Python + GTK +Glade

Tabla de Contenidos

Ing. Carlos Zager Fernández
Corvus / Latinux de Venezuela
czager@corvus.com.ve

Tabla de Contenidos

•Lo primero que debemos hacer es crear en *Glade* una ventana muy sencilla que contenga un **Label** al cual le vamos a poner el mensaje “Hola mundo!”. También debemos asegurarnos que la ventana sea visible. Salvamos el proyecto con el nombre que deseemos (ejm: “hola.glade”).

•Debemos crear un archivo de código *Python* que se encargará de controlar nuestra interfaz. Para ello creemos con el editor de nuestra preferencia un archivo con el siguiente contenido.

```
import pygtk
pygtk.require('2.0')
import gtk
from gtk import glade
glade.XML('hola.glade')
gtk.main()
```

```
#python hola.py
```

•Deberíamos ver una ventana muy parecida a la que está de muestra. Fíjese que no responde a ningún evento, por lo que debe mandarle una señal de **kill** al programa para finalizarlo.

LaTinux

Tabla de Contenidos

•Ya sabemos crear una ventana. Procedamos a diseñar nuestra interfaz. Para ello vamos de nuevo a *Glade* y creamos una ventana.
•Agreguemosle un **Vbox** de tamaño 3 y lo rellenamos de la siguiente manera:

1. **MenuBar**
   - Eliminamos los **MenuItems** *Edit y View*
2. **Horizontal Button Box**
   - Tamaño 2
   - Lo alineamos a la izquierda
   - Le desactivamos el expandir
   - Le agregamos dos botones tipo stock:add,remove
3. **Scrolled Window**
   - **TreeView**
     - Activamos *Rules Hint*
     - Colocamos *Search Column* en 1
     - Lo renombramos como tree

•Con esto ya hemos creado la base de nuestra interfaz y al igual que en el ejemplo anterior podemos hacer un simple archivo de *Python* para probarla. Deberíamos ver algo parecido a la imagen de ejemplo
•Recuerde que no hemos conectado ningun evento todavía, así que no espere ver nada atractivo ya que los botones no van a hacer nada cuando los clickee

Recordatorio: si no ve nada...probablemente es que se le olvido activar la visibilidad de la ventana

LaTinux

Tabla de Contenidos

Ahora que tenemos nuestra interfaz, necesitamos darle funcionalidad a la misma. Para ello necesitamos conectar los eventos a funciones que nosotros implementemos.

Procedemos a abrir nuestra interfaz en *Glade* y hacemos lo siguiente:

- En el arbol de inspección seleccionamos el **MenuItem** *New* y le asignamos a la señal “activate” el valor “new”
- Lo mismo hacemos para *Open,Save,Save As,Quit y About*.
- Ahora necesitamos las señales que van a permitir crear o eliminar una tarea. Para ello asociamos en los botones *Add y Remove* las señales “clicked” con los valores “add” y “remove” respectivamente.
- También debemos asociar la señal “destroy” de la ventana principal con el valor “quit” (la misma que asociamos al **MenuItem** “Quit”). Esta se encuentra en el submenu **GtkObject** de las señales.

Ahora debemos hacer que las señales sean atendidas por funciones de *Python*

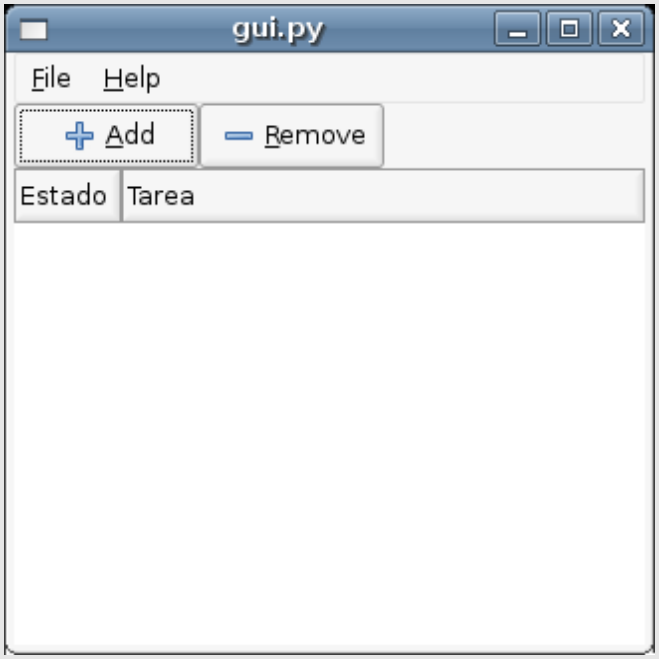
Tabla de Contenidos

Para darle funcionalidad a los eventos nos vamos al codigo de *Python* y debería quedar algo como esto...

```
import pygtk
pygtk.require('2.0')
import gtk,gobject
from gtk import glade

class gui:
	def __init__(self):
		self.glade=glade.XML('gui.glade')
		self.glade.signal_autoconnect({
			'quit':self.quit,
			'about':self.about,
			'add':self.add,
			'remove':self.remove,
			'open':self.open,
			'new':self.new,
			'save_as':self.save_as,
			'save':self.save,
		})

	def quit(self,b):
		gtk.main_quit()

	def add(self,b):
		#Aqui agregariamos una nueva tarea
		pass


	...

gui()
gtk.main()
```

LaTinux

Tabla de Contenidos



Un **TreeView** es una estructura de la librería **GTK** que nos permite tener un objeto que despliega su contenido como una lista de objetos divididos por columnas o como un arbol

En este caso estamos interesados en mostrar los objetos en forma de lista. Para ello necesitamos añadir el siguiente código al final de la funcion *__init__*

```
...
        self.tree=self.glade.get_widget('tree')

        self.model=gtk.TreeStore(gobject.TYPE_STRING,
                                 gobject.TYPE_STRING, gobject.TYPE_STRING)

        cellp = gtk.CellRendererText()
        col1=gtk.TreeViewColumn('Estado',cellp)
        col1.add_attribute(cellp,'background',0)

        cellt = gtk.CellRendererText()
        col2=gtk.TreeViewColumn('Tarea',cellt)
        col2.add_attribute(cellt,'text',1)

        self.tree.append_column(col1)
        self.tree.append_column(col2)

        self.tree.set_model(self.model)
...
```

Si ejecutamos el código ahora se debería ver algo como la imagen

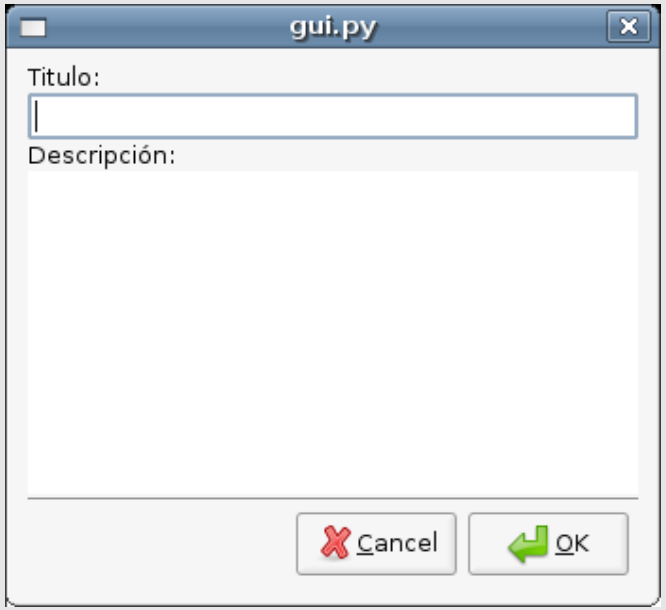
LaTinux

Tabla de Contenidos

Vamos a ver como trabajar con diálogos, para ello debemos diseñar nuestro diálogo en *Glade* y queremos que se vea como el de la imagen.

Lo hacemos con **Dialog Box**, **Vbox**, 2 Botones stock (Cancel,Ok), **Text Entry**, **Text View** y 2 **Label**. Al final renombramos el diálogo como dialogo

Una vez que tenemos nuestro diálogo listo, debemos de tener una manera de ejecutarlo. Cambiemos la funcion “add” por:



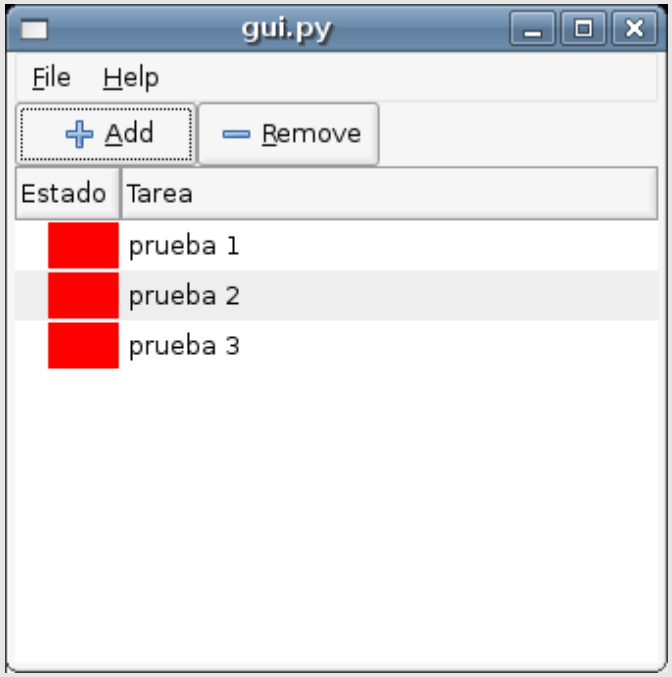

```
...

        def add(self,b):
                code=glade.XML('gui.glade','dialogo')
                dialogo=code.get_widget('dialogo')
                dialogo.show()


...
```

Hasta aqui ya podemos probar, si ejecutamos el código podemos ver que al hacer click en nuestro botón “add” se levanta este diálogo que acabamos de crear.

Pero esto así solo no es de mucha utilidad, así que vamos a ver como podemos utilizar el diálogo para guardar los datos del usuario en la lista

LaTinux

Tabla de Contenidos



Lo primero va a ser volver a *Glade* y colocar las señales y los nombres a los widgets para luego poder consultarlos.

Añadimos a las señales “clicked” de los botones los valores cancel y ok. Y renombramos el **TextArea** como titulo y el **TextView** como descripcion.

Luego debemos conectar estas señales a funciones en nuestro código. Quedaría asi:

```
...
        def add(self,b):
                code=glade.XML('gui.glade','dialogo')
                dialogo=code.get_widget('dialogo')
                code.signal_autoconnect({
                        'ok': lambda x: self.add_data(code),
                        'cancel':lambda x:dialogo.destroy()
                })
                dialogo.show()

        def add_data(self,code):
                dialogo=code.get_widget('dialogo')
                titulo=code.get_widget('titulo')
                descripcion=code.get_widget('descripcion')
                t1=titulo.get_text()
                buf=descripcion.get_buffer()
                start=buf.get_start_iter()
                end=buf.get_end_iter()
                t2=buf.get_text(start,end,True)
                self.model.append(None,['red',t1,t2])
                dialogo.destroy()
...
```

LaTinux

Tabla de Contenidos

Bueno ya en este punto tenemos cierta funcionalidad, obviamente falta mucho para que la aplicación este completa pero el resto es mecánico. Por ejemplo habría que hacer la funcionalidad de borrar, el diálogo “Acerca de”, etc.

Ahora vamos a imaginarnos que todo esto está hecho y que queremos que nuestra aplicación pueda guardar y cargar los datos para poder ser usados luego.

Para ello vamos a utilizar el módulo “csv” de *Python* para exportar los datos como archivos divididos por coma.

Sería algo así...

```
...
    def save(self,b):
        import csv
        writer = csv.writer(open(“~/.to-dos”, “w”))
        writer.writerows(self.model)

     def open(self,b):
        import csv
        reader = csv.reader(open(“~/.to-dos”, "r"))
        for row in reader:
                self.model.append(None,row)
...
```

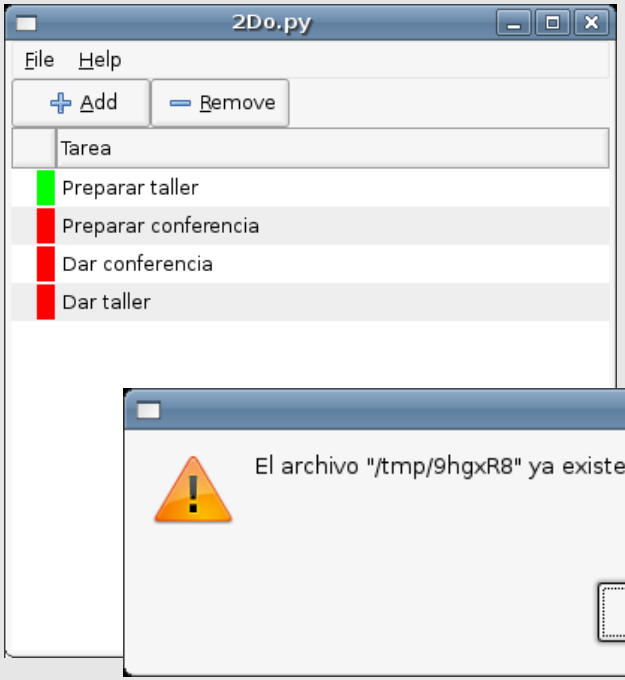
Tabla de Contenidos

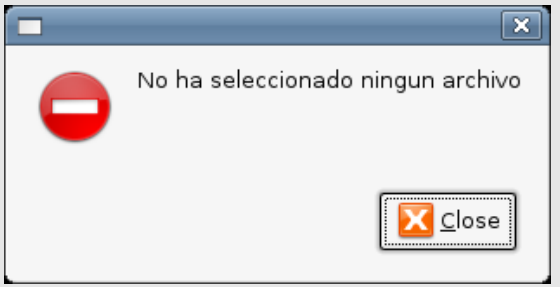
Bueno hasta ahora todo ha sido muy sencillo...

Pero debemos tomar en cuenta que no se ha hecho ninguna clase de verificicación, ni se levantan ventanas de diálogo de advertencia, de error, etc

Falta mucho para que esta aplicación este completa, pero por algún lado hay que empezar...

Ahora veamos como se vería esta aplicación un poco mas completa...





LOTINUX