Depuración).** Lo usamos cuando estemos depurando la aplicación. Muestra el código y la información de depuración.
   5.5. **Modo Projects (Proyectos).** Permite ver y configurar las opciones del proyecto. Normalmente no necesitaremos tocarlo mucho.
   5.6. **Modo Analyze (Análisis).** Sirve para medir el tiempo consumido en distintas operaciones del programa.
   5.7. **Modo Help (Ayuda).** Muestra la ayuda de Qt. Desafortunadamente, no incluye ayuda de C/C++ ni de las STL. Podemos usar Internet:
      - http://www.manpagez.com Páginas MAN online (para C).
      - http://www.cplusplus.com Referencia completa de C++ y las STL.

6. **Método de trabajo.** El proceso normal de trabajo empezará con la creación de un nuevo proyecto. Después diseñaremos el aspecto gráfico de la ventana (o ventanas) de nuestra aplicación. Escribimos el código usando el modo Edición. Después depuramos y ejecutamos, hasta estar seguros del funcionamiento correcto del programa.

NUESTRO PRIMER PROGRAMA QT

7. Vamos a crear nuestro primer programa con Qt Creator y a analizar los ficheros de los que consta un proyecto.

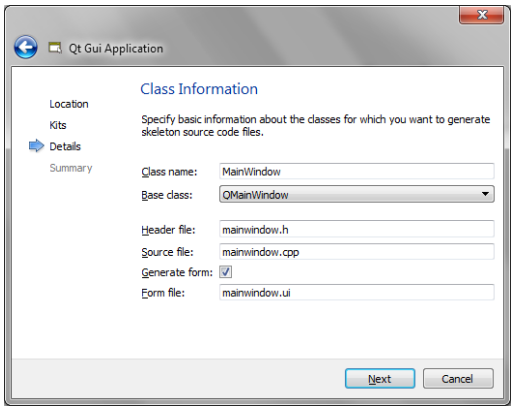
8. Dentro de Qt Creator, si tenemos abierto algún proyecto lo cerramos con: **File | Close All Projects and Editors**. A continuación hacemos: **File | New File or Project…** (o también, en el modo **Welcome**, solapa **Develop**, botón **Create Project**).

9. Dentro de la ventana **New Project** elegimos en **Projects** la entrada **Qt Gui Application**. Pinchamos en **Choose…**

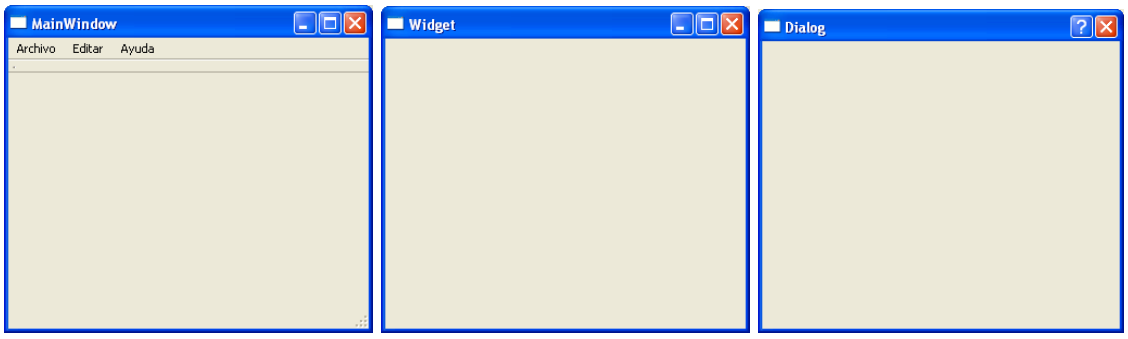
10. En la siguiente ventana, dentro de **Name:** ponemos el nombre de nuestro proyecto. Por ejemplo: HolaMundoQt (sin espacios). En **Create in:** seleccionamos el directorio base (bajo ese directorio se creará un subdirectorio HolaMundoQt con todos los ficheros de nuestro proyecto). Pinchamos en **Next**.

11. Seguidamente aparece otra ventana, **Kit Selection**, para elegir el kit con el que queremos compilar el programa. El kit se refiere al compilador usado, opciones de compilación, entorno de destino, y modos de compilación (Debug y Release). Por defecto solo tendremos la opción de elegir el kit **Desktop Qt 5.1.0 MinGW 32 bit**.

12. A continuación viene otra ventana, solicitando el nombre de la ventana principal de la aplicación (nombre de la clase y nombre de los ficheros asociados .h, .cpp y .ui), y el tipo de ventana: QMainWindow, QWidget o QDialog.



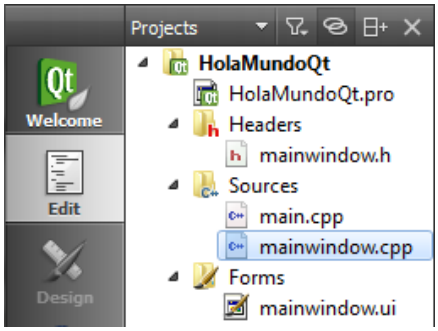
13. **QMainWindow** es una ventana con menú superior y barra de estado debajo. **QWidget** es una ventana vacía, con botones de maximizar, minimizar y cerrar. **QDialog** es una ventana con botones de cerrar y ayuda. Dejamos QMainWindow, dejamos los nombres que están y pinchamos en **Next**.



14. Finalmente pinchamos en **Finish** y se genera nuestro proyecto. ¡Ya hemos creado nuestro primer proyecto Qt! Ahora vamos a ver la estructura, el contenido y el significado de los ficheros de un proyecto.

ESTRUCTURA DE UN PROYECTO QT

15. Dentro de Qt Creator seleccionamos el modo **Edit** para ver los ficheros creados en el proyecto. Debe aparecer algo parecido a lo siguiente (después de desplegar las carpetas **Headers**, **Sources** y **Forms**).



16. **Archivo: HolaMundoQt.pro**. Contiene la definición de todos los ficheros y elementos de los que se compone el proyecto. Es un archivo de texto. En concreto, define el nombre de los formularios del proyecto (**FORMS**), de los ficheros de cabecera (**HEADERS**) y de implementación (**SOURCES**). También puede contener otra información, como la ruta de los ficheros *include* (**INCLUDEPATH**), las librerías externas utilizadas (**LIB**) y los ficheros de recursos (**RESOURCES**).

17. **Directorio: Forms**. Dentro de este directorio se incluyen todos los formularios (ventanas) que contiene nuestra aplicación. La aplicación puede contener varias ventanas, pero solo una de ellas puede ser la ventana principal (la que se muestra al ejecutar el programa).
    17.1. **Archivo: mainwindow.ui**. Es la ventana principal de la aplicación. Las ventanas tienen siempre extensión **.ui**. Internamente son archivos de texto en formato XML, que describen los elementos visuales de la ventana (botones, etiquetas, menús, etc.). No la editaremos "manualmente" sino de forma visual usando el editor de ventanas de Qt Creator (modo **Design**).

18. **Directorio: Headers**. Dentro de este directorio están todos los archivos de cabecera (es decir, los **.h**). Recordar que los archivos de cabecera contienen la definición de las clases, constantes, variables y procedimientos públicos. En general tendremos dos tipos de ficheros de cabecera: asociados a las ventanas, o asociados a módulos no visuales.
    18.1. **Archivo: mainwindow.h**. Contiene la declaración de la ventana principal de nuestra aplicación. Nuestras ventanas serán siempre clases, declaradas como subclases de QMainWindow, QWidget o QDialog. Observar el contenido del fichero mainwindow.h.

    El archivo mainwindow.ui (que no es código C/C++ sino XML) se compila para generar automáticamente el fichero ui_mainwindow.h, donde se define una clase Ui::MainWindow, que es la que contiene los botones, etiquetas, etc. Después, dentro de mainwindow.h se define la clase MainWindow que incluye un atributo Ui::MainWindow *ui;

19. **Directorio: Sources**. Dentro de este directorio está la implementación de las clases y de las funciones del programa. Contiene archivos de código C++ (con extensión **.cpp**). Pueden ser los asociados a las ventanas o asociados a módulos no visuales.

    19.1. **Archivo: mainwindow.cpp**. Aquí es donde va la implementación de los *slots* (los métodos asociados a los eventos de las ventanas), así como de cualquier otro método que queramos añadir a la clase. Contiene algo de código generado automáticamente. También podemos definir procedimientos fuera de las clases, si lo necesitamos.

    19.2. **Archivo: main.cpp**. Es el programa principal de nuestra aplicación, el procedimiento main. ¿Y qué hace el main?

main.cpp

```
1  #include "mainwindow.h"
2  #include <QApplication>
3
4  int main(int argc, char *argv[])
5  {
6      QApplication a(argc, argv);
7      MainWindow w;
8      w.show();
9      return a.exec();
10 }
```

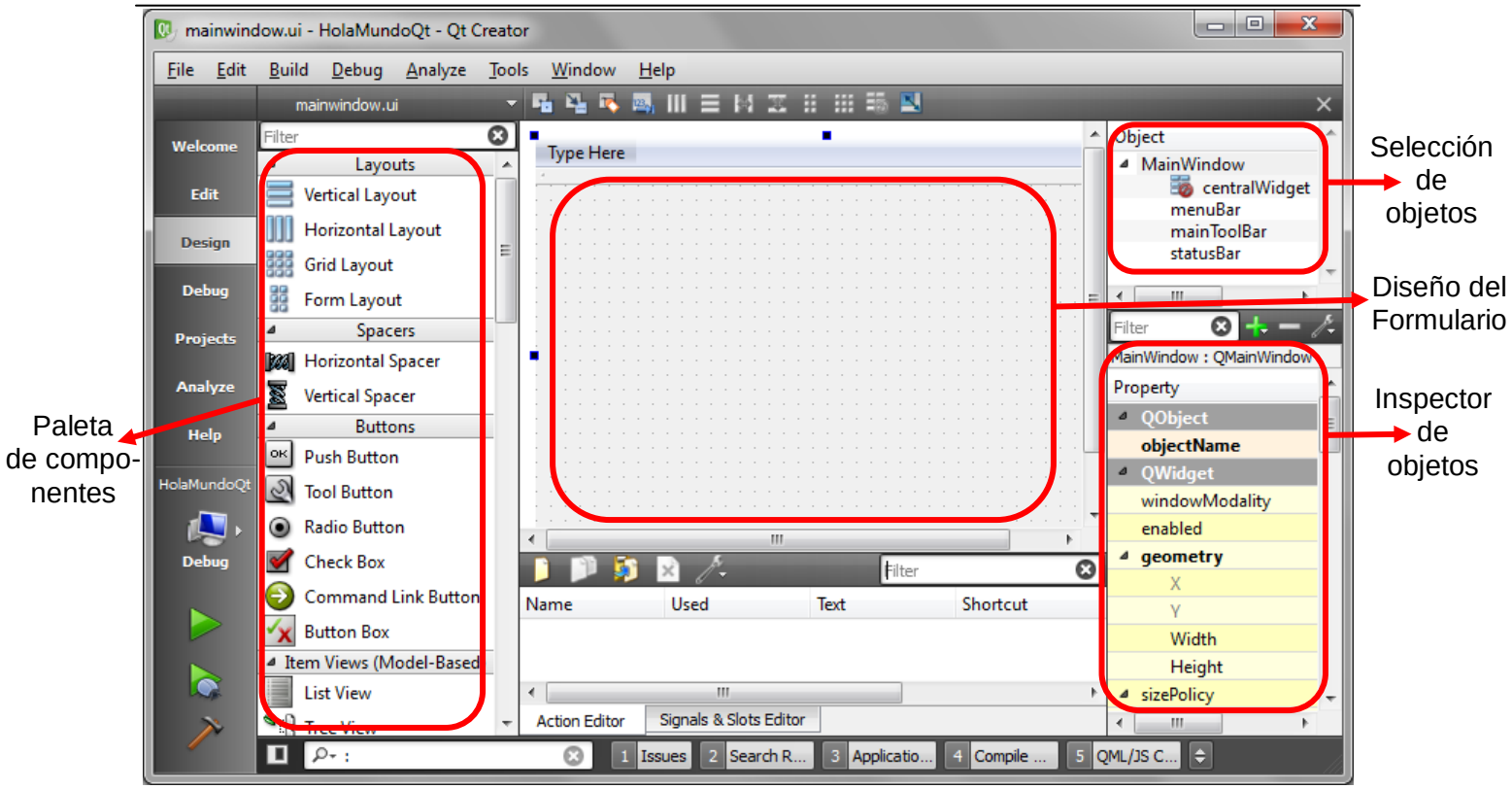
    Se define una QApplication **a**, se define una MainWindow **w**, se muestra la ventana **w** y se ejecuta la aplicación **a**. Sencillo, ¿verdad?

20. **Resumen**. Podemos crear un nuevo proyecto con **Welcome | New File or Project… | Qt Gui Application**. El proyecto tiene un fichero de proyecto (.pro), un programa principal (main.cpp), una o varias ventanas, y posiblemente módulos adicionales. Las ventanas constan de un fichero con el formulario de la ventana (.ui), un fichero de cabecera (.h) y uno de implementación (.cpp). Los módulos no visuales contendrán el fichero de cabecera (.h) y el de implementación (.cpp).

¡HOLA MUNDO!

21. Vamos a hacer un sencillo programa “¡Hola Mundo!” y luego implementaremos algo un poco más avanzado.

22. Partimos del proyecto **HolaMundoQt** que hemos creado en los pasos del 7 al 14. Seleccionamos el modo **Edit** (en los botones de modos de visualización). En la lista de archivos del proyecto que aparece a la izquierda, seleccionamos **HolaMundoQt | Forms | mainwindow.ui** y hacemos doble clic encima.



23. Se abrirá el **editor de formularios** (como se muestra arriba). A la izquierda aparece la **paleta de componentes**, con todos los elementos (*widgets*) que podemos añadir a nuestro formulario. En el centro aparece el **diseño del formulario** con el que estamos trabajando. Y a la derecha aparece el **inspector de objetos**, donde podemos editar las propiedades de los objetos que contiene el formulario.

24. Dentro del inspector de objetos, nos vamos a la propiedad **windowTitle** y escribimos: **Mi primer programa Qt**.

25. A continuación, en la paleta de componentes, pinchamos con el ratón en el componente llamado **Push Button** y lo arrastramos dentro del formulario. Podemos cambiar su tamaño y posición según queramos; y también el texto. Para ello podemos usar el inspector de objetos (propiedad **text**), o bien hacer doble clic en el botón. Vamos a poner: **Saludar**.

26. Ahora vamos a poner algo de código asociado al botón “Saludar”. Pinchamos con el botón derecho del ratón sobre el botón “Saludar”. En el menú desplegable que aparece, seleccionamos **Go to slot…** Nos aparecerá una ventana con la lista de eventos que se pueden asociar al botón. Seleccionamos **clicked()** y pulsamos **OK**.

27. Se abrirá automáticamente el fichero **mainwindow.cpp** dentro del editor de código, donde se habrá creado un nuevo método de la ventana, asociado a la pulsación del botón “Saludar”. El cursor se queda esperando a que escribamos el código del método.

```
void MainWindow::on_pushButton_clicked()
{

}
```

28. Dentro de las llaves escribimos:

```
QMessageBox::information(this, "Mensaje", "Hola Mundo");
```

29. Dentro del menú principal de Qt Creator, guardamos el proyecto con **File | Save All**. En general, siempre es aconsejable grabar con cierta regularidad.

30. Ahora ejecutamos el programa con **Build | Run**, o bien con el botón 

31. Y el resultado es… ¡Uppps! Error de compilación: *'QMessageBox' has not been declared*. Claro, **QMessageBox** es una clase que está definida en la librería QMessageBox. En general, todas las clases de Qt están definidas en librerías que tienen el mismo nombre que la clase. En definitiva, debemos añadir al principio:

```
#include <QMessageBox>
```

32. De nuevo, guardar y ejecutar. Ahora sí, ¡ya hemos creado nuestro primer Hola mundo con Qt! Acuérdate de cerrarlo antes de seguir.

33. La ejecución conlleva implícitamente la compilación del proyecto (compilación + enlace). También se puede hacer explícitamente con **Build | Build All**.

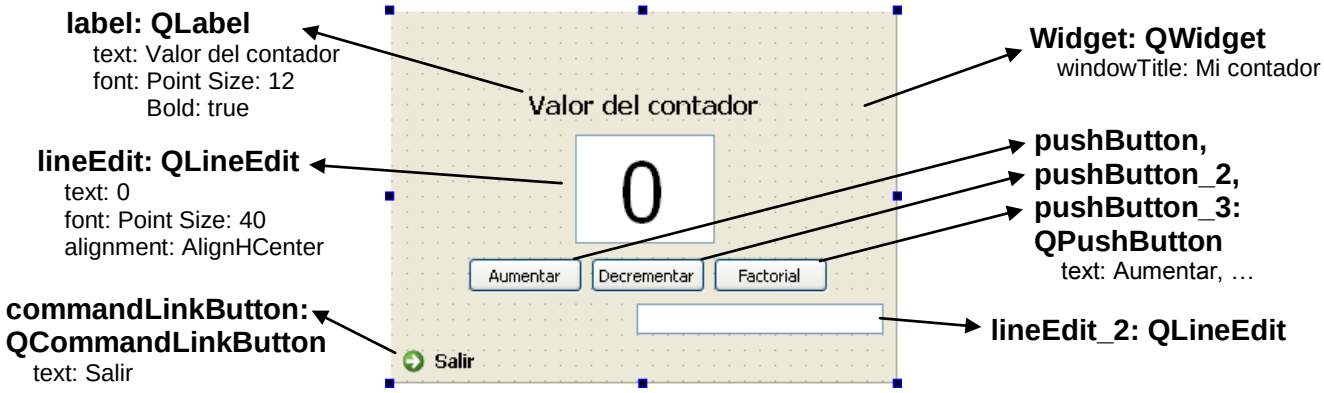
34. Observar los ficheros que se han creado dentro del directorio de nuestro proyecto:
    - 34.1. HolaMundoQt.pro: fichero principal del proyecto.
    - 34.2. main.cpp: programa principal de la aplicación.
    - 34.3. mainwindow.ui, mainwindow.h, mainwindow.cpp: ficheros asociados a la ventana principal.

    Y en el directorio: build-HolaMundoQt-Desktop_Qt_5_1_0_MinGW_32bit-Debug tenemos el resultado de la compilación:
    - 34.4. ui_mainwindow.h: fichero de código creado automáticamente a partir del mainwindow.ui.
    - 34.5. Makefile, Makefile.Debug, Makefile.Release: ficheros makefile del proyecto, en modo *debug* o *release*.
    - 34.6. Directorio debug: archivos de compilación del proyecto en modo depuración, código objeto (.o) y el ejecutable (.exe).
    - 34.7. Directorio release: lo mismo pero para el modo versión final.

UN SENCILLO CONTADOR

35. Ahora vamos a hacer algo un poco más avanzado: un contador manual. Creamos un proyecto nuevo (repetir los pasos del 7 al 14) de tipo **QWidget** y le damos el nombre **Contador**. (¡Cuidado! No meterlo como un subdirectorio dentro de HolaMundoQt. Si está abierto el proyecto HolaMundoQt, cerrarlo primero.)

36. En el **modo Edit**, nos vamos al fichero **Forms | widget.ui**, y se abre el editor de formularios. Seleccionar componentes de la paleta de componentes y ponerlos en el formulario, hasta crear una ventana con el siguiente aspecto. Algunas propiedades deben ser ajustadas con el inspector de objetos.



37. Podemos guardar y luego ejecutar para ver el aspecto que tendrá el programa.

38. Ahora vamos a meter el código asociado a los eventos de interés. Seleccionamos el botón **Salir**, pinchamos en el botón derecho del ratón, y elegimos **Go to slot…** Luego seleccionamos el evento **clicked()** y **OK**.

39. Ahora debemos estar dentro del editor de código, editando el fichero **widget.cpp**, y dentro del método creado **on_commandLinkButton_clicked()**. Escribimos el siguiente código:

```
close();
```

40. La operación **close()** es un método de la clase **Widget** (dentro de la cual nos encontramos), e indica que se cierre la ventana.

41. Vamos ahora a programar el efecto de los botones. Para ello necesitamos una variable entera que nos sirva de contador. Para simplificar, la vamos a definir como una variable global (es decir, fuera de la clase ventana). En **widget.cpp** nos vamos justo después de los *includes* y escribimos:

```
int contador= 0;
```

42. Igual que hemos hecho con el botón Salir, hacemos con el botón **Aumentar**. Nos vamos a **widget.ui**. Creamos para este botón su evento **clicked()** y escribimos:

```
void Widget::on_pushButton_clicked()
{
    contador++;
    ui->lineEdit->setText(QString::number(contador));
}
```

43. El atributo **ui** (*user interface*) está dentro de nuestro formulario (en la clase Widget) y a su vez **ui** contiene todos los elementos del formulario (tenemos ui->label, ui-> lineEdit, ui->pushButton, etc.). Por otro lado, **QString::number** es una operación estática para convertir un número en un QString (el tipo cadena usado en Qt).

44. De la misma forma, para el botón **pushButton_2** le asociamos el evento:

```
void Widget::on_pushButton_2_clicked()
{
    contador--;
    ui->lineEdit->setText(QString::number(contador));
}
```

45. Finalmente, queremos que el botón **pushButton_3** calcule el factorial del valor actual del contador. Así que en el código de su evento escribimos:

```
void Widget::on_pushButton_3_clicked()
{
    long resultado= 1;
    for (int i= 1; i<=contador; i++)
        resultado*= i;
    ui->lineEdit_2->setText(QString::number(resultado));
}
```

46. Guardamos el proyecto y lo ejecutamos. (Por cierto, ¿qué ocurre si ejecutamos sin guardar primero?) Si hay problemas, repasar todos los pasos y preguntar al profesor.

DEPURACIÓN CON QT CREATOR

47. La forma más **sencilla y común de depurar** es escribir mensajes en ciertos puntos de interés del programa. Para ello, podemos usar la función **qDebug**. Tiene el mismo formato que el printf de C (por ejemplo, podemos poner: `qDebug("La variable a vale %d y b vale %d", a, b);` El resultado se mostrará en la parte inferior del entorno de Qt Creator, seleccionando: **3 Application Output**.

48. Pero también tenemos herramientas para hacer depuración más avanzada. Los proyectos pueden ser construidos en dos modos diferentes: el **modo Debug** y el **modo Release**. El modo se selecciona en el botón de seleccionar modo de compilación.
    48.1. **Modo Debug.** Es más adecuado cuando estamos en la fase de desarrollo del proyecto. Se genera toda la información de depuración.
    48.2. **Modo Release.** Se usa una vez acabado el proyecto, para entregar el programa al cliente. El código está más optimizado en tiempo y espacio.

49. Para usar las funciones avanzadas de depuración: (1) debemos tener activado el modo Debug; y (2) hay que ejecutar el proyecto pinchando el botón  (o F5).

50. Procedimiento para hacer depuración de programas con Qt Creator:
    50.1. En primer lugar, señalar dentro del código los **puntos de ruptura** (los sitios donde queremos que se detenga la ejecución). Para añadirlos/eliminarlos, debemos pinchar en el código, a la izquierda de los números de las líneas; se muestra una bola roja  sobre los puntos de ruptura definidos.
    50.2. Ejecutar el programa estando en **modo Debug** y con el botón  .
    50.3. Cuando la ejecución llega a un punto de ruptura, se detiene y podemos empezar la depuración. Tenemos la posibilidad de: **ejecutar paso a paso** (F10), ejecutar paso a paso **entrando en las llamadas** a procedimientos (F11), ver el valor de las variables locales (cuadrado **Name|Value|Type**), ver el valor de otras variables (en la misma solapa, con el botón derecho del ratón, opción **Insert New Expression Evaluator**), ejecutar hasta una línea concreta, etc.
    50.4. Una vez comprobado un trozo de código, podemos continuar la ejecución  o detener la depuración  .

AÑADIR UN ICONO AL PROGRAMA

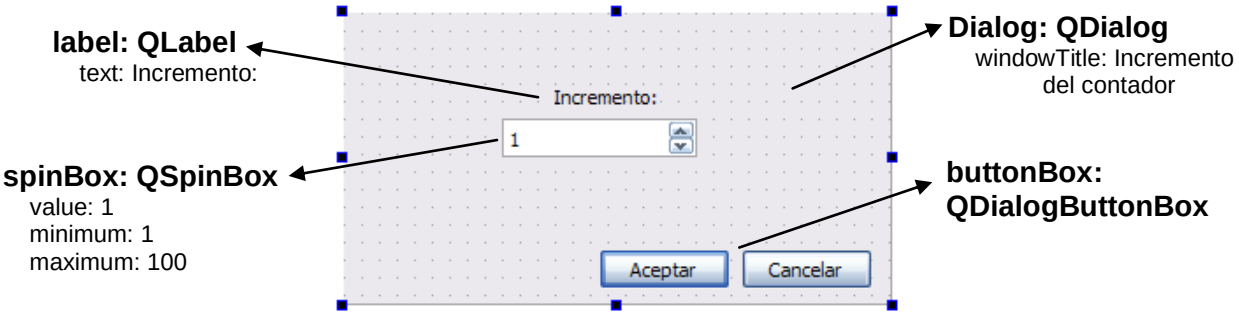
51. A todos nos gusta poder personalizar al máximo nuestros programas. ¿Cómo podemos **cambiar el icono** de nuestra aplicación creada con Qt Creator?

    51.1. Abrimos el proyecto **HolaMundoQt**.
    51.2. Usamos la opción de menú **File | New File or Project…**
    51.3. En la ventana que aparece, seleccionamos **Qt | Qt Resource File** y le damos a **Choose…**
    51.4. Nos pide que le asignemos un nombre al archivo de recursos. Por ejemplo, podemos escribir: **recursos**, y luego **Next…** Y después **Finish**.
    51.5. Se ha añadido a nuestro proyecto una nueva carpeta **Resources** con el archivo de recursos. Además, se ha abierto el editor dentro del fichero de recursos. Pinchamos dentro del editor en **Add | Add Prefix**.
    51.6. A continuación, seleccionamos **Add | Add Files**.
    51.7. Seleccionamos un archivo con el icono deseado (extensión **.ico** o **.bmp**) y **Abrir**.
    51.8. Guardamos el proyecto: **File | Save All**.
    51.9. Ahora seleccionamos en el editor el archivo **Forms | mainwindow.ui**
    51.10. Teniendo seleccionado el objeto ventana (**MainWindow**) nos vamos a la propiedad **windowIcon** dentro del inspector de objetos.
    51.11. Pinchar a la derecha de **windowIcon** y sale la opción **Choose Resource…**
    51.12. Para acabar, sólo tenemos que seleccionar nuestro icono en la ventana que aparece y darle a **OK** (puede ser necesario pinchar en la **flecha verde**, para actualizar la ventana y que salga el icono).
    51.13. Ya podemos guardar el proyecto y ejecutarlo.

MÚLTIPLES FORMULARIOS

52. Normalmente los programas no tienen una única ventana, sino que hacen uso de muchos formularios auxiliares. ¿Cómo construir una **aplicación con varios formularios**?

    52.1. Abrimos el proyecto **Contador** (si teníamos algún proyecto abierto, cerrarlo). Vamos a añadir un formulario que nos permita seleccionar el tamaño del incremento.
    52.2. Usamos la opción de menú **File | New File or Project…**
    52.3. En la ventana que aparece, seleccionamos **Qt | Qt Designer Form Class** y le damos a **OK**.
    52.4. Se nos pide que seleccionemos el tipo de formulario. Elegimos el primer tipo, **Dialog with Buttons Bottom** y le damos a **Next**.
    52.5. Nos pide el nombre de la ventana y de los ficheros asociados. Dejamos los que están (class Dialog). Le damos a **Next** y luego **Finish**.
    52.6. Abrimos **dialog.ui** en el editor y diseñamos una ventana como la siguiente.



52.7. Todos los **QDialog** tienen un método **exec()** para ejecutarlos de forma *modal* (bloquear el padre hasta que no se cierre) y que devuelve true/false según el usuario pulse aceptar/cancelar (también podemos cerrar el cuadro de diálogo mediante código, con las funciones **accept()** y **reject()**).

52.8. Vamos a añadir a la clase **Dialog** un método **getValor()** para obtener el valor seleccionado por el usuario en el incremento. Por lo tanto, editamos **dialog.h** y dentro de la clase **Dialog** añadimos:

```
public:
    ...
    int getValor(void);
private:
    ...
    int valor;
```

52.9. Editamos ahora **dialog.cpp** y ponemos:

```
Dialog::Dialog(QWidget *parent) :
    ...
    valor= 1;   // Inicialización
}

int Dialog::getValor(void)
{
    return valor;
}
```

52.10. Volvemos a **dialog.ui**, seleccionamos **buttonBox** y en **Go to slot...** seleccionamos el evento **accepted()** y escribimos:

```
valor= ui->spinBox->value();
accept();
```

52.11. Y en el slot **rejected()** del **buttonBox** escribimos simplemente:

```
reject();
```

52.12. Ahora queremos usar ese cuadro de diálogo desde el contador. Nos vamos a **widget.ui**, añadimos un botón al formulario, creamos su slot **clicked()** y en el código asociado ponemos:

```
Dialog d;       // Nos definimos una ventana de tipo Dialog
if (d.exec())   // y la ejecutamos
    incremento= d.getValor();
```

52.13. La variable **incremento** se supone que es una variable global que debemos añadir en **widget.cpp**. Además, dentro de este fichero **widget.cpp** debemos hacer el *include* de dialog.h. Por lo tanto, nos vamos al principio del código de **widget.cpp** y escribimos:

```
#include <dialog.h>
int incremento= 1;
```

52.14. Para acabar, en los sitios donde ponía: `contador++;` escribimos: `contador+= incremento;` Y donde ponía: `contador--;` escribimos: `contador-= incremento;`

52.15. Ya podemos darle a guardar todo y ejecutar.

DISTRIBUCIÓN DE PROGRAMAS CREADOS CON QT CREATOR

53. Cuando creamos un programa no suele ser para quedárnoslo nosotros mismos, sino para que lo pueda usar otra gente. Pero, ¿cómo **distribuir un programa** creado con Qt Creator? Debemos tener en cuenta algunas cosas.

54. En primer lugar, generar el proyecto con el **modo Release**. El archivo ejecutable aparecerá en un directorio cuyo nombre será del estilo: **build-NombreProyecto-Desktop_Qt_5_1_0_MinGW_32bit-Release** y será mucho más pequeño que el ejecutable en modo Debug.

55. En segundo lugar: incluir los **archivos DLL** necesarios. El código de las librerías Qt está dentro de archivos DLL (*Dynamic Link Library*), que se instalan con Qt Creator. Para distribuir una aplicación a una persona que no tenga Qt Creator, es necesario entregarle los archivos DLL que se usan. Se deben copiar en el mismo directorio donde esté nuestro ejecutable (aunque lo ideal sería que se copiaran a una zona compartida que esté en el PATH del sistema).

    En concreto, *normalmente* se deben copiar las siguientes DLL al distribuir nuestros programas Qt (es muy conveniente comprobar si falta alguna otra más):

C:\Qt\Qt5.1.0\5.1.0\mingw48_32\bin\Qt5Core.dll	(4,18 Mbytes)
C:\Qt\Qt5.1.0\5.1.0\mingw48_32\bin\Qt5Gui.dll	(4,24 Mbytes)
C:\Qt\Qt5.1.0\5.1.0\mingw48_32\bin\Qt5Widgets.dll	(5,86 Mbytes)
C:\Qt\Qt5.1.0\5.1.0\mingw48_32\bin\libgcc_s_dw2-1.dll	(532 Kbytes)
C:\Qt\Qt5.1.0\5.1.0\mingw48_32\bin\libstdc++-6.dll	(966 Kbytes)
C:\Qt\Qt5.1.0\5.1.0\mingw48_32\bin\libwinpthread-1.dll	(73 Kbytes)
C:\Qt\Qt5.1.0\5.1.0\mingw48_32\bin\icuin51.dll	(3,21 Mbytes)
C:\Qt\Qt5.1.0\5.1.0\mingw48_32\bin\icuuc51.dll	(53 Kbytes)
C:\Qt\Qt5.1.0\5.1.0\mingw48_32\bin\icudt51.dll	(21,3 Mbytes)

56. Observar que lo anterior implica que, incluso para el programa más simple, necesitamos entregar al usuario unos 40 Mbytes en archivos DLL. Y todo eso además de las DLL correspondientes a OpenCV… En cualquier caso, es conveniente asegurarse que no le faltan otras DLL al usuario. Para eso lo mejor es ejecutar el programa en un ordenador sin Qt y ver las DLL que va pidiendo.

COSAS PARA SEGUIR EXPLORANDO

57. **Compleción automática de código.** El entorno de Qt Creator tiene algunas utilidades muy interesantes en la edición de código. Una de ellas es la capacidad de autocompletar código cuando estamos a medio de escribirlo. Una forma de usarlo es cuando escribimos el nombre de un objeto y nos muestra la lista de métodos y atributos del mismo. Pero también se puede usar en cualquier momento, pulsando **Control + Espacio**.

58. **Autoindentación del código.** Muchas veces nos descuidamos al indentar el código fuente. Después de escribir muchas líneas, es difícil volver a editar todos los espacios para dejar el código bien indentado. Solución: usar la indentación automática. O bien con **Control + I**, o bien con **Edit | Advanced | Autoindent Selection**.

59. **Comentado/descomentado rápido.** Otra interesante funcionalidad del editor es la posibilidad de comentar o descomentar rápidamente un fragmento del código. Para ello, simplemente debemos seleccionar el trozo que queremos comentar (o descomentar) y pulsar **Control + /** (es decir, **Control + Mayús. + 7**).

60. **Ayuda sensible al contexto.** En el modo de visualización **Help** podemos encontrar ayuda sobre las funciones de Qt (recordar, no hay ayuda general sobre C, C++ ni las STL). También podemos usar la ayuda en el modo **Edit**. Cuando nos movemos con el ratón sobre el texto, aparece información sobre el elemento que hay debajo. Si aparece **F1**, significa que podemos pulsar F1 para obtener ayuda de esa clase o método.

61. **Errores de compilación.** Desafortunadamente, no siempre escribimos todo perfectamente bien a la primera, y surgen los errores de compilación. El editor de Qt Creator nos señala en rojo o en verde los errores de sintaxis mientras escribimos. Otra utilidad, pulsando con el botón derecho sobre un símbolo podemos ver su definición. Por otro lado, al compilar podemos ver los errores de compilación (debajo, en el panel **1 Issues**) o bien podemos ver la salida extendida del compilador (en el panel **4 Compile Output**).

62. **Reconstruir todo.** A veces (raramente) es conveniente usar la opción **Build | Rebuild All**. En ocasiones (muy pocas) hay que ver la salida extendida del compilador para poder entender la causa del problema.

63. **Propiedades del proyecto.** Dentro del modo de visualización **Projects** podemos ver los detalles sobre la compilación y ejecución del proyecto (en modo debug y release). Normalmente no necesitamos cambiar nada. Obsérvese que en la ejecución se incluyen en el PATH los directorios necesarios de Qt y MinGW; por eso, si ejecutamos nuestro .exe fuera del entorno, nos dará un error (no se encuentran algunas DLL necesarias). Ver el punto 55 para saber cómo solucionar este problema.

64. **Usando el localizador.** Cuando tenemos muchos ficheros abiertos y queremos localizar dónde se encuentra la definición de una clase o de un método concreto, podemos usar el **localizador** (que aparece en la parte inferior de Qt Creator). Si lo

que queremos es buscar un trozo de texto, usaremos el menú **Edit | Find/Replace**. Podemos buscar en un fichero o en todos los del proyecto.

65. **Y mucho más por explorar.** Es conveniente que eches un vistazo a los componentes existentes en la paleta de componentes, sus propiedades, los cuadros de diálogo existentes, los tipos de datos de Qt que pueden ser de ayuda, etc.

65.1. **Componentes de utilidad**: label, line edit, push button, radio button, check button, spin box, combo box, slider, etc.

65.2. **Cuadros de diálogo predefinidos**: existen algunos cuadros de diálogo muy interesantes, como los QColorDialog (para seleccionar un color) y los QFileDialog (para seleccionar un nombre de fichero). Ejemplo, seleccionar un nombre de fichero para abrir:

```
QString nombre= QFileDialog::getOpenFileName();
```

Seleccionar un nombre de fichero para guardar:

```
QString nombre= QFileDialog::getSaveFileName();
```

Abrir un fichero incluyendo filtros para que la selección sea un archivo de vídeo:

```
QString nombre;
nombre= QFileDialog::getOpenFileName(this, "Abrir archivo de vídeo"
      ,"", tr("Archivos AVI (*.avi);; Otros vídeos (*.mpg *.mov
            *.wmv);;Todos los archivos (*.*)"));
```

65.3. **Tipo de datos QString**: para almacenar cadenas. Ejemplo, convertir un entero seleccionado en un spin box, a un QString:

```
int numero= ui->spinBox->value();
QString cadena= QString::number(numero);
```

Almacenar en un QString una cadena dada en un char *:

```
char str[]= "Hola";
QString cadena= str;
```

Convertir un QString a un string de C++ y a un char* de C:

```
char achar[500];
string str;
QString cadena= "¡Adiós!";
str= cadena.toStdString();
strcpy(achar, str.c_str());
```

65.4. **Leer y escribir archivos**: para leer y escribir en archivos del disco se usan los tipos QFile, QTextStream y QDataStream. Buscar más información en la ayuda de Qt Creator.