Índice general

Parte I

Contenido

Capítulo 1

Introducción

1.1. ¿Qué son las bibliotecas gráficas?

Las bibliotecas gráficas no son más que un conjunto de clases y funciones encapsuladas en una biblioteca que pueden ser utilizadas por programas o usuarios para mostrar o dibujar gráficos en la pantalla de manera simple.

¿Qué es wxWidgets?

wxWidgets es una biblioteca gráfica en C++ que posee del GUI (Interfaz Gráfica de Usuario) y otras facilidades en distintas plataformas. La versión 2 es soportada por todas las versiones de sobremesa de Windows, Unix (con GTK+ o Motif), y MacOS.

Algunas de las ventajas de wxWidgets son:

Soporta varios sistemas operativos, pero con una pequeña particularidad: de acuerdo a la plataforma en que se ejecute la aplicación, esta tomara el estilo del SO donde se esté ejecutando. Por ejemplo, si se ejecuta en MS-Windows, la aplicación tendrá la apariencia de una aplicación hecha con el API de Windows, pero si se ejecuta en GNU/Linux, tendrá la apariencia de una aplicación GTK2. Es decir que la parte de GUI es solo una capa para el API nativa de cada SO. Además de lo anterior, wxWidgets cuenta con una versión embebida para dispositivos como PDAs y teléfonos celulares.

Es cien por ciento libre en todas las plataformas que soporta, con una licencia LGPL modificada. Tiene un API orientada a objetos que es, en extremo, fácil de aprender y utilizar. Posee funcionalidades para muchos aspectos aparte de la construcción de interfaces gráficas. Como gráficos 2D, 3D con OpenGL, Bases de Datos (ODBC), redes, impresión, hilos, entre muchas otras.

Pese a todas estas ventajas, wxWidgets también tiene algunas desventajas: Su diseño orientado a objetos no es el mejor que uno pueda ver. A veces abusa de la utilización de macros para realizar ciertas operaciones (como las tablas de eventos), aunque esto hace que codificar sea más fácil, también complica la labor de depuración, especialmente para los mismos desarrolladores de wxWidgets. Por otro lado, debido a que el framework se comenzó a desarrollar hace bastante tiempo, no cuenta con soporte para algunas características relativamente modernas del lenguaje C++ como por ejemplo el manejo de excepciones, y la STL.

1.2. ¿Cómo programar en wxWidgets?

Para empezar a utilizar la librería, con sus componentes y utilidades, tenemos que incluirla: *wx.h* y además crearemos el bloque **IFNDEF - DEFINE - ENDIF** para la definición de las clases.

A continuación, declaramos la clase más importante de la aplicación: MyApp que hereda de wxApp. Es la clase principal y más importante ya que con dicha clase implementamos el evento: *OnInit()* el cual es el encargado de iniciar la aplicación.

Aparte, en la clase derivada de wxApp, en nuestro caso MyApp, debe de aparecer todas las ventanas tipo Frame, las ventanas avanzadas, y los diálogos que compone toda la aplicación completa.

```
class MyApp public wxApp
{
 public:
   virtual bool OnInit();
 private:
  MyFrame *frame;
};
```

Por último en este fichero, se debe declarar que existe una clase que hereda de wxApp y que tiene la funcionalidad de iniciar nuestra aplicación. Aunque parezca mentira, con esta instrucción lo que estamos realizando es la declaración de la función principal (*main*) que toda aplicación debe tener. Luego, en el fichero .cpp implementaremos esa función Principal que expicaremos en su lugar.

```
DECLARE_APP(MyApp)
```

Definimos una clase nueva llamada *MyFrame* que hereda de *wxFrame*.

Con esta clase definiremos nuestro Frame, es decir, el contenido y estilo de la ventana. Luego, hay que definir las funcionalidades que tanto la ventana como los componentes vamos a querer que tengan, es decir, le daremos una funcionalidad por cada evento la aplicación que queramos.

1.3. El entorno wxDev-C++

wxDev-C++ es un entorno de desarrollo integrado libre basado en el popular Dev-C++.

Hay varias características nuevas no encontradas en el Dev-C++ original. Uno de ellas es un diseñador rápido de aplicaciones visual que trabaja como el C++Builder para crear aplicaciones wxWidgets. La última versión, la 7.0, agrega soporte para compiladores de Microsoft. Se está trabajando para el soporte para otros compiladores.

Características del wxDev-C++

Diseñador de forma de wxWidgets:

- Genera recursos XRC XML..
- Paradigma de diseño de arrastrar y soltar.
- Soporta disposiciones de objetos basadas en sizer para los wxWidgets.
- Conecta los eventos a las funciones miembro dentro del editor.

Depurador integrado

- Soporte para GDB.
- El soporte de CDB (WinDbg) está en el CVS.
- Visores de variables.
- Trazo automático de la pila.
- Lista local de variables.
- Muestra desensamblador de registros del CPU.

Características del editor

- Browser de Clases.
- Completado de código.
- Manejo de proyecto.
- Perfiles de proyecto.
- Resaltado de sintaxis personalizable.
- Resaltado automático de ensamblado en línea.
- Lista "To do".

Compatibilidad de aplicaciones

- Soporte incorporado para CVS.
- Soporta los compiladores MinGW y Visual C++ (6, 2003 y 2005).

Rápidamente crea aplicaciones para Windows y de cónsola, bibliotecas estáticas y DLLs

- Soporte para plantillas de proyecto para acelerar la creación de nuevos tipos de proyecto.
- Manejador de paquetes (con el uso de DevPaks), para la instalación fácil de bibliotecas adicionales.

1.4. Otras bibliotecas gráficas

Veamos a continuación alguna de las otras bibliotecas que existen:

Qt

Es el framework en el cual se basa el escritorio de GNU/Linux KDE. Como ventajas tiene el hecho de tener una buena velocidad, debido a que está escrito en C++ y tener un muy buen diseño orientado a objetos que hacen muy placentero programar con Qt. Por otro lado, el problema con Qt consiste principalmente en que solo es software libre en su versión para GNU/Linux, todas las demás versiones (Win32, MacOS y otros) son propietarias (y además con alto costo).

GTK+

Tiene varias ventajas, como por ejemplo que está disponible para muchísimos sistemas operativos y es completamente libre, con licencia LGPL. Sin embargo, GTK+ tiene algunos inconvenientes: En primer lugar, GTK+ solo posee funcionalidad para desarrollar interfaces gráficas de usuario y deja un lugar vacio a aspectos como bases de datos, redes, etc. Por otro lado, el API de GTK+ está escrito de forma estructurada en lenguaje C, lo cual dificulta trabajar con programación orientada a objetos. Aunque existe GTKmm, tiene el mismo diseño estructurado pero usando clases en C++.

Capítulo 2

Conceptos Básicos

2.1. Componentes

Hasta ahora habíamos hecho programas en los que los datos y las funciones estaban perfectamente separadas, cuando se programa con objetos esto no es así, cada objeto contiene datos y funciones. Y un programa se construye como un conjunto de objetos, o incluso como un único objeto.

En C++, un objeto es sólo una instancia (un ejemplar) de una clase determinada. Es importante distinguir entre objetos y clases, la clase es simplemente una declaración, no tiene asociado ningún objeto, de modo que no puede recibir mensajes ni procesarlos, esto únicamente lo hacen los objetos.

En C++ un método no es otra cosa que una función o procedimiento perteneciente a un objeto.

En wxWidgets llamaremos componente a cada una de las clases con las que contamos y podemos utilizar siempre que queramos en nuestros programas. De estas clases podremos crear objetos (o instancias) y diseñar nuestros programas.

2.2. Tipos de datos

wxString

Esta clase representa una cadena simple. Es uno de los componentes más utilizados en wxWidgets. Tiene varios constructores, veremos a continuación los más utilizados.

El primero es el constructor por defecto, que crea una cadena vacía:<

```
wxString()
```

Después podemos crear una cadena de un tamaño de derminado pasandole un caracter y un tamaño:

```
wxString(char ch, size_t n = 1)
```

Por último, podemos crear un objeto wxString a partir de una cadena de caracteres de bajo nivel:

```
wxString(const char* psz, wxMBConv& conv, size_t nLength = wxSTRING_MAXLEN)
```

Este tipo de dato cuenta con muchos métodos, de los cuales destacamos: *Alloc()*, que permite reservar memoria para una cadena, *Append()*, que le concatena una cadena al final, *Clear()*, que limpia la cadena y libera la memoria ocupada, *Cmp()*, que la compara con otra cadena, *Empty()*, que pone la cadena vacía, *Find()*, que busca un caracter en la cadena, *FromAscii()*, que convierte desde ASCII, *GetChar()*, que devuelve el caracter de la posición dada, *IsAscii()*, *IsEmpty()*, *IsNull()* y *IsNumber()*, que realizan comprobaciones en la cadena, *Length()*, que devuelve la longitud, *Prepend()*, que concatena una cadena al inicio, *Remove()*, que borra desde la posición dada hasta el final de la cadena, *Replace()*, que reemplaza las ocurrencias de la cadena dada por otra cadena también dada, *SetChar()*, que modifica el caracter de la posición dada, y *ToAscii()*, *ToDouble()* y *ToLong()*, para conversiones de tipo.

wxFile

Este componente simula un archivo, y contiene los métodos necesarios para trabajar con ellos en bajo nivel.

Para abrir un fichero necesitamos indicar, además del nombre, el modo en que se desea abrir, a elegir entre: **wxFile::read**, para leer, **wxFile:write**, para escribir eliminando el contenido anterior, **wxFile::read_write**, para leer y escribir, y **wxFile::write_append**, para escribir sin eliminar el contenido anterior.

```
wxFile(const char* filename, wxFile::OpenMode mode = wxFile::read)
```

Los métodos más destacados de esta clase son: *Access()*, para verificar el acceso en un modo determinado, *Close()*, para cerrar el archivo, *Create()*, para crear un archivo para escritura, *Eof()*, que comprueba si se ha llegado a fin de archivo, *Exists()*, que comprueba si el archivo existe, *GetKind()*, que devuelve el tipo del archivo, *IsOpened()*, que comprueba si el archivo está abierto o no, *Length()*, que devuelve la longitud del archivo, *Open()*, que abre un archivo, *Read()*, para leer, *Seek()*, que se desplaza a una determinada posición, *SeekEnd()*, que va al final del archivo, y *Write()*, para escribir.

2.3. Eventos

Un evento es una acción que es reconocida por un objeto y provocada bien por el usuario al interactuar con la interfaz del programa (la pulsación de un botón del ratón, la pulsación de una tecla, etc.), o bien por el propio sistema.

Muchos de los objetos tienen un conjunto de eventos predefinidos que se pueden reconocer y, si ocurre alguno de ellos, se ejecuta un manejador de evento (función) como respuesta, por lo tanto, una aplicación para Windows en realidad lo que hace es ejecutar los manejadores de los distintos eventos que se van produciendo.

Eventos estáticos

En wxWidgets, los componentes son simplemente atributos de una clase contenedor, mientras que los manejadores de eventos son métodos de esta misma clase, por lo que surge la siguiente pregunta, ¿cómo sabemos qué evento de qué componente activa qué manejador?.
Para solucionar este problema tenemos la tabla de eventos, la cual tenemos que declarar siempre dentro de una clase contenedor, y que enlazará cada evento con su respectivo manejador.

La tabla de eventos se definirá en el fichero fuente, entre las macros **BEGIN_EVENT_TABLE** y **END_EVENT_TABLE**.

Eventos dinámicos

Aunque no es usual, también podemos enlazar eventos con sus manejadores en tiempo de ejecución, es lo que llamamos eventos dinámicos.

Una posible aplicación sería el querer utilizar en el mismo evento diferentes manejadores dependiendo de que se cumpla una determinada condición o no. Otro posible uso sería el querer usar lenguajes de programación que no admitan eventos estáticos (como Python).

Asociado a este tipo de eventos existen dos funciones, *Connect()* y *Disconnect()*, que nos permiten enlazar y desenlazar manejadores con eventos. A estas funciones se les pasan por este orden: el identificador de ventana, el identificador de evento y un puntero al manejador de evento correspondiente.

2.4. Primer ejemplo

Para finalizar este primer capítulo en el que hemos tratado los conceptos básicos de wxWidgets, trabajaremos en un ejemplo simple, que nos servirá como introducción a la programación con wxWidgets.

El ejemplo consistirá en una ventana que contendrá una etiqueta y dos botones. La etiqueta estará inicialmente vacía, aunque tendrá aplicado un estilo. El primer botón se utilizará para mostrar en la etiqueta anterior el mensaje "¡Hola mundo!", mientras que el segundo botón nos permitirá salir de la aplicación.

El primer punto a tener en cuenta es que utilizaremos dos ficheros (un fichero de cabecera (.h) y un fichero de código (.cpp)), ya que tenemos una única ventana. El segundo punto será que los tamaños de la ventana y los componentes no son importantes en este ejemplo. Se utilizarán unos por defecto, pudiéndose modificar siempre que el usuario lo desee.

Empezaremos definiendo las clases que necesitaremos en el archivo de cabecera, en este caso necesitaremos dos, una clase que modelará la aplicación en sí, y otra que modelará la ventana. Cómo definir estas clases, sus atributos y sus métodos se verán con más claridad en el siguiente capítulo.

En la clase que modela la aplicación (*MyApp*) necesitamos un método que arranque la aplicación y en la clase que modela la ventana (*MyFrame*) necesitamos un constructor y un método por cada manejador de evento que debamos programar dentro de esta ventana. Además, dentro de la clase MyFrame, debemos definir los componentes que vayamos a modelar dentro de la ventana, en este caso, una etiqueta y dos botones.

Por último, dos cosas muy importantes. La primera, debemos declarar la tabla de eventos para esta ventana, en la parte privada de la clase, utilizando la macro **DECLARE_EVENT_TABLE()**. La segunda, debemos declarar la aplicación, fuera de cualquier clase y como última instrucción del fichero, utilizando la macro **DECLARE_APP(*nombre_clase*)**.

Ahora debemos implementar todo esto en el fichero ".cpp", pero antes debemos declarar siempre la implementación de la aplicación con la macro **IMPLEMENT_APP(*nombre_clase*)** y la tabla de eventos, donde debemos enlazar cada uno de los eventos con su correspondiente manejador.

A continuación, implementamos todos y cada uno de los métodos de las clases que hayamos definido anteriormente.

Capítulo 3

Componentes Básicos

3.1. wxApp, wxFrame y wxButton

wxApp

Todas las aplicaciones de wxWidgets están definidas en clases derivadas de la clase wxApp. Una aplicación en ejecución no es más que una instancia de ella.

Como ya hemos visto en el capítulo anterior, cada clase que definamos derivada de wxApp debe de tener un método *OnInit()* que será llamado cuando wxWidgets esté listo para ejecutar el código. Sería equivalente a las funciónes *main()* de C o *WinMain()* de Win32.

Aquí tenemos un ejemplo de como se definiría una clase desde wxApp:

```
class MyApp : public wxApp
{
  public:
     virtual bool OnInit();
};
```

La implementación del método *OnInit()* contendrá las sentencias que deseamos que se ejecuten al cargar la aplicación, o sea, lo que podríamos llamar "la configuración inicial"de la aplicación. Esta función devolverá un valor booleano en función de si se ha ejecutado correctamente o no.

Entonces, ¿en qué parte del código debemos decir que creamos una instancia de esta clase? wxWidgets la crea internamente, pero nosotros debemos decirle que clase de objeto deseamos crear. Por lo que debemos utilizar una macro en nuestro fichero de código, **IMPLEMENT_APP(*nombre_clase*)**.

Cuando wxWidgets crea un objeto aplicación derivado de wxApp se le asigna el resultado a la variable global wxTheApp. Esta se puede usar en la aplicación, pero no es conveniente. Para ello podemos insertar la macro **DECLARE_APP(*nombre_clase*)** después de la definición de la clase en el fichero de cabecera. Esto nos permite usar la función *wxGetApp()* que nos devuelve una referencia al objeto aplicación que hemos creado.

wxFrame

Es una ventana de bajo nivel, una clase derivada de wxWindow. Puede contener otras ventanas y suele tener barra de menú y barra de título.

Su definición sería de la siguiente manera:

```
class MyFrame : public wxFrame
{
public:
     //Constructor.
     //Manejadores de eventos.
private:
     DECLARE_EVENT_TABLE()
};
```

El prototipo del constructor es:

```
wxFrame(wxWindow* parent, wxWindowID id, const wxString& title,
        const wxPoint& position, const wxSize& size,
        long style, const wxString& name);
```

Los eventos que genera esta clase son los siguientes: **EVT_ACTIVATE**, cuando se activa o desactiva el wxFrame, **EVT_CLOSE**, cuando se intenta cerrar el wxFrame (ya sea el usuario o el sistema), **EVT_ICONIZE**, cuando se intenta minimizar o restaurar, y **EVT_MAXIMIZE**, cuando se maximiza.

Algunos de los principales métodos de la clase wxFrame son: *CreateStatusBar()*, *SetStatusText()* y *SetStatusWidths()* para gestionar la barra de estado, *CreateToolBar()*, *GetMenuBar()* y *SetMenuBar()* para gestionar barras de herramientas, *GetTitle()* y *SetTitle()* para ver y editar el título, *Iconize()*, *IsIconized*, *Maximize()* e *IsMaximized* para comprobar y modificar el estado de la ventana, *SetIcon()* establece el icono de la ventana y *ShowFullScreen()* muestra el componente al tamaño máximo posible.

wxButton

Este componente es un control representado como un botón pulsable con una etiqueta de texto

Para crearlo debemos pasarle al constructor la ventana padre, el identificador, la etiqueta, la posición, el tamaño y el estilo.

```
wxButton(wxWindow* parent, wxWindowID id,
         const wxString& label = wxEmptyString,
         const wxPoint& pos = wxDefaultPosition,
         const wxSize& size = wxDefaultSize, long style = 0)
```

Este componente cuentas con numerosos estilos: **wxBU_LEFT**, **wxBU_RIGHT**, **wxBU_TOP**, **wxBU_BOTTOM**, para alinear el texto respecto al botón, **wxBU_EXACTFIT**, que crea el botón de tamaño mínimo para que entre el texto, y **wxNO_BORDER**, que crea un botón sin bordes.

El único evento que genera este componente es **EVT_BUTTON**, cuando se pulsa el botón.

Este componente tiene pocos métodos, SetLabel() y GetLabel(), para ver y modificar la etiqueta, y SetDefault(), para restaurar el botón a uno prefeterminado.

Además, este componente cuenta con las llamadas etiquetas predeterminadas, que son una serie de macros ya definidas que podemos utilizar como identificador del botón, modificando automáticamente la etiqueta del mismo. A continuación se muestran unas cuantas: **wxID_ADD**, **wxID_CANCEL**, **wxID_CLOSE**, **wxID_DELETE**, **wxID_NO**, **wxID_OK**, **wxID_EXIT**, **wxID_SAVE**, **wxID_YES**, etc.

3.2. wxStaticText y wxTextCtrl

wxStaticText

Este componente muestra una o más líneas de texto en modo sólo lectura.

Para crear este componente, debemos pasar al constructor, la ventana padre, el identificador, la etiqueta, la posición, el tamaño y el estilo.

```
wxStaticText(wxWindow* parent, wxWindowID id, const wxString& label,
             const wxPoint& pos = wxDefaultPosition,
             const wxSize& size = wxDefaultSize, long style = 0)
```

Los estilos que podemos usar en este componente son los siguientes: **wxALIGN_LEFT**, **wxALIGN_RIGHT**, **wxALIGN_CENTRE** y **wxALIGN_CENTER**, para centrar el texto en la etiqueta, y **wxST_NO_AUTORESIZE**, para que la etiqueta no se ajuste al texto y siempre mantenga su tamaño original (normalmente se ajusta al texto).

Al ser un componente tan simple, sólo tiene dos métodos destacables: *GetLabel()* y *SetLabel()*, que sirven para ver y modificar el contenido de la etiqueta.

wxTextCtrl

Este componente es parecido al anterior, pero en este caso podemos editar y escribir texto en él.

Para crearlo, debemos pasar al constructor, la ventana padre, el identificador, la etiqueta, la posición, el tamaño y el estilo, además del texto inicial que queremos que muestre.

```
wxTextCtrl(wxWindow* parent, wxWindowID id, const wxString& value = "",
           const wxPoint& pos = wxDefaultPosition,
           const wxSize& size = wxDefaultSize, long style = 0)
```

Existen multitud de estilos para este componente, a continuación veremos algunos: **wxTE_PROCESS_ENTER** y **wxTE_PROCESS_TAB** para controlar el cambio de un wxTextCtrl a otro con las teclas de enter y tabulación respectivamente, **wxTE_MULTILINE**, para que permita varias líneas de texto, **wxTE_PASSWORD**, para que muestre asteriscos en lugar de lo que escribimos, **wxTE_READONLY**, para que sea de sólo lectura, **wxHSCROLL**, para que muestre una barra horizontal, **wxTE_LEFT**, **wxTE_CENTRE**, **wxTE_RIGHT**, para la alineación del texto, y **wxTE_NO_VSCROLL** para eliminar la barra vertical que por defecto muestra.

Son tres eventos los que este componente genera, **EVT_TEXT**, cuando el texto cambia, **EVT_TEXT_ENTER**, cuando se pulsa la tecla enter, y **EVT_TEXT_MAXLEN**, cuando el texto introducido supera el límite indicado.

De los muchos métodos que tiene este componente destacaremos los siguientes: *AppendText()*, para

introducir texto al final del existente (si lo hay), *WriteText()*, con el que se puede escribir en cualquier punto del mismo, *SetValue()* y *GetValue()*, que trabajan con el texto completo, *GetLineText()*, que utiliza líneas para mostrar el texto, *Copy()*, *Cut()* y *Paste()*, que trabajan con el portapapeles, *GetNumberOfLines()*, que devuelve el número de líneas, *Clear()*, para limpiar el texto, y *LoadFile()* y *SaveFile()*, para trabajar con ficheros.

3.3. wxMenu

Un menú es una lista de comandos que se despliegan de una barra de menu o de una ventana arbitraria.

Un item de un menú puede ser un comando normal, o un comando seleccionable con una etiqueta asociada. También existen separadores para una mejor organización.

Para crear un menú sólo debemos de pasarle el nombre que llevará y el estilo.

```
wxMenu(const wxString& title = "", long style = 0)
```

Los eventos más importantes que genera son los siguientes: **EVT_MENU** y **EVT_MENU_RANGE**, que se generan dependiendo de si se activa un comando, o varios, **EVT_UPDATE** y **EVT_UPDATE_UI_RANGE**, que se activan cuando un comando seleccionable, o varios para el segundo, cambia su estado, y **EVT_MENU_OPEN** y **EVT_MENU_CLOSE**, que se generan cuando el menu se abre/despliega o se cierra/pliega.

La principal función de este componente es *Append()*, que añade items o componentes al menú. Además podemos encontrar otras como: *AppendCheckItem()* o *AppendRadioItem()*, para insertar items de estos tipos en el menú, *AppendSeparator()*, para insertar un separador, *Insert()* e *InsertSeparator()*, para insertar en una posición determinada, *Remove()*, para eliminar items, *GetTitle()* y *SetTitle()* para trabajar con el nombre del menú, *GetMenuItems()* que devuelve una lista con los items del menú, *GetMenuCount()*, que devuelve el número de items, etc.

Capítulo 4

Componentes de Imagen

4. Componentes imagen

wxImage

Este componente encapsula una imagen en una plataforma independiente. Una imagen puede ser creada desde datos, o usando el método *ConvertToImage()*, del componente wxBitmap, que veremos más adelante. Una imagen puede ser cargada desde un fichero en variedad de formatos.

Al existir más de una forma de crear una imagen, veremos más de un constructor, el primero es a partir de un wxBitmap:

```
wxImage(const wxBitmap& bitmap)
```

El segundo, a partir de un archivo, donde además de este, indicaremos el tipo y el índice (el índice servirá para ordenas todas las imágenes que tengamos):

```
wxImage(const wxString& name, long type = wxBITMAP\_TYPE\_ANY, int index = -1)
```

Y por último, mostraremos como crear una imagen nueva, a partir de su anchura y su altura, indicando además si la queremos en negra (true) o sin color (false):

```
wxImage(int width, int height, bool clear=true)
```

Existen manejadores predeterminados para las imágenes, como por ejemplo: **wxBMPHandler**, **wxPNGHandler**, **wxJPEGHandler**, **wxGIFHandler** o **wxTIFFHandler**, entre otros.

Además, este componente cuenta con métodos como: *ConvertToMono()*, para convertir una imagen en monocromática, *Copy()* y *Paste()*, para copiar una imagen o pegarla en un lugar determinado, *Scale()* y *Rescale()*, para escalarla, *Rotate()*, para rotarla, *LoadFile()* y *SaveFile()*, para trabajar con ficheros, *GetWidth()* y *GetHeight()*, que devuelven la anchura y altura de la imagen, y *Size()* y *Resize()*, que trabajan con el tamaño de la imagne.

wxBitmap

Esta clase encapsula el concepto de plataforma dependiente de mapa de bits, ya sea monocromo, en color o soportado por algún canal.

Como en el caso anterior, también existen varios constructores. El primero, es a partir de datos dados, siempre que se le indique también, el tamaño (anchura y altura), el tipo y la profundidad:

```
wxBitmap(void* data, int type, int width, int height, int depth = -1)
```

También podemos crear un nuevo Bitmap, con el siguiente constructor, pasandole la anchura, altura y profundidad:

```
wxBitmap(int width, int height, int depth = -1)
```

Y por último, podemos crearlo a partir de una imagen

```
wxBitmap(const wxImage& img, int depth = -1)
```

Existen diferentes tipos predeterminados, que podemos pasara como parámetro al constructor, según el tipo de imagen que queramos crear, como por ejemplo: **wxBITMAP_TYPE_BMP**, **wxBITMAP_TYPE_JPEG**, **wxBITMAP_TYPE_TIF**, **wxBITMAP_TYPE_PNG**, **wxBITMAP_TYPE_GIF**, **wxBITMAP_TYPE_XPM**, etc.

Los métodos que destacamos de este componente son los siguientes: *ConvertToImage()*, que convierte un wxBitmap en un wxImage, *CopyFromIcon()*, que crea un wxBitmap desde un wxIcon, *GetWidth()* y *GetHeight()*, que devuelven el tamaño del componente, *GetDepth()*, que devuelve la profundidad, y *LoadFile()* y *SaveFile()*, que trabajan con archivos.

wxStaticBitmap

Este componente muestra un wxBitmap. Podemos utilizarlo para mostrar iconos en cuadros de diálogos, pero no como componente para mostrar imágenes de forma común, ya que bajo Windows el tamaño de los wxBitmaps está limitado a 64x64 pixeles.

Para crearlo, debemos de pasar al constructor, la ventana padre, el identificador, la etiqueta, la posición, el tamaño y el estilo.

```
wxStaticBitmap(wxWindow* parent, wxWindowID id, const wxBitmap& label,
               const wxPoint& pos = wxDefaultPosition,
               const wxSize& size = wxDefaultSize, long style = 0)
```

Destacamos pocos métodos para este componente, principalmente *GetBitmap()*, *SetBitmap()*, *GetIcon()* y *SetIcon()*, que permiten trabajar tanto con objetos wxBitmaps como con wxIcon.

wxBitmapButton

Este componente es un botón simple que contiene un bitmap, o lo que sería lo mismo, un wxButton con un wxBitmap como etiqueta. Permite insertar diferentes wxBitmap para los diferentes estados posibles del botón, estos son: normal, desactivado, seleccionado, con el foco y con el ratón encima (sin pulsar).

Para crearlo, necesitamos pasar al constructor, los parámetros habituales, la ventana, identificador, posición, tamaño y estilo, y además el wxBitmap que deseemos.

```
wxBitmapButton(wxWindow* parent, wxWindowID id, const wxBitmap& bitmap,
               const wxPoint& pos = wxDefaultPosition,
               const wxSize& size = wxDefaultSize, long style = wxBU_AUTODRAW)
```

Los estilos que acepta este componente son: **wxBU_AUTODRAW**, que crea un botón con un estilo por defecto, y **wxBU_LEFT**, **wxBU_RIGHT**, **wxBU_TOP** y **wxBU_BOTTOM**, para seleccionar la alineación del wxBitmap en el botón.

Al ser un botón, su único evento es **EVT_BUTTON**, como el de estos.

Los métodos son para ver y modificar los wxBitmap de cada uno de los posibles estados del botón: *GetBitmapDisabled()*, *GetBitmapFocus()*, *GetBitmapHover()*, *GetBitmapLabel()*, *GetBitmapSelected()*, *SetBitmapDisabled()*, *SetBitmapFocus()*, *SetBitmapHover()*, *SetBitmapLabel()* y *SetBitmapSelected()*.

Capítulo 5

Componentes de Elección

5.1. wxChoice, wxComboBox y wxListBox

wxChoice

El componente wxChoice es un cuadro de texto de sólo lectura en la que se muestra la opción elegida de las posibles, que se muestran en una lista desplegable. La lista desplegable sólo se muestra cuando se hace click en el componente, quedando oculta el resto del tiempo.

Para crear este componente debemos pasar al constructor la ventana padre, el identificador, la posición, el tamaño y estílo y como específico del componente un array de cadenas con la lista de opciones a elegir.

```
wxArrayString strings;
strings.Add(wxT("Opción 1"));
strings.Add(wxT("Opción 2"));
strings.Add(wxT("Opción 3"));
wxChoice* choice = new wxChoice(panel, ID_COMBOBOX, wxDefaultPosition,
                                wxDefaultSize, strings);
```

En la mayoría de plataformas, este componente es similar al wxComboBox, que veremos a continuación, excepto porque en este el usuario no puede editar el texto. Lo cual puede subsanarse configurando el wxComboBox en modo sólo lectura.

wxChoice no tiene estilos propios, y genera el evento **EVT_CHOICE** cuando el usuario selecciona un elemento de la lista, procesado como un **wxEVT_COMMAND_CHOICE_SELECTED**.
Los métodos de los que cuenta esta clase son por ejemplo, *Clear()* para limpiar todas las opciones, *Delete()* para eliminar una de ellas, *Insert()* para insertar una opción nueva, *FindString()* para buscar una determinada opción, *GetCount()* para obtener el número de opciones disponibles, *IsEmpty()* para ver si hay alguna opción seleccionada y varios métodos para establecer y editar opciones como *GetSelection()*, *SetSelection()*, *GetString()*, *SetString()*, *GetStringSelection()* o *SetStringSelection()*.

wxComboBox

El componente wxComboBox es una combinación de dos componentes, un wxListBox (que veremos a continuación) y un campo de texto (wxStaticText) que te permite establecer y obtener el texto de manera

totalmente independiente del wxListBox.

El campo de texto puede ser de sólo lectura, actuando como un wxChoice como vimos antes. Normalmente la lista está oculta hasta que el usuario hace click o pulsa un botón. Esto es una eficiente manera de que el usuario pueda tanto introducir una opción como seleccionarla de una lista de posibles.

Para crear el wxComboBox, debemos pasar al constructor la ventana padre, el identificador, la posición, el tamaño y estilo y además, el string con las opciones que queremos mostrar y si queremos o no que haya una opción incial establecida.

```
wxArrayString strings;<br>
strings.Add(wxT("Opción 1"));<br>
strings.Add(wxT("Opción 2"));<br>
strings.Add(wxT("Opción 3"));<br>
strings.Add(wxT("Opción 4"));<br>
wxComboBox* combo = new wxComboBox(panel, ID_COMBOBOX, wxT("Opción 1"),
                                   wxDefaultPosition, wxDefaultSize,
                                   strings, wxCB_DROPDOWN);
```

Los estilos con los que cuenta esta clase son **wxCB_SIMPLE** que crea un wxComboBox ordinario, **wxCB_DROPDOWN** que crea el wxComboBox con la lista desplegable hacia abajo, **wxCB_READONLY** que es como la anterior pero el campo de texto es de sólo lectura, por lo que sólo pueden seleccionarse las opciones predefinidas, y por último **wxCB_SORT**, que ordena la lista alfabéticamente.

Los eventos disponibles para este componente son dos, **EVT_TEXT** para cuando el texto es editado y **EVT_COMBOBOX** para cuando se selecciona una de las opciones predefinidas.

Los métodos con los que cuenta este componente son los mismos que los de wxChoice, añadiendo algunos como *Copy()* que copia el texto del campo de texto al portapapeles, *Paste()* que pega el texto del portapapeles en el campo de texto, *GetValue()* que obtiene el valor del campo de texto y *SetValue()*, que lo establece, *SetSelection()* que selecciona el texto del campo de texto entre dos posiciones dadas, *Replace()* que reemplaza el texto entre dos posiciones dadas por otro dado o *Remove()* que borra el texto entre dos posiciones dadas.

wxListBox

Un wxListBox se usa para seleccionar una o más opciones de una lista, numeradas desde cero. Las opciones se muestran en un cuadro con barra de desplazamiento, con las opciones marcadas en color inverso. Un wxListBox puede ser monoseleccionable, permitiendo sólo una selección posible, o multiseleccionable, permitiendo más de una.

El constructor funciona de la misma manera que el del wxComboBox, pasandole la ventana padre, el identificador, la posición, el tamaño, el estilo y el string con la lista de opciones.

```
wxArrayString strings;
strings.Add(wxT("Primera opción"));
strings.Add(wxT("Segunda opción"));
strings.Add(wxT("Tercera opción"));
strings.Add(wxT("Cuarta opción"));
wxListBox* listBox = new wxListBox(panel, ID_LISTBOX, wxDefaultPosition,
                                   wxSize(180, 80), strings, wxLB_SINGLE);
```

Los estilos que podemos usar son **wxLB_SINGLE**, que crea un wxListBox ordinario (monoselección), **wxLB_MULTIPLE**, que permite la selección múltiple, **wxLB_EXTENDED**, que ademas de permitir más de una permite usar atajos de teclado para la selección, **wxLB_HSCROLL**, que crea una barra de desplazamiento horizontal si el contenido es demasiado ancho, **wxLB_ALWAYS_SB**, que siempre muestra una barra vertical (normalmente se inserta automáticamente cuando se sobrepasa un número determinado de opciones), **wxLB_NEEDED_SB**, que sólo crea la barra vertical cuando es necesaria y **wxLB_SORT**, que ordena la lista de opciones alfabéticamente.

Respecto a los eventos, existen dos, **EVT_LISTBOX**, que se genera cuando se selecciona un item, y **EVTLISTBOX_DCLICK**, que se genera cuando se hace doble click en una de las opciones.

Por último, los métodos con los que cuenta son: *Deselect()*, para quitar la selección de una opción, *GetSelections()*, que devuelve un vector de enteros con las posiciones de los elementos seleccionados, *InsertItems()*, que inserta los items dados antes de la posición dada, *Selected()* que devuelve un booleano dependiendo de si la posición dada está ocupada o no y *Set()* que vacia el wxListBox y añade las opciones dadas en un string.

Además, este componente cuenta con todos los métodos del componente wxChoice.

5.2. wxCheckBox y wxRadioBox

wxCheckBox

Un wxCheckBox es un componente que normalmente tiene dos estados posibles, activado o desactivado (se pueden llamar de diferentes maneras). Se representa como una caja conteniendo una cruz, si está seleccionada, o vacía, si no lo está. Opcionalmente, puede tener un tercer estado, no permitiendo ni la activación ni la desactivación de la opción.

El constructor sería así de simple, pasandole la ventana padre, el identificador, el nombre de la opción, la posición, el tamaño y el estilo.

```
wxCheckBox* checkbox = new wxCheckBox(panel, ID_CHECKBOX, wxT("&Check me"),
                                      wxDefaultPosition, wxDefaultSize);
```

Los estilos disponibles para wxCheckBox son: **wxCHK_2STATE**, que crea un wxCheckBox de dos estados (por defecto), **wxCHK_3STATE**, que crea un wxCheckBox de tres estados, **wxCHK_ALLOW_3RD_STATE_FOR_USER**, permite al usuario cambiar al tercer estado haciendo click (no puede usar el tercer estado por defecto), y **wxALIGN_RIGHT**, que crea un wxCheckBox pero con la caja a la derecha de la etiqueta (por defecto está en el lado izquierdo).

Este componente sólo genera un evento, **EVT_CHECKBOX**, que se genera cada vez que el usuario hace click en el wxCheckBox y cambia de estado.

Respecto a los métodos, *SetLabel()* y *GetLabel()* dan acceso a la etiqueta (permite el uso de teclas de acceso rápido con el '&'), *GetValue()* y *SetValue()* trabajan con booleanos para activar o desactivar el wxCheckBox en sus dos estados principales y *Set3StateValue()* y *Get3StateValue()* trabajan con el tercer estado.

Además, existen los métodos consultores *IsChecked()* e *Is3State()* para ver el estado del wxCheckBox.

wxRadioBox

Un componente wxRadioBox se usa para seleccionar una opción, y sólo una, de entre varias. Se muestra en una columna vertical o en una fila horizontal de botones con etiquetas encerrados en un marco con o sin etiqueta.

El constructor es fácil de manejar, sólo hay que pasarle la ventana padre, el identificador, la etiqueta del marco, la posición, el tamaño, un string con las opciones y el número de filas y de columnas máximas (a la hora de mostrarlo).

```
wxArrayString strings;<br>
strings.Add(wxT("&One"));<br>
strings.Add(wxT("&Two"));<br>
strings.Add(wxT("T&hree"));<br>
strings.Add(wxT("&Four "));<br>
strings.Add(wxT("F&ive "));<br>
strings.Add(wxT("&Six "));<br>
wxRadioBox* radioBox = new wxRadioBox(panel, ID_RADIOBOX, wxT("Radiobox"),
                                      wxDefaultPosition, wxDefaultSize,
                                      strings, 3, wxRA_SPECIFY_COLS);
```

Existen dos estilos para este componente, **wxRA_SPECIFY_ROWS**, que especifica el número dado como el máximo número de filas para mostrar, y **wxRA_SPECIFY_COLS**, que especifica el número dado como el máximo número de columnas para mostrar.

Sólo tenemos un evento en este componente, **EVT_RADIOBOX**, que se lanza cuando el usuario hace click en alguno de las opciones.

Para terminar, los métodos que dispone este componente son: *Enable()*, que con un índice y un booleano activa o desactiva las diferentes opciones, *FindString()*, que devuelve el index de una cadena buscada (si existe en el wxRadioBox), *GetCount()*, que devuelve el número de opciones que hay, *GetString()* y *SetString()* que permiten acceder a la etiqueta de una opción según un índice, *GetLabel()* y *SetLabel()* que acceden a la etiqueta del wxRadioBox, *GetSelection()* que devuelve el índice de la opción seleccionada, *GetStringSelectcion()*, que devuelve la etiqueta de la opción seleccionada, *SetSelection()* y *SetStringSelection()* que modifican la opción seleccionada sin generar un evento, y *Show()*, que permite mostrar u ocultar las opciones dentro del wxRadioBox.

Capítulo 6

Componentes Contenedores

6.1. wxPanel

Este componente es un contenedor, diseñado para albergar tanto a otros elementos visuales, como a otras ventanas hijas o gráficos dibujados.

El constructor de este componente no difiere mucho de los vistos hasta ahora. Debemos proporcionarle la ventana padre, el identificador, el tamaño, la posición, el estilo, y por último, el nombre.

```
wxPanel(wxWindow* parent, wxWindowID id,
        const wxPoint& pos = wxDefaultPosition,
        const wxSize& size = wxDefaultSize,
        long style = wxTAB_TRAVERSAL|wxNO_BORDER,
        const wxString& name = wxT("panel"))
```

Este componente no cuenta con estilos propios, pero puede utilizar los estilos del componente wxWindow, algunos de los cuales son: **wxSIMPLE_BORDER**, **wxDOUBLE_BORDER** o **wxNO_BORDER**, para la existencia o no de bordes, **wxTAB_TRAVERSAL**, para permitir el cambio entre componentes wxPanel con la tecla "tabulador", **wxWANTS_CHARS**, para activar todos los eventos producidos por teclas o combinaciones de ellas, o **wxHSCROLL** y **wxVSCROLL**, para mostrar barras de desplazamientos.

Tampoco cuenta con métodos propios, pero también podemos utilizar los del componente wxWindow, ya que wxPanel hereda de él. Alguno de los métodos más útiles son: *Centre()*, *CentreOnParent()* y *CentreOnScreen()*, para posicionar el componente, *Close()*, para cerrar, *Destroy()*, para cerrar y destruir el componente, *Enable()* y *Disable()*, para activar y desactivar, *GetFont()*, *SetFont()*, *GetName()*, *SetName()*, *GetPosition()*, *GetSize()*, *SetSize()*, para trabajar con las distintas propiedades del componente, *GetParent()*, que devuelve un puntero a la ventana padre, *GetWindowStyle()* y *SetWindowStyle()*, para trabajar con los estilos, *Move()*, para mover el componente, *SetFocus()*, para que el foco apunte a este componente, o *Show()* e *Hide()*, para mostrar y ocultar el componente.

6.2. wxNotebook

Esta clase representa una especie de bloc, con varias páginas, mostradas como pestañas, por las que se navega haciendo click en ellas o con la tabulación. Cada una de las páginas será un componente wxPanel o algún otro heredado de él, aunque existe la posibilidad de utilizar más componentes.

Este componente cambia un poco respecto de los vistos hasta ahora, por el hecho de que se puede requerir crear una o más páginas dentro de él. Lo primero será crear el wxNotebook, pasando al contructor los parámetros habituales: la ventana padre, el identificador, la posición y el tamaño.

A continuación creamos las páginas (componentes wxPanel) como hemos visto anteriormente y por último se las añadimos al wxNotebook utilizando la función *AddPage()*.

```
// Create the notebook
wxNotebook* notebook = new wxNotebook(parent, wxID_ANY,
                                      wxDefaultPosition,
                                      wxSize(300, 200));

// Create and add the pages
wxPanel1* window1 = new wxPanel(notebook, wxID_ANY);
wxPanel2* window2 = new wxPanel(notebook, wxID_ANY);
wxPanel3* window3 = new wxPanel(notebook, wxID_ANY);

notebook->AddPage(window1, wxT("Tab one"), true, 0);
notebook->AddPage(window2, wxT("Tab two"), false, 1);
notebook->AddPage(window3, wxT("Tab three"), false 2);
```

Este componente puede tener, además de alguno de los estilos del componente wxWindow, los siguientes estilos propios: **wxNB_TOP**, **wxNB_LEFT**, **wxNB_RIGHT** y **wxNB_BOTTON**, para indicar el lugar en donde aparecerán las pestañas de las distintas páginas, **wxNB_FIXEDWIDTH**, para que todas las pestañas tengan el mismo grosor, y **wxNB_MULTILINE**, que permite varias filas de pestañas.

El evento principal de este componente es **EVT_NOTEBOOK_PAGE_CHANGED**, que se genera cuando se cambia de una página a otra.

Respecto a los métodos con los que cuenta el componente wxNotebook, caben a destacar el método *AddPage()*, que añade páginas al wxNotebook (como hemos visto antes), que necesita como parámetros, el componente (normalmente un wxPanel), la cadena que mostrará la etiqueta, un booleano que nos dirá si será la página por defecto del notebook o no, y un entero que indicará el orden dentro de las páginas. Otros métodos son: *DeletePage()*, que elimina y destruye páginas, *RemovePage()*, que sólo elimina, *DeleteAllPages()* que elimina todas las páginas, *AdvanceSeletion()* que pasa a la siguiente página, *SetSelection()*, que muestra una página determinada, *GetPage()*, que que devuelve una página según un índice dado, *GetPageCount()* que devuelve el número de páginas, y *SetPageText()* y *GetPageText()* que trabajan con la etiqueta de la página.

6.3. wxHtmlWindow

Componente específico para mostrar texto en formato HTML, permitiendo mostrar gifs animados, tablas, hiperenlaces, fuentes, etc.

Para utilizarlo basta con crear como hasta ahora el componente y utilizar el método *LoadFile()* para cargar el archivo que deseemos. El constructor recibe la ventana padre, el identificador, la posición, el tamaño y los estilos.

```
wxHtmlWindow* html = new wxHtmlWindow(parent, wxID_ANY, wxDefaultPosition,
                                      wxSize(380, 160),
                                      wxHW_SCROLLBAR_NEVER)
```

Los estilos que permite este componente son **wxHW_SCROLLBAR_NEVER**, no mostrando nunca barras de desplazamiento, **wxHW_SCROLLBAR_AUTO**, que muestra barras de desplazamiento, cuanto son estrictamente necesaria porque el tamaño del texto excede del de la ventana, y **wxHW_NO_SELECTION**, que no permite al usuario seleccionar texto (normalmente sí puede).

Los métodos que podemos destacar de este componente son: *LoadFile()*, que como hemos dicho permite cargar un archivo html en el componente, *LoadPage()*, permite cargar urls, *SetPage()*, que devuelve en un string la ruta o url del archivo cargado.

Además cuenta con métodos para seleccionar texto, como: *SelectAll()*, *SelectLine()*, y *SelectWord()*, y método para devolver texto una vez seleccionado, como *SelectionToText()* o *ToText()*, devolviendo este último todo el contenido en texto plano.

Capítulo 7

Componentes de Diálogo

7.1. wxDialog

Un diálogo es una ventana de alto nivel, que se usa para presentar información al usuario, opciones o selecciones. Al contrario que las ventanas normales, este tipo puede no tener título, botones para minimizar o maximizar, etc.

Existen dos tipos de diálogos, modales y no modales. Los diálogos modales son aquellos que interrumpen el flujo natural del programa hasta que son respondidos, sin dar la posibilidad de realizar otra acción. Los no modales, son los que no tienen este comportamiento.

Para crear un diálogo, debemos pasar al constructor los siguientes parámetros: ventana padre, identificador, título, posición, tamaño y estilo.

```
wxDialog(wxWindow* parent, wxWindowID id, const wxString& title,
         const wxPoint& pos = wxDefaultPosition,
         const wxSize& size = wxDefaultSize,
         long style = wxDEFAULT_DIALOG_STYLE)
```

Este componente admite los siguienes estilos: **wxDEFAULT_DIALOG_STYLE**, que contiene tres estilos: **wxCAPTION**, que muestra una etiqueta, **wxCLOSE_BOX**, que añade el icono de cerrar, y **wxSYSTEM_MENU**, que añade un menú del sistema. **wxMINIMIZE_BOX**, que añade el icono de minimizar, **wxMAXIMIZE_BOX**, que añade el icono de maximizar, **wxSTAY_ON_TOP**, que hace que el diálogo se situe como la ventana enfocada más alta, **wxRESIZE_BORDER**, añade un borde alrededor de la ventana, **wxDIALOG_NO_PARENT**, que crea un diálogo "huerfano".

Este componente cuenta con cinco eventos, **EVT_ACTIVATE**, que se genera cuando se activa o desactiva el diálogo, **EVT_CLOSE**, que se genera cuando el usuario o el sistema intenta cerrar el diálogo, **EVT_ICONIZE**, que se genera cuando el diálogo es minimizado, **EVT_MAXIMIZE**, que se genera cuando el diálogo es maximizado, y **EVT_INIT_DIALOG**, que se genera cuando el diálogo se inicializa a si mismo.

Respecto a los métodos con los que cuenta este componente: *GetTitle()* y *SetTitle()*, para obtener y modificar el título del diálogo, *Iconize()* que minimiza y restaura el diálogo, *Maximize()*, que maximiza el diálogo, *ShowModal()* y *EndModal()*, que inicia y finaliza el diálogo como modal.

7.2. Dialogos de entrada

Estos diálogos preguntan información al usuario.

wxTextEntryDialog

Es un diálogo simple de una línea que permite escribir al usario.

```
wxTextEntryDialog(wxWindow* parent, const wxString& message,
                  const wxString& caption = "Please enter text",
                  const wxString& defaultValue = "",
                  long style = wxOK | wxCANCEL | wxCENTRE,
                  const wxPoint& pos = wxDefaultPosition)
```

Los estilos aplicables son pocos, y permiten mostrar botones en el diálogo como OK (**wxOK**) o cancel (**wxCANCEL**), y mostrarlos centrados (**wxCENTER**).

Podemos utilizar métodos como *GetValue()* o *SetValue()*, para obtener o establecer el texto en el diálogo, y *ShowModal()*, para lanzarlo como modal.

wxPasswordEntryDialog

Es como el anterior, pero no permite ver al usuario lo que escribe.

```
wxPasswordEntryDialog(wxWindow* parent, const wxString& message,
                      const wxString& caption = "Enter password",
                      const wxString& defaultValue = "",
                      long style = wxOK | wxCANCEL | wxCENTRE,
                      const wxPoint& pos = wxDefaultPosition)
```

Los estilos aplicables son exactamente los mismos que en el componente anterior, permitiendo mostrar botones en el diálogo como OK (**wxOK**) o cancel (**wxCANCEL**), y mostrarlos centrados (**wxCENTER**).

Podemos utilizar métodos como *GetValue()* o *SetValue()*, para obtener o establecer el texto en el diálogo, y *ShowModal()*, para lanzarlo como modal.

wxFindReplaceDialog

Este componente representa a un diálogo no modal que permite al usuario buscar texto y reemplazarlo a su deseo

```
wxFindReplaceDialog(wxWindow * parent, wxFindReplaceData* data,
                    const wxString& title, int style = 0)
```

Tiene varios eventos: **EVT_FIND**, **EVT_FIND_NEXT**, **EVT_FIND_REPLACE**, **EVT_FIND_REPLACE_ALL** y **EVT_FIND_CLOSE**, que se generan con cada una de las opciones del diálogo, buscar, siguiente, reemplazar, reemplazar todos y cerrar, respectivamente.

El único método a destacar de este componente es *GetData()*, que devuelve el dato utilizado por el diálogo.

7.3. Dialogos de salida

Estos diálogos muestran información al usuario.

wxMessageDialog

Este componente muestra un diálogo con un mensaje y unos botones seleccionados que pueden ser, Sí, No, Aceptar y Cancelar.

```
wxMessageDialog(wxWindow* parent, const wxString& message,
               const wxString& caption = "Message box",
               long style = wxOK | wxCANCEL,
               const wxPoint& pos = wxDefaultPosition)
```

Podemos asignarle varios estilos, por ejemplo **wxOK**, **wxCANCEL**, **wxYES_NO**, para mostrar unos botones u otros, **wxYES_DEFAULT** y **wxNO_DEFAULT**, que se utilizan junto con **wxYES_NO** para indicar que opción es la elegida por defecto, **wxICON_EXCLAMATION**, que sirve para mostar una marca de exclamación para dar énfasis. Además podemos insertar algunos iconos predefinidos en el diálogo, con los estilos **wxICON_ERROR**, **wxICON_HAND**, **wxICON_QUESTION** o **wxICON_INFORMATION**.

Como único método, comentar *ShowModal()*, que ya sabemos que lanza el diálogo en modo modal y devuelve el valor del botón que pulse el usuario.

wxPrintDialog

Este componente es un diálogo especialmente implementado para imprimir con una impresora.

```
wxPrintDialog(wxWindow* parent, wxPrintDialogData* data = NULL)
```

Los métodos que utiliza son los siguientes: *GetPrintDialogData()*, que devuelve el dato asociado a la ventana (lo que deseamos imprimir), *GetPrintDC()*, que devuelve el dispositivo desde el que vamos a imprimir, y *ShowModal()*, que lanza el diálogo en modo modal y devuelve la respuesta que pulse el usuario.

7.4. Dialogos de fichero

Estos diálogos permiten obtener información al usuario de ficheros y directorios.

wxFileDialog

Este componente sirve para manejar la selección de uno o más archivos, y realizar acciones como abrir o guardar.

Debemos pasar al constructor la ventana padre, el mensaje que queramos mostrar, un directorio, un archivo por defecto (si se desea), una máscara para el tipo de archivo (*.*, *.jpeg, etc.) y la posición.

```
wxFileDialog(wxWindow* parent, const wxString& message = "Choose a file",
             const wxString& defaultDir = "",
             const wxString& defaultFile = "",
             const wxString& wildcard = "*.*",
             long style = 0, const wxPoint& pos = wxDefaultPosition)
```

Tiene estilos propios, como **wxSAVE**, para guardar el archivo, **wxOPEN**, para abrirlo, **wxOVERWRITE_PROMPT**, que avisará si al archivo a guardar ya existe, **wxFILE_MUST_EXIT**, que forzosamente se tendrá que seleccionar un archivo existente, y **wxMULTIPLE**, que permite seleccionar varios archivos.

En cuanto a los eventos, destacamos: *GetDirectory()* y *SetDirectory()*, para trabajar con los directorios, *GetFilename()* y *SetFilename()*, para trabajar con el nombre del archivo, *GetFilenames()* y *GetPath()*, que sirven cuando la selección es multiple (*GetPath()* devuelve el conjunto directorio y archivo), *GetWildcard()* y *SetWildcard()*, para trabajar con la mascara, y *GetMessage()* y *SetMessage()* para cuando queramos trabajar con el mensaje del diálogo.

wxDirDialog

Este diálogo permite al usuario seleccionar un directorio, ya sea local o en red. Además con el estilo **wxDD_NEW_DIR_BUTTON**, te permite crear un nuevo directorio.

Para contruirlo, debemos pasar al constructor la ventana padre, el mensaje, el path (directorio y archivo), el estilo, la posición y el tamaño.

```
wxDirDialog(wxWindow* parent, const wxString& message = "Choose a directory",
            const wxString& defaultPath = "",
           long style = wxDD_DEFAULT_STYLE,
           const wxPoint& pos = wxDefaultPosition,
           const wxSize& size = wxDefaultSize)
```

Cuenta con dos estilos, **wxDD_DEFAULT_STYLE**, que crear un diálogo por defecto (dependiendo del sistema), y **wxDD_NEW_DIR_BUTTON**, que crea un botón en el diálogo que permite crear nuevos directorios.

Destacamos algunos métodos de este componente, como *GetPath()* y *SetPath()*, para obtener y modificar el par directorio-fichero, *GetMessage()* y *SetMessage()*, para trabajar con el mensaje que muestra, y *ShowModal()*, para mostrarlo como modal.