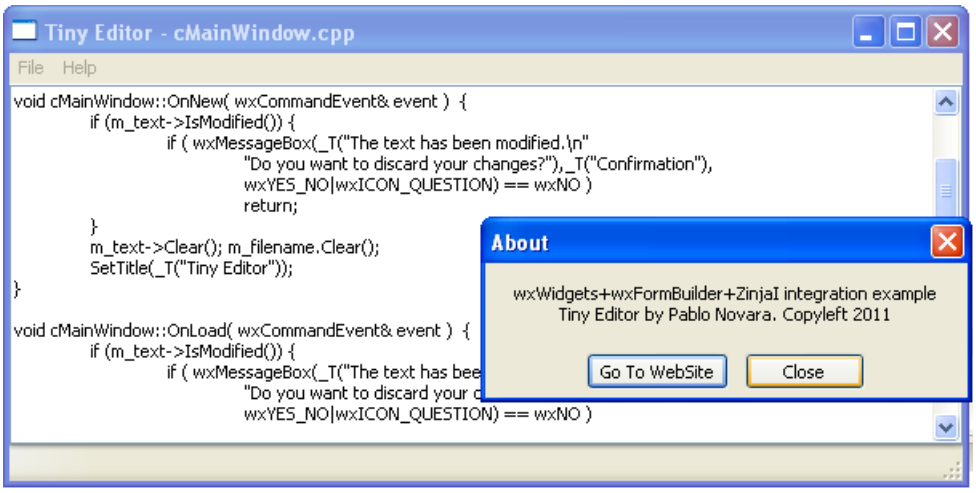
Developing wxWidgets applications with ZinjaI and wxFormBuilder

An Introductory Example: TinyEditor

by Pablo Novara
zaskar_84@yahoo.com.ar
Last revision: 22-07-2011

Introduction

This tutorial will show you how to easily create a simple portable C++ application with a Graphical User Interface (GUI), combining wxWidgets+wxFormBuilder+ZinjaI. The example application is a very basic text editor (similar to notepad), but it's enough to illustrate how this tool chain works, and how fast you can build GUI apps, “drawing” the interface with your mouse and writing very few lines of code to connect events.



Example Application: TinyEditor

I'll assume that you have some knowledge on C++ and object-oriented programming but I won't request any experience on writing apps with GUIs (in case you have it, it can help a lot).

If you don't want to read the whole explanation and want to go directly to the action, you can install the tools (ZinjaI, wxWidgets, and wxFormBuilder) and read just the key steps (text in blue) ignoring everything else and try to figure out yourself the integration mechanism. For an experienced programmer this should be enough.

Now you should start familiarizing with event-driven programs. It means that your program will be waiting for an event to occur (for instance, clicking on a button, typing in text box, closing a window, are user actions that generate events), and when the event happens it do something to react to that event (open a file, show another window, display a message, whatever). wxWidgets will handle what we call “event-loop”, so, most of the time, the library has the control, and you don't worry about it. The library handle a lot of events itself and acts in the expected way if you don't want to replace them, but when something interesting goes on (a particular event you are waiting for), the library will tell you by calling a function you choose in order to give you the control (actually, this will be a method that will be attached to the event). There you can do whatever you want with the GUI or anything else you usually do in a program, and when you're done, you return the control to the library and sit down to wait for another meaningful event.

In this tutorial you'll be defining windows and setting events with wxFormBuilder. Here is the procedure's summary:

1) create a wxFormBuilder project from ZinjaI

2) open wxFormBuilder and draw your interface “with your mouse” without writing a single line of code

3) choose a name for the visual components you will need to refer from the code, choose the events you want to handle and choose name for the methods associated to those events

4) use ZinjaI to derive a class that has the components as attributes and the events as methods with three clicks

5) write something inside those methods to get your app finally working

The general idea is as follows: you draw some nice window in the designer so the designer can write for you the code for the class that really generate that window. Then you tell the designer witch events you need so the designer can write in the window's class virtual methods for them and the corresponding associations. Finally, you inherit from that class and rewrite those virtual methods. The inheritance is needed so the designer can regenerate the window class in the future if you change something without overwriting the methods you already coded. It isn't that hard, is it?

Before you go on, a little request: if you think that some part of this tutorial looks awful, is a mess to understand, or my English went so down, please write me and I'll try to improve it. I'm also ZinjaI's developer, so if you can imagine some tool or feature that will ease your work with wxFormBuilder and wxWidgets also let me know.

License information

Part 0: Getting the tools and preparing the environment

In order to create your first wxWidgets application, in this tutorial you are going to use a visual designer named wxFormBuilder (the one that lets you draw the interface without writing a single line of code) and an specific IDE named ZinjaI (the big editor where you create the project and write your part of the code). Of course, you'll also need wxWidgets library.

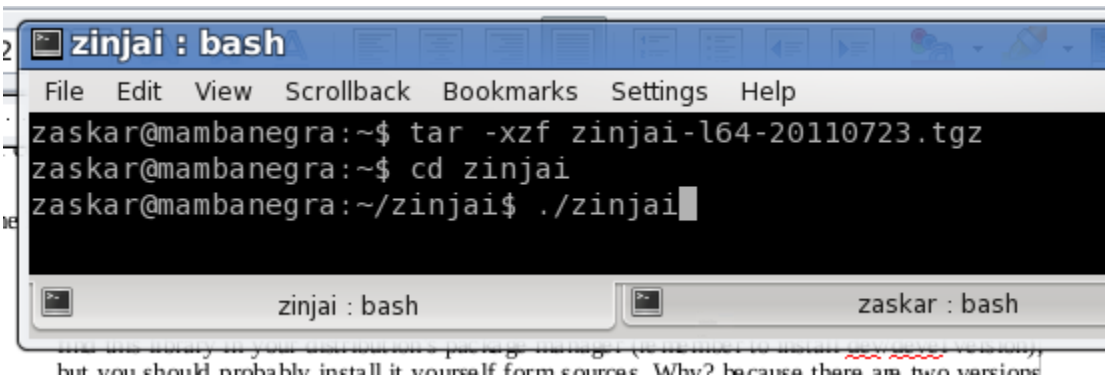
The library: **wxWidgets**

You can get wxWidgets from several sources. You can always grab sources from http://www.wxwidgets.org, but there are easier ways. If you are a Windows user just skip this step, because the needed files will be installed with the IDE. If you are a Linux user, you can find this library in your distribution's package manager (remember to install dev/devel version), but you should probably install it yourself form sources. Why? because there are two versions of the library: unicode and ansi. The difference is in how they handle strings. Let's just say that unicode should be always better (so most repositories have this one), but ansi is easier for beginners (you won't need to convert from/to the usual cstrings as with unicode). So, if you want to build your own ansi library, download the source, extract them somewhere, get into its folder and from a terminal do: `./configure –enable-ansi --disable-unicode && make && sudo make install`

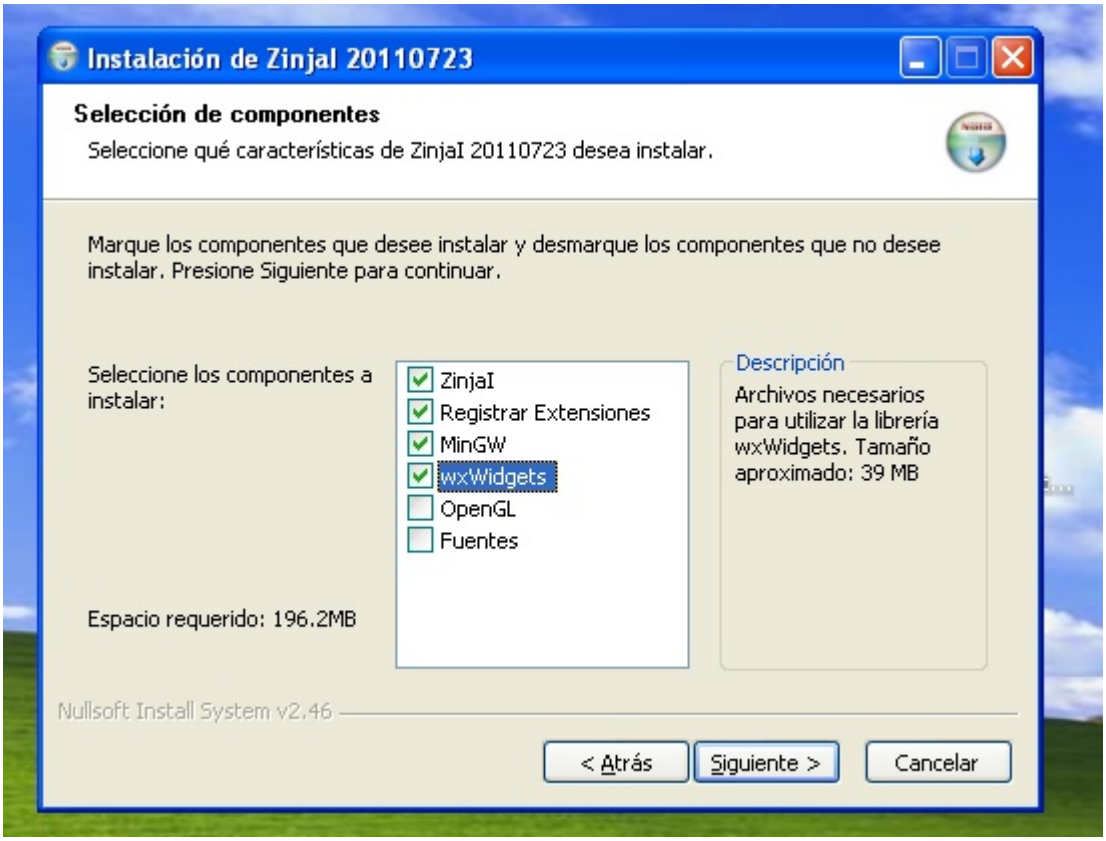
We'll use wxWidgets because it is the one I use, and I really like it. There are other alternatives like QT, fltk, gtk+, and many more. Each one has pros and cons, but after messing around with some of them I ended writing wxWidgets apps. It's object-oriented, it's fully featured, it's free and very portable, it's seems to be very efficient, so why not?

The IDE: **ZinjaI**

You can download the IDE from http://zinjai.sourceforge.net. Linux version is just a tgz file. Extract it somewhere and just run the “zinjai” file in the uncompressed “zinjai” folder. Windows version is a classic installer where you normally click “next” several times without reading. But wait, when it asks you what components to install you should select wxWidgets, that is unselected by default. If you already have it, check you have 20110802 or newer version.



extracting and running ZinjaI from linux terminal



Installing ZinjaI with wxWidgets support in Windows XP

You could use wxWidgets and wxFormBuilder with any other IDE, but you will notice that ZinjaI will do some extra work for you: it will call wxFormBuilder to ask for the window's code before compiling, it will realize if you changed something there to add the new event-methods in your child classes, it will set up compiling and linking commands in order to use wxWidgets, it will provide you quick access to wxFormBuilder and wxWidgets' reference, and some more random stuff.

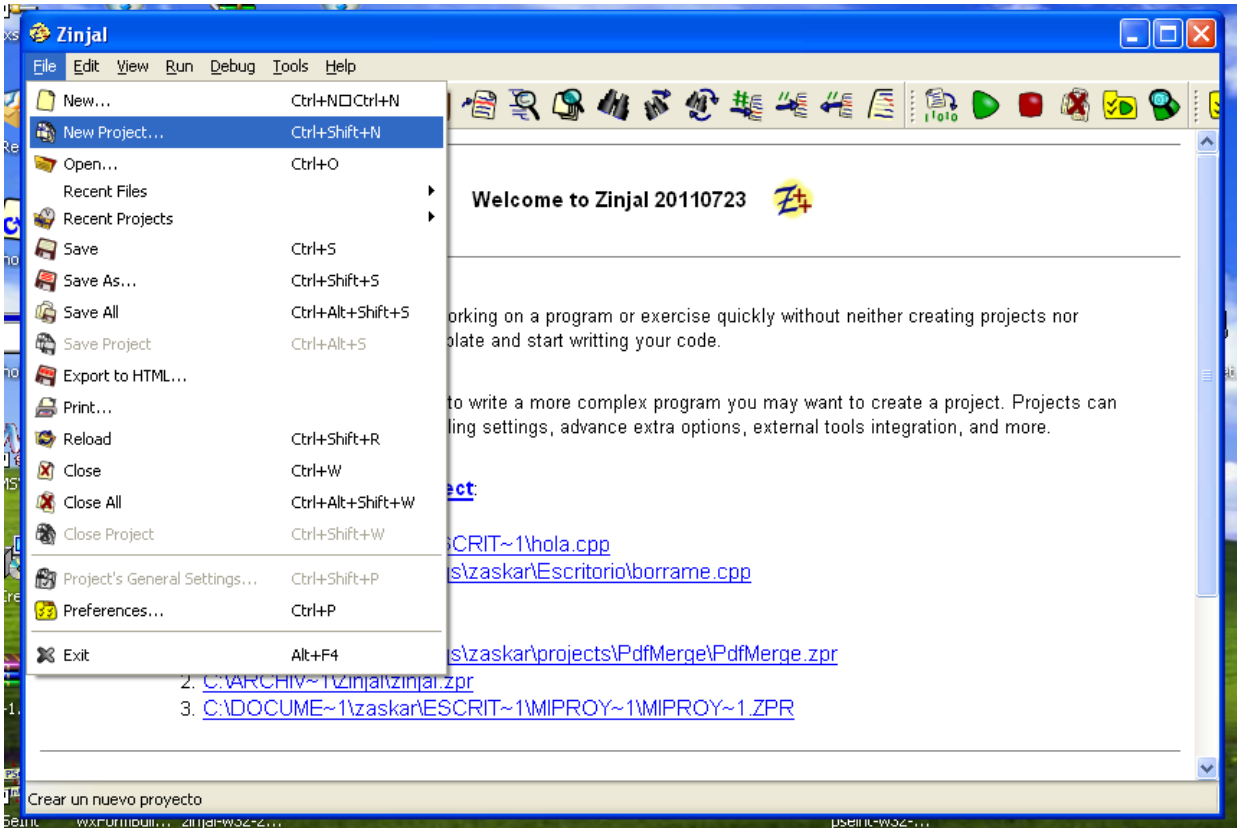
The designer: **wxFormBuilder**

Download and install it from your distribution's package manager, or get it from http://wxformbuilder.org/. Latest stable version today is 3.1.70. Installation is straight forward, and if you do it without changing the destination folder, ZinjaI will find it automatically. If it does not, you'll see a warning message. In that case, you can set wxFormBuilder's path from “Paths 2” tab in “Preferences” dialog (File menu->Preferences).

Part 1: Creating the project

Lets start with the project. The first thing to do is loading ZinjaI. Once you're in create a new project:

0) Launch ZinjaI

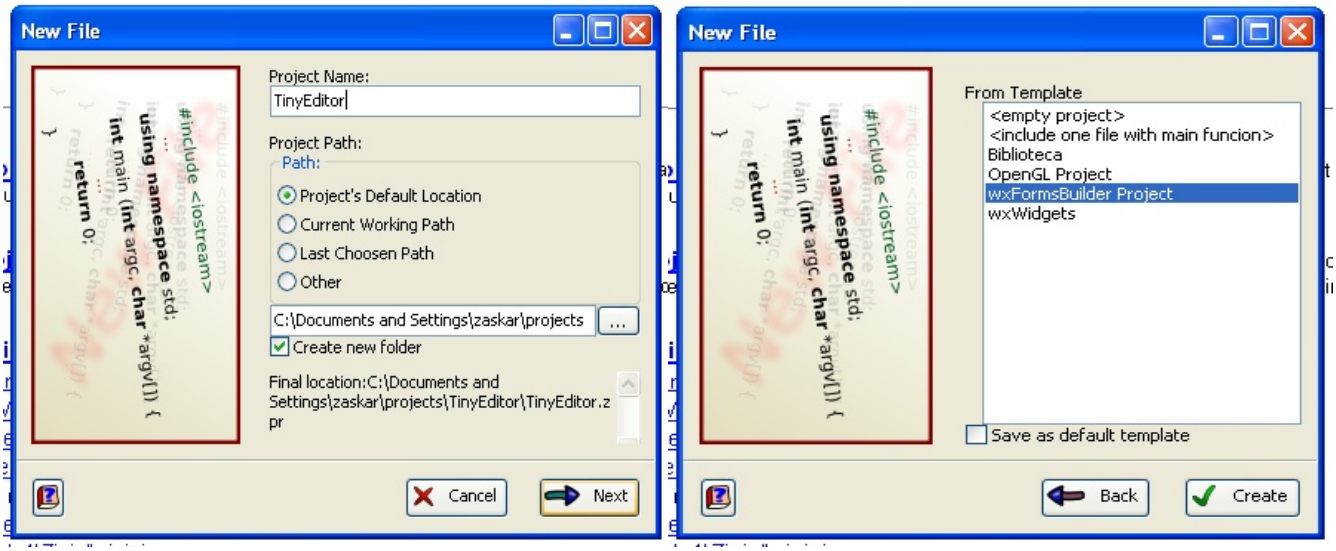


ZinjaI's main window

1) Click on “Create New Project” in welcome panel if you see it, or chose “New Project...” item from “File” menu.

2) In the first step you should choose where to place the project and how to name it. I recommend you choosing names and a paths without withe-spaces. A new folder with that name will be created and all projects files will be in that folder. Type “TinyEditor” and click “Next” to move to next step.

3) In the second step of the wizard you can choose a template for your project. It means default compiling and linking options, some initial code and source files, etc. Choose “wxFormBuilder Project” and click “Create”.



creating a wxFormBuilder project from ZinjaI

If you see some question about old versions and new versions of wxFormBuilder projects just say yes. It means that the template was made with a version of wxFormBuilder that is older than the one you have now. Saying yes, wxFormBuilder will update the project file to the current version.

Now you see a txt file with some description and tips about the project. Just close it, you'll find all that information in this tutorial.

In the left panel, you'll see a tree with three main items: “Sources”, “Headers”, and “Other Files”. The first one contains the files that will be compiled. The second one header files, useful for the auto-completion system, and the third one contains among others the wxFormBuilder file, called “Ventanas.fbp”. Double clicking this file will launch wxFormBuilder, but don't do it yet.

The sources and headers contains the following files:

Application.cpp/.h: a method from this class will be your main, I'll explain later.

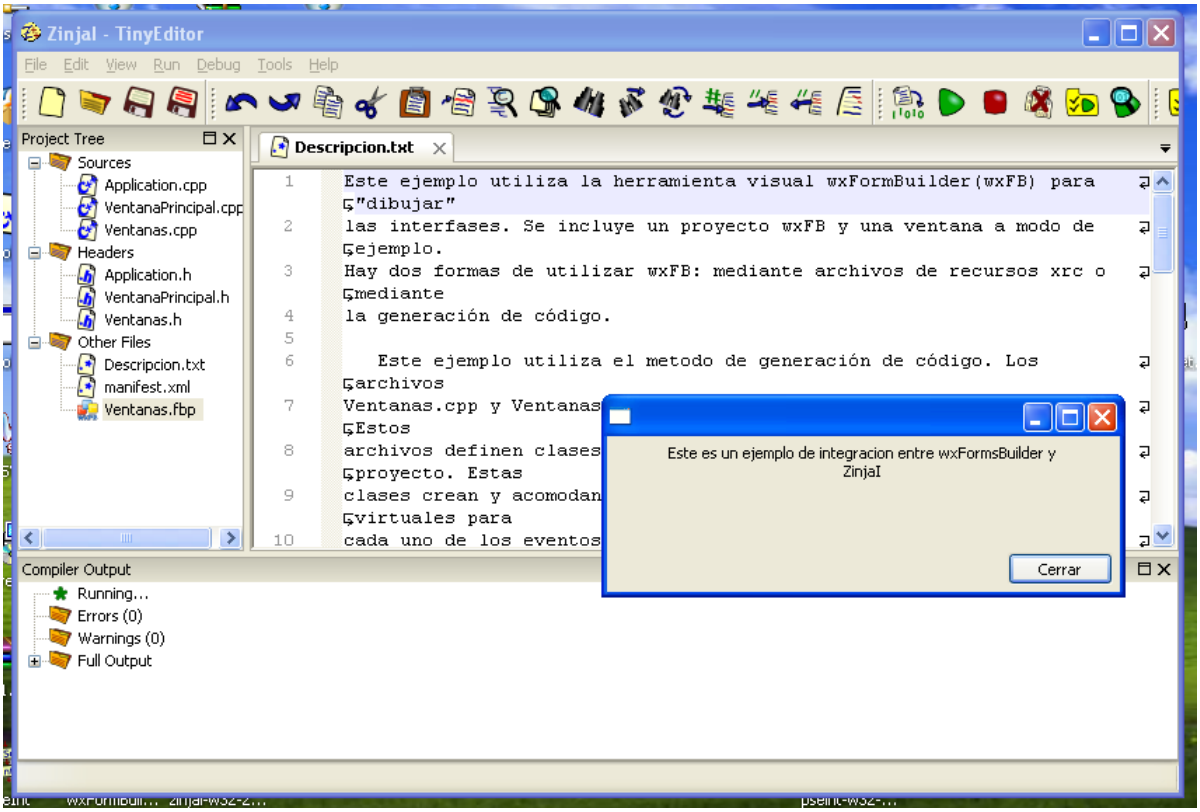
Ventanas.cpp/.h: code generated automatically by wxFormBuilder.

VentanaPrincipal.cpp/.h: a sample window with some events already programmed.

Descripcion.txt: a readme file that you can safely delete.

manifest.xml: a file for windows platform that make your controls looks nicer (it asks the system to apply the theme in common controls).

Try to run this example project by hitting F9 and see what happens. There's an example window with a simple message and a close button.

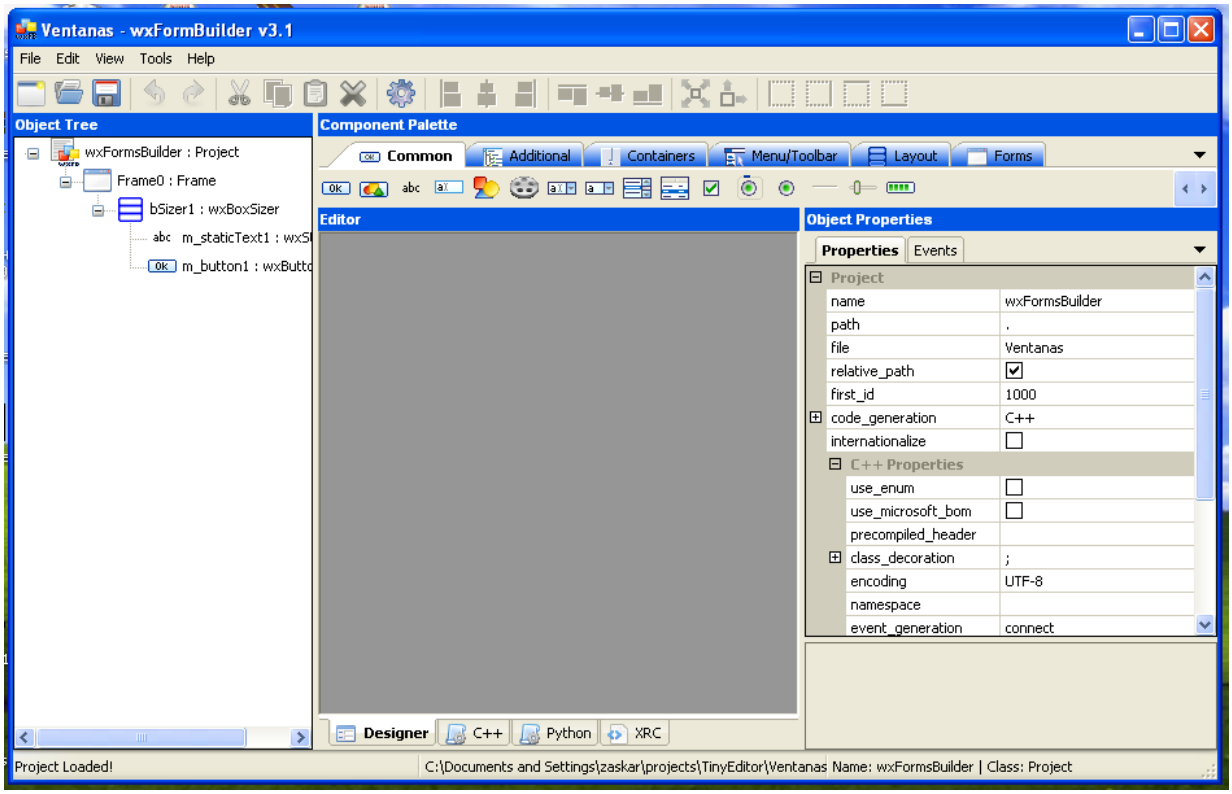


wxFormBuilder project running

Part 2: Drawing the interface

In this part, you'll work with the designer:

1) Double click on Ventanas.fbp in Other Files will open the designer. There, you'll see a screen like this one:



wxFormBuilder's main window

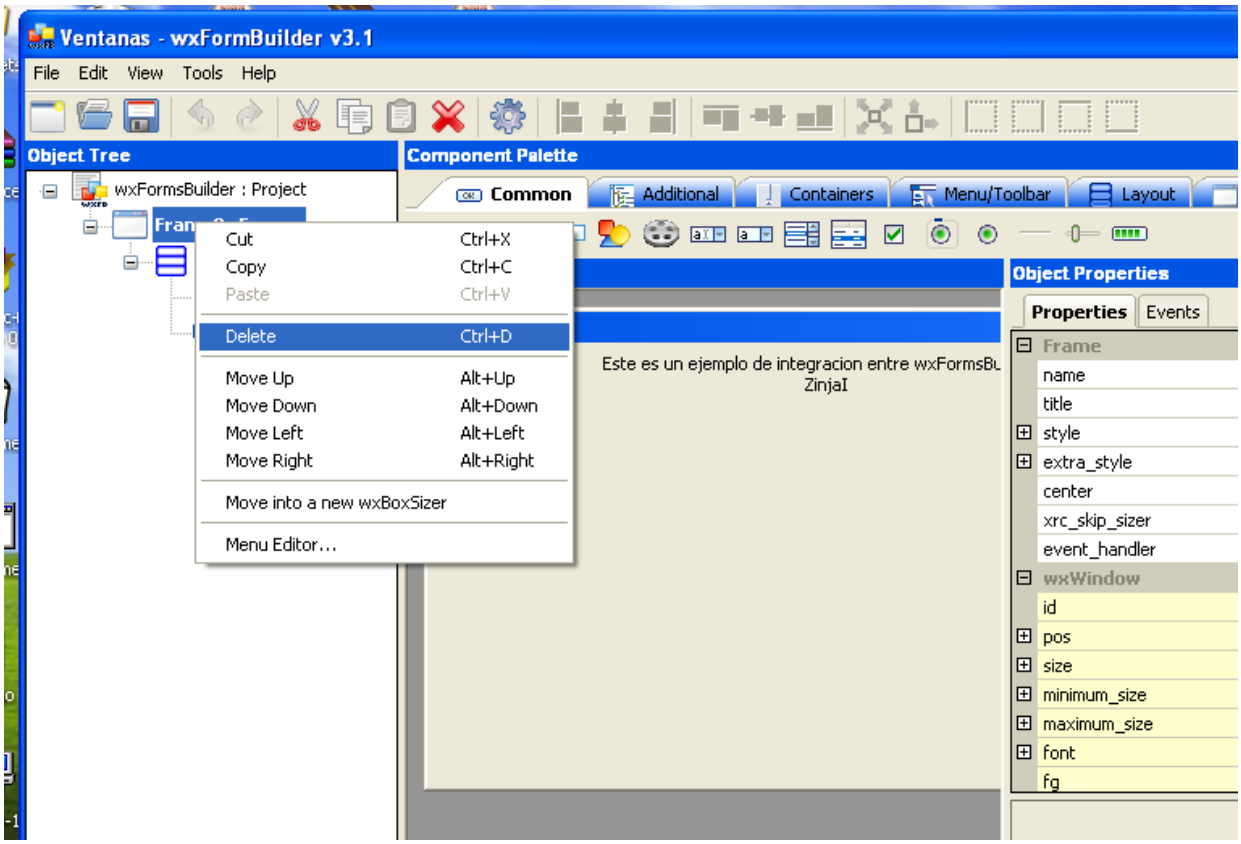
The window have five main areas: a toolbar on the top, the Object Tree on the left, the Object Properties panel on the right, the Editor space in the center and the Component Palette just above it. To create your windows you must choose components from the palette (starting with a frame or a dialog), and then define its properties. The components will show up in the Editor area and you'll see the components hierarchy (wich one has wich other one inside) in the Object Tree. By selecting one of the tree's items you'll see the associated properties in the Object Properties panel. There are have two tabs: the first one to control appearance, basic behavior, name, and other stuff, the second to choose wich events are interesting for your application.

There is some policy about how components are arranged inside a window that you should know: unlike other designers, components won't have an absolute fixed position and size. Instead, you must use sizers. Sizers are some kind of invisible components whose task is to arrange other components. For instance, you have one sizer to arrange components one next to the other horizontally, or one below the other vertically. So you add components to the sizer and you tell it how to treat them, so the sizer

will calculate for you minimal and optimal sizes for every system. This way, you won't worry about the differences between platforms (operative systems) for a single component, or the different themes in a platform, the sizer will do for you. For instance, you say to the sizer: draw these three components one next to the other, expand the first one horizontally, set the second to its minimal size, and expand the third one in both directions to complete the windows size, and the sizer will try to do it in a clever way. If you resize the window, the sizer will automatically recalculate each child component's size. So, when you create a window (frame or dialog), the first component inside it will be a sizer (even if the window contains just one other component).

Let's start with main window. Main window will have a menu bar and a text component that will fill the whole window area. Menu bars, like status bars are some exceptions that should go directly as window's childs, without the need of a sizer. But text control will be inside a sizer that will be responsible of keeping it as big as possible when the window shows up or changes its size:

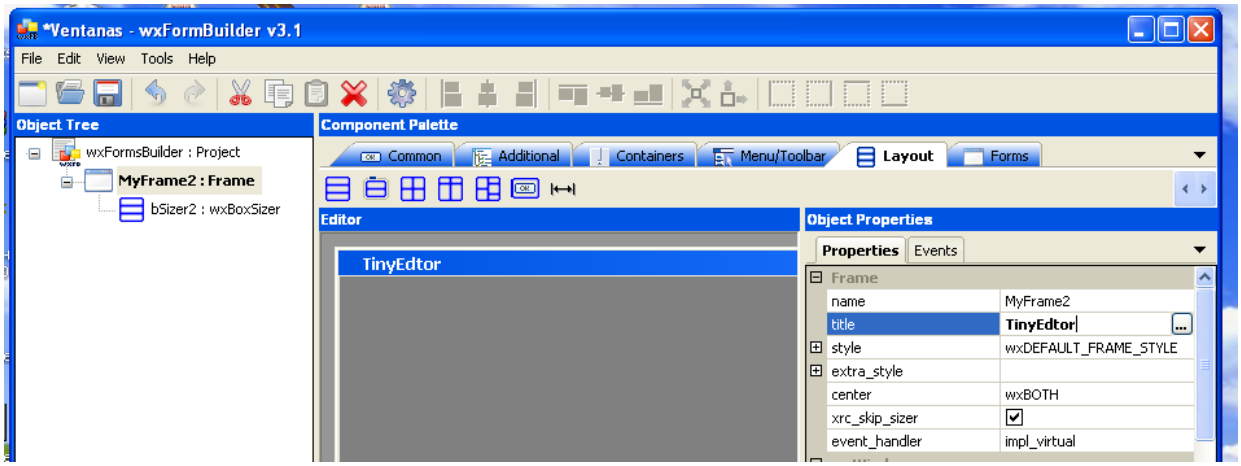
1) The example project already has a window named “VentanaPrincipal”. Select it in the Object Tree and hit Ctrl+D (or use tree's contextual menu with right click) to delete it and start with an empty project.



2) Create a new wxFrame clicking the first button from the “Forms” tab in Component Palette. An empty window will show up in the Editor area. Set its title in Object Properties panel to “Tiny Editor”.

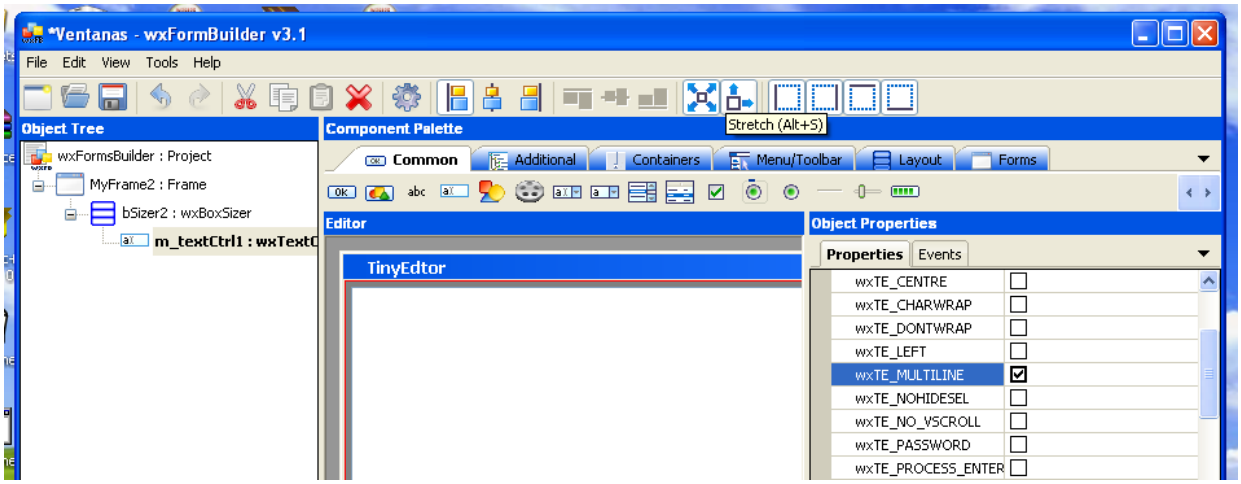
3) Go to “Layout” tab in Component Palette and choose the first item. It's the simplest sizer (wxBoxSizer). It can arrange component horizontally or vertically. There are more complex

ones but you can build almost any configuration just by combining wxBoxSizers (a sizer can be inside other sizer). You won't see any change in the window, but you can see the sizer in the Object Tree.



4) Now add the main component to the sizer: the text control. Use the fourth button (wxTextCtrl) in the “Common” tab in the palette. You'll see a little text box in the top-left corner of the window. To tell the sizer you want to expand it in both directions you can use the Expand (Alt+W) and Stretch (Alt+S) buttons from toolbar. You can see that those buttons change the “proportion” field and the “wxEXPAND” flag in the property panel.

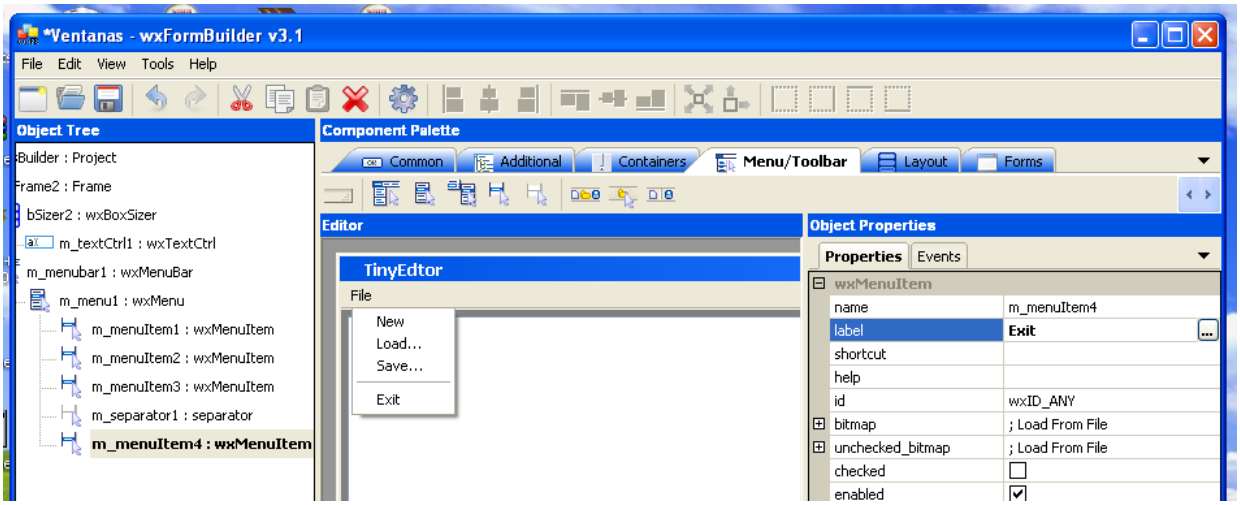
5) The text control is now in place and with full size. To enable multiline text (default text control shows a single line), you must find “style” item in Properties and check “wxTE_MULTILINE”.



6) Now go to “Menu/Toolbar” tab in palette and click wxMenuBar button (the second one). This will add an empty menu bar on the window. With the third button (wxMenu) you can add the actual Menus. Click it one time to add the first menu and set its “label” property to “File”.

7) Now you have and empty menu that should contain “New”, “Load...”, “Save...”, and “Exit” items inside. Select it in the Object Tree and use the fifth button from “Menu/Toolbar” tab in Components Palette (wxMenuItem) to add those items. Again, use “label” property to set their text. You can click the sixth element from the palette before inserting the last menu item to add

a little separator between New/Load/Save items and exit one.

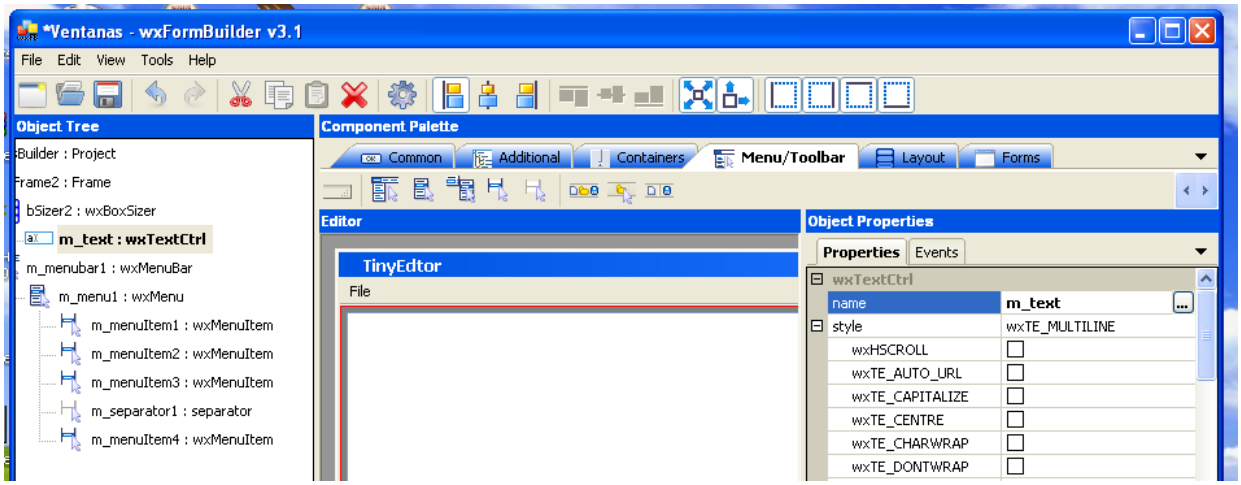


Now the main window is complete. At least visually. Lets give it some life.

Part 3: Dealing with controls and events

To use the window you created in the designer from the IDE you need to follow two more steps: define names for the components you need to reference in the code, and define names for the methods that will be associated with events in the final class. You can type the name for a control in the first entry of Object Properties panel. For a window (frame, dialog, or anything from Forms tab in Component Palette), the name is the name of the class that wxFormBuilder will generate for you. For another component, this name will be the name of a pointer attribute in the class and you can use it later to do things with the component (get/set its value, change its properties, show/hide, anything). As this is a variable or class name, it cannot contains spaces, operators nor any other forbidden character. In this example you'll need to reference two components: the whole window and the text control:

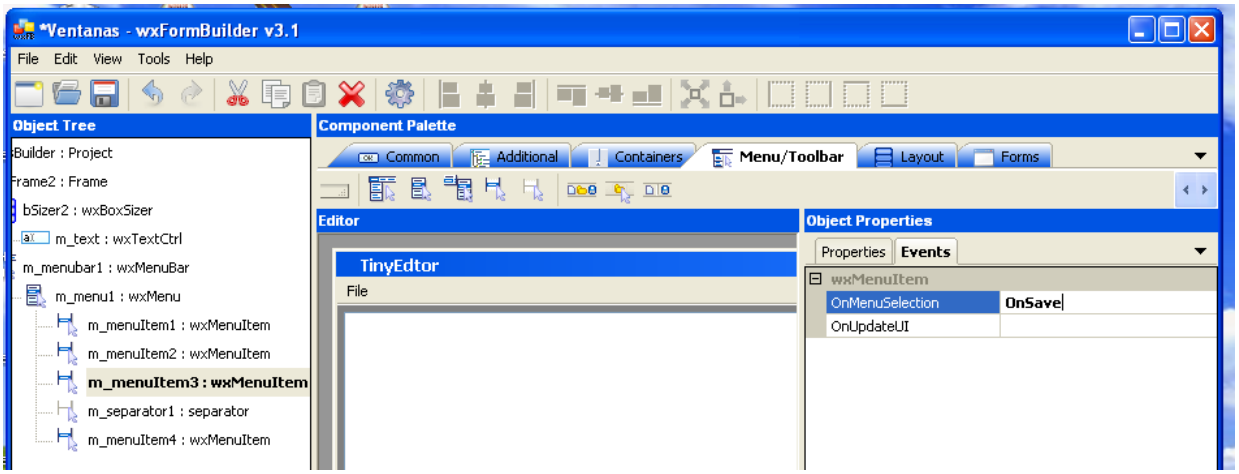
1) Select the frame in the Object Tree an set the name “bMainWindow” in its properties. Since you'll make a new class derived from this one later, you need to think two names for the window, one for the base class, (that's why there is a preceding b), and one for the child class, that will be cMainWindow. Here you enter the base class name.



2) Select the text control in the tree an set the name “m_text” in properties. There are many conventions about how to choose names. Most of wxWidgets examples use m_ prefix for attributes. Also uncheck “wxALL” option in “flags” item (the last one) from Object Properties.

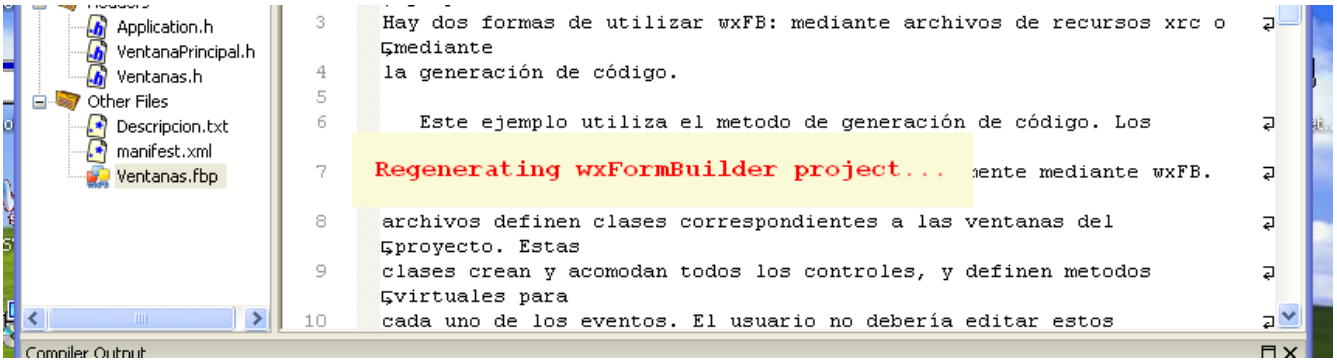
Now you have to choose wich events you want to control and define names for their associated methods. When you select an item in the Object Tree or in Editor area and go to “Events” tab in Object Properties panel you can see the available events for that kind of item, and you can define names for the corresponding methods (empty names are methods you won't care about):

3) Select “New”, “Load...”, “Save...” and “Exit” menu items and set their “OnMenuSelection” event to “OnNew”, “OnLoad”, “OnSave” and “OnExit” respectively. Again, it's a common convention in wxWidgets programs to start events names with On prefix.



4) Finally, save the project (hit Ctrl+S or click on the save button in toolbar) and close wxFormBuilder.

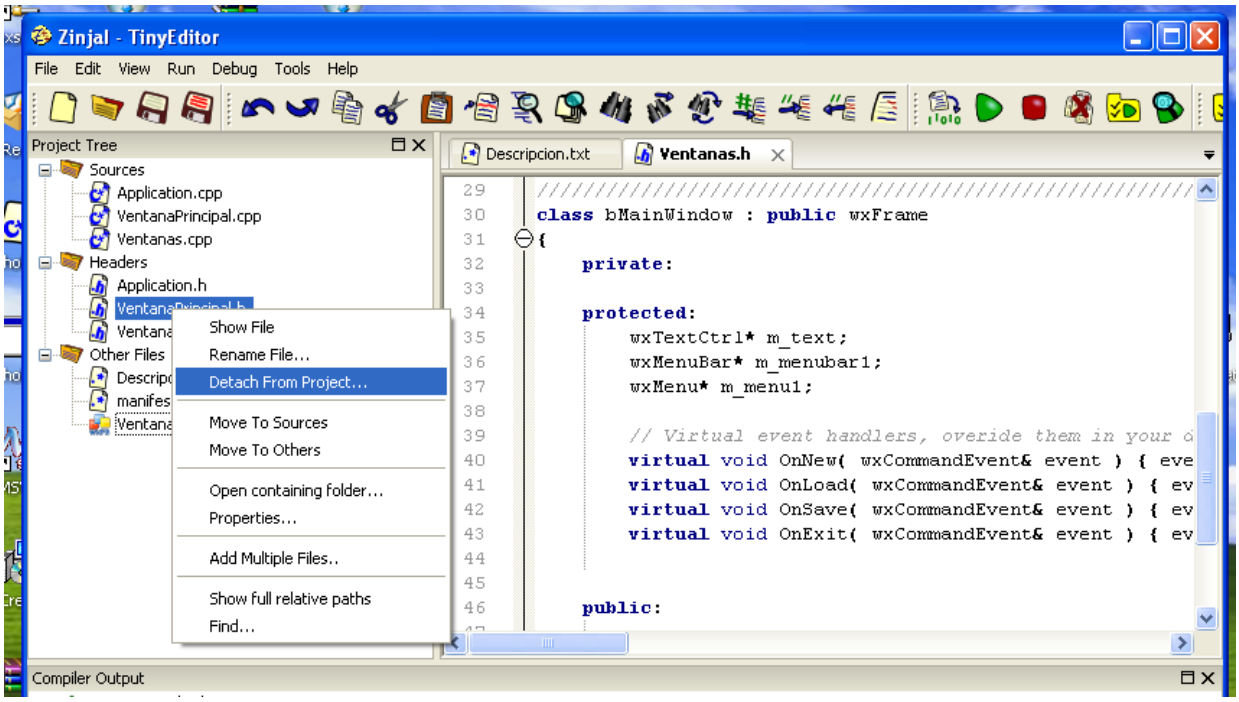
When you go back to ZinjaI it will automatically detect that there is some change and it will ask wxFormBuilder to regenerate the code.



Part 4: Writing some code

Now you have the window's base class written by wxFormBuilder in files Ventanas.h and Ventanas.cpp. By creating a bMainWindow object you create a new window just like the one you have drawn in parts 2 and 3. Files VentanaPrincipal.cpp and VentanaPrincipal.h where related to the example window you deleted at the beginning of part 2. So you should delete this old code:

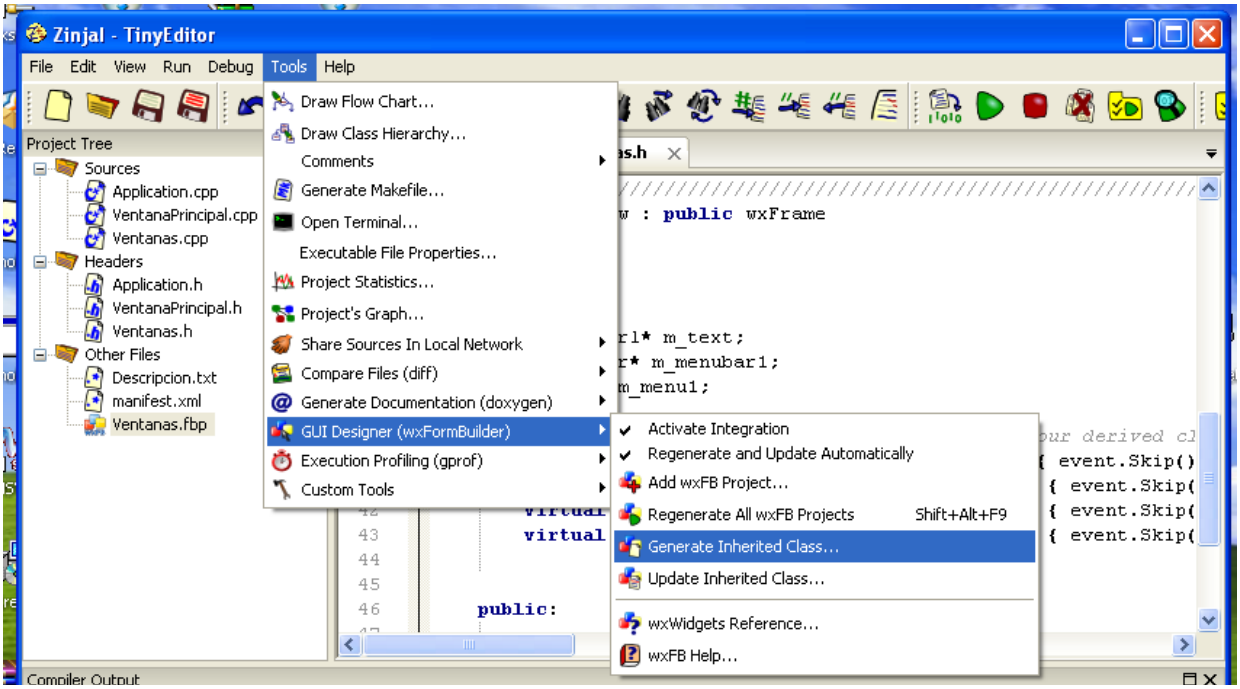
1) Select VentanaPrincipal.cpp in Project Tree and hit Delete key or choose “Detach from project” in its contextual menu. You can also safely delete it from disk by selecting the checkbox in confirmation window. When it asks for the second file (VentanaPrincipal.h) you should also say yes.



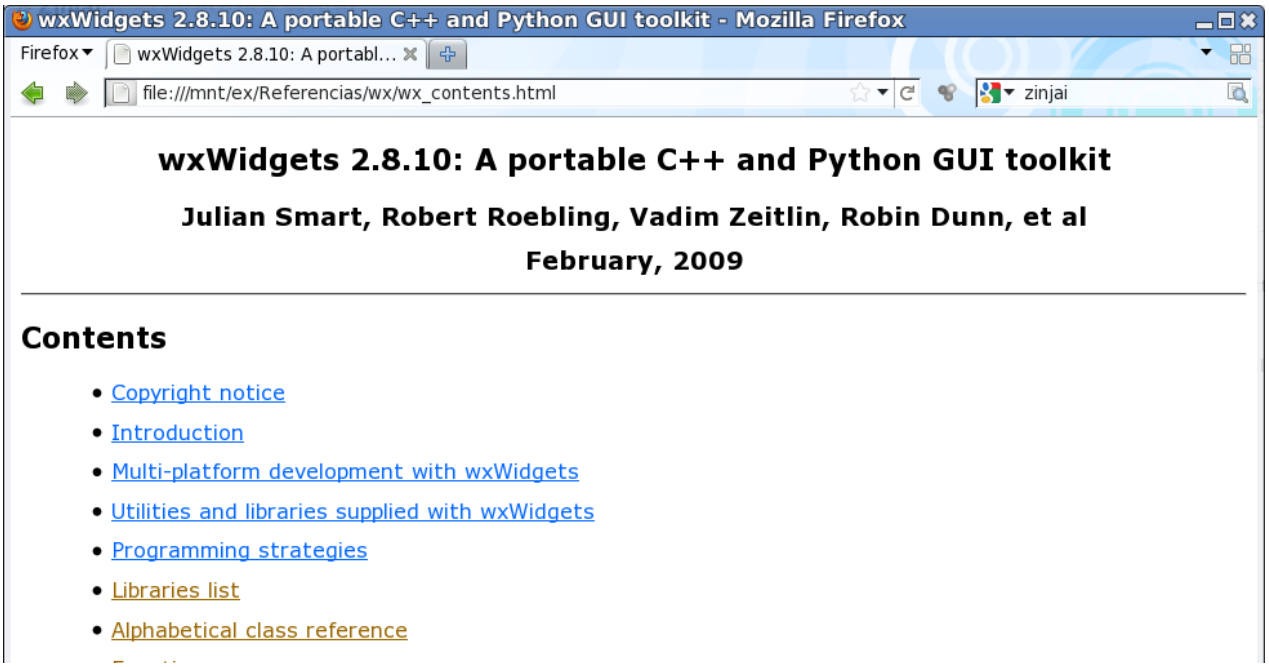
left side: removing old files from project, right side: recently generated bMainWindow class

You can explore Ventanas.h and Ventanas.cpp to see the code. You'll also see some methods for the events. You should not modify those files, because if you do so, when you regenerate wxFormBuilder project (you'll need to do it if you want to add something to the GUI) you will loose those changes. So, to avoid this problem, you can create a derived class from bMainWindow in a separated file. Actually, ZinjaI can create this class for you:

2) Go to “Tools” menu, select “GUI Designer” submenu and finally pick “Generate Inherited Class...” item. You'll see a little dialog where you can choose the base class among wxFormBuilder generated ones and a text field where you can enter the new derived class name. Enter cMainWindow and click “Ok”.

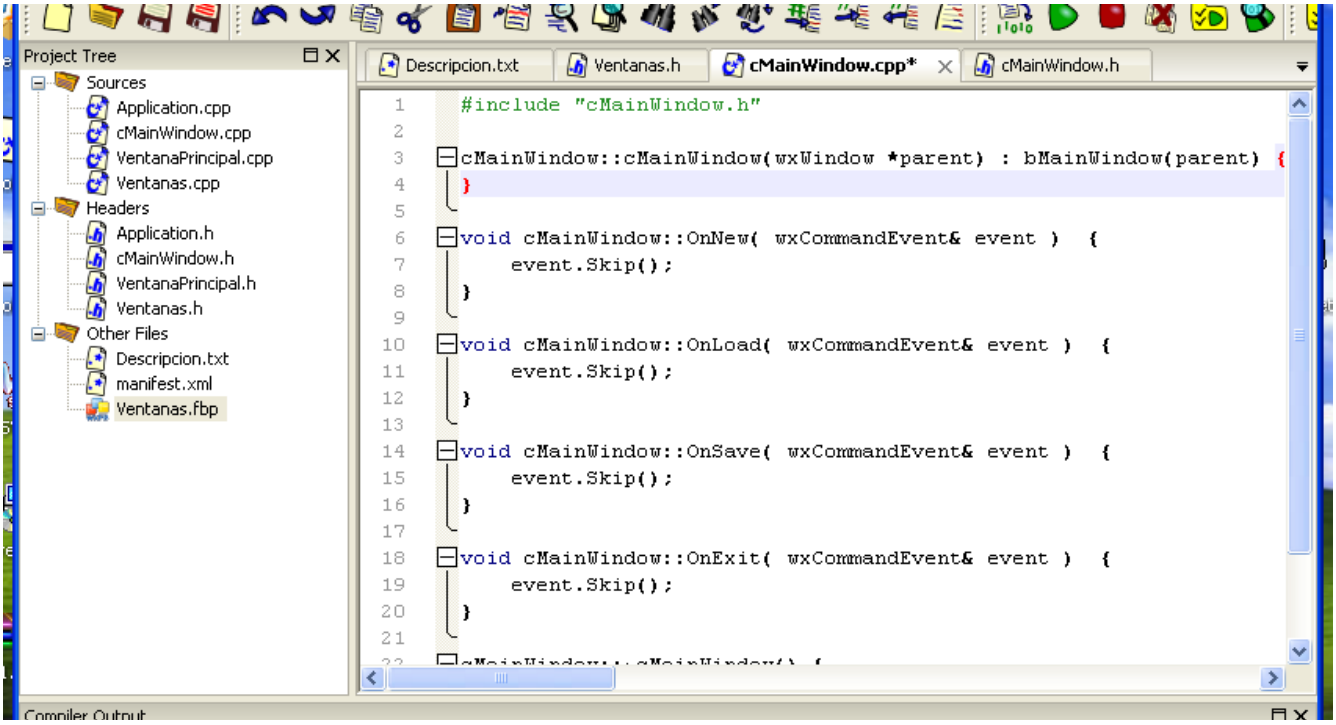


Now you'll see two new files in the project. The cpp one will have some empty methods (the one you defined for events). There is where you must write your code. In order to interact with the library, you should familiarize with its classes, functions and methods. You can open wxWidget's reference from ZinjaI by clicking on item “wxWidgets Reference...” from “GUI Designer” submenu from “Tools” menu (in Windows version it will open help without problem, in Linux or Mac versions you have to download and extract HTML Docs from http://www.wxwidgets.org/downloads/).



wxWidget's html reference index

Lets analyze the different methods:



OnNew: this one should erase all content from the text control (remember that you can access the text control with the m_text pointer inherited from class bMainWindow). To achieve that you can use Clear method from wxTextCtrl.

3) Write the following code inside cMainWindow::OnNew method in cMainWindow.cpp, replacing “event.Skip();”:

```
m_text->Clear();
```

OnLoad: this one should let user pick a file and load its content to text control. You can use LoadFile method from wxTextCtrl class to load the content of a file to the text control, but you need some dialog to let the user enter the file's path. For most common dialogs (open, save, find, print, color picker, font picker) wxWidgets have some special classes. You can use wxFileDialog in this event. The class represents a classic Open/Save file dialog and have a ShowModal() method to actually run the dialog and a GetPath() method to retrieve the path the user selects.

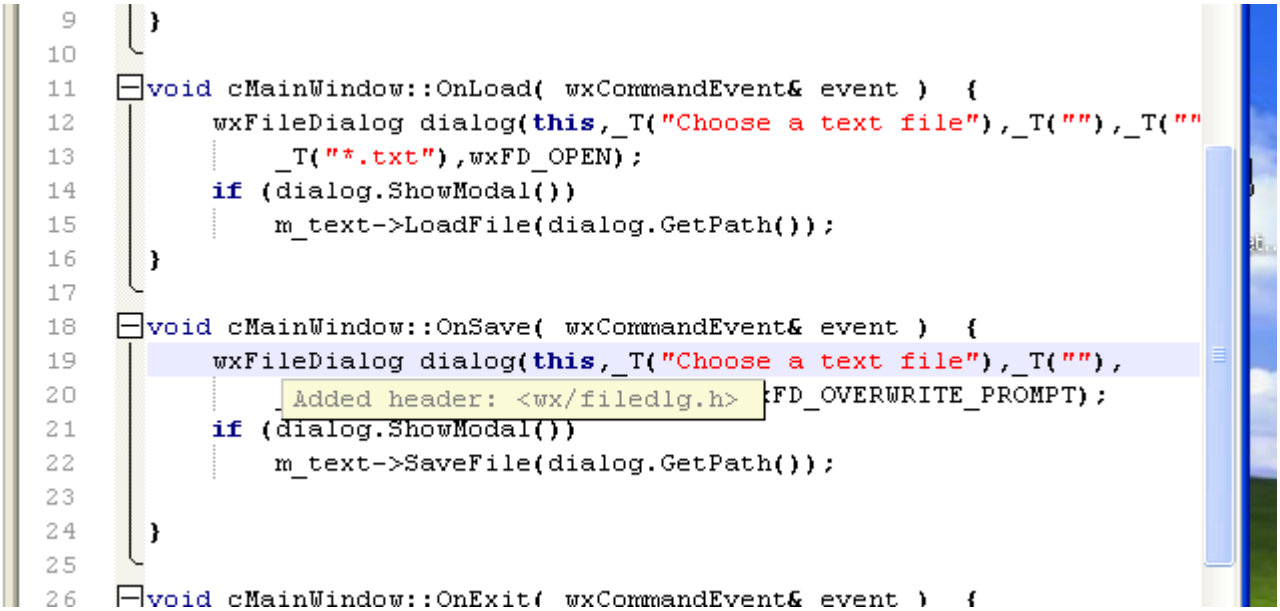
4) Write the following code inside cMainWindow::OnNew method in cMainWindow.cpp;

```
wxFileDialog dialog(this,_T("Choose a text file"),_T(""),_T(""),
                    _T("Text files|*.txt|All files|*”),wxFD_OPEN);
if (dialog.ShowModal()==wxID_OK)
     m_text->LoadFile(dialog.GetPath());
```

wxFileDialog's constructor takes the parent window, a text message, the default folder, the default filename, the filename wildcard (for filtering files, by extension mainly), and same flags to control

details about the dialog. If you installed ansi wxWidgets version you can't write text literals without surrounding them with _T macro. Dialogs are usually shown with ShowModal() method, while Frames are usually shown with Show() method. The first one will pause the calling event until the dialog is closed (and main window won't be able to receive any other event until then). The second one will show the dialog and continue with the event, keeping both windows enabled and receiving events at the same time. In addition, ShowModal can return a number indicating the success or cancellation of the dialog action. wxFileDialog is defined in <wx/filedlg.h> header. You need to add an include, but ZinjaI can do it for you.

5) Move the text cursor to wxFileDialog word and hit Ctrl+H.



methods code and automatic #include insertion

OnSave: this one should let user pick a file name and save the text control content to that file. You'll do it very similar to OnLoad:

6) Write the following code inside cMainWindow::OnSave method in cMainWindow.cpp;

```
wxFileDialog dialog(this,_T("Choose a text file"),_T(""),_T(""),
                                  _T("Text files|*.txt|All files|*"),
                                    wxFD_SAVE|wxFD_OVERWRITE_PROMPT);
if (dialog.ShowModal()==wxID_OK)
     m_text->SaveFile(dialog.GetPath());
```

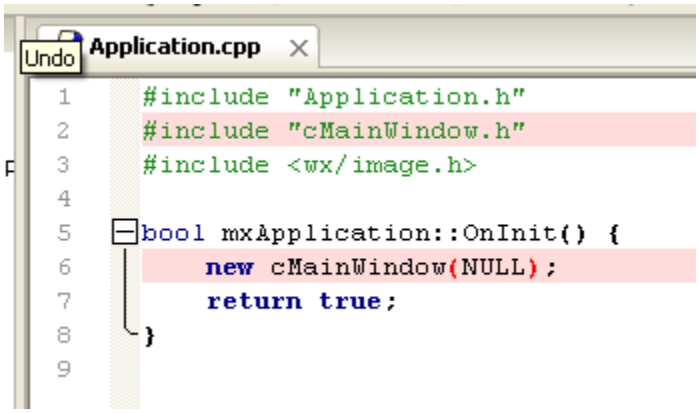
OnExit: this one should end the application. There are two ways: call wxExit function (declared in wx/app.h), or close the main window.

7) Write the following code inside cMainWindow::OnExit method in cMainWindow.cpp;

```
Close();
```

Now you have a very functional window, but if you still can't test your app. There is one more thing to do. At this time, you may be wondering about C/C++ main function. If you look carefully there's no main function in your project. So, how does it works? The fact is that main function is inside the library and you cant touch it. This function will initialize the library and create a wxApp object. That's why you have Application.cpp and Application.h. Your main function is now OnInit method from Application. That is the point where you take control of your app when it is loaded. There you should load and display the main window, and then wait for events. To load the main window you can just create an instance of cMainWindow (the derived class). Doing that you load it to memory but its not visible yet. You should use Show method to actually display the window. So, summing up:

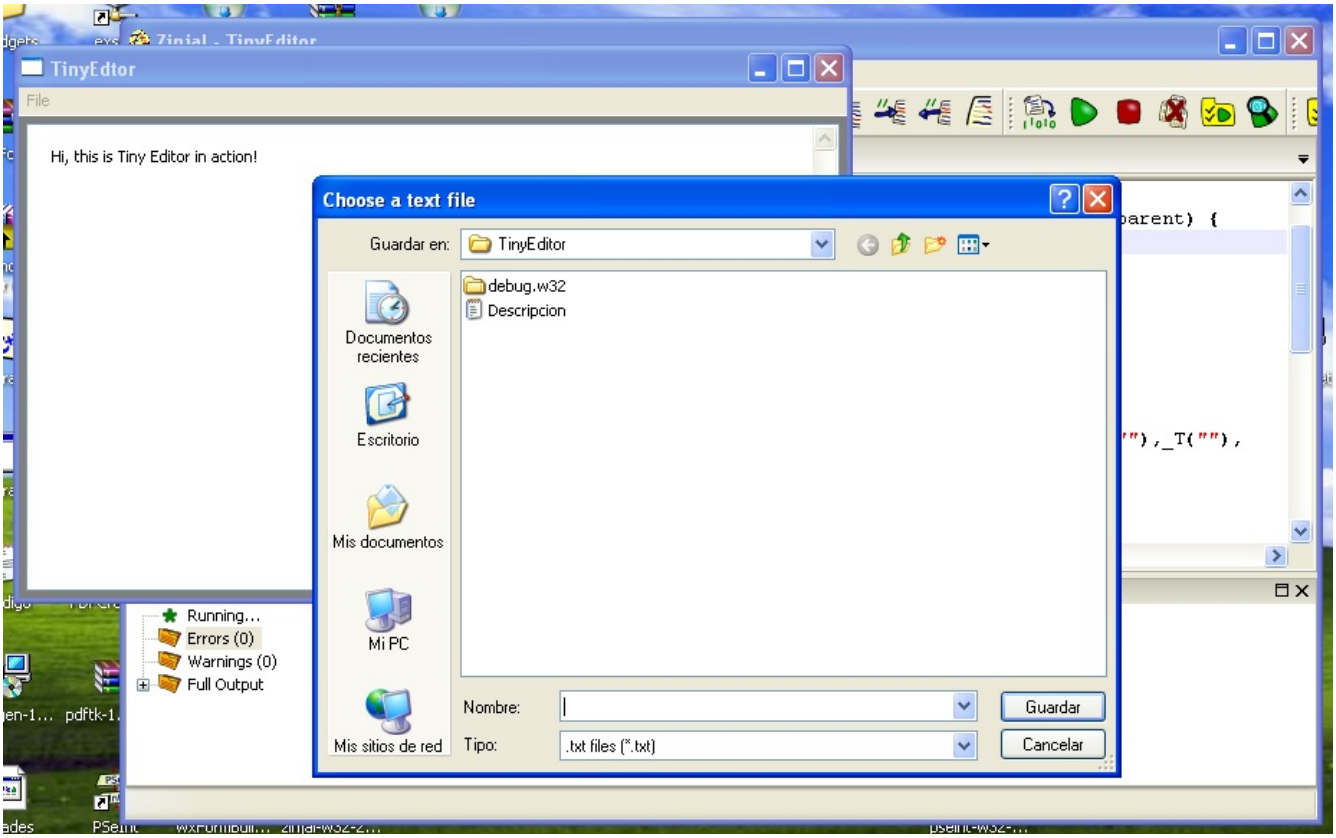
8) Go to to Application.cpp and replace VentanaPrincipal by cMainWindow in the include line and in the new line just before the return. You can also safely delete the lines starting with wxImage if your program wont need to load file with those image formats.



9) Go to cMainWindow constructor and add a call to Show() inside it.

You'll notice there's no delete for the new in Application::OnInit. wxWidgets controls all the dynamically created windows and components. If you want to delete a window, you just call its Destroy method. This will safely delete the window and all its components after processing all its remaining events. This eliminate the need to keep a pointer to every window and allow of self-destruction.

Now you can run the app hitting F9 key.



recently born TinyEditor running from ZinjaI and try to save its first text file.

Part 5: Little improvements

At this point you should be running your first TinyEditor's beta version, and you should have learned some basic skills that will let you integrate wxWidgets GUIs in your applications. You can stop the tutorial right now and start working on your own projects, but, for completeness, you can tweak a few details without much work to improve TinyEditor a little more and get some extra tips.

Adding an About dialog

Lets see how to modify the existing GUI. Assume you want to add an about dialog to the app. You need to modify main window to add a menu item for it, and you need to create a new window (a wxDialog this time) with the information you want to display. The first step is going to wxFormBuilder and adding those component's:

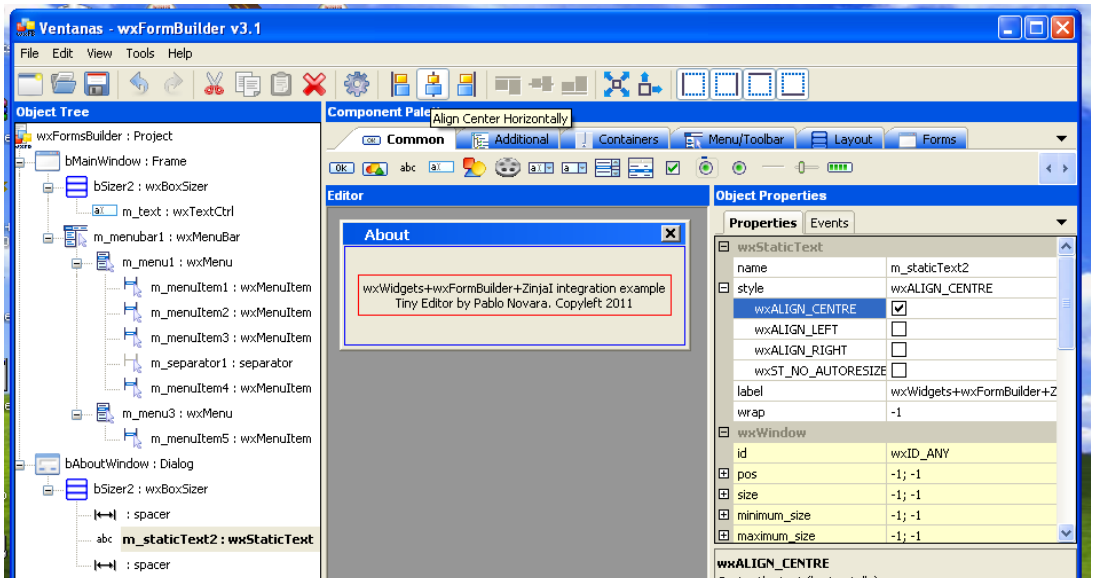
1) Double click Ventanas.fbp in ZinjaI's project's tree. It will open wxFormBuilder again

2) Select the menu bar in the tree and add a new wxMenu with the palette. Set its label to “Help”.

3) Add a new wxMenuItem from palette to Help menu, set its label to “About...” and it's “OnMenuSelection” event to “OnAbout”.

4) Create a new Dialog, clicking the third button (wxDialog) in “Forms” tab from palette. Set its title to “About” and its name to bAboutWindow, and its “bg” property (background colour) to “ButtonFace” (same color as buttons).

5) Add a wxBoxSizer to bAboutWindow, a spacer (last button in Layout tab), a wxStaticText (the third from “Common” tab in palette), and another spacer to the sizer. The spacers will consume the extra space in the window centering vertically the text. To center it horizontally, use the Align button in toolbar (Shift+Alt+H).



6) Use properties panel to set following label (text):

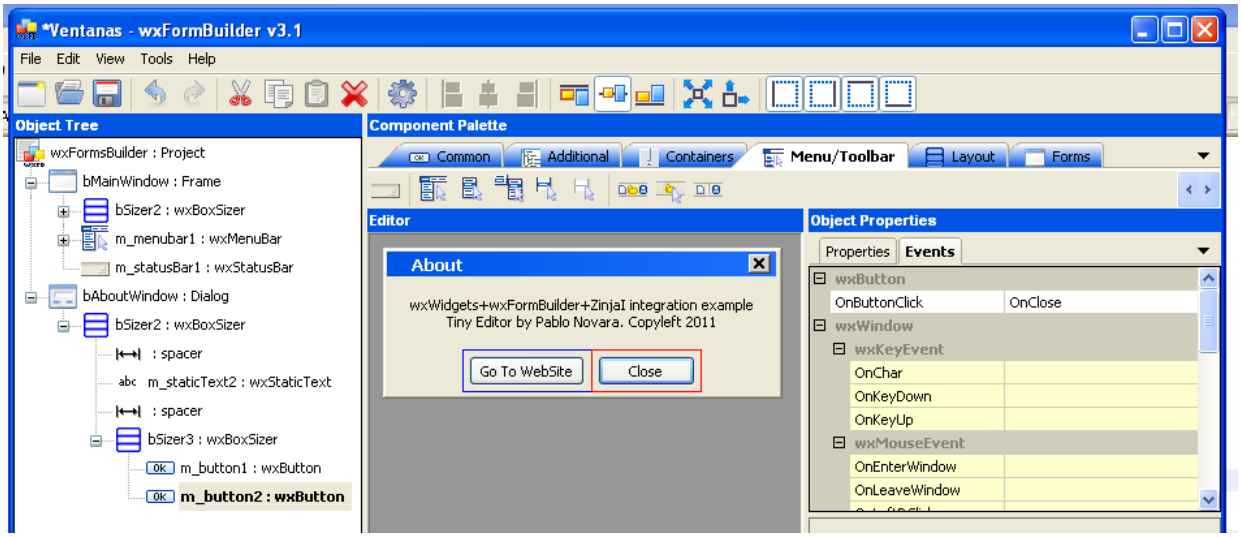
wxWidgets+wxFormBuilder+ZinjaI integration example

Tiny Editor by Pablo Novara. Copyleft 2011

8) Add a new sizer to the already existing sizer by clicking again the wxBoxSizer button in Component Palette, and unset its Stretch and Expand properties by hitting Alt+S and Alt+W, or with toolbar items.

9) Add two buttons to this last sizer: the first one with label “Open Website” and the second one with label “Close”. Also set OnClick event for both buttons writing “OnGotoWebsite” for the first one in Event tab in Object Properties panel, and “OnClose” for the second one.

10) You can see both buttons arranged vertically (one over the other). To arrange them horizontally (one next to the other in a single row) select the last sizer and change its “orient” property to “wxHorizontal”.



11) Save the project and close wxFormBuilder.

You'll see that ZinjaI will automatically add the new method to cMainWindow class. There you should write the code to display About window, but first you must create cAboutWindow class.

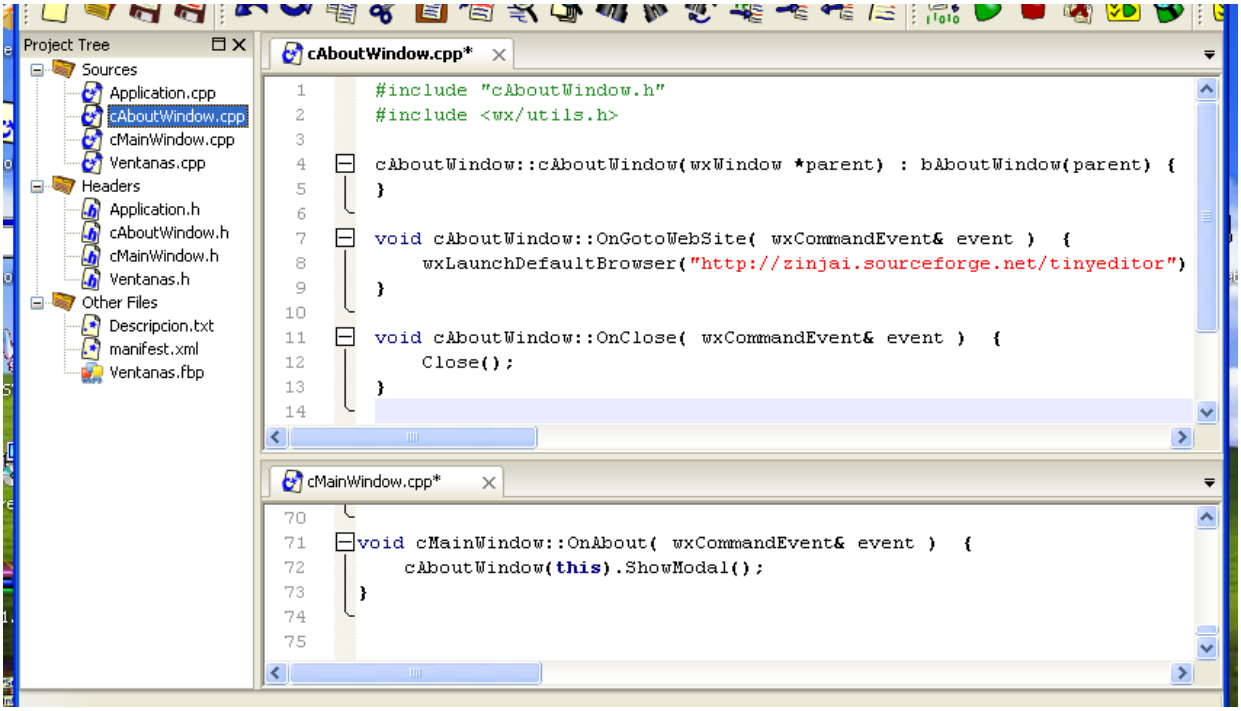
12) Go to “Tools” menu, select “GUI Designer” submenu and finally pick “Generate Inherited Class...” item. Select “bAboutWindow” as base class and enter “cMainWindow” in the inherited class name. Then click “Ok”.

13) Write `Close();` in OnClose method from cAboutWindow, and write:

```
wxLaunchDefaultBrowser("http://zinjai.sf.net/tinyeditor.html");
```

in OnOpenWebsite method from cAboutWindow. You'll need to add an `#include <wx/utils.h>` line for this function, placing text cursor on it and hitting Ctrl+H.

17) Finally, to actually show the window when the user clicks on the menu item, write `cAboutWindow(this).ShowModal();` in OnAbout method from cMainWindow, and add the `#include “cAboutWindow.h”` line for this class by placing text cursor on the word “cAboutWindow” hitting Ctrl+H again.



Notice that modal dialogs are usually created as static objects. If you create it as a dynamic object (with new operator) you must call Destroy somewhere (a good place is OnClose event). Destroy method replaces delete operator (actually, you shouldn't use delete operator for wxWidgets components, always Destroy instead), with the advantage that an object can safely delete itself with it.

Remembering opened file name and updating window's caption

If you want an editor that remembers the file name when you load a file and set it as default to save dialog when you go to save, you can add an attribute to cMainWindow to store it. This should be a wxString for convenience. All the strings that go in/out from/to wxWidget's functions/methods are wxString instances. This class shares properties from strings and streams, and if you have ansi version they can be easily created from cstring with a specific constructor that takes the cstring as its only argument, or with = operator, and can be easily converted to cstring with c_str() method.

So, to remember file name you need to add the attribute (for instance m_filename), and to modify OnNew, OnLoad and OnSave methods. Also, having the file name, it's a good idea to show it in the titlebar, and you can do it with SetTitle method.

1) Go to cMainWindow class definition in cMainWindow.h and add a private attribute:

```
wxString m_filename;
```

2) Go to OnNew method from cMainWindow and modify it to reset file name:

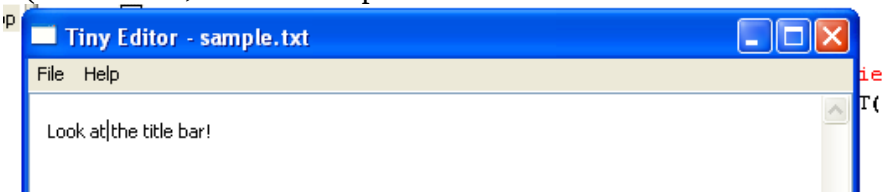
```
m_text->Clear(); m_filename.Clear();
SetTitle(_T("Tiny Editor"));
```

3) Go to OnLoad method from cMainWindow and modify it to remember file name:

```
wxFileDialog dialog(this,_T("Choose a text file"),_T(""),_T(""),
                    _T("Text files|*.txt|All files|*"),wxFD_OPEN);
if (dialog.ShowModal()==wxID_OK) {
     m_filename=dialog.GetPath();
     m_text->LoadFile(m_filename);
     SetTitle(wxString(_T("Tiny Editor - "))<<
                              wxFileName(m_filename).GetFullName());
}
```

You should add an `#include <wx/filename.h>` line, but again you can do it with Ctrl+H.

wxFileName is a class that handles file paths and here it is used to split the filename from the whole path (for instance, extract "sample.txt" from "C:\users\zaskar\Documents\sample.txt").



4) Go to OnSave method and modify it to use the remembered file name and remember the new one if user accepts the dialog:

```
wxFileDialog dialog(this,_T("Choose a text file"),_T(""),
                    m_filename,_T("Text files|*.txt|All files|*"),
                                  wxFD_SAVE|wxFD_OVERWRITE_PROMPT);
if (dialog.ShowModal()==wxID_OK) {
     m_filename=dialog.GetPath();
     m_text->SaveFile(m_filename);
     SetTitle(wxString(_T("Tiny Editor - "))<<
                              wxFileName(m_filename).GetFullName());
}
```

Note that you can add any method or attribute you need to derived classes without interfering with the designer.

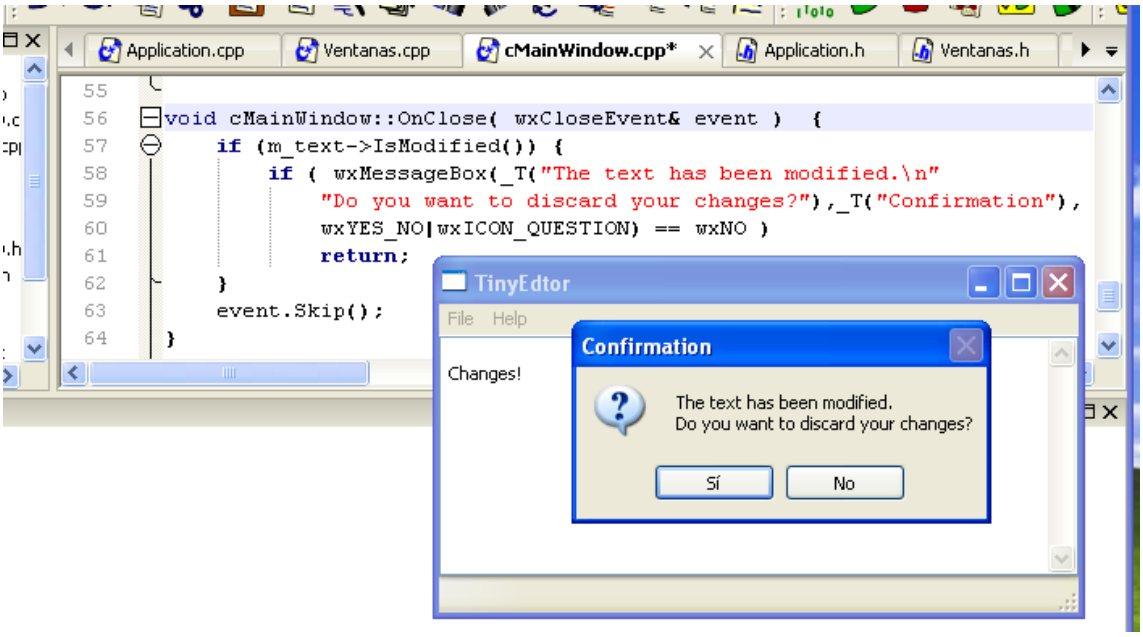
Avoiding unintentional data loss

It is a good practice to request user confirmation before loosing any data. This application, as almost any editor, should warn the user before creating or opening a new file if previous one have unsaved changes. You can use GetModified method from wxTextCtrl class to ask the control if there was any change. You could use SetModified method to reset its status but it wont be necessary because Clear, LoadFile and SaveFile methods already do it. To display the confirmation message you can use wxMessageDialog function, that creates and run as modal a wxMessageWindow.

1) Add the following code at the beginning of OnNew and OnLoad methods:

```
if (m_text->IsModified()) {
	if ( wxMessageBox(_T("The text has been modified.\n"
			"Do you want to discard your changes?"),
		_T("Confirmation"),wxYES_NO|wxICON_QUESTION) == wxNO )
	return;
}
```

Last argument in wxMessageBox functions controls wich buttons and wich image to show in the dialog.



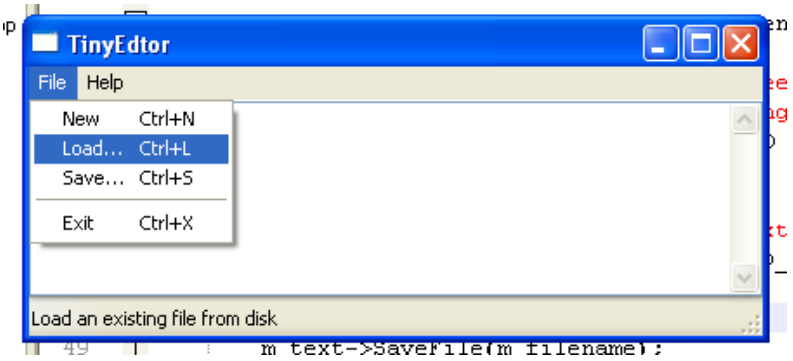
3) Finally, the editor should also ask before closing the window, so open the designer, select the main window and add enter “OnClose” for the event “OnClose” in Object Properties panel. Return to ZinjaI and write the following code in OnClose method:

```
if (m_text->IsModified()) {
	if ( wxMessageBox(_T("The text has been modified.\n"
			"Do you want to discard your changes?"),_T("Confirmation"),
			wxYES_NO|wxICON_QUESTION) == wxNO )
		return;
}
event.Skip();
```

OnClose event is called when the user tries to close the window, but before actually closing it. Note that you must keep the “event.Skip()” line at the end. It tells wxWidgets to proceed with the event's default action. In this case, the default action is to actually close the window. If the user answers “No” to the confirmation question, the method returns without reaching the Skip line. In this situation, the window won't close.

Improving menu appearance

You can implement a few changes from the designer that will improve your menus without any extra code. Those changes are: add shortcuts to menu items, add help texts to be displayed in a status, and add icons for them. Menu items have two kind of shortcuts, the real shortcut wich is a combination of keys that triggers the menu item action, and a more simple and local one that is the one you use with Alt key and see as an underlined letter in its text. The first one is defined with “shortcut” property, and the second one adding an '&' character in “label” property before the letter to be underlined. To display a help text in status bar when the mouse arrow goes over an item, you need to add an status bar to the window and enter the text in the menu item's “help” property. Finally, to define an image for the menu item you can use “file path” property, wich is inside “Bitmap” property, but we won't set images now.

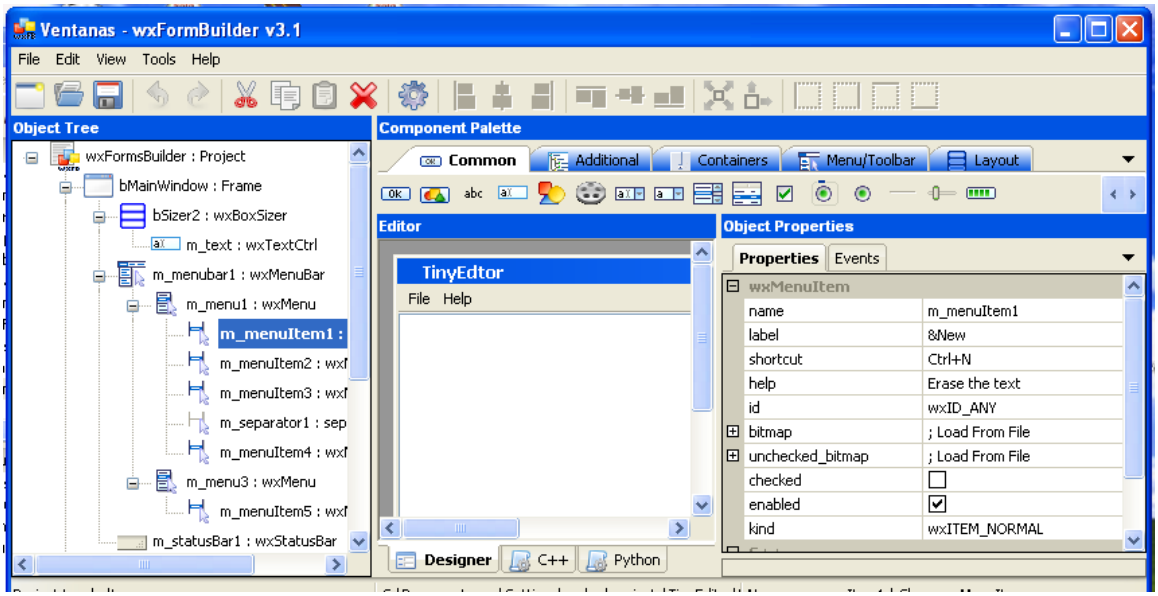


1) Open the designer, select the main window and click the first item from “Menu/Toolbar” tab in Palette. This will add an empty status bar to the window.

2) Select menu items and change its properties as follows:

- For “File” menu, set “name” to “&File”

- For “New” item, set “name” to “&New”, shortcut to “Ctrl+N” and “help” to “Erase the whole text”.



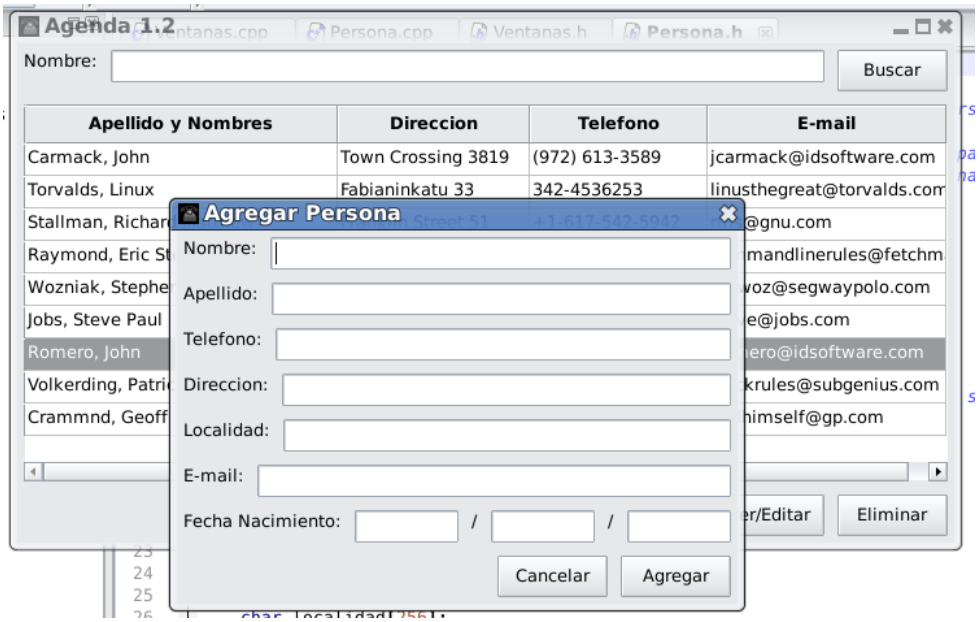
- For “Load” item, set “name” to “&Load”, shortcut to “Ctrl+L” and “help” to “Load an existing file from disk”.
- For “Save” item, set “name” to “&New”, shortcut to “Ctrl+N” and “help” to “Save current text in a file”.
- For “Exit” item, set “name” to “&New”, shortcut to “Ctrl+N” and “help” to “Closes the editor”.
- For “Help” menu, set “name” to “&Help”
- For “About” item, set “name” to “&About”, shortcut to “F1” and “help” to “Display license and credits”.

Final remarks

As you can see now, developing a simple GUI with wxWidgets and wxFormBuilder is a simple task. You won't need to write much code to have your windows running, and you can use ZinjaI to speed up even more the whole development process.

To go further you should familiarize with wxWidgets interface. Luckily, wxWidgets has an excellent reference in html format wich you can access from ZinjaI (in Windows version its shipped in the installer, go to Tools menu->GUI Designer->wxWidgets reference, in Linux you have to download from wxWidgets site and tell ZinjaI where you extract it from Preferences dialog). Also, if you need a complex example for an specific component you can download wxWidgets sources and explore its “sample” folder.

If you have mastered this simple example you can explore my other examples. An interesting one is wxAgenda. It's a simple address book, but it has more windows that communicates between, a grid (that is a component a little more complex) and the main difference: an object-oriented class model written in plain standard C++, wich is interface independent. If you have a good set of classes that solves your problem, connecting them to a GUI should be pretty straightforward, because you only need a few lines for a translation layer from events and wxWidgets' interface to your classes interface, but you can avoid writing “business logic” on the GUI side, keeping code clearer and modular. The only drawback is that this example and it's tutorial are written in Spanish language, but you can read the code and understand it anyway.



address book example running

I hope you found this text useful. Find more info and example code in Documents section from http://zinjai.sourceforge.net