Tutorial wxPython + wxGlade

John Alexis Guerra Gómez

Tutorial wxPython + wxGlade

por John Alexis Guerra Gómez

Tabla de contenidos

Lista de figuras

Capítulo 1. Introducción

El objetivo de este tutorial es sentar las bases para el desarrollo de aplicaciones GUI *(Graphic User Interface)* usando la librería wxPython (http://www.wxpython.org) y el lenguaje de programación Python (http://www.python.org), utilizando la ayuda del diseñador de interfaces wxGlade (http://wxglade.sourceforge.net).

Se espera que el lector tenga conocimientos básicos en Python, y en caso de que aún no los tenga, le recomendamos revisar el tutorial oficial de python (http://www.python.org/doc/).

Este tutorial fue diseñado para realizarse utilizando Python >2.3, wxPython >2.6 y wxGlade >0.4.

Capítulo 2. ¿Cómo empezar?

Para empezar a realizar una aplicación con wxwindows necesitamos seguir los siguientes pasos:

- Incluir la librería, para esto bastará agregar el siguiente fragmento de código:

```
import wx
```

> **Aviso**
>
> Al realizar esta instrucción estamos incluyendo un módulo completo que incluye todos los elementos del wxpython, esta característica es relativamente reciente y en internet aún se encuentran tutoriales que utilizan widgets cómo *wxApp* o *wx.wxApp*, este modo de utilizar ha quedado en desuso y se recomienda utilizar los widgets usando el nuevo módulo, es decir que por ejemplo para acceder a la clase *App* que en C++ sería *wxApp* se debería usar *wx.App*

- La librería wxPython está completamente basada en la teoría de programación orientada objetos, es por esto que para poder crear cualquier programa siempre se necesita “sobrecargar” el “método” *OnInit* de una clase clase de tipo *wx.App* (es decir que puede ser de una clase que herede de ella, para hacer esto basta con escribir esto:

```
class MiApp(wx.App):
   def OnInit(self):
      return True
```

  Este fragmento de código crea una clase llamada *MiApp*, que hereda de la clase *wx.App* cuyo único método, por el momento es *OnInit*, método éste que será llamado en el momento que iniciemos nuestra aplicación, es importante tener en cuenta que este método debe retornar *True*.

> **Aviso**
>
> Todo programa en *wxPython* debe tener por lo menos una instancia de la clase *wx.App*, que puede ser una clase que herede de ella, o por ejemplo una creada con el método *wx.PySimpleApp*, que crea una *wx.App* sencilla.

- Una vez esté lista la clase debemos instanciarla, y hacer un llamado al método MainLoop, que es heredado de la clase *wx.App*. Este método se encargará de entregarle el control de programa al wxPython permitiendo la captura de eventos (más sobre esto más adelante) entre otras cosas. Es decir que necesitamos un código cómo:

```
app=MiApp()
app.MainLoop()
```

Aviso

No se debe colocar código después de un *MainLoop*, ya que este llamado bloque la aplicación al entregarle el control al *wxPython*, y cualquier instrucción que se coloque luego de este llamado será ejecutada solo cuando termine la aplicación.

Hasta el acá ya tendríamos una aplicación en *wxPython* completa, pero aunque la ejecutáramos, y no saliera ningún error, no aparecería nada en la pantalla, y ¿qué pasó con el “hola mundo”?, pues bueno, para poder que nuestra aplicación muestre algo necesitamos crear una ventana

2.1. Nuestra primer ventana

Para crear ventanas debemos crear instancias de la clase *wx.Frame*, un ejemplo de un programa que muestre una ventana sencilla que salude en su título sería el siguiente:

ventana_saludona.py (ejemplos/ventana_saludona.py)

```
#-*- coding: iso8859-1 -*-
import wx

class VentanaSaludona(wx.Frame):
    def __init__(self):
        #Llamo al constructor de wx.Frame
        wx.Frame.__init__(self,
                          None, #El padre de esta ventana
                          -1, #Un número para identificarla
                          u"Hola ¿cómo estás?" #El título de la ventana
                          )
        #Le digo a la ventana que se muestre
        self.Show(True)

class MiApp(wx.App):
    def OnInit(self):
        #Creo una instancia de la ventana como atributo de miApp
        self.ventana=VentanaSaludona()
        #Le digo a miApp quien es la Ventana Principal
        self.SetTopWindow(self.ventana)
        return True

app=MiApp()
app.MainLoop()
```

Este programa crea una nueva clase llamada *VentanaSaludona* que hereda de la clase *wx.Frame*, de esta clase solo hemos definido su constructor, que lo único que hace es llamar al constructor de la clase

wx.Frame, pasándole 3 parámetros: el primero es el padre de la ventana, en *wxPython* todo *widget* tiene una ventana padre, pero como esta es nuestra ventana padre será la excepción (*None* será su padre); el segundo parámetro es un número que usaremos para identificar la ventana, como tampoco necesitamos identificarla por ahora le asignaremos -1 y por último recibe una cadena (unicode, por eso tiene una *u* antes de las comillas) que será el título. Para conocer más datos de los parámetros del constructor de la clase *wxFrame* se puede consultar la documentación oficial del *wxPython*, este debe ser el procedimiento a seguir cada que queramos conocer algo más de la librería. Por último le decimos a la ventana que se muestre utilizando el método *Show* con un parámetro *True*, si este método es llamado con un valor *False* la ventana se ocultará (no se destruirá, eso es otra cosa) y saldrá de la vista del usuario aunque estará disponible con todos los valores que tenía al desaparecer para cuando la vuelvan a llamar con *Show*

Más adelante en el código hay que notar que estamos creando una instancia de la clase *VentanaSaludona* y la estamos guardando como un atributo de la clase *MiApp*, de esta manera podremos usarla más adelante, luego le decimos a *MiApp* que ella será su ventana principal

> **Importante:** Una aplicación puede tener muchas ventanas, pero debe tener siempre una que sea la principal, cuando la ventana principal de una aplicación se cierra, la aplicación termina sin importar si hay más ventanas abiertas.
>
> De igual manera trabajan las ventanas entre ellas, si una ventana es padre (o madre) de otras, cuando ella se cierre se cerraran todas sus hijas.

Figura 2-1. Mira mamá una ventana que saluda!

Capítulo 3. Simplificando el trabajo con el wxGlade

Este tutorial podría continuar mostrando como construir aplicaciones directamente en el código utilizando cualquier editor, y es completamente factible, sin embargo, desde el punto de vista del autor, el proceso de creación de aplicaciones con *wxPython* sin ayuda de un diseñador puede ser bastante traumático. Es por esto que apartir de este punto empezaremos a utilizar el diseñador de interfaces (no un IDE completo, es decir que no servirá para escribir código, solo para generarlo) wxGlade (http://wxglade.sourceforge.net).

El *wxGlade* es un programa que ayuda a diseñar las interfaces gráficas, permitiendo que el proceso de construcción sea mucho más veloz que al hacerlo manualmente.

3.1. Iniciando el wxGlade

Figura 3-1. El inicio del wxGlade, ventana de íconos (arriba a la izquierda), ventana de propiedades (abajo), ventana de árbol (derecha)

Al iniciar el wxGlade éste nos presenta una interfaz que puede ser un poco extraña para las personas que vienen de trabajar en otros editores “visuales”, sin embargo con un poco de información es fácil empezar a trabajar con el. En su presentación inicial aparecen tres ventanas:

- *La ventana de íconos*, en ella se encuentran todos los elementos visuales o *widgets* que el *wxGlade* pone a nuestra disposición para crear interfaces, ventanas, botones, cuadros de texto, árboles, listas, en fin.
- *La ventana de íconos*, acá encontrará propiedades del widget que se encuentre seleccionado en ese momento, al iniciar contiene las propiedades de la aplicación, el equivalente a nuestro *MiApp*.
- *La ventana de árbol* en ella se encuentra una estructura jerárquica donde se representa la interfaz que estamos diseñando, donde aparece la aplicación como padre y apartir de ella podremos encontrar ventanas como sus hijas, y a su vez botones, cajas de texto, y demás como hijos de las ventanas, pero con una estructura especial que veremos más adelante.

3.2. Nuestra primer ventana

Para llegar con el wxGlade hasta donde íbamos en el capítulo anterior lo primero que debemos hacer es crear nuestra ventana saludona, para eso basta con dar click en el icono de ventana de la ventana de íconos. Al hacerlo aparecerá un cuadro de dialogo pidiéndonos el nombre de la clase base, y el nombre de la clase, seleccionemos *wxFrame* para la primer opción y *VentanaSaludona* para la segunda.

Figura 3-2. Crear una ventana

Al aceptar aparecerá una nueva ventana, esta es la *Ventana de Diseño* tendremos una de estas por cada ventana que estemos diseñando y en ella se colocan los diferentes elementos que compondrán la ventana.

3.3. Cambiando parámetros

Bueno ya tenemos nuestra ventana, pero queremos que salude, ¿no es así?, entonces debemos ir a la ventana de propiedades, (con la nueva ventana seleccionada, y en la pestaña *widget* cambiamos el valor de la propiedad *Title* por “Hola ¿cómo estás?”, ya que estamos aquí aprovechemos cambiemos también el nombre de la instancia en la aplicación, para eso vamos a la pestaña *Common* y allí cambiamos la propiedad *Name* por ventana.

Figura 3-3. Cambiar el título de la ventana

En la ventana de propiedades existen un amplio número de características que podemos cambiar, alterando así como se ve nuestra aplicación, por ejemplo cambiar su tamaño, su color, o su icono, tareas estas que serían un poco dispendiosas directamente en código. Por ejemplo modifiquemos su tamaño para que aparezca por defecto de 400x300, para esto cambiemos la propiedad *Size* de la pestaña *Common*, activemos la opción y escribamos “400, 300”

3.4. Generar Código

Ya tenemos nuestra interfaz lista, para generar el código que la ejecute debemos cambiar las propiedades de la aplicación, para hacer esto debemos primero seleccionar la aplicación,dando click en *Application* en la ventana de árbol, luego en la ventana de propiedades seleccionamos las opciones *Name* y *Class*, en la opción *Top Window* seleccionamos “"ventana"”, escogemos el nombre del archivo donde va a quedar nuestra aplicación y por último damos click en el botón *Generate Code*.

Figura 3-4. Generar Código

Al hacer esto obtendremos un nuevo código Python, completamente generado por el wxGlade que lucirá así:

```
#!/usr/bin/env python
# -*- coding: ISO-8859-1 -*-
# generated by wxGlade 0.4.1 on Mon May  1 22:20:17 2006

import wx

class VentanaSaludona(wx.Frame):
```

```
    def __init__(self, *args, **kwds):
        # begin wxGlade: VentanaSaludona.__init__
        kwds["style"] = wx.DEFAULT_FRAME_STYLE
        wx.Frame.__init__(self, *args, **kwds)

        self.__set_properties()
        self.__do_layout()
        # end wxGlade

    def __set_properties(self):
        # begin wxGlade: VentanaSaludona.__set_properties
        self.SetTitle(u"Hola ¿Cómo estás?")
        self.SetSize((400, 300))
        # end wxGlade

    def __do_layout(self):
        # begin wxGlade: VentanaSaludona.__do_layout
        sizer_1 = wx.BoxSizer(wx.VERTICAL)
        self.SetAutoLayout(True)
        self.SetSizer(sizer_1)
        self.Layout()
        # end wxGlade

# end of class VentanaSaludona


class MyApp(wx.App):
    def OnInit(self):
        wx.InitAllImageHandlers()
        ventana = VentanaSaludona(None, -1, "")
        self.SetTopWindow(ventana)
        ventana.Show()
        return 1

# end of class MyApp

if __name__ == "__main__":
    app = MyApp(0)
    app.MainLoop()
```

3.5. Entendiendo el código generado

El código generado por el wxGlade es muy similar al que escribimos en el capítulo anterior, pero hay unas cuantas diferencias que se marcan, por ejemplo la clase *VentanaSaludona* contiene un nuevo par de métodos *__set_properties* y *__do_layout* estos sirven para definir las propiedades de nuestros widgets el primero (como por ejemplo cambiar el título) y el segundo sirve para definir como se organizan las cosas en una ventana pero de eso hablaremos más adelante.

Otra diferencia es que la parte principal del código es decir donde instanciamos nuestra aplicación y lanzamos el *MainLoop* se encuentra incrustada dentro de un condicional que garantiza que la aplicación no se creará si el archivo es cargado como una librería y no como un programa principal, es un viejo truco “pythónico”.

El resto de código no debería asustarnos, porque por lo general, no vamos a tener la necesidad de modificarlo.

Capítulo 4. ¿Cómo organizar los widgets?, una introducción a los *sizers*

4.1. Estático vs. Dinámico

Hasta ahora hemos creado una ventana con un título, pero por lo general las aplicaciones son bastante más complejas que eso, normalmente incluyen botones, cajones de texto, listas, árboles, imágenes, en fin amplio conjunto de elementos que nos permiten interactuar con las ellas. En *wxPython* estos elementos son llamados widgets (de ahí el nombre de la librería en la que está basada *wxWidgets*.

En *wxPython* existen dos maneras de incorporar los widgets, la primera es mediante tamaños y posiciones fijas, y la segunda es mediante proporciones. El primer método es el utilizado por muchos diseñadores de interfaces monoplataforma, que mediante un entorno de desarrollo permiten arrastrar controles a ventanas para ubicarlos en una ubicación específica, este mecanismo puede ser fácil de utilizar al principio, pero es poco configurable si necesitamos incorporar widgets dinámicamente y no es muy flexible porque en el momento que cambia un tipo de letra (por ejemplo cuando se cambia de plataforma), todo se puede descuadrar, estas aplicaciones se caracterizan por ser muy estáticas ya que por lo general no se pueden maximizar, o si lo hacen su distribución no cambia en lo absoluto. El otro mecanismo es la utilización de proporciones, mediante este método, no se le dice a un botón que ubicará una posición absoluta, sino que se le ubica mediante un algoritmo que calcula el espacio que le puede brindar, dependiendo del tamaño de la aplicación, así por ejemplo si un botón ocupa media ventana de 40x40 pixels, pues tendrá un espacio disponible de 20*40 pixels, y si la ventana cambia de tamaño a por ejemplo 200*200 pixels, pues el botón contará ahora con 100*200 pixels, para reubicarse.

En *wxPython* se encuentran disponibles los dos métodos. Si se desea trabajar con el primer modelo se puede utilizar un entorno de desarrollo llamado *boa-constructor*, y si prefiere, tal como lo recomienda el autor, trabajar con proporciones, puede utilizar el diseñador de interfaces *wxglade*, tal como lo haremos en el tutorial

4.2. Los sizers

Los *sizers* son un *widget* especial utilizado para organizar las interfaces. Posee la característica de poder contener otros *widgets*, organizándolos según unos parámetros que se les puede definir.

Existen dos tipos de *sizer*, el *wxBoxSizer* (de caja) y el *wxGridSizer* (de malla). El primero puede ser de vertical u horizontal y representa una columna o una fila de widgets respectivamente, mientras que el segundo es una especie de cuadrícula en la cual se pueden insertar wigdets cual si fuera un tablero de ajedrez.

Los *wxBoxSizer* pueden contener tantos widgets como se necesite, he inclusive permiten la inclusión de nuevos widgets en tiempo de ejecución, en cambio a los *wxGridSizer* se les debe definir cuantas filas y cuantas columnas van a tener en el momento de su creación.

4.3. Los sizers en *wxGlade*

En *wxGlade* se añaden sizers utilizando o de la barra de íconos, para *wxBoxSizer* y *wxGridSizer* respectivamente.

Cada que se agrega una ventana, la herramienta añade un sizer por nosotros automáticamente, ese era el widget *sizer_1* que se ubicaba por debajo de la *VentanaSaludona* en la ventana de árbol.

Vamos a trabajar de nuevo en nuestra ventana saludona, vamos a agregar un cuadro de texto (*wxTextCtrl*)para escribir un saludo y un botón (*wxButton*) que al ser presionado saludará con el texto del cuadro.

Para esto seleccionamos y luego damos click dentro de la ventana saludona (en la vista de diseño), al hacerlo debería de aparecer una nueva caja de texto dentro de la ventana y un nuevo hijo en la ventana de árbol, por debajo del sizer, Le damos un nombre al widget (*tSaludo*, recomiendo utilizar letras antes de los nombres de variables para identificar el tipo del widget, en este caso texto) en la pestaña *Common* de la ventana de propiedades. Para que contenga un texto inicial que diga *Hola mundo* insertamos esa cadena en el atributo *Value* de la pestaña *Widget* y seleccionamos la propiedad *wxTE_MULTILINE* que nos permitirá insertar varias líneas de texto. Por último para que se vea más bonito, seleccionamos, de la pestaña *Layout*, la opción *wxExpand* y damos el valor de *1* a *Option*.

> **Sugerencia:** *wxPython* utiliza un solo widget para los cuadros de texto de una o más líneas controladas por la propiedad *wxTE_MULTILINE*.

Para acabar de decorar, le cambiamos el nombre al *sizer_1* por *sPal* para que no quede con ese nombre tan genérico, recuerde que para esto debemos seleccionar primero el sizer en la ventana de árbol y luego usar la ventana de propiedades. Si todo ha ido bien deberíamos tener algo como:

Figura 4-1. Un cuadro de texto

¡Perfecto!, ahora agreguemos el botón, podríamos intentar utilizando su ícono y luego dar click de nuevo en la ventana para agregarlo, pero ¿qué pasa?, ¿por qué no aparece?. Lo que pasa es que el *wxglade* no administra dinámicamente los sizers, tenemos que hacerlo manualmente, para esto vamos a la ventana de árbol, seleccionamos el sizer *sPal* y damos click de la derecha, en el menú que sale seleccionamos *Add Slot*, al hacer esto debería aparecer un nuevo espacio donde podremos agregar el botón, agreguémoslo repitiendo el procedimiento.

De nuevo para evitar nombre genéricos cambiemos su nombre a *bSaludar* y en la pestaña *Widget* cambiamos *Label* a *Saludar* (este es el texto que aparece en el botón. Para ajustar un poco el aspecto del botón, digámosle al *wxGlade* que deje un borde de 5 pixels entre el y el resto de cosas, para eso vamos a la pestaña *Layout*, en el valor *Border* colocamos 5, seleccionamos la opción *wxAll*, y la opción *wxALIGN_CENTER_HORIZONTAL*.

> **Sugerencia:** El sistema de bordes nos permite seleccionar bordes arriba (*wxTOP*), abajo (*wxBOTTOM*), a la izquierda (*wxLEFT*), a la derecha (*wxRIGHT*), en una combinación de los anteriores, o en todos al tiempo (*wxALL*).

Si hemos seguido todos los pasos correctamente podremos generar código para obtener una ventana como la siguiente, con el código fuente que sigue:

Figura 4-2. VentanaSaludona con un cuadro de texto y un botón

ventana_saludona_wxglade.py (ejemplos/ventana_saludona_wxglade.py)

```
#!/usr/bin/env python
# -*- coding: ISO-8859-1 -*-
# generated by wxGlade 0.4.1 on Sun May  7 18:28:10 2006

import wx

class VentanaSaludona(wx.Frame):
    def __init__(self, *args, **kwds):
        # begin wxGlade: VentanaSaludona.__init__
        kwds["style"] = wx.DEFAULT_FRAME_STYLE
        wx.Frame.__init__(self, *args, **kwds)
        self.tSaludo = wx.TextCtrl(self, -1, "Hola mundo", style=wx.TE_MULTILINE)
        self.bSaludar = wx.Button(self, -1, "Saludar")

        self.__set_properties()
        self.__do_layout()
        # end wxGlade

    def __set_properties(self):
        # begin wxGlade: VentanaSaludona.__set_properties
        self.SetTitle(u"Hola ¿Cómo estás?")
        self.SetSize((400, 300))
        # end wxGlade

    def __do_layout(self):
        # begin wxGlade: VentanaSaludona.__do_layout
        sPal = wx.BoxSizer(wx.VERTICAL)
        sPal.Add(self.tSaludo, 1, wx.EXPAND|wx.ADJUST_MINSIZE, 0)
        sPal.Add(self.bSaludar, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL|wx.ADJUST_MINSIZE, 5)
        self.SetAutoLayout(True)
        self.SetSizer(sPal)
        self.Layout()
        # end wxGlade
```

```
# end of class VentanaSaludona


class MyApp(wx.App):
    def OnInit(self):
        wx.InitAllImageHandlers()
        ventana = VentanaSaludona(None, -1, "")
        self.SetTopWindow(ventana)
        ventana.Show()
        return 1

# end of class MyApp

if __name__ == "__main__":
    app = MyApp(0)
    app.MainLoop()
```

4.4. ¿Cómo hacer que la aplicación haga algo?

Hasta ahora tendremos una aplicación con todas las funcionalidades generales de cualquier otra, maximiza, minimiza, se cierra, cambia de tamaño, podemos escribir texto en su caja, inclusive podemos copiar y pegar, pero su botón, pese a que se unde cada que lo presionan, ¡no hace nada!, ¿no se supone que los botones hacen algo?, pues sí, pero el botón no es adivino, nosotros tenemos que decirle que hacer. Para esto debemos modificar el código fuente generado por el *wxGlade*.

4.4.1. Agregando eventos al código

En un programa que utiliza la librería *wxPython* la línea de ejecucción del programa se ejecuta un poco diferente a otros programas a los que quizá estemos más acostumbrados. Lo usual es que un programa en *Python* las instrucciones se ejecutan una por una, en un orden específico, que solo se ve alterado por estructuras de control como ciclos o condicionales, o métodos de clases y demás herramientas de programación, pero en todo momento un programador, o por lo menos uno bueno, sabrá en todo momento en que fragmento del código se encuentra un programa en un momento dado y cual será el siguiente fragmento a ejecutar.

En un programa orientado a eventos (la técnica utilizada por el *wxPython*), el programador escribe “pequeños” fragmentos de código, en métodos, que se deben ejecutar cada que cierta acción se realice, esta acción es lo que se conoce como un *evento*, y el método se conocerá como un manejador de evento (*Event handler* en inglés). Maximizar la ventana, cambiar su tamaño, cerrarla, son todos eventos sobre el widget *wxFrame* que tienen un manejador por defecto, que hacen lo que esperamos que hagan, maximizar, cambiar tamaño y cerrar. Por otro lado existen otros eventos que no tienen manejadores, como por ejemplo el evento presionar botón (*wx.EVT_BUTTON*) de nuestro botón *bSaludar*, la tarea de un programador de aplicaciones orientadas a eventos es construir manejadores de eventos.

Sugerencia: Inclusive para eventos como cerrar, se puede definir un manejador que haga otra cosa diferente a la esperada.

Sugerencia: Cada widget tiene sus propios eventos característicos de su naturaleza (por ejemplo expandir árbol de los *wxTreeCtrl*) pero muchos comparten eventos en común, por ejemplo el evento dar click, existe tanto para *VentanaSaludona* como para *bSaludar*, es importante tener en cuenta a quien se le asigna el manejador, ya que si lo asignamos al botón solo se activará cuando le den click a él, pero si lo asignamos a la ventana se activará sobre la ventana y sobre el botón, ya que éste último pertenece a la ventana (es hijo de ella)

Aviso

Es importante definir los métodos lo suficientemente livianos para que no se bloquee la aplicación, esas aplicaciones que se quedan con un botón undido y sin responder son una muestra de manejadores pesados. Si requerimos manejadores que toman mucho tiempo es recomendable la utilización de hilos para no bloquear la interfaz, sin embargo hay que tener cuidado cuando se trabajan hilos con *wxPython* ya que desde un hilo no se puede modificar la interfaz directamente, pero eso es un tema que no cubre este tutorial, todavía.

Vamos a construir pues un manejador para el evento de presionar el botón *bSaludar*. Para agregar un evento se requieren dos pasos:

1. Lo primero que hay que hacer es construir un manejador de evento, como ya se había mencionado este debe ser un método, en este caso de la clase *VentanaSaludona* ya que a un miembro de ella (*bSaludar*) es a quien afectará el método. Para eso escribimos un nuevo método, que como cualquier otro método de una clase recibe el parámetro *self* para poder utilizar la instancia de la clase, pero además debe recibir otro parámetro que llamaremos *evento* que incluirá información útil del estado de la aplicación al momento de ocurrir el evento (por ejemplo la posición del mouse, o el item seleccionado), nuestro método imprimirá en pantalla un sencillo “Hola Mundo”.

```
def Saludar(self,evento=None):
    """Manejador del evento click sobre bSaludar, saluda al usuario"""
    print "Hola Mundo"
```

2. El segundo paso es informarle a la *ventanaSaludona* que el manejador para el evento *wx.EVT_BUTTON* de el botón *bSaludar* será el método *Saludar* para eso utilizamos un método llamado Bind (que lo tienen la mayoría de widgets), esto lo hacemos al final del constructor de la ventana, después de la línea que indica que allí termina el código generado por el wxGlade.

```
self.bSaludar.Bind(wx.EVT_BUTTON,self.Saludar)
```

Nótese que no estamos realizando un llamado al método Saludar, sino que lo pasamos como parámetro de una función, esto para que el *wxPython* pueda realizar el llamado cuando presionen el botón.

Aviso

El método *Bind* fue introducido en la versión 2.6 del *wxPython* así que si está utilizando una versión anterior, debería incluir un nuevo paso que consiste en asignarle un identificador al botón así:

```
self.bSaludar = wx.Button(self, wx.NewId(), "Saludar")
```

Y hacer el enlace del método de la siguiente manera:

```
wx.EVT_BUTTON(self,self.bSaludar.GetId(),self.Saludar)
```

Juntando todo quedaría un código como el que sigue:

```
#!/usr/bin/env python
# -*- coding: ISO-8859-1 -*-
# generated by wxGlade 0.4.1 on Sun May  7 18:28:10 2006

import wx

class VentanaSaludona(wx.Frame):
    def __init__(self, *args, **kwds):
        # begin wxGlade: VentanaSaludona.__init__
        kwds["style"] = wx.DEFAULT_FRAME_STYLE
        wx.Frame.__init__(self, *args, **kwds)
        self.tSaludo = wx.TextCtrl(self, -1, "Hola mundo", style=wx.TE_MULTILINE)
        self.bSaludar = wx.Button(self, -1, "Saludar")

        self.__set_properties()
        self.__do_layout()
        # end wxGlade

        self.bSaludar.Bind(wx.EVT_BUTTON,self.Saludar)

    def __set_properties(self):
        # begin wxGlade: VentanaSaludona.__set_properties
        self.SetTitle(u"Hola ¿Cómo estás?")
        self.SetSize((400, 300))
        # end wxGlade

    def __do_layout(self):
        # begin wxGlade: VentanaSaludona.__do_layout
        sPal = wx.BoxSizer(wx.VERTICAL)
        sPal.Add(self.tSaludo, 1, wx.EXPAND|wx.ADJUST_MINSIZE, 0)
        sPal.Add(self.bSaludar, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL|wx.ADJUST_MINSIZE, 5)
        self.SetAutoLayout(True)
        self.SetSizer(sPal)
        self.Layout()
        # end wxGlade
```

```
    def Saludar(self,evento=None):
        """Manejador del evento click sobre bSaludar, saluda al usuario"""
        print "Hola Mundo"

# end of class VentanaSaludona


class MyApp(wx.App):
    def OnInit(self):
        wx.InitAllImageHandlers()
        ventana = VentanaSaludona(None, -1, "")
        self.SetTopWindow(ventana)
        ventana.Show()
        return 1

# end of class MyApp

if __name__ == "__main__":
    app = MyApp(0)
    app.MainLoop()
```

Con esto tendremos una aplicación muy amigable que cada que le presionan el botón saludar imprime en la consola “Hola Mundo”, es importante resaltar que existen una gran cantidad de eventos, varios para cada widget, para saber que evento utilizar podemos mirar la documentación oficial de *wxPython* en la ficha de cada widget está la lista de los eventos que soporta, son todos aquellos que empiezan por wx.EVT

4.4.2. Saludando con ventanas

Hasta el momento tenemos una aplicación que por lo menos hace algo, sin embargo no parece muy útil, además es un gesto un poco feo que siendo una aplicación GUI, salude por la consola y para colmo de males ¿para qué sirve el cuadro de texto si no hace nada?. En esta sección corregiremos todo esto.

Primero vamos a hacer que el evento lea el contenido de la caja de texto y muestre el mensaje que allí se encuentre, para eso debemos utilizar un método que tienen los *wx.TextCtrl* que se llama *GetValue*, éste retorna una cadena unicode (si el *wxPython* fue compilado con dicha opción). Lo siguiente que haremos es utilizar un nuevo widget que se encarga de mostrar información casual de errores o de información en un diálogo emergente, es el *wx.MessageDialog*. *wxPython* tiene un amplio conjunto de diálogos de propósito específico como la selección de archivos *wx.FileDialog* o de directorios *wx.DirDialog*, o la selección de color *wx.ColourDialog*, o un díalogo de impresión *wx.PrintDialog*. El método se vería así:

`ventana_saludona_con_texto_y_boton.py (ejemplos/ventana_saludona_con_texto_y_boton.py)`

```
#!/usr/bin/env python
# -*- coding: ISO-8859-1 -*-
# generated by wxGlade 0.4.1 on Sun May  7 18:28:10 2006
```

```
import wx

class VentanaSaludona(wx.Frame):
    def __init__(self, *args, **kwds):
        # begin wxGlade: VentanaSaludona.__init__
        kwds["style"] = wx.DEFAULT_FRAME_STYLE
        wx.Frame.__init__(self, *args, **kwds)
        self.tSaludo = wx.TextCtrl(self, -1, "Hola mundo", style=wx.TE_MULTILINE)
        self.bSaludar = wx.Button(self, -1, "Saludar")

        self.__set_properties()
        self.__do_layout()
        # end wxGlade

        self.bSaludar.Bind(wx.EVT_BUTTON,self.Saludar)

    def __set_properties(self):
        # begin wxGlade: VentanaSaludona.__set_properties
        self.SetTitle(u"Hola ¿Cómo estás?")
        self.SetSize((400, 300))
        # end wxGlade

    def __do_layout(self):
        # begin wxGlade: VentanaSaludona.__do_layout
        sPal = wx.BoxSizer(wx.VERTICAL)
        sPal.Add(self.tSaludo, 1, wx.EXPAND|wx.ADJUST_MINSIZE, 0)
        sPal.Add(self.bSaludar, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL|wx.ADJUST_MINSIZE, 5)
        self.SetAutoLayout(True)
        self.SetSizer(sPal)
        self.Layout()
        # end wxGlade

    def Saludar(self,evento=None):
        """Manejador del evento click sobre bSaludar, saluda al usuario"""
        #Obtengo el saludo de la caja de texto
        saludo=self.tSaludo.GetValue()

        #Creo un diálogo para mostrarlo
        dlg=wx.MessageDialog(self, # El padre en este caso la misma ventana
                             saludo, # El mensaje a mostrar
                             u"Un saludo para tí", # El título de la ventana
                             wx.OK | # Muestre solo un botón de aceptación
                             wx.ICON_INFORMATION  # Muestre un ícono de información
                             )

        #Con crear un diálogo no básta para que aparezca, hace falta
        #llamar este método, que retorna la salida del mismo
        salida=dlg.ShowModal()
        #Esto es útil cuando se crea un diálogo con más de un botón
        if salida==wx.ID_OK:
            print u"Presionaron ok en el diálogo"
```

```
# end of class VentanaSaludona


class MyApp(wx.App):
    def OnInit(self):
        wx.InitAllImageHandlers()
        ventana = VentanaSaludona(None, -1, "")
        self.SetTopWindow(ventana)
        ventana.Show()
        return 1

# end of class MyApp

if __name__ == "__main__":
    app = MyApp(0)
    app.MainLoop()
```

Figura 4-3. Un saludo para tí