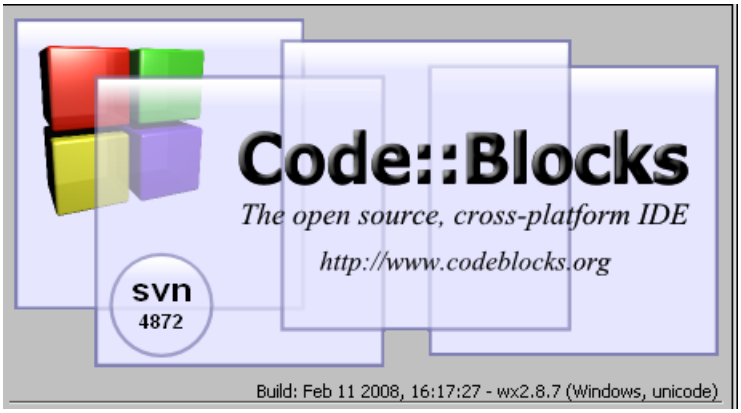
23/02/2008

Introducción a la programación con wxWidgets y Code::Blocks

Instalación y ejemplo.



Jhon James Quintero Osorio

Introducción a la programación con wxWidgets y Code::Blocks

Introducción

Existen muchas herramientas a la hora de programar aplicaciones en C++, que nos permiten construir interfaces de usuario, una de ellas es wxWidgets, un toolkit que contiene módulos para muchas de las necesidades, GUI's, redes, multithread entre otros. Dos de las características de los wxWidgets que los hacen especiales (comparados con VCL o MFC) son:

- Multiplataforma (Windows, Unix, MaxOs y otros). Adquiriendo la apariencia de la plataforma.
- Libres, licencia GPL.
- Orientados a objetos.
- Bonitos.

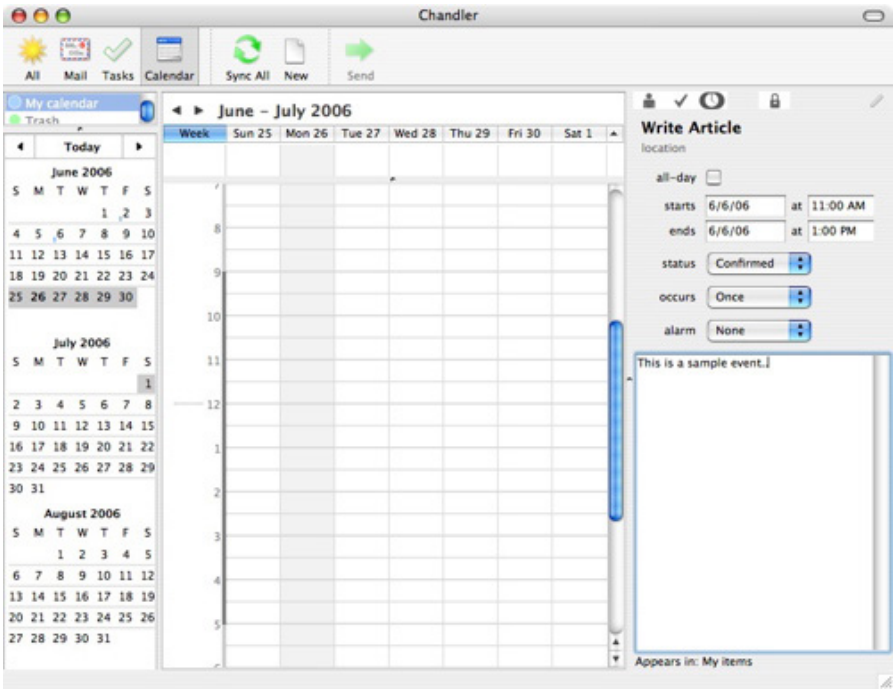


Figura 1 Ejemplo de aplicación con wxWidgets

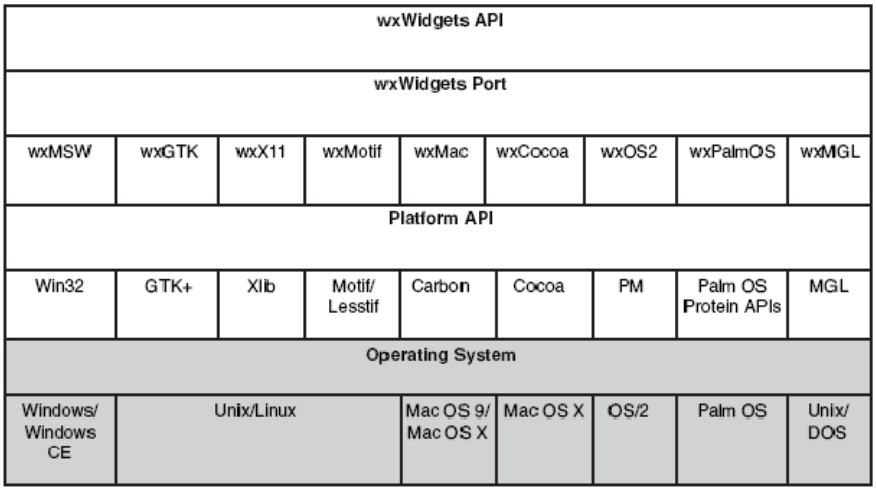


Figura 2 La estructura de los ports de wxWidgets.

Los wxWidgets están desarrollados en C++, y es posible programar con ellos usando varios compiladores de C++, GCC, Visual C++, Borland, etc, dado que, a diferencia de otros toolkits como Qt o VCL, estos no usan ninguna extensión de C++, usa solo C++ estándar.

En este documento se indica como tener un entorno de desarrollo ***libre*** para Windows en el que se usarán los wxWidgets para la creación de las interfaces de usuario (o lo que se necesite).

Lo que se necesita

- Un compilador de C++, usaremos *mingw*.
- Los *wxWidgets* para Windows, aquí la versión 2.8.
- *wxAdditions* (opcional), más widgets (scintilla, plots, led y otros).
- *wxFormBuilder*. Constructor de interfaces para wxWidgets, versión 3.0.
- *Code::Blocks*. Un IDE multiplataforma para programar en C/C++ (hecho con wxWidgets).

Instalando mingw

Es necesario del sitio web de mingw (usar un motor de búsqueda), bajar los siguientes instaladores:

- binutils.
- gcc-core.
- mingw-runtime.
- mingw-api
- gcc-g++.
- gdb
- mingw32-make

Generalmente estos archivos son archivos con extención *gz* por lo cual es necesario utilizar un programa de descompresión como winrar, winace o cualquiera que soporte este tipo de compresión.

Creamos un directorio digamos en C, llamado mingw y allí descomprimimos los archivos mencionados.

Instalando wxWidgets (wxAdditions) y wxFormBuilder

Para instalar los wxWidgets, wxAdditions y el wxFormBuilder para Windows es posible conseguir los instaladores individuales, sin embargo existe un instalador que contiene todo lo anterior denominado wxPack, este permite seleccionar que se desea instalar.

Code::Blocks

En este momento (febrero de 2008) no existe un instalador actualizado (pronto habrá uno disponible) pero es posible usar la versión del repositorio de versiones, en mi caso usé CB_20080211_rev4872_win32.7z, para que funcione correctamente se necesitan además, mingwm10_gcc421.7z y wxmsw28u_gcc_cb_wx287.7z, se descomprimen en algún directorio y listo. Para descomprimir este tipo de archivos (.7z) se puede conseguir un programa libre en la red llamado 7-Zip.

wxFormBuilder y los sizers

Existen dos fomas para el programador de poner los widgets en un contenedor, manualmente o usando algún algoritmo, la primera forma es la que se usa comúnmente en interfaces pequeñas en las que generalmente no se permite el redimensionamiento, pues tiene el inconveniente que un cambio de tamaño de hace que los widgets se queden las posiciones originales haciendo que la apariencia de la interfaz no se la mejor tal y como se muestra en las figuras siguientes.

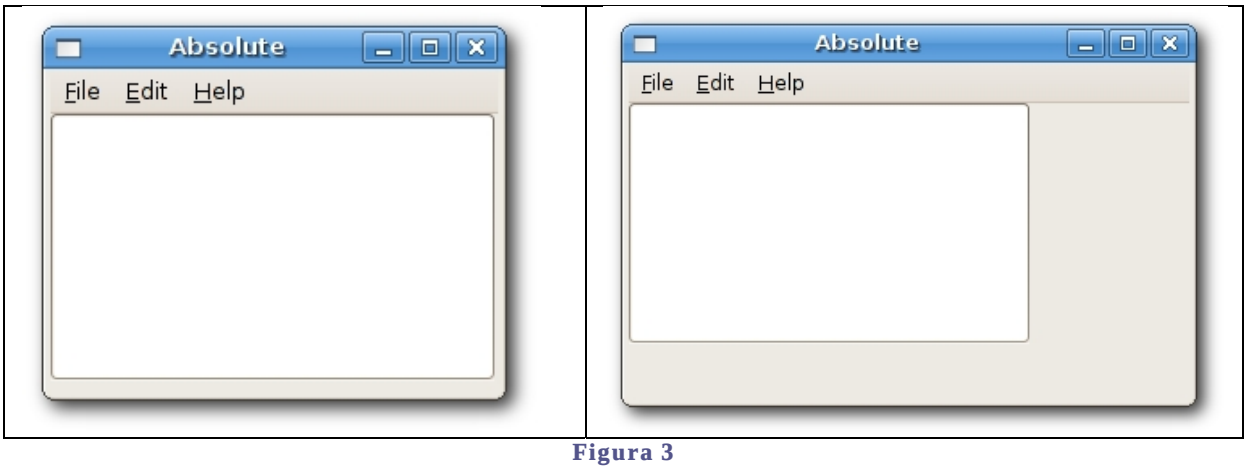


Figura 3

Además de lo mencionado un cambio de plataforma (o de tema en una misma plataforma) puede hacer que nuestra interfaz se convierta en una especie de mutante.

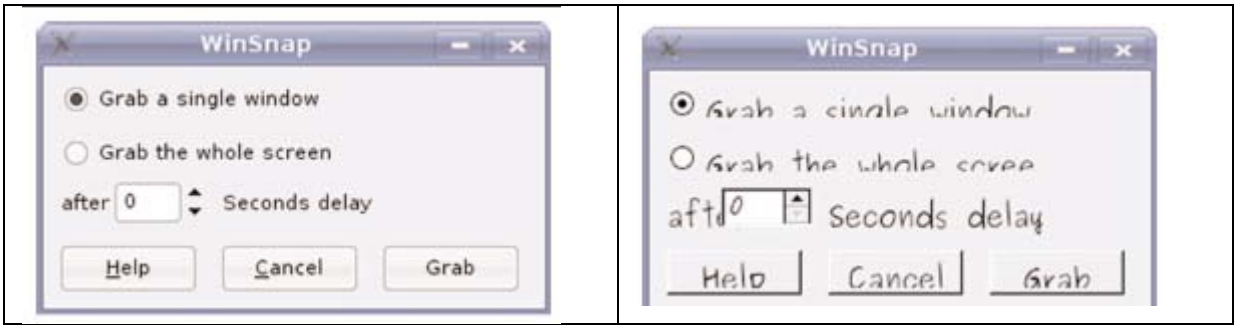


Figura 4 Efectos de cambio de plataforma

¿Cómo hacemos para que esto no suceda? Aja, la respuesta es con *layouts*, lo que en wxWidgets se denominan sizers.

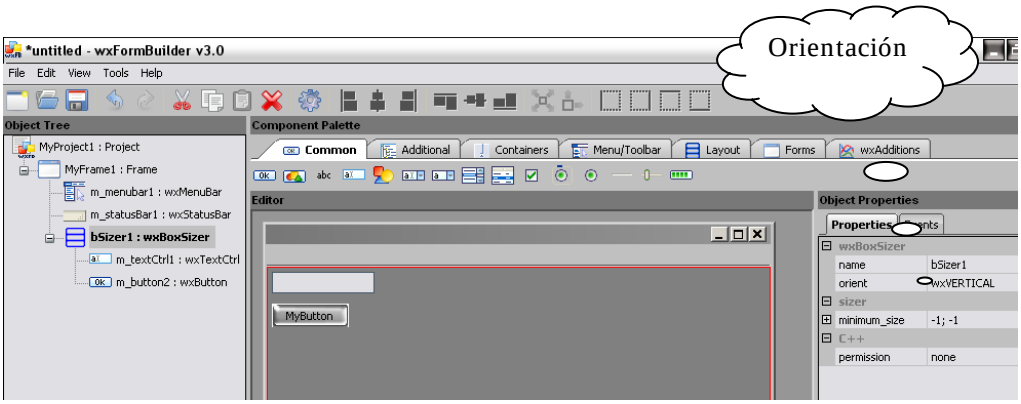
Trabajando con sizers en wxFormBuilder

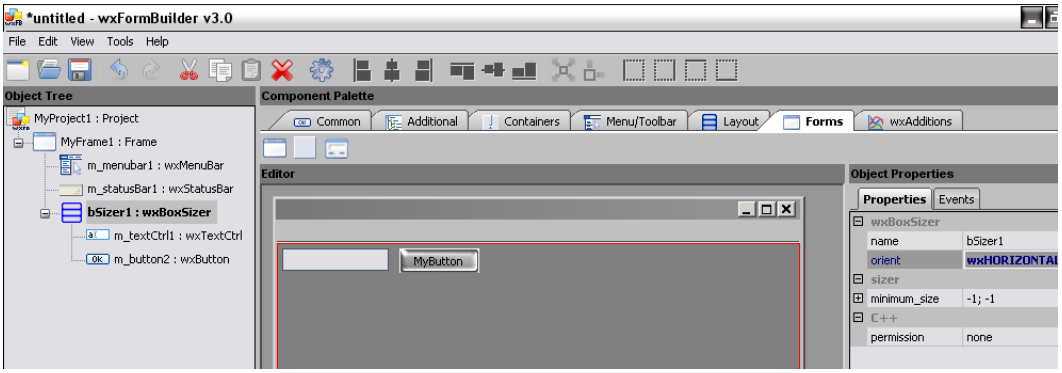
Los sizers, son objetos que se encargan de ubicar de forma automática los widgets en un contenedor, siguiendo unas deteminadas reglas para ello, las reglas están definidas según el tipo de sizer que se utilice. wxFormBuilder no permite ubicar los widgets usando posiciones absolutas, siempre es preciso usar sizers.

Cuando se abre el wxFormBuilder tenemos un proyecto *MyProyect1* vacio, para comenzar a agregar widgets debemos crear un Frame primero. A este último le podemos agregar un sizer el cual se encuentra en la pestaña de *Layout.*

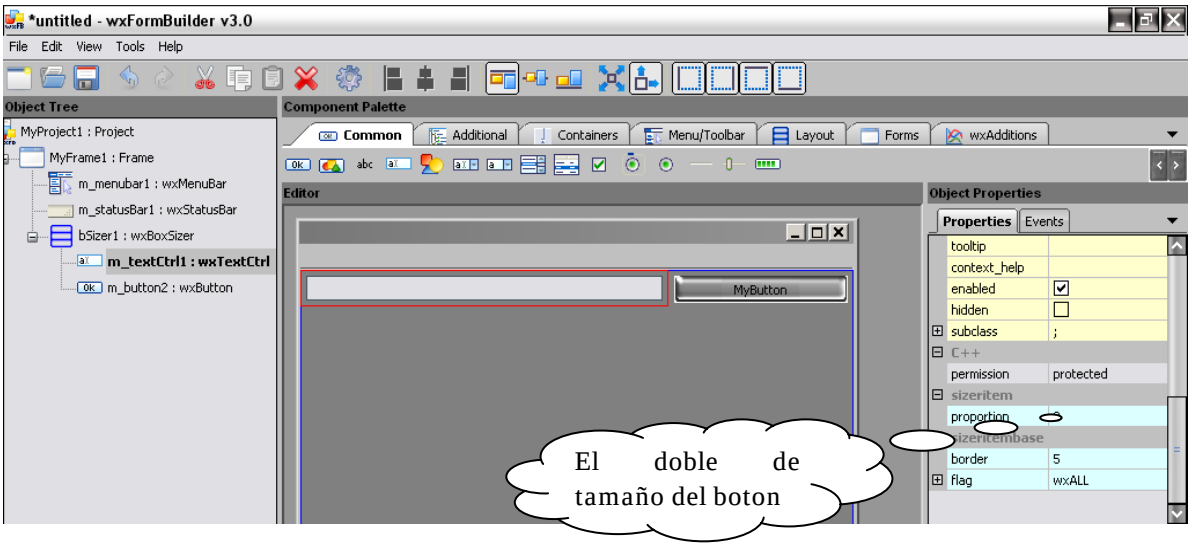
wxBoxSizer

Permite ubicar widgets en filas o columnas, dependiendo del valor de propiedad *orient*. Esto se muestra en las figuras siguientes





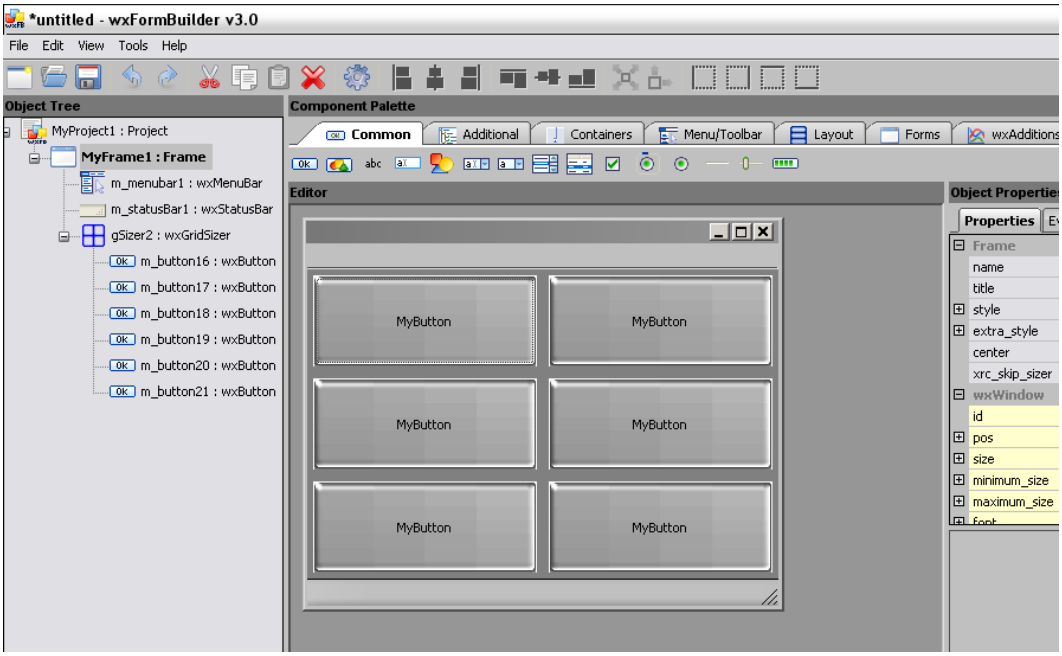
En wxFormBuilder es posible indicar la proporción que ocupa cada widgets en relación con los otros en un wxBoxSizer usando la propiedad *proportion* de cada ítem, por ejemplo modificándola en la caja de texto de la forma de las figuras anteriores de cero a uno (0-2) y de cero a uno (0-1) la del botón, obtenemos lo siguiente:



Es posible mediante los botones en la barra herramientas (o modificando los flags del sizer) alinear los widgets en la fila o columna dependiendo de la orientación del sizer.

wxGridSizer

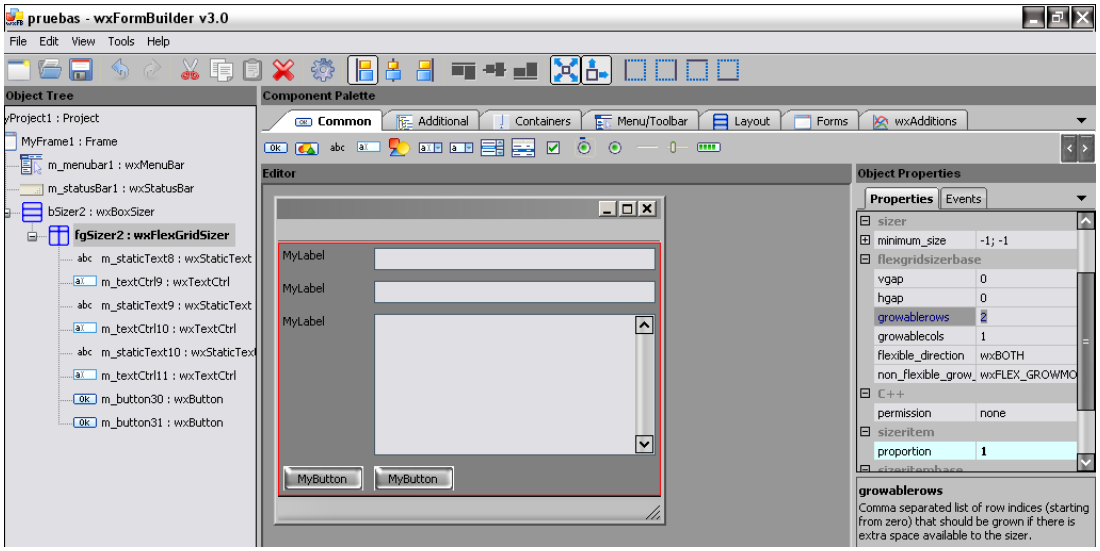
Este sizer se encarga de ubicar los widgets en cuadriculas. Los widgets son agregados por por filas cada vez que se llama el método Add.



Las celdas en el sizer tiene el mismo tamaño.

wxFlexGridSizer

Este tipo de sizer es similar al anterior con la ventaja de que permite que las columnas tengan anchos diferentes y que las filas tengan alturas diferentes. Para ello es necesario usar una lista en la que se inican las filas y columnas que pueden .....



Un ejemplo

Vamos a realizar un ejemplo con wxWidgets, wxFormBuilder y Code::Blocks, el ejemplo está basado en el desarrollado por el profesor Eduardo Giraldo que luce como se muestra a continuación.

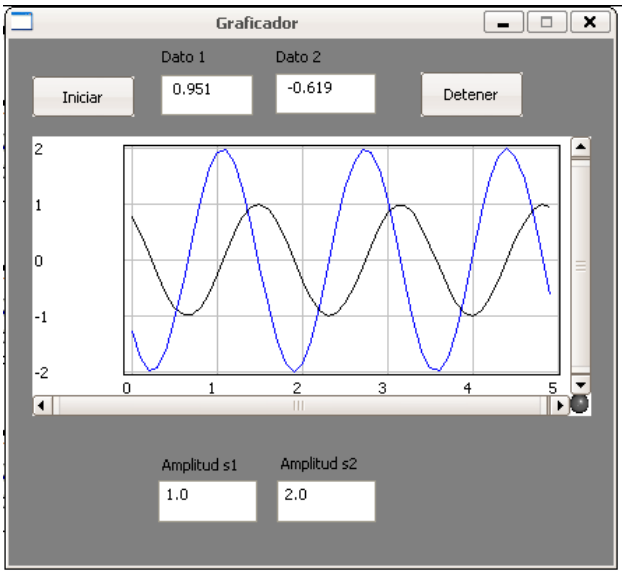


Figura 5 Programa de ejemplo (GTimer)

Lo primero que vamos a hacer es crear un nuevo proyecto wxWidgets con Code::Blocks

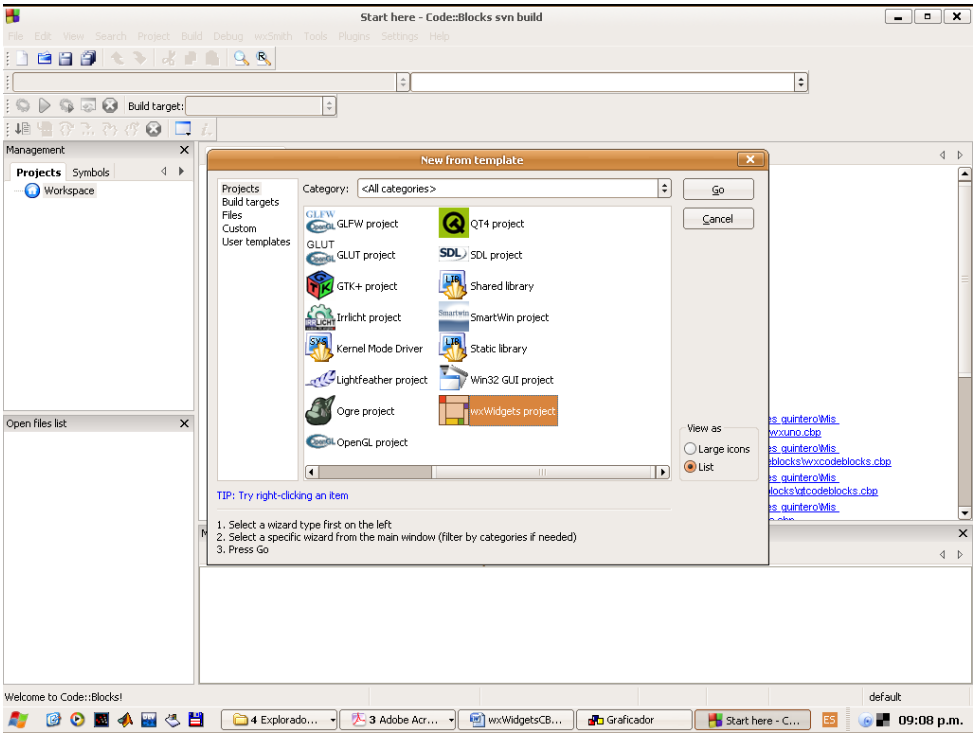


Figura 6 Nuevo proyecto wxWidgets en Code::Blocks

Damos click en *Go* y después de una pantalla de bienvenida al wizard nos pregunta por la versión de wxWidgets que se van a usar, en este caso 2.8, luego le indicamos al wizard el nombre del proyecto, en nuestro caso gtimer2.

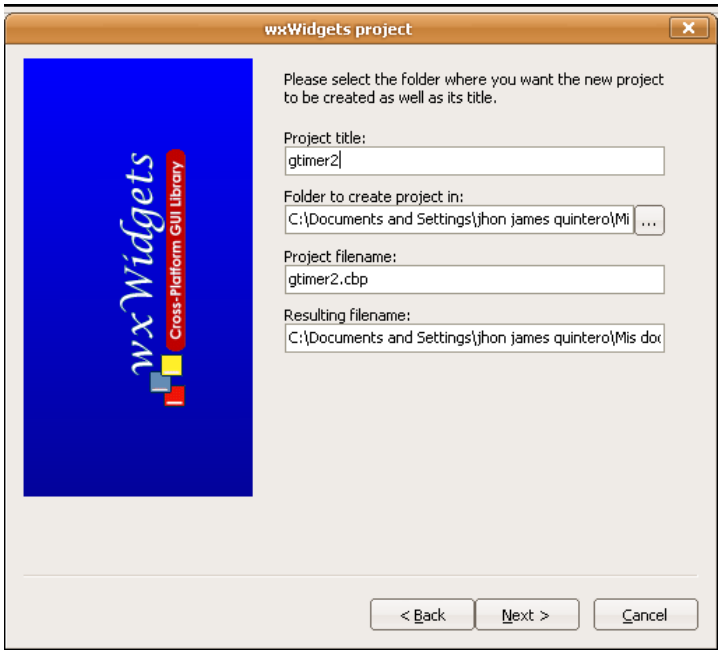


Figura 7 El nombre del proyecto

Luego indicamos el tipo de constructor de interfaces de usuario que deseamos usar en nuestro proyecto.

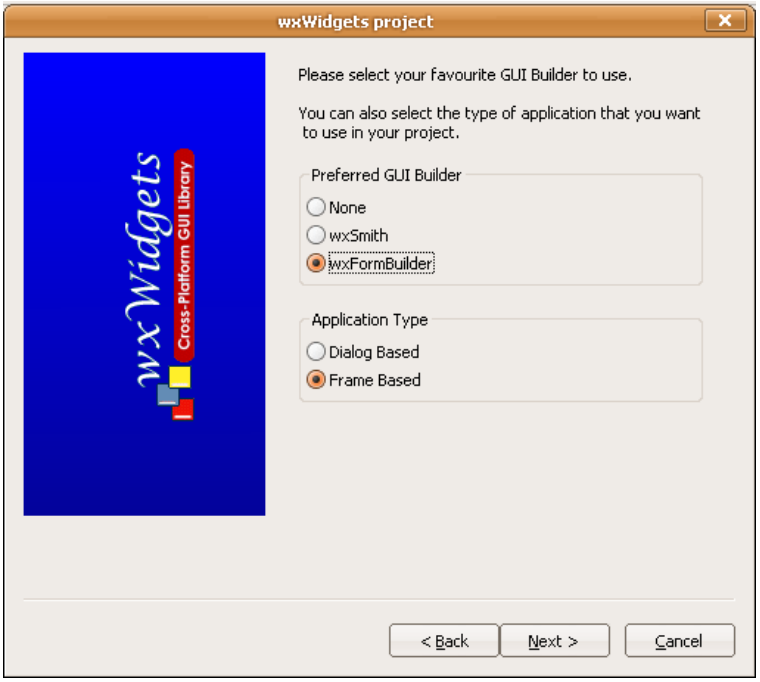


Figura 8 Seleccionar el constructor de GUI´s

Posteriormente indicamos la ruta donde quedaron instalados los wxWidgets.

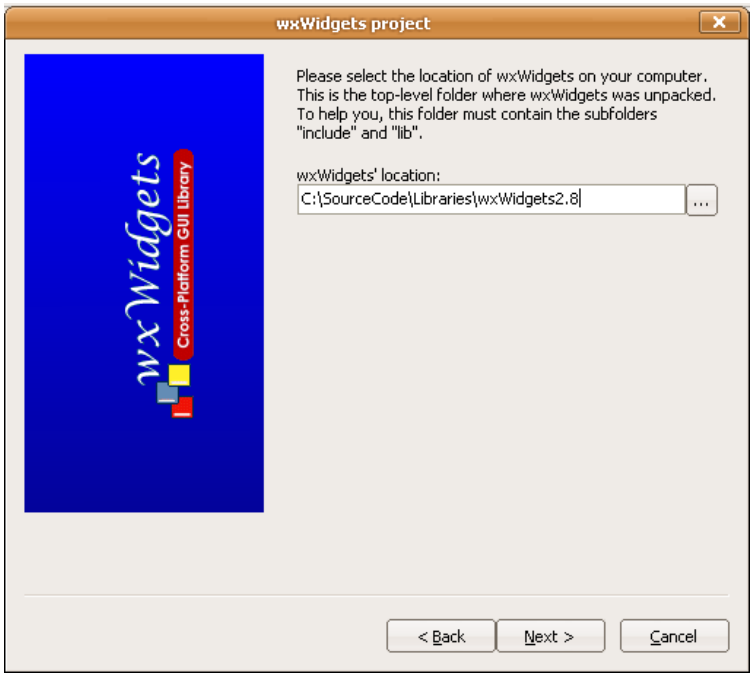


Figura 9 La ruta de los wxWidgets

Luego podemos configurar si queremos que los wxWidgets se usen como *dll* o como librería monolítica en la aplicación.

Listo debemos tener algo como los siguiente

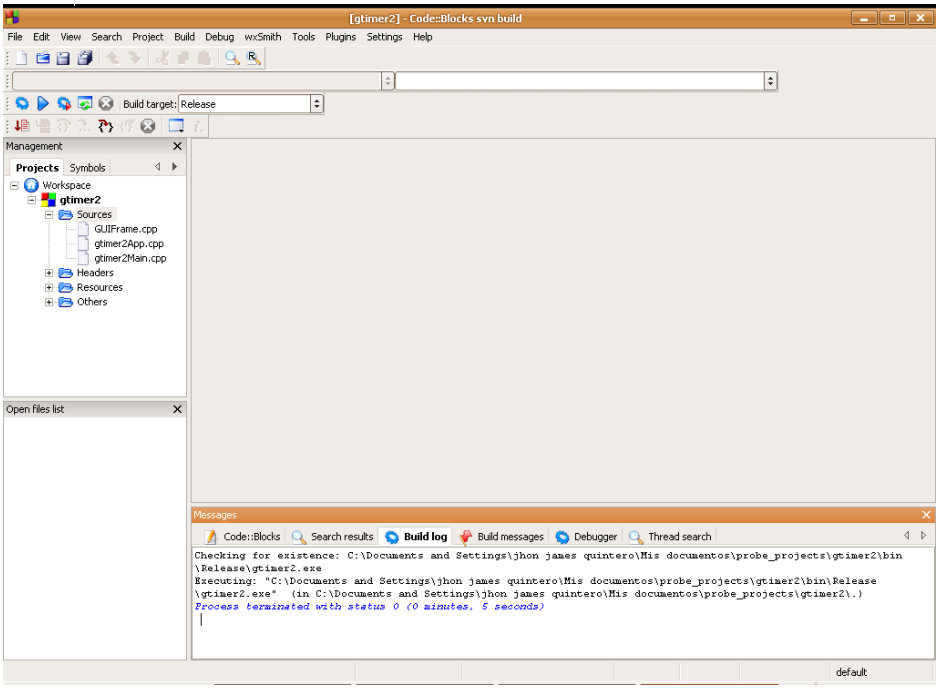
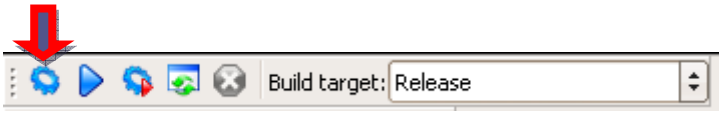


Figura 10 Proyecto listo para ser construido y ejecutado

Podemos dar click en  con eso construimos y ejecutamos el proyecto recién creado, obteniendo algo como:

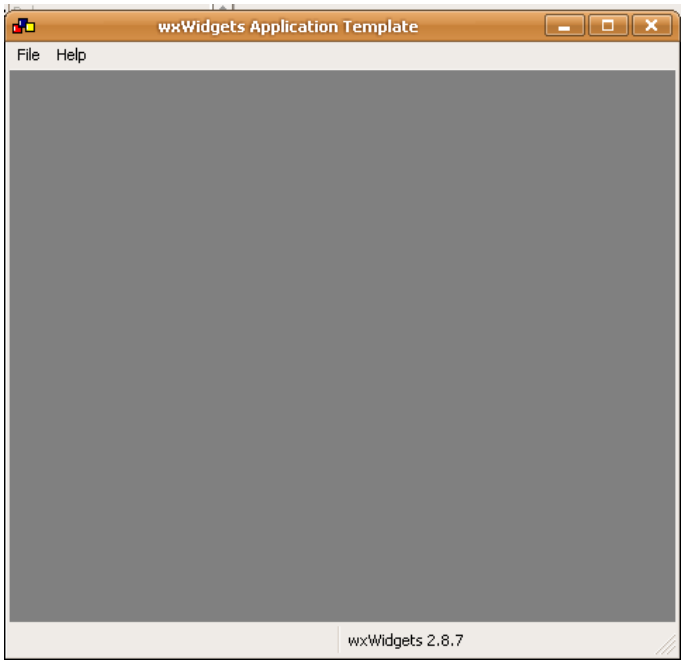


Figura 11 Nuestra aplicación de ejemplo

Ahora vamos a modificar el archivo con la interfaz de usuario, este aparece en el grupo *others* del visualizador de proyectos, allí debe aparecer un archivo llamado *WxWizFrame.fbp* el cual es un archivo de proyecto para wxFormBuilder, haciendo doble click sobre ese archivo el Code::Blocks nos pregunta con que programa abrirlo, debemos seleccionar el ejecutable del wxFormBuilder y listo, cada vez que queramos abrirlo desde Code::Blocks se lanzará de forma automática como se muestra en la figura.

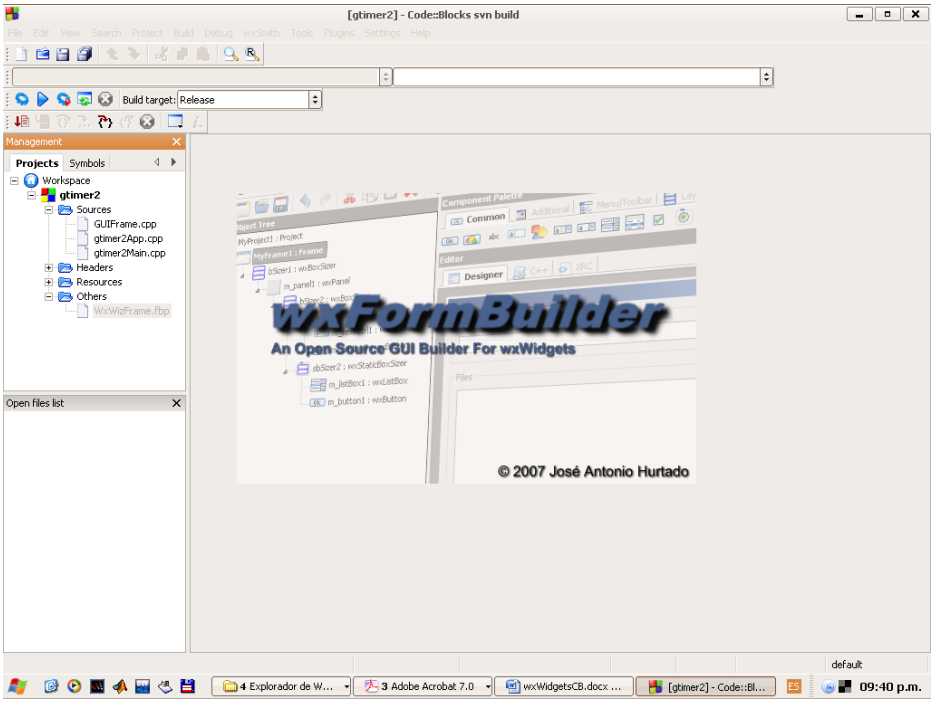


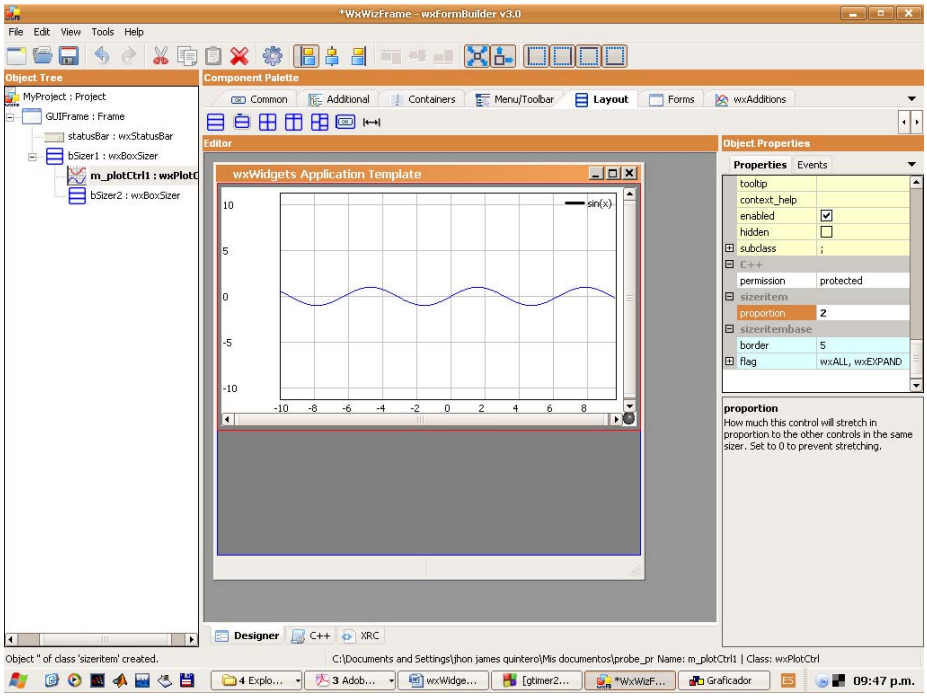
Figura 12 Lanzando del wxFormBuilder desde Code::Blocks

El wxFormBuilder nos informará que la versión del proyecto de GUI no corresponde, le indicamos que lo convierta y podemos editar la interfaz como nosotros queramos.

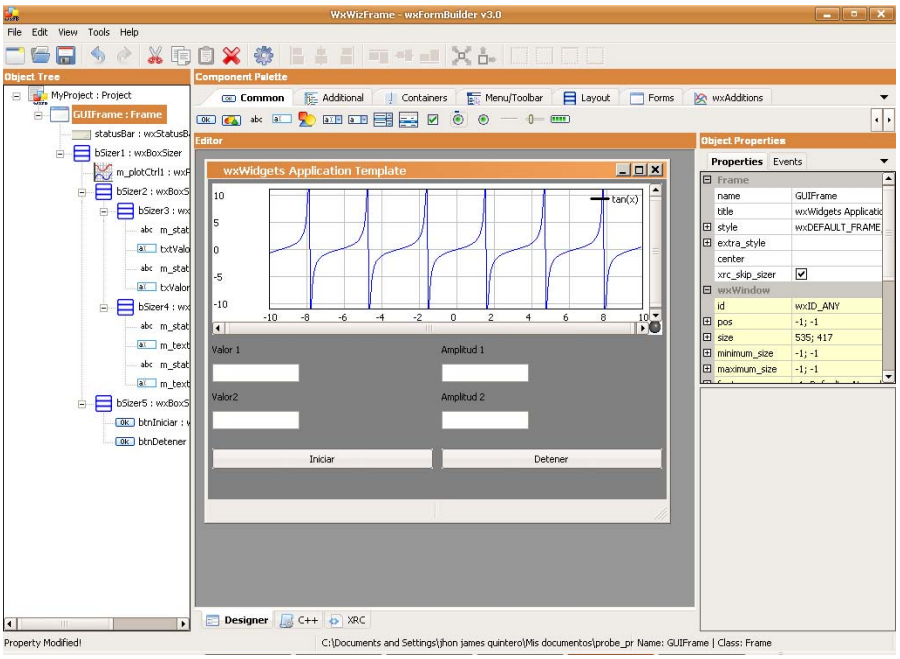
Dado que no necesitamos menú lo eliminamos seleccionándolo y dando *crtl-d*.

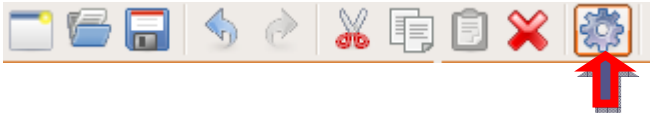
Se muestran a continuación algunos de los pasos para construir la interfaz, recordemos que todo se hace con los sizers.

Primero agregamos un wxBoxSizer y en el un widget para gráficos 2D llamado wxPlotCtrl, le indicamos a este que tenga una proporción de 2, luego agregamos otro wxBoxSizer en el Frame, y obtenemos lo siguiente.



Después de jugar un poco obtenemos (fijarse en la jerarquía de los sizer).

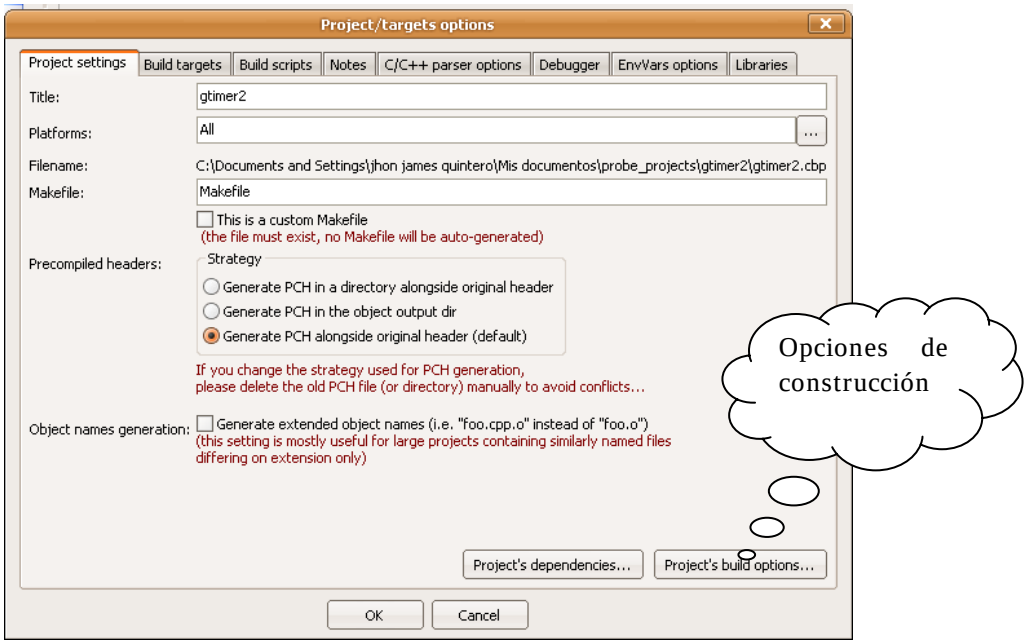


Ahora generamos el código que implementa la interfaz que acabamos de diseña dando click sobre el botón indicado  .

Ahora en Code::Blocks si tenemos abierto el archivo *GUIFrame.cpp* o *GUIFrame.h* se nos indicará que el archivo ha cambiado que si lo deseamos recargar, decimos que si y vemos el código en C++, generado.

Ahora como hacernos para usar el código generado en nuestro programa, la solución en este caso usaremos el mecanismo de herencia (ver un librillo de programación orientada a objetos). La ventaja es que nuestro proeyecto ya tiene una clase que desciende de la clase que implementa la GUI, en nuestro caso se llama gtimer2Main (debido al nombre que le dimos al proyecto), a mi personalmente no me gustan las clases con nombre que inicien en minúsculas entonces los cambié (el *.h,.cpp* y el nombre de la clase por *Main*).

Ahora si tratamos de compilar el proyecto,..., aja, no compila, necesitamos indicarle al compilador donde encontrar el archivo de cabecera para el *wxPlotCtrl*, esto lo hacemos de la siguiente manera, abrimos las propiedades del proyecto.



Damos click en el botón de opciones de construcción y agregamos la ruta de los archivos de cabecera donde se instalaron los widgets adcionales, generalmente

C:\SourceCode\Libraries\wxWidgets2.8\additions\include

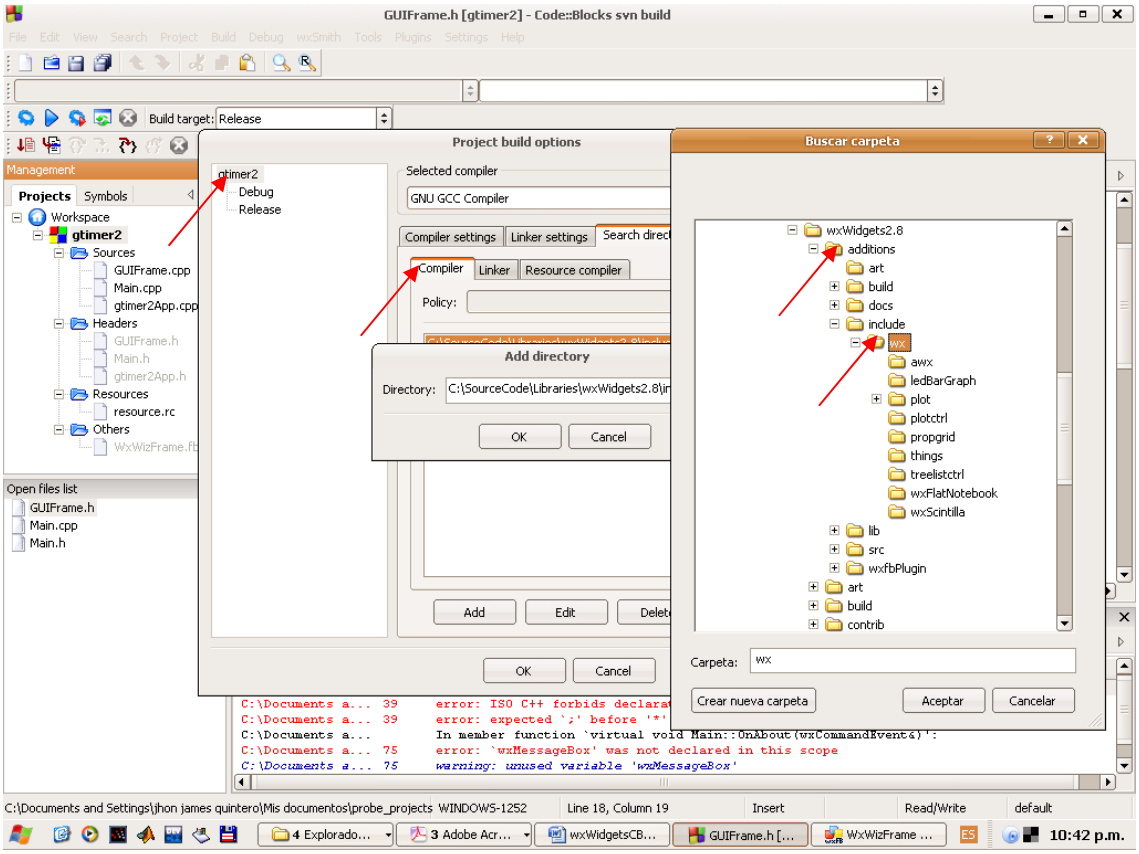
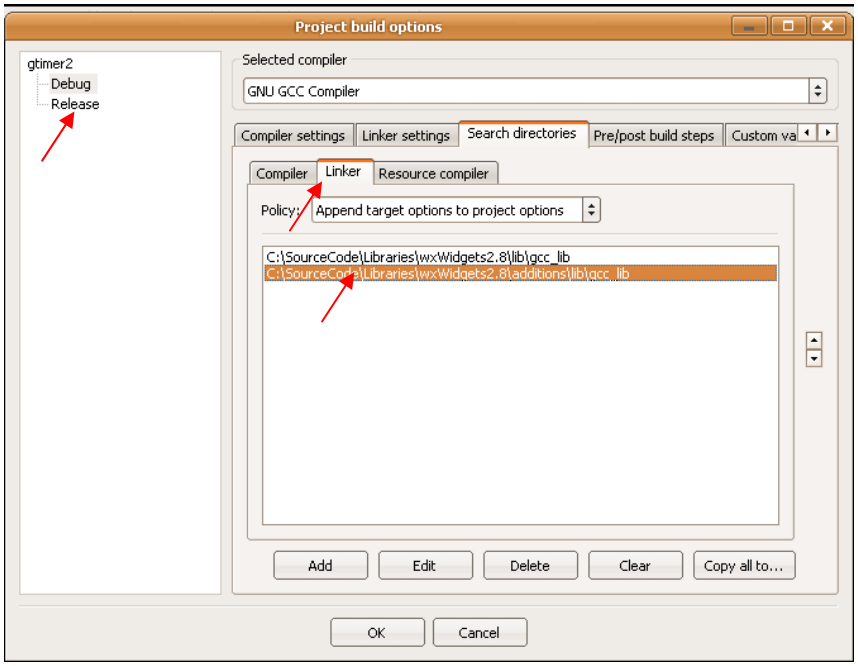


Figura 13 Agregando un directorio para los archivos de cabecera

Ahora para cada agregamos un directorio para las librerías, para que el enlazador encuentre las que necesitamos, en nuestro caso las del wxPlotCtrl. Para ello hacemos lo que se muestra en la siguiente figura, es de notar que esto es necesario hacerlo tanto para *debug* como para *release*.



Ya casi estamos listos, nos falta agregar las librerías.

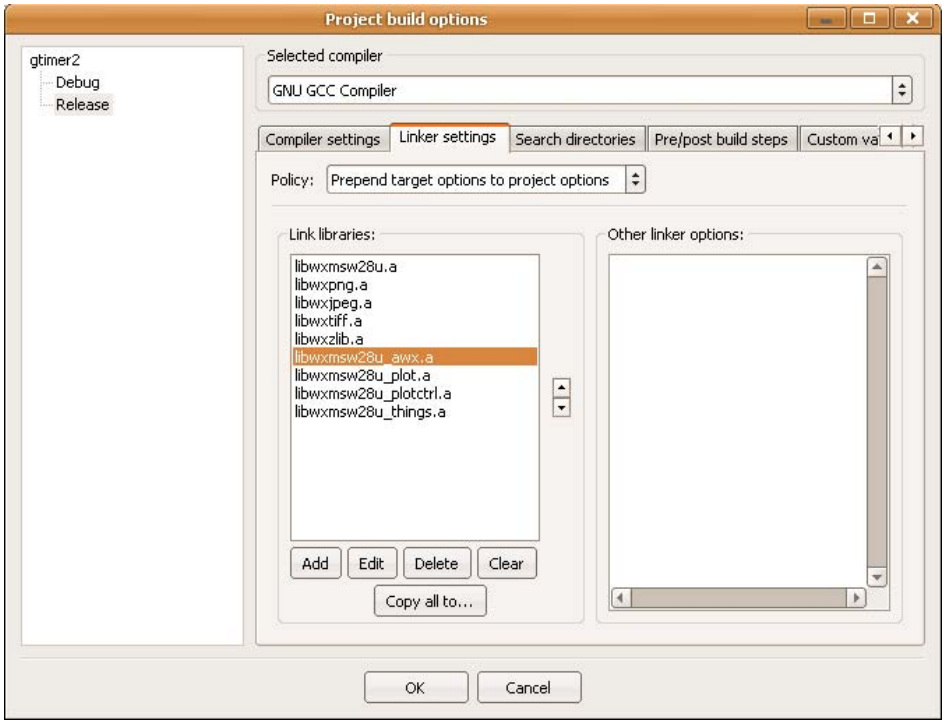


Figura 14 Librerias para el wxPlotCtrl (release)

Ahora lo mismo pero para la versión de depuración (*debug*).

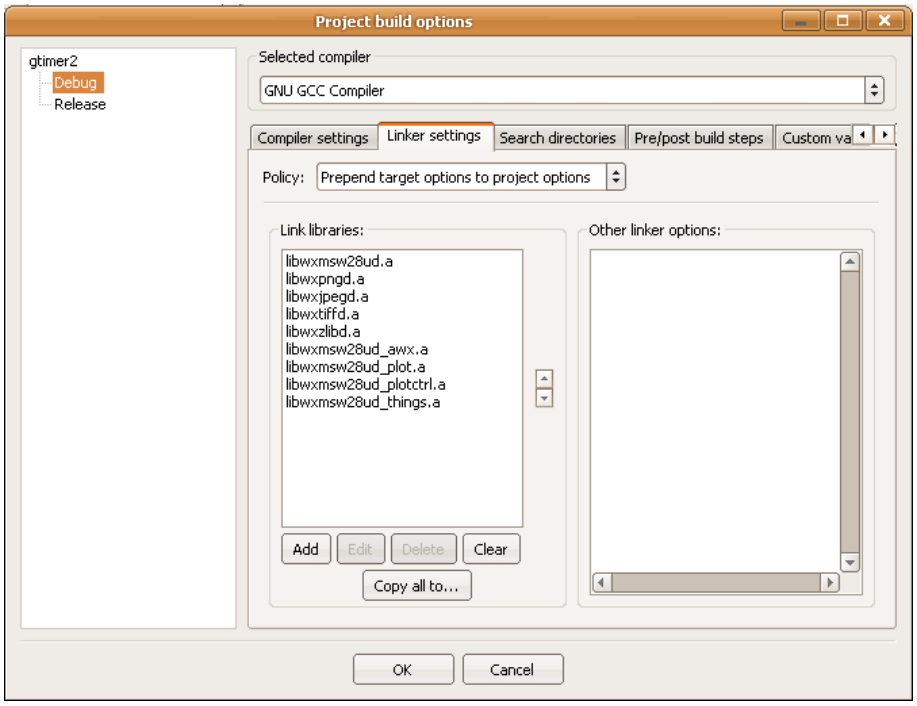


Figura 15 Librerias para el wxPlotCtrl (debug)

Si construimos y ejecutamos el proyecto, podemos obtener lo siguiente:

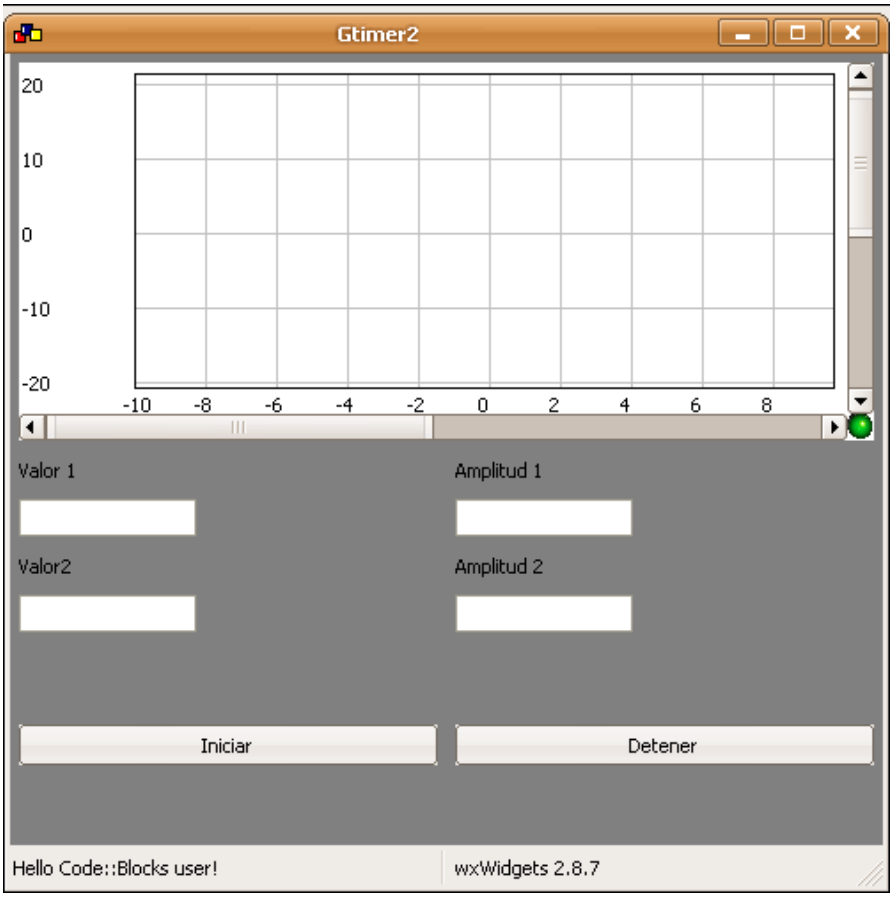


Figura 16 La base del ejemplo en ejecución

Ahora agreguemos el resto de cosas al proyecto, primero copiamos los archivos *lrtimer.h y lrtimer.cpp*, necesario para el timer de tiempo real y los agregamos al proyecto, y trabajamos algo sobre el código (ver código fuente) y tenemos.

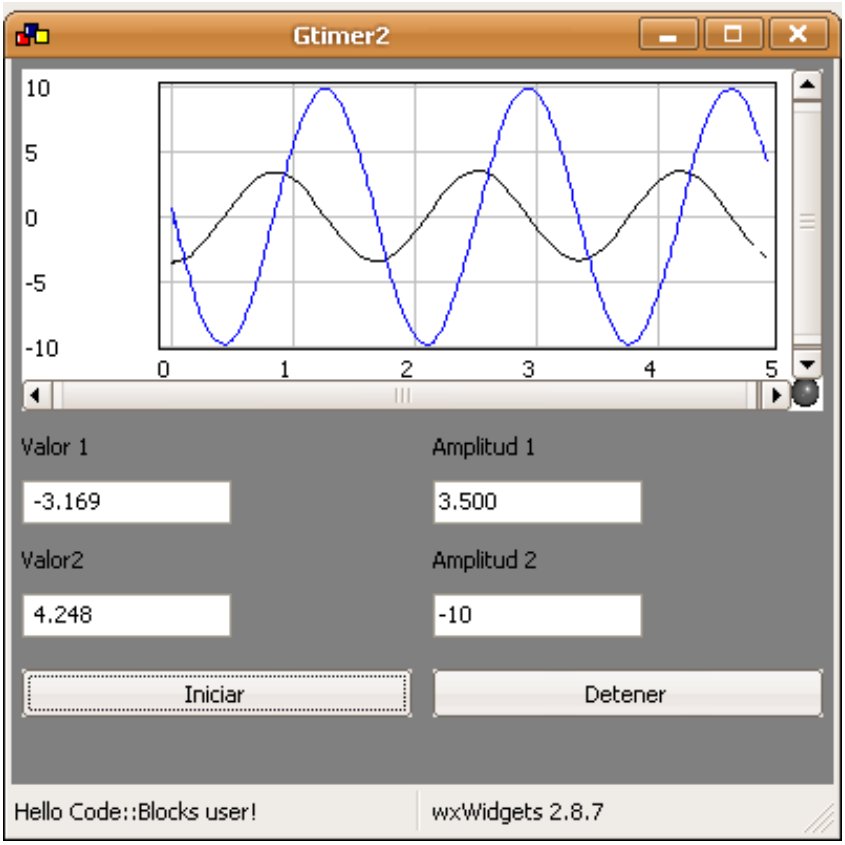


Figura 17 El programa funcionado