Creación de Interfaz Gráfica de Usuario (GUI) en C++

Creación de interfaz gráfica de usuario (GUI) en C++

Cristobal Rodas

Copyright: (c) Cristobal Rodas

23/06/2018

Introduction

En este documento se tratara de explicar al lector los aspectos esenciales del desarrollo de una GUI, en el lenguaje de programación C++ haciendo uso de las siguientes tecnologías.

- ZinjaI
- wxWidgets
- wxFormBuilder

ZinjaI

ZinjaI es un IDE (entorno de desarrollo integrado) libre y gratuito para programar en C/C++. Pensado originalmente para ser utilizado por estudiantes de programación durante el aprendizaje, presenta una interfaz inicial muy sencilla, pero sin dejar de incluir funcionalidades avanzadas que permiten el desarrollo de proyectos tan complejos como el propio ZinjaI.

WxWidgets

Las wxWidgets son unas bibliotecas multiplataforma y libres, para el desarrollo de interfaces gráficas programadas en lenguaje C++. Están publicadas bajo una licencia LGPL, similar a la GPL con la excepción de que el código binario producido por el usuario a partir de ellas, puede ser propietario, permitiendo desarrollar aplicaciones empresariales sin coste de licencias.

Las wxWidgets proporcionan una interfaz gráfica basada en las bibliotecas ya existentes en el sistema (nativas), con lo que se integran de forma óptima y resultan muy portables entre distintos sistemas operativos. Están disponibles
para Windows, MacOS, GTK+, Motif, OpenVMS y OS/2.

También pueden ser utilizadas desde otros lenguajes de programación, aparte del C++: Java, JavaScript, Perl, Python, Smalltalk, Ruby, Erlang.

WxFormBuilder

WxFormBuilder es un diseñador de interfaz gráfica de usuario para el marco de aplicaciones wxWidgets, que ayuda a la creación y mantenimiento rápido de aplicaciones multiplataforma.

He decidido utilizar estas tres tecnologías porque contienen una compatibilidad que facilita la creación de GUI en C++, haciendo que los desarrolladores menos experimentados (Como yo XD) se centren en hacer funcionar la aplicación y no en la implementación del código para la creación de la interfaz.

Instalación de las herramientas

Ahora pacemos a la explicación de la instalación de estas tecnologías

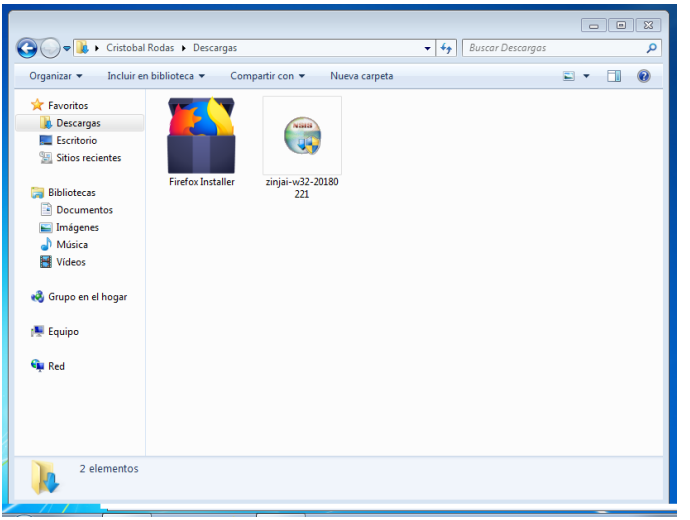
- ZinjaI

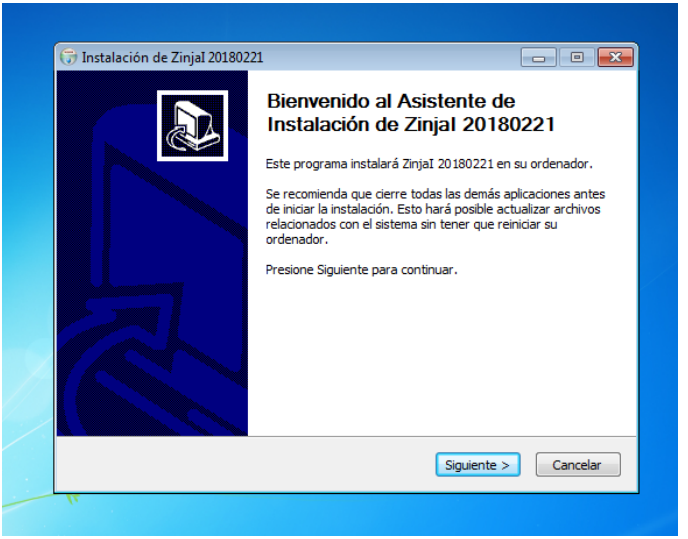
  Para la instalación de este IDE primero tendremos que irnos a la página oficial en el apartado de descargas.

  http://zinjai.sourceforge.net/index.php?page=descargas.php

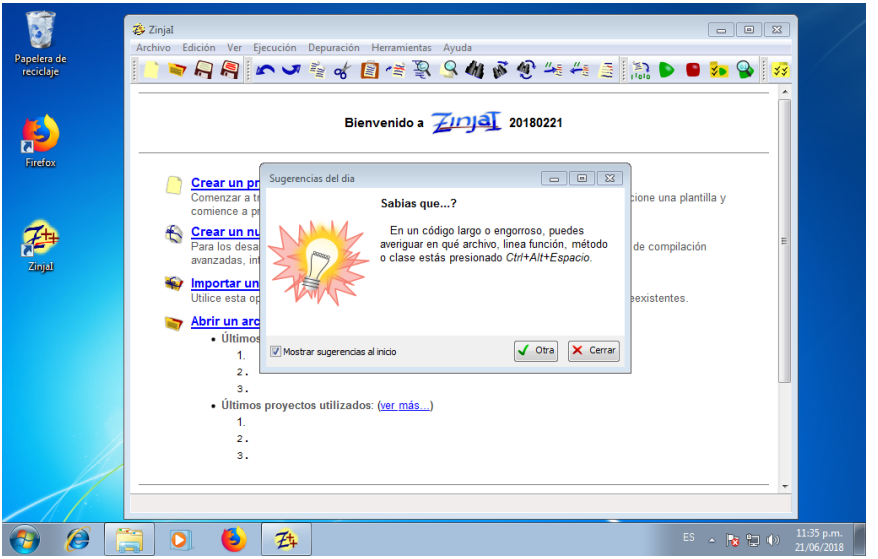
  como podrán observar este IDE tiene soporte para las plataformas **Windows**, **Mac OS i686** y **GNU/Linux.**

  (Windows) Después de descargar el ejecutable procedemos a instalarlo,





Es una instalación como cualquiera, siguiente, acepto, siguiente e instalar, por ultimo le damos al botón finalizar y listo ya tendremos instalado ZinjaI en nuestro pc



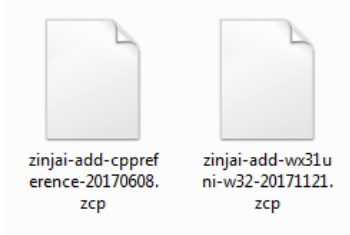
- WxWidgets
  La verdad la instalación de esta biblioteca puede ser un tanto complicada para usuarios de C++ sin experiencia ya que tendrán que compilar la librería colocar los flags correspondientes si quieren que la librería se a estática o dinámica etc…

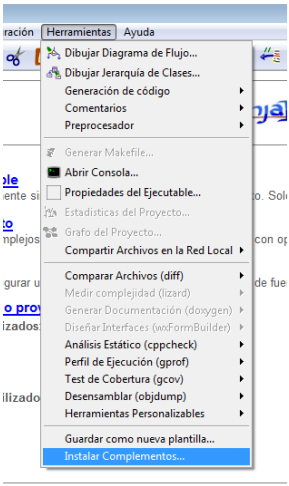
  Por suerte ZinjaI nos facilita esta tarea (solo para usuarios de Windows) dado que en su página oficial en el apartado de complementos nos facilitan uno para la compilación, plantillas, autocompletado de esta librería

  http://zinjai.sourceforge.net/index.php?page=downextras.php

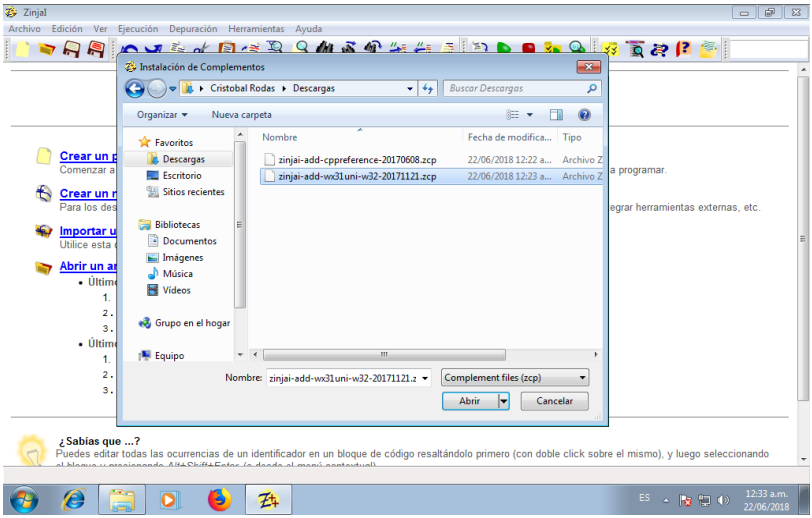
  El complemento que utilizaremos será el de la Biblioteca wxWidgets versión 3.x, además aprovecharemos para descargar el complemento Referencia C/C++ que contiene plantillas, auto completado etc… que nos facilitaran el trabajo con ZinjaI

  Nos quedaran dos ficheros .zcp los cuales son muy sencillos de instalar en ZinjaI nos dirigimos a Herramientas->Instalar Complementos…

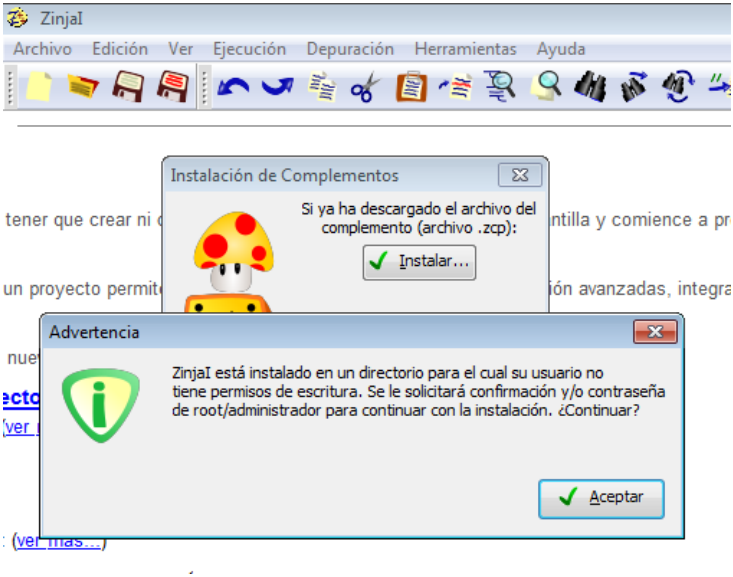




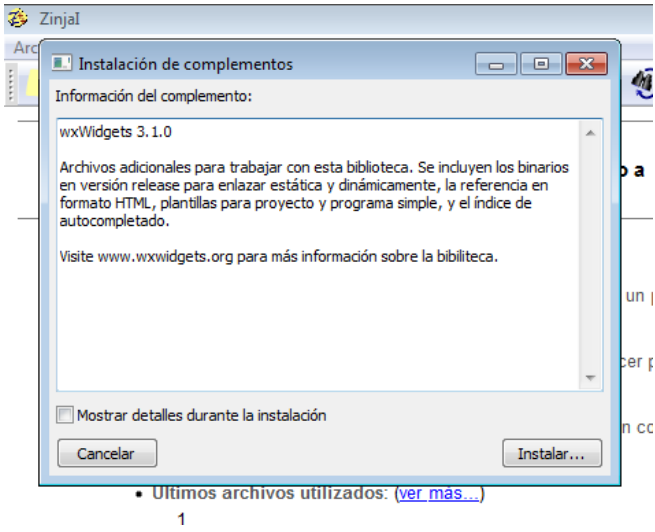
Nos abrirá una nueva ventana con la cual instalaremos nuestros complementos dando click en el botón de instalar, los buscaremos y los instalaremos de a uno comenzando porque más les guste XD.



Ahora nos saldrá un cartel que nos informa que tenemos que darle permiso de administrador para instalar el complemento esto es debido a que zinjai instala los complementos en su carpeta raíz que se encuentra en C:\Program Files (x86)\Zinjal y para ello necesita los permisos de administrador.



Después de eso nos sale otro cartelito donde tendremos que darle a instalar por segunda y última vez XD, para poder hacer uso de nuestro complemento, además esta ventana nos da información del contenido del mismo, al terminar la instalación nos sale otro cartelito diciéndonos que los cambios se verán al reiniciar ZinjaI



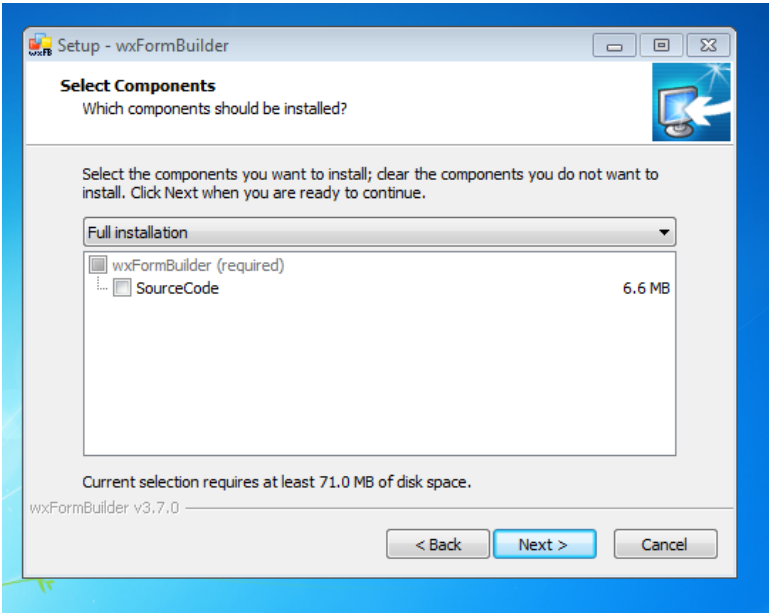
Una vez instalado los 2 complementos ya estamos listos para compilar proyectos con la librería wxWidgets

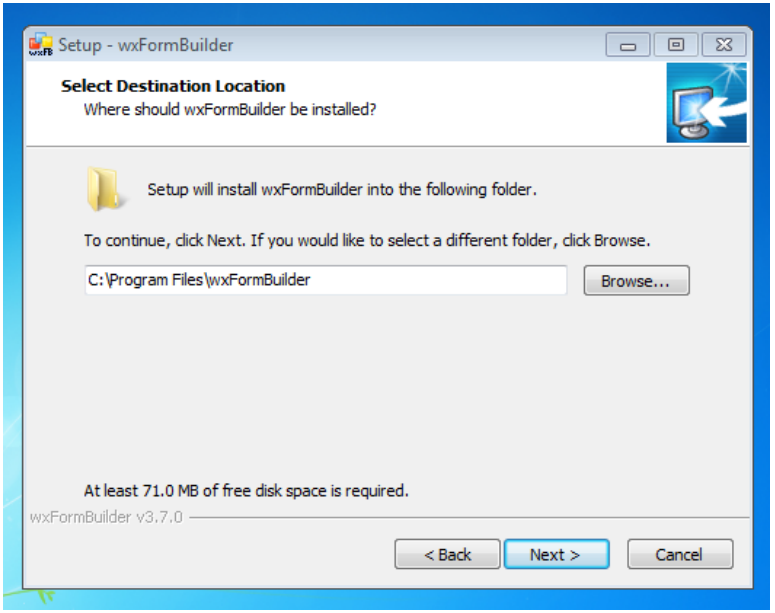
- WxFormBuilder

  Para instalar este programa necesitamos dirigirnos al repositorio github donde se encuentra alojado

  https://github.com/wxFormBuilder/wxFormBuilder/releases

  Nos descargamos wxFormBuilder_v3.7.0.exe para Windows
  Y lo instalamos con la configuración por defecto

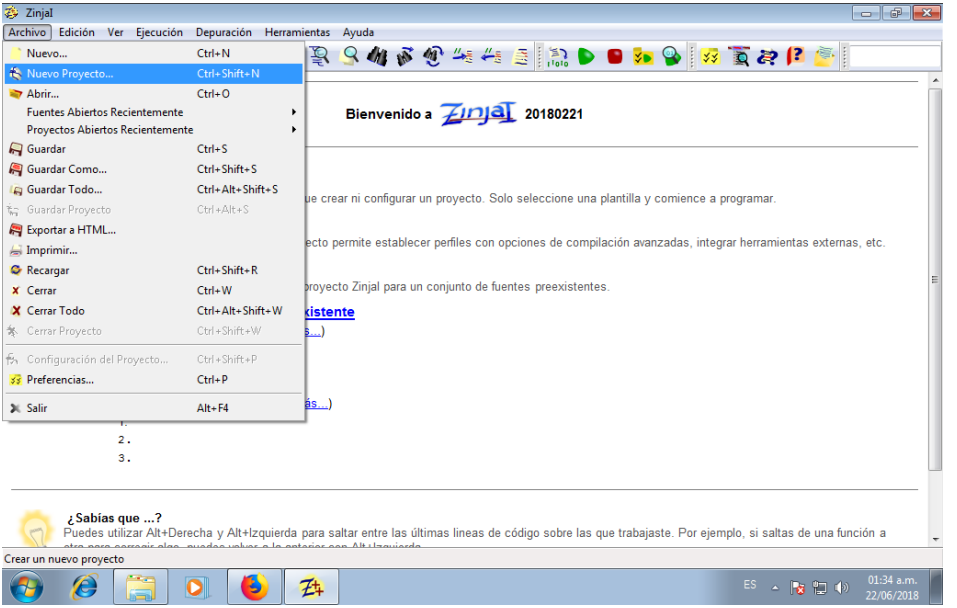



Setup - wxFormBuilder
Select Destination Location
Where should wxFormBuilder be installed?
Setup will install wxFormBuilder into the following folder.
To continue, click Next. If you would like to select a different folder, click Browse.
C:\Program Files\wxFormBuilder
Browse...
At least 71.0 MB of free disk space is required.
wxFormBuilder v3.7.0
< Back
Next >
Cancel

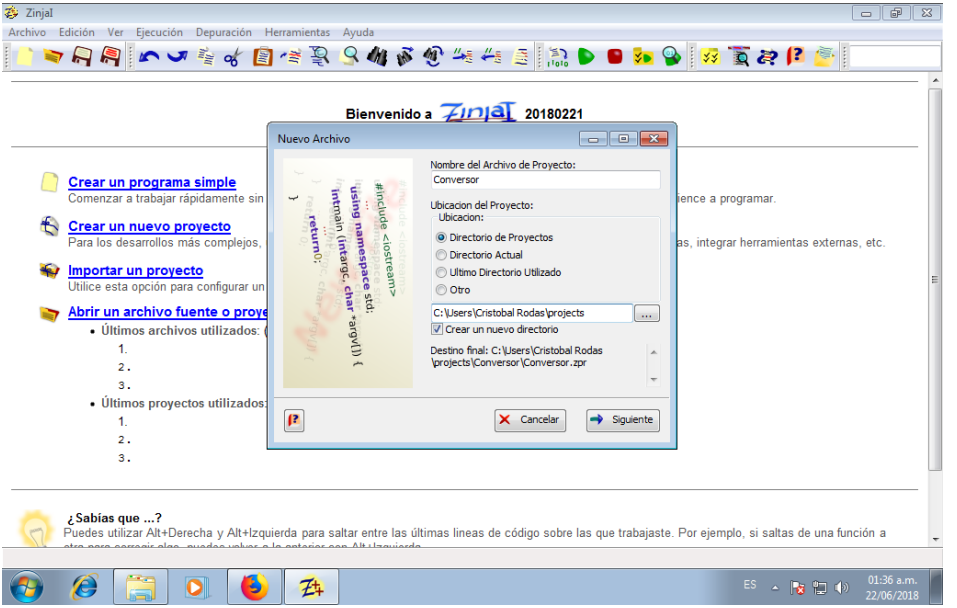

Setup - wxFormBuilder
Completing the wxFormBuilder Setup Wizard
Setup has finished installing wxFormBuilder on your computer. The application may be launched by selecting the installed shortcuts.
Click Finish to exit Setup.
Launch wxFormBuilder
< Back
Finish

Creación de proyectos con ZinjaI

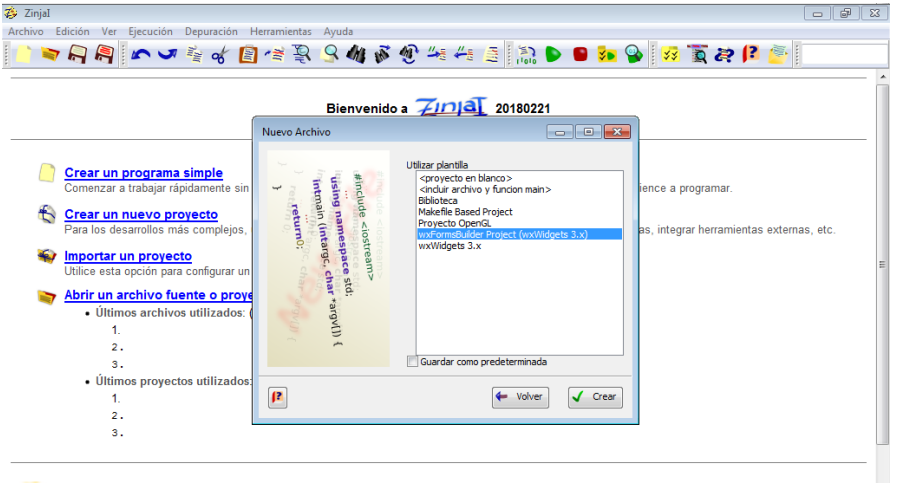
Bien ahora ya estamos listos para crear nuestra pequeña aplicación en C++ para ello abrimos Zinjal y nos dirigimos a Archivo->Nuevo Proyecto…



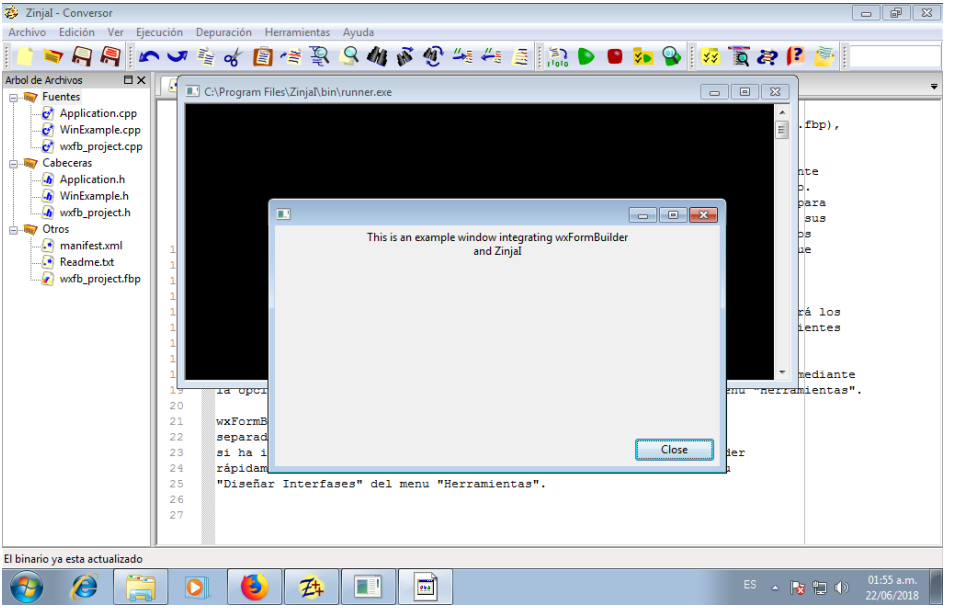
Nos desplegara una ventana de ayuda para la creación de nuestro directorio de trabajo y el nombre de nuestro proyecto, en el nombre pondremos Conversor (así es crearemos un conversor de unidades) en ubicación lo dejaremos por defecto al igual que las demás configuraciones.



Ahora escogeremos una plantilla usaremos la wxFromsBuilder Project (wxWidgets 3.0)



Ahora se nos abre el proyecto con algunos ficheros fuetes, cabeceras y otros, podemos compilar este proyecto para verificar que las anteriores instalaciones se efectuaron de forma correcta

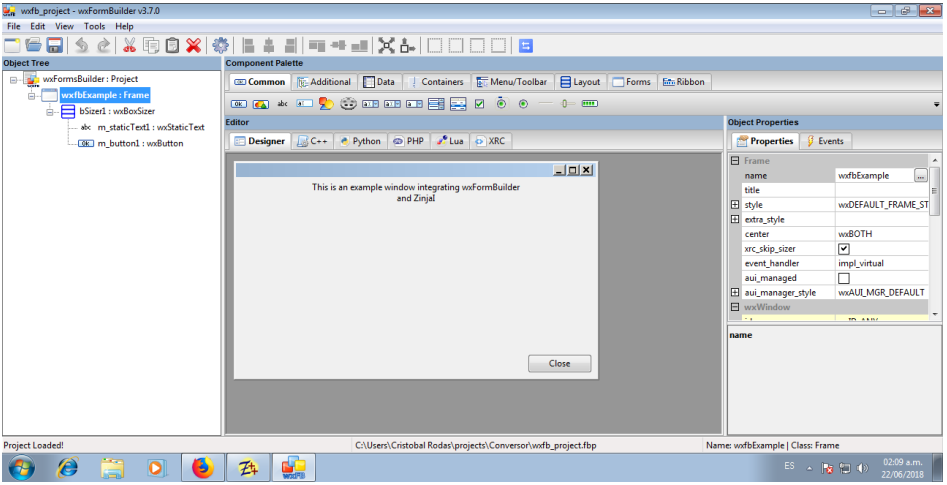


Si nos compila y ejecuta sin que Zinjal nos dé ni un error o warning, significa que todo anda bien ☺

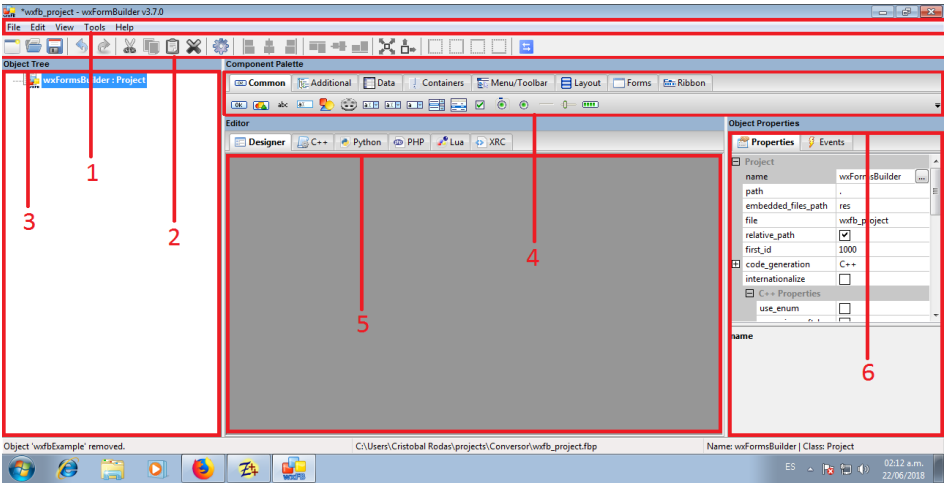
Los ficheros que realmente nos interesan son 5,

- Application.cpp
- Application.h
- wxfb_project.cpp
- wxfb_project.h
- wxfb_project.fbp
- manifest.xml

los demás ficheros los borraremos del proyecto, ahora haremos doble click sobre el fichero wxfb_project.fbp para editar su contenido, eliminaremos el frame wxfbExample con Ctrl+d, para dejar el proyecto en blanco.



Haremos una pausa para explicar por encima la interfaz de wxFormBuilder



1. Barra de menús

   En ella encontraremos lo típico en casi todos los programas, crear, guardar, deshacer, rehacer, eliminar, cerrar etc…

2. Barra de herramientas

   En ella encontraremos herramientas más utilizadas a la hora de crear nuestras interfaces graficas

3. Árbol de objetos

   En el se irán agregando los elementos que conformaran nuestro proyecto

4. Paleta de componentes

   Esta es sin duda una de las herramientas que más utilizaremos en esta aplicación, ya que contiene todos los componentes con lo que podremos crear infinidad de proyectos, en ella se encuentran los freames, textcontrol, botones, menús etc…

5. Editor

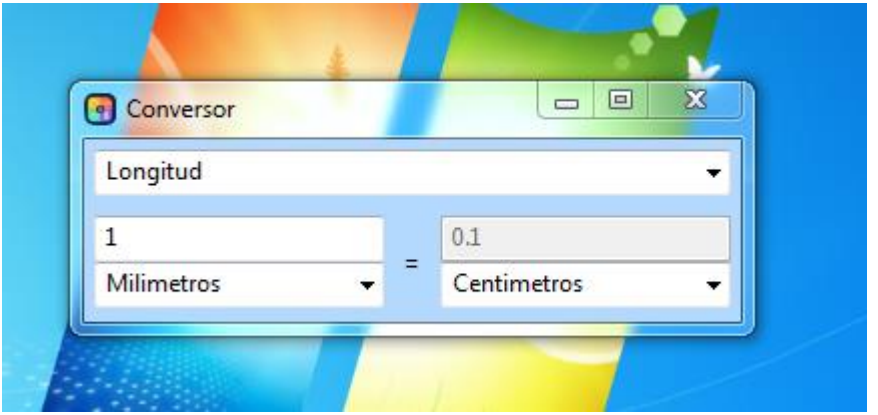
   La verdad que de editor tiene poco ya que en el solo podremos seleccionar los elementos agregados y poder visualizar el resultado final

6. Propiedades de objeto

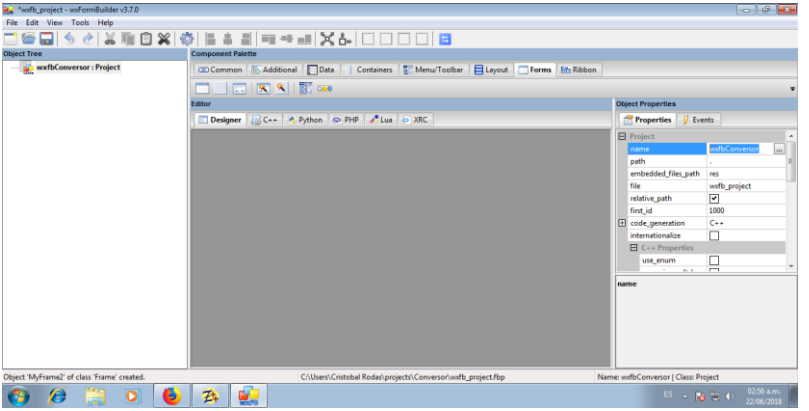
   Este es la segunda herramienta que más utilizaremos en la aplicación, en ella podremos modificar los atributos de cada componente como su nombre, apariencia, interactividad con los eventos etc…

Construyendo la Interfaz Grafica

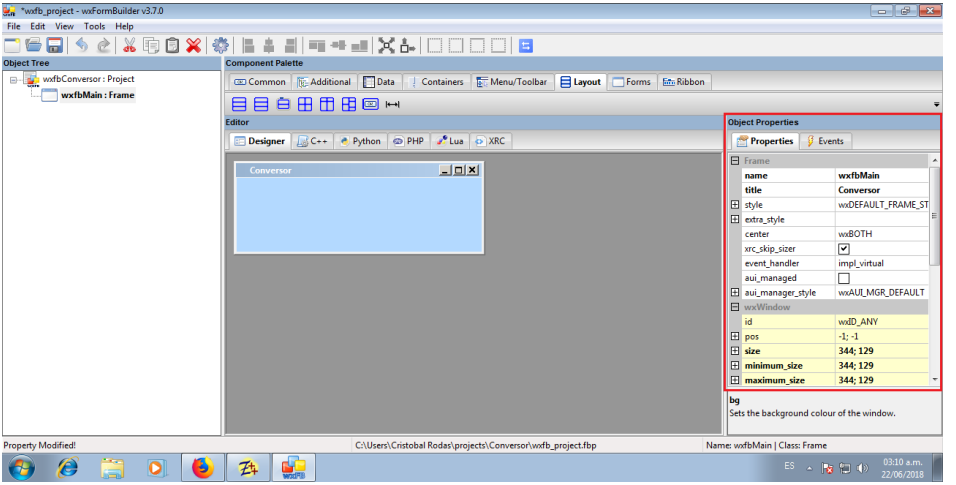
Ya podemos empezar a construir la parte grafica de nuestro proyecto, para ello debemos tener una idea de que es lo que queremos, yo he pensado en una interfaz simple ya que es un proyecto pequeño que no requiere de mucho.



Esta será la apariencia que tendrá nuestra aplicación, ahora que sabemos cómo se verá nuestra aplicación ya podemos empezar a agregar cosas a nuestro proyecto, lo primero es seleccionar el único objeto que está en el árbol de objetos y cambiarle el nombre por uno más descriptivo por ejemplo wxfbConversor (wxfb es wxFormBuilder)



Ahora nos dirigimos a la paleta de componentes en la ficha Forms elegimos el objeto Frame (el primero), en las propiedades de objeto le cambiamos el nombre a wxfbMain, title Conversor, bg Custom (r 179, g 217, b 255), size 344; 129, minimum_size 344; 129, máximum_size 344; 129, lo de los tamaños es para bloquearlo y que el usuario final no pueda modificarlo.



Ahora nos dirigimos a la paleta de componentes en la ficha Layout agregamos un wxBoxSizer (el primero) y dentro de ese agregamos otro wxBoxSizer al que le desactivamos el Stretch (Barra de herramientas) ahora dentro de nuestro segundo wxBoxSizer agregamos un wxComboBox en la ficha Common al que le activamos el Expand y le desactivamos el Stretch (Barra de herramientas), en las propiedades de objeto le cambiamos el nombre a unidades y en la ficha eventos en el evento OnCombobox ponemos unidades_onClick.

Ahora en el árbol de objetos seleccionamos el primer wxBoxSizer que agregamos y le agregamos un tercer wxBoxSizer activamos Expand

desactivamos Stretch en propiedades de objeto cambiamos la horientacion a wxHORIZONTAL,a este le agregamos un wxBoxSizer al que le activamos el Expand y Stretch, le activamos los cuatro márgenes los que están a la par del Stretch, a este wxBoxSizer le agregamos un wxComboBox y un wxTextCtrl de la ficha Common.

Al wxComboBox le cambiamos nombre a unidadEntrada, en la ficha eventos en el evento OnCombobox ponemos unidadEntrada_onClick

Al wxTextCtrl le cambiamos nombre a entrada, en la ficha eventos en el evento OnCombobox ponemos entrada_update

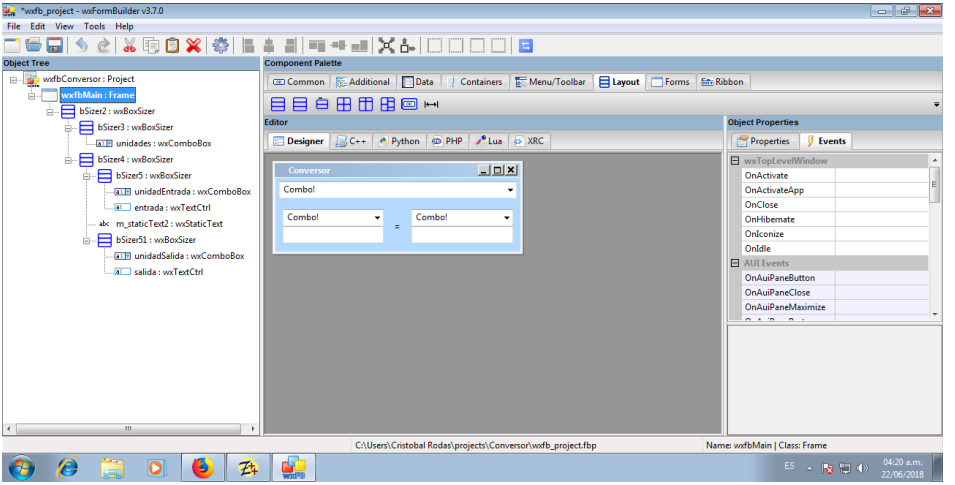
Ahora en el árbol de objetos un wxBoxSizer arriba del cual nos encontramos (el penultimo) agregamos un wxTextCtrl en la barra de herramientas lo alineamos al centro, en las propiedades le cambiamos el nombre por un =

Por ultimo solo nos hace fala colocar a la derecha los objetos que representaran la salida pero como podrán observar en la imagen de ejemplo que coloque al principio de este capítulo es idéntico al que tenemos a la izquierda así que lo copiaremos desde el bSizer5 y pegaremos en bSizer4 en el árbol de objetos con algunas modificaciones

Primero al wxBoxSizer que acabamos de pegar le activaremos todos los márgenes

Segundo al wxComboBox y wxTextCtrl cambiaremos donde de diga entrada lo cambiaremos a salida quedando unidadSalida,

unidadSalida_onClick, salida y al wxTextCtrl en los eventos lo dejamos vacío, quedándonos de la siguiente forma (ver el árbol de objetos)



Al guardar el proyecto y regresar a nuestro ya olvidado ZinjaI veremos cómo nos regenera el proyecto creando una clase para la ventana (wxfbMain) que hemos creado en wxFormBuilder

Ficheros generados

Bien en ZinjaI tenemos los siguientes ficheros

- Application.cpp
- Application.h
- wxfb_project.cpp
- wxfb_project.h
- wxfb_project.fbp
- manifest.xml
- win_main.cpp
- win_main.h

En los ficheros wxfb_project tenemos el nuevo código que nos generó WXFB aparte de 2 nuevos ficheros win_main que no es más que una clase que hereda de la clase generada por WXFB el motivo de esto es porque si queremos agregar más peculiaridades a nuestro proyecto en WXFB este remplazara el código que tengamos por el que él nos proporciona borrando todo el código que a él le parece que sobra XD.

Con la creación de una nueva clase que herede de la que nos da WXFB nos evitamos ese problema tan grave ya que ZinjaI no borra el código que se encuentre en esos ficheros, simplemente chequea que tengamos todas las funciones miembro de dicha clase para que no tengamos que hacerlo nosotros.

Manos a la obra

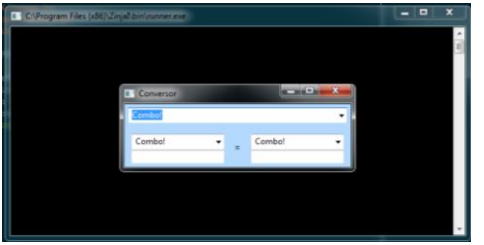
Ahora si por fin podremos empezar a codear para sentirnos como verdaderos programadores.

Lo primero que haremos será dirigirnos al fichero Application.cpp en el que cambiaremos el include "WinExample.h" por el nuestro “win_main.h”, y en la línea 9 donde pone WinExample *win = new WinExample(NULL) pues como no nuestra clase win_main *win = new win_main(NULL), quedándonos así.

`Application.cpp`

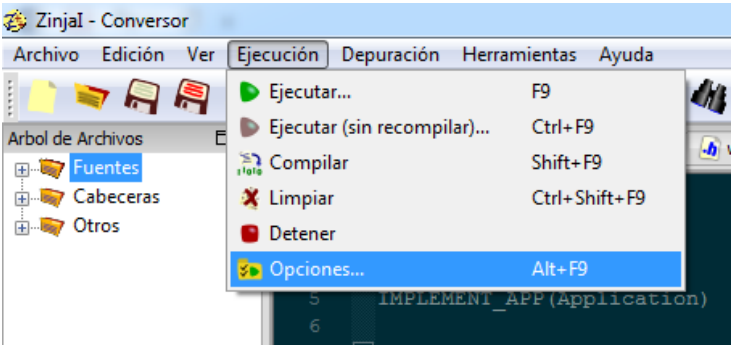
```
#include <wx/image.h>
#include "Application.h"
#include "win_main.h"

IMPLEMENT_APP(Application)

bool Application::OnInit() {
    wxInitAllImageHandlers();
    win_main *win = new win_main(NULL);
    win->Show();
    return true;
}
```

De pues de haber hecho esos pequeños cambios ya podemos compilar nuestro proyecto para admirar la belleza de su interfaz XD.

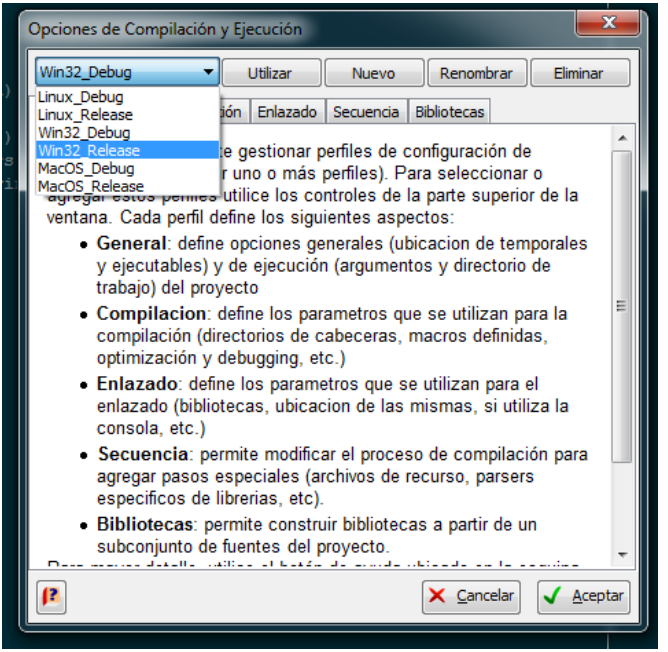


Antes de continuar haremos unos pequeños cambios como quitar la Shell que aparece detrás de nuestra aplicación.

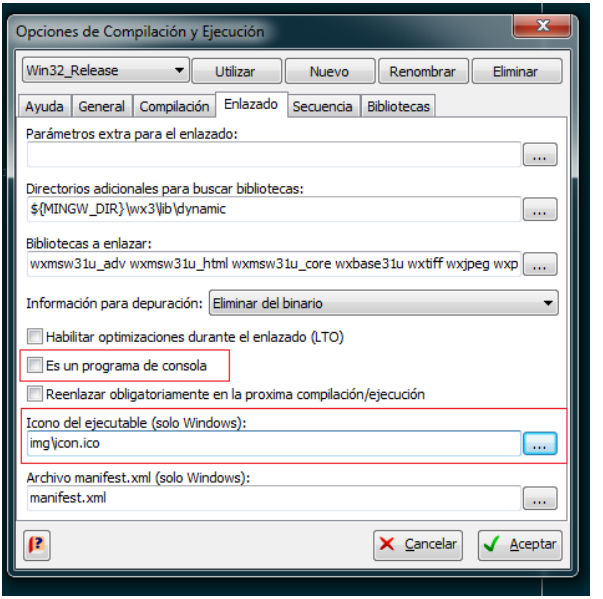
Nos dirigimos a la barra de menús->ejecución->opciones



En las plantillas de compilación escogemos la de Release



En la ficha enlazado verificamos que este desmarcada la opción Es un programa de consola, además aprovecharemos para colocar el icono del ejecutable (solo para Windows), yo he colocado el icono en una carpeta que he llamado img en el directorio de trabajo, las imágenes las pueden descargar de esta carpeta de mega https://mega.nz/#F!kksVCYYB!5DkO1AxWQsoPNwZXUOYE7A



Como podrán observar el icono está compuesto de cuatro partes peso, medidas, tiempo y energía que serán las unidades que nuestra aplicación podrá manejar (luego le pueden agregar mas unidades de conversión si gustan)

Ahora devuelta en Zinjal crearemos un fichero de cabecera y uno de implementación llamado utilidades donde colocaremos variables y funciones de apoyo

Dentro del fichero de utilidades.cpp crearemos las variables que contendrán los nombres de las unidades de conversión

`Utilidades.cpp`

```
#include "utilidades.h"
```

```
using namespace std;

string unidades[] = {
    "Longitud",
    "Masa",
    "Energia",
    "Tiempo",
    ""
};

string unidadesL[] = {
    "Milimetros",
    "Centimetros",
    "Pies",
    "Metros",
    "Yardas",
    "Pulgadas",
    ""
};

string unidadesM[] = {
    "Gramos",
    "Miligramos",
    "Libras",
    "Onzas",
    "Kilogramo",
    ""
};

string unidadesE[] = {
    "Julio",
    "Kilojoule",
    "Gram Calorie",
    "Kilocaloria",
    "Vatio-Hora",
    "Kilovatio-Hora",
    ""
};

string unidadesT[] = {
    "Segundo",
    "Minuto",
```

```
    "Nanosegundo",
    "Microsegundo",
    "Milisegundo",
    "Hora",
    "Dia",
    "Semana",
    ""
};
```

Bien ahora haremos que se visualicen en nuestra aplicación, primero comenzaremos con los tipos de unidades, para ello en utilidades haremos una función que retorne las unidades luego en win_main aremos una función que las cargue al combobox y llamar a la función en el constructor de nuestra aplicación

Utilidades.cpp

```
string* listUnidades(){
    return unidades;
}
```

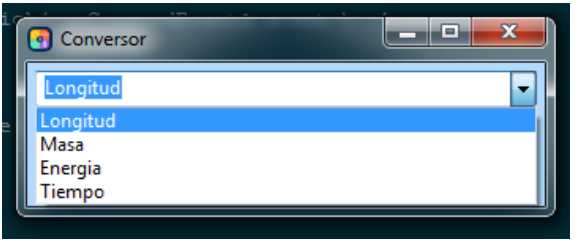
Win_main.cpp

```
win_main::win_main(wxWindow *parent) : wxfbMain(parent) {
    SetIcon(wxIcon(wxIconLocation("img/icon.ico")));

    tUnidades(unidades);
    unidades->SetValue(unidades->GetString(0));
}

void win_main::tUnidades(wxComboBox* c){
    string* list = listUnidades();

    for(int i=0; !list[i].empty(); i++){
        c->Append(list[i]);
    }
}
```

Como podrán observar la función que carga la lista de unidades tiene una peculiaridad y es que en el for en donde debería ir la comprobación del iterador i, se comprueba si la posición i de la lista list esta vacia, este siclo for agrega los elementos de la lista a mi combobox mientras la posición del iterador i contenga elementos

Y también en el constructor he agregado el icono que se muestra en la parte superior izquierda de nuestra aplicación

Al compilar y ejecutar podremos ver reflejados los cambios que atenido nuestra aplicación



Ahora haremos lo mismo con los otros dos combobox que hacen falta con la excepción que estos últimos su contenido varía según el tipo de unidad que seleccionemos, para ello crearemos un mapa que nos retorne la lista de unidades según la que escojamos, agregaremos los elementos del mapa con una función que llamaremos desde el constructor

Utilidades.cpp

```
map<int, string*> Munidades;//mapa unidades

void iniciaMapa(){
    Munidades[0] = unidadesL;
    Munidades[1] = unidadesM;
    Munidades[2] = unidadesE;
    Munidades[3] = unidadesT;
}
```

Win_main.cpp

```
win_main::win_main(wxWindow *parent) : wxfbMain(parent) {
    SetIcon(wxIcon(wxIconLocation("img/icon.ico")));
    iniciaMapa();

    tUnidades(unidades);
    unidades->SetValue(unidades->GetString(0));
}
```

Ahora primero haremos una función que tendrá como parámetro el índice del elemento del map que queremos retornar

Utilidades.cpp

```
std::string* listTipoUnidades(int t){
    return Munidades[t];
}
```

Segundo aremos una función miembro en nuestra aplicación, que reciba un combobox y lo rellene con las unidades del tipo que ese encuentra seleccionado

Win_main.cpp

```
void winConv::updateUnidades(wxComboBox* c){
    string* list = listTipoUnidades(unidades->FindString(unidades->GetValue()));

    for(int i=0; !list[i].empty(); i++){
        c->Append(list[i]);
    }
}
```

Como podrán observar en la llamada a la function listTipoUnidades(int t) el parámetro que indica la posición la lista de unidades lo conseguimos del combobox unidades, dado que los ítems del mismo son una lista con

índice, la función FindString(string) nos retorna ese índice y el estring que le pasamos a dicha función también lo sacamos del mismo combobox con la función GetValue()

Ahora solo nos hace falta llamar a la función en los lugares adecuados como el constructor y cada vez que el combobox unidades se actualice, y como tendríamos que hacer lo mismo en dos sitios haremos una función que se encargue de limpiar los combobox, actualizar los parámetros y aparte coloque las unidades por defecto y el valor inicial de la entrada

Win_main.cpp

```
void win_main::ActualizaParametros(){
    unidadEntrada->Clear();
    unidadSalida->Clear();

    updateUnidades(unidadEntrada);
    updateUnidades(unidadSalida);

    unidadEntrada->SetValue(unidadEntrada->GetString(0));
    unidadSalida->SetValue(unidadSalida->GetString(1));
    entrada->SetValue("1");
}

void win_main::unidades_onClick( wxCommandEvent& event )  {
    ActualizaParametros();
    event.Skip();
}

win_main::win_main(wxWindow *parent) : wxfbMain(parent) {
    SetIcon(wxIcon(wxIconLocation("img/icon.ico")));
    iniciaMapa();

    tUnidades(unidades);
    unidades->SetValue(unidades->GetString(0));

    ActualizaParametros();
}
```

Nota:

> La función void win_main::unidades_onClick(wxCommandEvent& event) no la tienen que crear ya la ha generado zinjai por nosotros solo tienen que poner la llamada a la función ActualizaParametros() dentro de ella;

Uff que poco nos falta ya para acabar nuestra aplicación y presumírsela a nuestros colegas XD ya solo nos queda programar la interacción con la entrada de datos y nosotros dar una salida lógica, ni pensar que hace 13 paginas estábamos diseñando la interfaz gráfica, quien me diría a mí que esto de crear GUI's sería tan fácil XD

Bien continuemos codeando nos queda por programar la parte mas importante de la aplicación, pero antes que nada tenemos que pensar un poco como funciona esto de hacer conversiones de unidad

La fórmula que utilizaremos nosotros es la regla de 3 simple

> x = a*b/c

Si tenemos:

> Tabla de unidades equivalentes a gramos
>
> 1 libra = 453.592 gramos
> 1 onza = 28.3495 gramos
> 1 kilogramo = 1000 gramos

Podemos hacer operaciones del tipo
Convertir 3 libras a gramos
(3L * 453.592 g)/1L = 1360.776 g
Convertir 6 gramos a onzas
(6g * 1z)/ 28.3495 g = 0.211644 z

De modo que si tenemos una tabla con el equivalente por cada unidad para todas las magnitudes podemos calcular el equivalente a la cantidad ingresada por el usuario?

Si podemos pero esa forma de resolver este problema nos daría mucho trabajo ya que porcada magnitud tendríamos que formar una tabla de unidades equivalentes ejemplo.

Nuestro conversor en la tabla de longitud contiene:
milímetros, centímetros, pies, metros, yardas, pulgadas

Entonces crearemos por cada elemento una tabla de equivalencia a todos los elementos

Tabla de unidades equivalentes a milímetros

1 Centímetros = 10
1 Pies = 304.8
1 Metros = 1000
1 Yardas = 914.4
1 Pulgadas = 25.4

Lo que nos daría un resultado de N*N-1 (N = cantidad de elementos) con un total de 6*5 = 30 (6 tablas de 5 unidades), y esto solo es para la magnitud de longitud ni que hablar de las demás

Para resolver este problema he decidido crear una sola tabla por cada tipo de magnitud, ya que si todas las unidades de longitud las podemos convertir a milímetros y de milímetros a todas las demás tendríamos resulto el problema (uno para todos y todos para uno XD)

Entonces primero que todo crearemos las tablas de equivalencia para cada magnitud, segundo una función que se encargue de encontrar a x (la infame XD), tercero una función que se encargue de convertir de una a todas y viceversa

utilidades.cpp

```
float longitud[] = {1, 10, 304.8, 1000, 914.4, 25.4};
float masa[] = {1, 0.001, 453.592, 28.3495, 1000};
float energia[] = {1, 1000, 4.184, 4184, 3600, 3600000};
float tiempo[] = {1, 60, 0.000000001, 0.000001, 0.001, 3600, 86400, 604800};


float reglaDe3(float a, float b, float c){
    return (a*b)/c;
}

float milimetrosTo(int s, float c){
    return reglaDe3(c, 1, longitud[s]);
}

float toMilimetros(int e, float c){
    return reglaDe3(c, longitud[e]);
}
```

Las función milimetrosTo(int s, float c), recibe 2 parámetros, el primero es la salida (la unidad de salida que daremos), el segundo es la cantidad

La función toMilimetros(int e, float c), recibe dos parámetros, el primero es la entrada (la unidad de entrada recibida), el segundo es la cantidad

Repetiremos esto con cada una de las tablas de magnitud

Utilidades.cpp

```
float gramosTo(int s, float c){
    return reglaDe3(c, 1, masa[s]);
}

float toGramos(int e, float c){
    return reglaDe3(c, masa[e]);
}
```

```
float kilojouleTo(int s, float c){
    return reglaDe3(c, 1, energia[s]);
}

float toKilojoule(int e, float c){
    return reglaDe3(c, energia[e]);
}

float segundosTo(int s, float c){
    return reglaDe3(c, 1, tiempo[s]);
}

float toSegundos(int e, float c){
    return reglaDe3(c, tiempo[e]);
}
```

Bien ya podemos hacer las conversiones entre unidades, solo nos hace falta obtener los datos ingresados por el usuario de modo que crearemos otra función que nos ayude a capturar esos datos y lo aremos dentro de la clase de la aplicación

Primero crearemos 5 variables miembro
e, s, c, res, magnitud

Win_main.h

```
class win_main : public wxfbMain {
    void tUnidades(wxComboBox* c);
    void updateUnidades(wxComboBox* c);
    void ActualizaParametros();
    int magnitud;
    int e, s; //(e)ntrada, (s)alida
    double c, res; //(c)antidad (res)ultado

protected:
    void unidadEntrada_update( wxCommandEvent& event ) ;
    void unidadSalida_update( wxCommandEvent& event ) ;
    void unidades_onClick( wxCommandEvent& event ) ;
    void entrada_update( wxCommandEvent& event ) ;

public:
    win_main(wxWindow *parent=NULL);
```

```
    ~win_main();
};
```

e = unidad de entrada
s = unidad de salida
c = cantidad designada a una unidad
res = resultado de la conversión
magnitud = la tabla de magnitud seleccionada

Segundo crear una función miembro que nos ayude a capturar esos parámetros

Win_main.cpp

```
void win_main::capturaParametros(){
    magnitud = unidades->FindString(unidades->GetValue());
    e = unidad1->FindString(unidad1->GetValue());
    s = unidad2->FindString(unidad2->GetValue());
    c = /*ToDo*/
}
```

Como verán la variable c tiene un poco de dificultad a la hora de capturarla, Primero que nada es porque el objeto que contiene su información es un wxTexCtrl y como podrán imaginar el tipo de dato que maneja es un wsString (ha creyeron que diría string pero no haha XD) el wxString es una clase que nos proporciona wxWidgets para el manejo de strings, entre ellas la conversión a números de punto flotante, les dejare un enlace donde podrán ver los diferentes tipos de conversión que tiene esta clase
https://wiki.wxwidgets.org/Converting_everything_to_and_from_wxString

Con lo que la captura de esta variable seria

```
entrada->GetValue().ToDouble(&c);
```

Fácil no XD, la única diferencia es que recibe la variable por referencia y retorna un booleano que indica el resultado de la conversión

Ahora que ya tenemos en nuestro poder todos los datos necesarios para poder hacer la conversión ya podemos hacer la función más importante, será la que se encargue de hacer las conversiones según los parámetros capturados anteriormente

```
void win_main::conversor(){
    switch(magnitud){
        case 0: res = milimetrosTo(s, toMilimetros(e, c)); break;
        case 1: res = gramosTo(s, toGramos(e, c)); break;
        case 2: res = kilojouleTo(s, toKilojoule(e, c)); break;
        case 3: res = segundosTo(s, toSegundos(e, c)); break;
    }

    salida->SetValue(wxString::Format(wxT("%lf"), res));
}
```

Esta función lo que hace es comprobar con un switch que tabla de magnitud se esta tratando, luego con una de las funciones lo convertimos a las unidad de salida ejemplo:

milimetrosTo(int s, float c), esta función convierte una cantidad de milímetros a las demás unidades de su magnitud, con lo cual en el segundo parámetro que es la cantidad necesitamos mandarlo en milímetros por lo que necesitamos convertir la cantidad inicial a milímetros con la función toMilimetros(int e, float c), esta función convierte una cantidad a milímetros.

Ahora solo nos hace falta llamarla en los lugares adecuados junto con la función de que captura los parámetros para asegurarnos de que los parámetros que estamos convirtiendo siempre serán los correctos

```
void win_main::entrada_update( wxCommandEvent& event )  {
    if(entrada->IsEmpty()){
        entrada->SetValue("0");
    }else{
        capturaParametros();
        conversor();
    }
    event.Skip();
}

win_main::~win_main() {
```

```
}

void win_main::unidadEntrada_update( wxCommandEvent& event )  {
    capturaParametros();
    conversor();
    event.Skip();
}

void win_main::unidadSalida_update( wxCommandEvent& event )  {
    capturaParametros();
    conversor();
    event.Skip();
}
```

Como verán en la función void entrada_update(wxCommandEvent& event), que es la que se encarga de capturar la cantidad ingresada por el usuario primero comprueba que este no se encuentre vacío de lo contrario coloca un 0, las otras funciones son las que capturan el tipo de unidad a convertir

Bueno ya puedo dar por finalizado esta aplicación, lo único que le falta es comprobar si el dato ingresado es un numero o no ha si que invito al lector a resolver ese problema por su cuenta, gracia por acompañarme hasta el final de este proyecto gracias y hasta la próxima.

Application.cpp

```
/////////////////////////////////////////////////////////////////
//Name:         Application.cpp
//Author:       Cristobal Rodas
//Created:      21/06/2018
//Copyright:    (c) Cristobal Rodas
//Licence:      wxWindows licence
/////////////////////////////////////////////////////////////////
#include <wx/image.h>
#include "Application.h"
#include "win_main.h"

IMPLEMENT_APP(Application)

bool Application::OnInit() {
    wxInitAllImageHandlers();
    win_main *win = new win_main(NULL);
    win->Show();
    return true;
}
```

Application.h

```
/////////////////////////////////////////////////////////////////
//Name:         Application.h
//Author:       Cristobal Rodas
//Created:      21/06/2018
//Copyright:    (c) Cristobal Rodas
//Licence:      wxWindows licence
/////////////////////////////////////////////////////////////////

#ifndef APPLICATION_H
#define APPLICATION_H

#include <wx/app.h>

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

```
};

#endif
```

Utilidades.cpp

```
///////////////////////////////////////////////////////////////////
//Name:         utilidades.cpp
//Author:       Cristobal Rodas
//Created:      21/06/2018
//Copyright:    (c) Cristobal Rodas
//Licence:      wxWindows licence
///////////////////////////////////////////////////////////////////

#include "utilidades.h"
using namespace std;

string unidades[] = {
    "Longitud",
    "Masa",
    "Energia",
    "Tiempo",
    ""
};

float longitud[] = {1, 10, 304.8, 1000, 914.4, 25.4};
string unidadesL[] = {
    "Milimetros",
    "Centimetros",
    "Pies",
    "Metros",
    "Yardas",
    "Pulgadas",
    ""
};

float masa[] = {1, 0.001, 453.592, 28.3495, 1000};
string unidadesM[] = {
    "Gramos",
    "Miligramos",
    "Libras",
```

```
    "Onzas",
    "Kilogramo",
    ""
};

float energia[] = {1, 1000, 4.184, 4184, 3600, 3600000};
string unidadesE[] = {
    "Julio",
    "Kilojoule",
    "Gram Caloric",
    "Kilocaloria",
    "Vatio-Hora",
    "Kilovatio-Hora",
    ""
};

float tiempo[] = {1, 60, 0.000000001, 0.000001, 0.001, 3600,
86400, 604800};
string unidadesT[] = {
    "Segundo",
    "Minuto",
    "Nanosegundo",
    "Microsegundo",
    "Milisegundo",
    "Hora",
    "Dia",
    "Semana"
    ""
};

map<int, string*> Munidades;//mapa unidades

void iniciaMapa(){
    Munidades[0] = unidadesL;
    Munidades[1] = unidadesM;
    Munidades[2] = unidadesE;
    Munidades[3] = unidadesT;
}

string* listUnidades(){
    return unidades;
}
```

```
string* listTipoUnidades(int t){
    return Munidades[t];
}

float reglaDe3(float a, float b, float c){
    return (a*b)/c;
}

float milimetrosTo(int s, float c){
    return reglaDe3(c, 1, longitud[s]);
}

float toMilimetros(int e, float c){
    return reglaDe3(c, longitud[e]);
}

float gramosTo(int s, float c){
    return reglaDe3(c, 1, masa[s]);
}

float toGramos(int e, float c){
    return reglaDe3(c, masa[e]);
}

float kilojouleTo(int s, float c){
    return reglaDe3(c, 1, energia[s]);
}

float toKilojoule(int e, float c){
    return reglaDe3(c, energia[e]);
}

float segundosTo(int s, float c){
    return reglaDe3(c, 1, tiempo[s]);
}

float toSegundos(int e, float c){
    return reglaDe3(c, tiempo[e]);
}
```

Utilidades.h

```
/////////////////////////////////////////////////////////////////////
//Name:         utilidades.h
//Author:       Cristobal Rodas
//Created:      21/06/2018
//Copyright:    (c) Cristobal Rodas
//Licence:      wxWindows licence
/////////////////////////////////////////////////////////////////////

#ifndef UTILIDADES_H
#define UTILIDADES_H

#include <iostream>
#include <map>

std::string* listUnidades();
void iniciaMapa();
std::string* listTipoUnidades(int t);
float reglaDe3(float a, float b, float c = 1);
float milimetrosTo(int s, float c);
float toMilimetros(int e, float c);
float gramosTo(int s, float c);
float toGramos(int e, float c);
float kilojouleTo(int s, float c);
float toKilojoule(int e, float c);
float segundosTo(int s, float c);
float toSegundos(int e, float c);

#endif
```

win_main.cpp

```
/////////////////////////////////////////////////////////////////////
//Name:         win_main.cpp
//Author:       Cristobal Rodas
//Created:      21/06/2018
//Copyright:    (c) Cristobal Rodas
//Licence:      wxWindows licence
/////////////////////////////////////////////////////////////////////

#include "win_main.h"
```

```
using namespace std;

win_main::win_main(wxWindow *parent) : wxfbMain(parent) {
    SetIcon(wxIcon(wxIconLocation("img/icon.ico")));
    iniciaMapa();

    tUnidades(unidades);
    unidades->SetValue(unidades->GetString(0));

    ActualizaParametros();
}

void win_main::unidades_onClick( wxCommandEvent& event )  {
    ActualizaParametros();
    event.Skip();
}

void win_main::entrada_update( wxCommandEvent& event )  {
    if(entrada->IsEmpty()){
        entrada->SetValue("0");
    }else{
        capturaParametros();
        conversor();
    }
    event.Skip();
}

win_main::~win_main() {

}

void win_main::unidadEntrada_update( wxCommandEvent& event )  {
    capturaParametros();
    conversor();
    event.Skip();
}

void win_main::unidadSalida_update( wxCommandEvent& event )  {
    capturaParametros();
    conversor();
    event.Skip();
}
```

```
void win_main::tUnidades(wxComboBox* c){
    string* list = listUnidades();

    for(int i=0; !list[i].empty(); i++){
        c->Append(list[i]);
    }
}

void win_main::updateUnidades(wxComboBox* c){
    string* list = listTipoUnidades(unidades-
>FindString(unidades->GetValue()));

    for(int i=0; !list[i].empty(); i++){
        c->Append(list[i]);
    }
}

void win_main::ActualizaParametros(){
    unidadEntrada->Clear();
    unidadSalida->Clear();

    updateUnidades(unidadEntrada);
    updateUnidades(unidadSalida);

    unidadEntrada->SetValue(unidadEntrada->GetString(0));
    unidadSalida->SetValue(unidadSalida->GetString(1));
    entrada->SetValue("1");
}

void win_main::capturaParametros(){
    magnitud = unidades->FindString(unidades->GetValue());
    e = unidadEntrada->FindString(unidadEntrada->GetValue());
    s = unidadSalida->FindString(unidadSalida->GetValue());
    entrada->GetValue().ToDouble(&c);
}

void win_main::conversor(){
    switch(magnitud){
        case 0: res = milimetrosTo(s, toMilimetros(e, c));
break;
        case 1: res = gramosTo(s, toGramos(e, c)); break;
```

```
        case 2: res = kilojouleTo(s, toKilojoule(e, c)); break;
        case 3: res = segundosTo(s, toSegundos(e, c)); break;
    }

    salida->SetValue(wxString::Format(wxT("%lf"), res));
}
```

win_main.h

```
///////////////////////////////////////////////////////////////////
//Name:         win_main.h
//Author:       Cristobal Rodas
//Created:      21/06/2018
//Copyright:    (c) Cristobal Rodas
//Licence:      wxWindows licence
///////////////////////////////////////////////////////////////////

#ifndef WIN_MAIN_H
#define WIN_MAIN_H
#include "wxfb_project.h"
#include <iostream>
#include "utilidades.h"

class win_main : public wxfbMain {
    void tUnidades(wxComboBox* c);
    void updateUnidades(wxComboBox* c);
    void ActualizaParametros();
    void capturaParametros();
    void conversor();
    int magnitud;
    int e, s; //(e)ntrada, (s)alida
    double c, res; //(c)antidad (res)ultado

protected:
    void unidadEntrada_update( wxCommandEvent& event ) ;
    void unidadSalida_update( wxCommandEvent& event ) ;
    void unidades_onClick( wxCommandEvent& event ) ;
    void entrada_update( wxCommandEvent& event ) ;

public:
    win_main(wxWindow *parent=NULL);
```

```
    ~win_main();
};

#endif
```

wxfb_project.cpp

```
///////////////////////////////////////////////////////////////////////
// C++ code generated with wxFormBuilder (version May 29 2018)
// http://www.wxformbuilder.org/
//
// PLEASE DO *NOT* EDIT THIS FILE!
///////////////////////////////////////////////////////////////////////

#include "wxfb_project.h"

///////////////////////////////////////////////////////////////////////

wxfbMain::wxfbMain( wxWindow* parent, wxWindowID id, const
wxString& title, const wxPoint& pos, const wxSize& size, long
style ) : wxFrame( parent, id, title, pos, size, style )
{
    this->SetSizeHints( wxSize( 344,129 ), wxSize( 344,129 ) );
    this->SetBackgroundColour( wxColour( 179, 217, 255 ) );

    wxBoxSizer* bSizer2;
    bSizer2 = new wxBoxSizer( wxVERTICAL );

    wxBoxSizer* bSizer3;
    bSizer3 = new wxBoxSizer( wxVERTICAL );

    unidades = new wxComboBox( this, wxID_ANY, wxT("Combo!"),
wxDefaultPosition, wxDefaultSize, 0, NULL, 0 );
    bSizer3->Add( unidades, 0, wxALL|wxEXPAND, 5 );


    bSizer2->Add( bSizer3, 0, wxEXPAND, 5 );
```

```
    wxBoxSizer* bSizer4;
    bSizer4 = new wxBoxSizer( wxHORIZONTAL );

    wxBoxSizer* bSizer5;
    bSizer5 = new wxBoxSizer( wxVERTICAL );

    unidadEntrada = new wxComboBox( this, wxID_ANY,
wxT("Combo!"), wxDefaultPosition, wxDefaultSize, 0, NULL, 0 );
    bSizer5->Add( unidadEntrada, 1,
wxEXPAND|wxTOP|wxRIGHT|wxLEFT, 5 );

    entrada = new wxTextCtrl( this, wxID_ANY, wxEmptyString,
wxDefaultPosition, wxDefaultSize, 0 );
    bSizer5->Add( entrada, 1, wxEXPAND|wxRIGHT|wxLEFT, 5 );


    bSizer4->Add( bSizer5, 1, wxALL|wxEXPAND, 5 );

    m_staticText2 = new wxStaticText( this, wxID_ANY, wxT("="),
wxDefaultPosition, wxDefaultSize, 0 );
    m_staticText2->Wrap( -1 );
    bSizer4->Add( m_staticText2, 0,
wxALL|wxALIGN_CENTER_VERTICAL, 5 );

    wxBoxSizer* bSizer51;
    bSizer51 = new wxBoxSizer( wxVERTICAL );

    unidadSalida = new wxComboBox( this, wxID_ANY,
wxT("Combo!"), wxDefaultPosition, wxDefaultSize, 0, NULL, 0 );
    bSizer51->Add( unidadSalida, 1,
wxEXPAND|wxTOP|wxRIGHT|wxLEFT, 5 );

    salida = new wxTextCtrl( this, wxID_ANY, wxEmptyString,
wxDefaultPosition, wxDefaultSize, 0 );
    bSizer51->Add( salida, 1, wxEXPAND|wxRIGHT|wxLEFT, 5 );


    bSizer4->Add( bSizer51, 1, wxEXPAND|wxALL, 5 );


    bSizer2->Add( bSizer4, 0, wxEXPAND, 5 );
```

```
    this->SetSizer( bSizer2 );
    this->Layout();

    this->Centre( wxBOTH );

    // Connect Events
    unidades->Connect( wxEVT_COMMAND_COMBOBOX_SELECTED,
wxCommandEventHandler( wxfbMain::unidades_onClick ), NULL, this
);
    unidadEntrada->Connect( wxEVT_COMMAND_COMBOBOX_SELECTED,
wxCommandEventHandler( wxfbMain::unidadEntrada_update ), NULL,
this );
    entrada->Connect( wxEVT_COMMAND_TEXT_UPDATED,
wxCommandEventHandler( wxfbMain::entrada_update ), NULL, this );
    unidadSalida->Connect( wxEVT_COMMAND_COMBOBOX_SELECTED,
wxCommandEventHandler( wxfbMain::unidadSalida_update ), NULL,
this );
}

wxfbMain::~wxfbMain()
{
    // Disconnect Events
    unidades->Disconnect( wxEVT_COMMAND_COMBOBOX_SELECTED,
wxCommandEventHandler( wxfbMain::unidades_onClick ), NULL, this
);
    unidadEntrada->Disconnect( wxEVT_COMMAND_COMBOBOX_SELECTED,
wxCommandEventHandler( wxfbMain::unidadEntrada_update ), NULL,
this );
    entrada->Disconnect( wxEVT_COMMAND_TEXT_UPDATED,
wxCommandEventHandler( wxfbMain::entrada_update ), NULL, this );
    unidadSalida->Disconnect( wxEVT_COMMAND_COMBOBOX_SELECTED,
wxCommandEventHandler( wxfbMain::unidadSalida_update ), NULL,
this );

}
```

wxfb_project.h

```
///////////////////////////////////////////////////////////////////////
///////////
```

```
// C++ code generated with wxFormBuilder (version May 29 2018)
// http://www.wxformbuilder.org/
//
// PLEASE DO *NOT* EDIT THIS FILE!
///////////////////////////////////////////////////////////////////////
///////////

#ifndef __WXFB_PROJECT_H__
#define __WXFB_PROJECT_H__

#include <wx/artprov.h>
#include <wx/xrc/xmlres.h>
#include <wx/string.h>
#include <wx/combobox.h>
#include <wx/gdicmn.h>
#include <wx/font.h>
#include <wx/colour.h>
#include <wx/settings.h>
#include <wx/sizer.h>
#include <wx/textctrl.h>
#include <wx/stattext.h>
#include <wx/frame.h>

///////////////////////////////////////////////////////////////////////
///////////


///////////////////////////////////////////////////////////////////////
///////////////
/// Class wxfbMain
///////////////////////////////////////////////////////////////////////
///////////////
class wxfbMain : public wxFrame
{
    private:

    protected:
        wxComboBox* unidades;
        wxComboBox* unidadEntrada;
        wxTextCtrl* entrada;
        wxStaticText* m_staticText2;
        wxComboBox* unidadSalida;
```

```
        wxTextCtrl* salida;

        // Virtual event handlers, overide them in your derived
class
        virtual void unidades_onClick( wxCommandEvent& event ) {
event.Skip(); }
        virtual void unidadEntrada_update( wxCommandEvent& event
) { event.Skip(); }
        virtual void entrada_update( wxCommandEvent& event ) {
event.Skip(); }
        virtual void unidadSalida_update( wxCommandEvent& event
) { event.Skip(); }


    public:

        wxfbMain( wxWindow* parent, wxWindowID id = wxID_ANY,
const wxString& title = wxT("Conversor"), const wxPoint& pos =
wxDefaultPosition, const wxSize& size = wxSize( 344,129 ), long
style = wxDEFAULT_FRAME_STYLE|wxTAB_TRAVERSAL );

        ~wxfbMain();

};

#endif //__WXFB_PROJECT_H__
```