# Capítulo 4

Fecha Release: Agosto 13, 2006
Versión: 1.0
Ultima Actualización: Agosto 13, 2006

## 4. Construyendo utilidades GTK para nuestras aplicaciones

En el capítulo 4 construimos el formulario principal de nuestra aplicación. En esta sección, diseñaremos una serie de utilidades comunes en las interfaces gráficas. Cual será la gran diferencia? Todas ellas estarán construidas para el GTK, lo cual garantiza que nuestras utilidades ejecutarán en la plataforma Windows o en Linux.

Las utilidades que construiremos serán las siguientes:

1. Una caja de diálogo que muestra un mensaje y se cierra cuando hacen clic en un botón [*Aceptar*].
2. Una caja de diálogo que muestra una pregunta y recibe una respuesta de Si o No.
3. Una caja de diálogo para solicitar un dato al usuario.

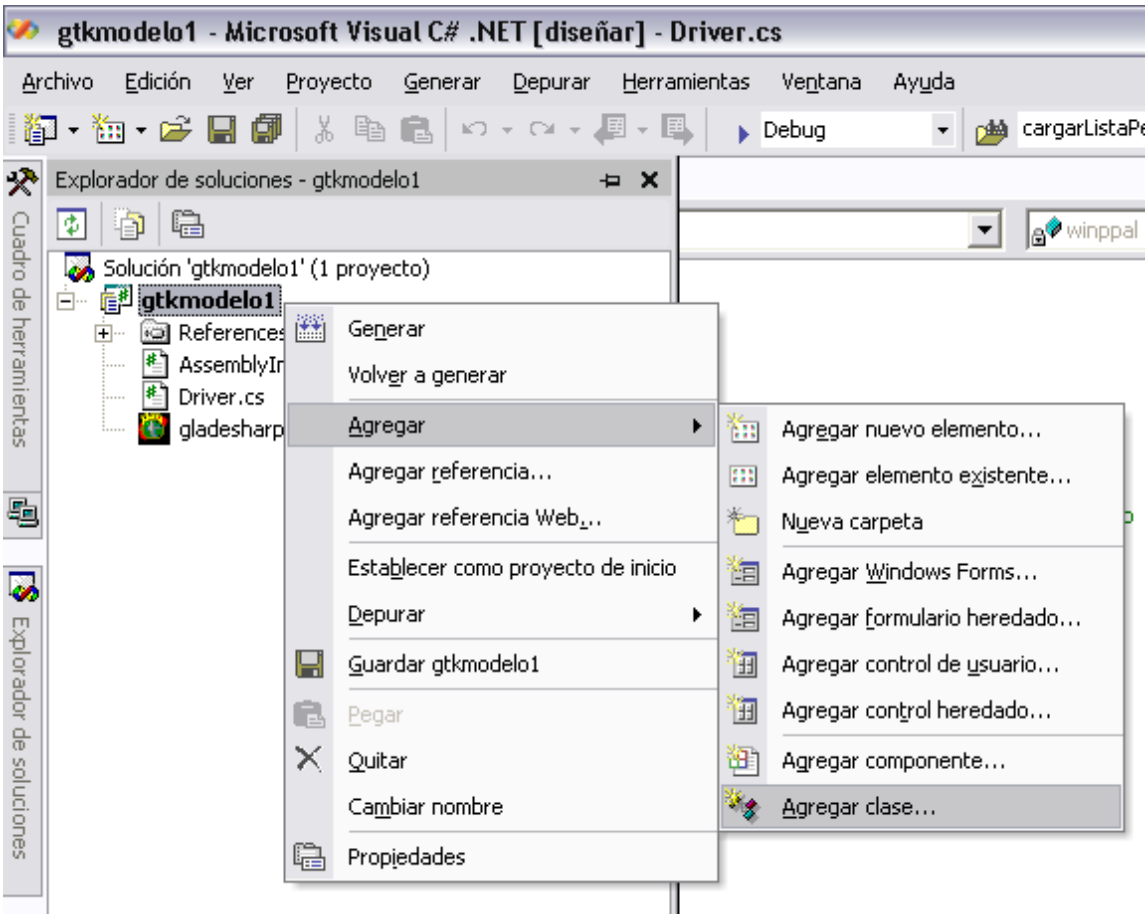
Para desarrollar estas utilidades necesitamos: el proyecto gtkmodelo1, el IDE de Visual Studio y dos cervezas. Por el momento, vaya al refrigerador, destape la primera cerveza y póngase cómodo mientras desarrollamos la primera utilidad.

## 4.1. La caja de diálogo con botón [*Aceptar*]

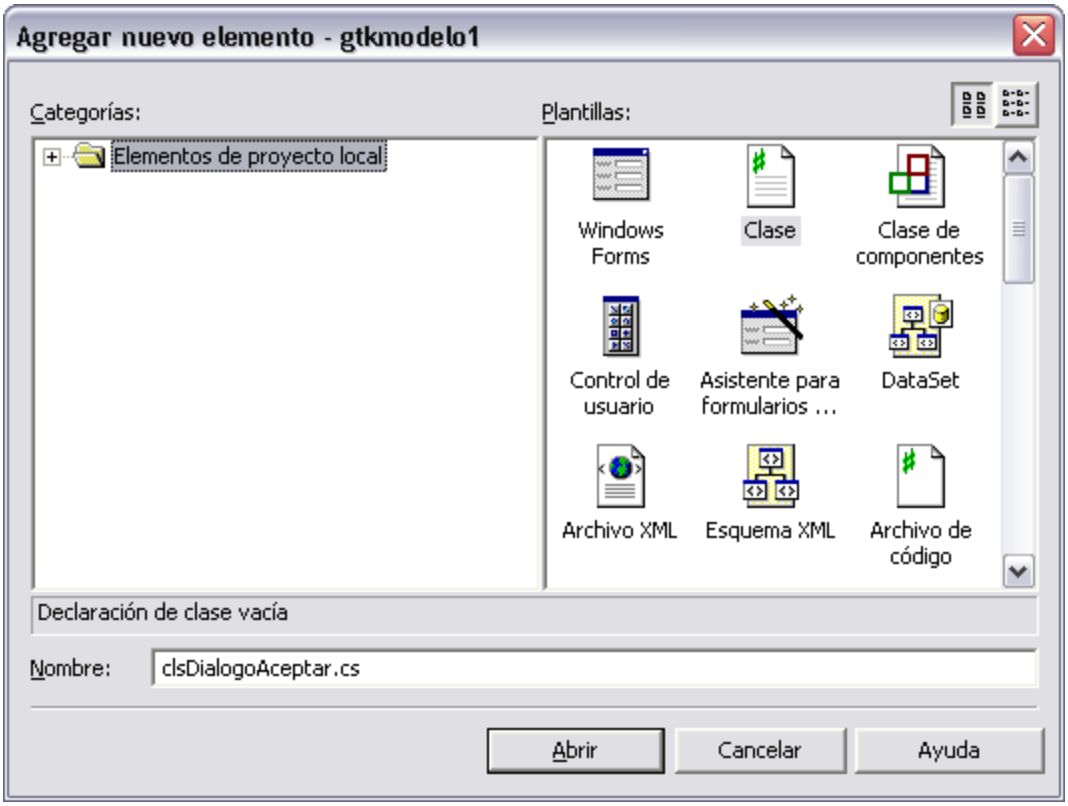
Es común en las interfaces gráficas, mostrarle mensajes informativos al usuario y esperar a que el usuario cierre la caja de diálogo haciendo clic en un botón [*Aceptar*].

La caja de diálogo que diseñaremos tendrá la siguiente presentación:

Cargue el IDE de Visual Studio y cargue el proyecto gtkmodelo1. Ahora, desde el explorador de proyecto haga clic derecho en *gtkmodelo1*, seleccione *agregar* del menú emergente, luego *Agregar clase*.




Especifique como nombre de clase el nombre clsDialogoAceptar. Damos este nombre para poderlo ubicar entre los archivos de forma sencilla, ya que una vez empieza a crecer nuestro proyecto en archivos, se hace difícil ubicar elementos si no se posee una técnica de nombres mnemónicos.




Ya tenemos una clase vacía dentro de nuestro proyecto. Esta clase, la definiremos a "puro código", sin utilizar el Glade, debido a que es bastante sencillo el contenido de esta clase, además, para demostrar que podemos combinar lo mejor de los dos mundos: utilizando el diseñador gráfico Glade y empleando widgets del Gtk a puro código.

Lo primero que haremos en la clase es escribir la referencia al gtk en la clase y definir el namespace de gtkmodelo. Luego, haremos que la clase herede de la clase de Gtk.Dialog; El código debe lucir de la siguiente manera (en negrita los cambios realizados):

```
using System;
using Gtk;

namespace gtkmodelo1
{
	/// <summary>
	/// Descripción breve de clsDialogoAceptar.
	/// </summary>
	public class clsDialogoAceptar : Gtk.Dialog
	{
		public clsDialogoAceptar() : base()
		{
			//
			// TODO: agregar aquí la lógica del constructor
			//
		}
	}
}
```

Ahora ya tenemos una definición de clase que incluye todos los atributos, métodos y eventos de un diálogo Gtk.

Lo siguiente que haremos será preparar a la clase *clsDialogoAceptar* para que reciba parámetros del diálogo: el título de la ventana, el mensaje a mostrar y el tipo de imagen a visualizar.

```
public clsDialogoAceptar( string titulo, string mensaje, string tipo) : base()
{

}
```

Ya tenemos una estructura de clase que recibe los parámetros que la clase necesita para no ser estática.

Lo que haremos ahora es empezar a programar la presentación de la clase en pantalla. El siguiente código, hace esa programación (this hace referencia a la misma clase de tipo Gtk.Dialog):

| **Instrucción** | **Descripción** |
|---|---|
| this.Title = titulo; | El título recibido como parámetro |
| this.HasSeparator = true; | Incluye un separador entre los botones Aceptar y el cuerpo del mensaje. |
| this.BorderWidth = 6; | Ancho del borde de la ventana |
| this.Resizable = false; | La ventana no permite cambiar su tamaño. Es fijo. |
| this.WindowPosition = Gtk.WindowPosition.Center; | Cuando se muestre, va a aparecer centrado en la pantalla. |

Ahora le programaremos el icono que tendrá nuestra ventana:

| **Instrucción** | **Descripción** |
|---|---|
| this.IconName = Stock.DialogInfo; | Tipo de icono por defecto para la caja de diálogo. En caso de que no se acierte el tipo que reconoce, muestra un icono por defecto. |
| if ( tipo == "ATENCION"){<br>this.IconName = Stock.DialogWarning;}; | Cuando recibe el texto “ATENCION” asocia un icono *warning* directamente del inventario de iconos del Gtk. |
| if ( tipo == "INFO"){<br>this.IconName = Stock.DialogInfo;}; | Cuando recibe el texto “INFO” asocia un icono *Info* directamente del inventario de iconos del Gtk. |
| if ( tipo == "PREGUNTA"){<br>this.IconName = Stock.DialogQuestion;}; | Cuando recibe el texto “PREGUNTA” asocia un icono *Question* directamente del inventario de iconos del Gtk. |
| if ( tipo == "ERROR"){<br>this.IconName = Stock.DialogError;}; | Cuando recibe el texto “ERROR” asocia un icono *Error* directamente del inventario de iconos del Gtk. |

Definimos a continuación el contenedor principal:

```
//contenedor principal
HBox h = new HBox();
h.BorderWidth = 6;
h.Spacing = 12;
```

Nuestro organizador será un contenedor Horizontal Box. Dentro de este, pondremos todos los objetos.

Ahora definimos un icono al lado del mensaje, según el tipo de mensaje que se desea mostrar:

```
//Imagen del mensaje
Image i = new Image();
i.SetFromStock( Stock.DialogInfo, IconSize.Dialog);
if ( tipo == "ATENCION"){ i.SetFromStock( Stock.DialogWarning, IconSize.Dialog);};
if ( tipo == "INFO"){ i.SetFromStock( Stock.DialogInfo, IconSize.Dialog);};
if ( tipo == "PREGUNTA"){ i.SetFromStock( Stock.DialogQuestion, IconSize.Dialog);};
if ( tipo == "ERROR"){ i.SetFromStock( Stock.DialogError, IconSize.Dialog);};


//agregar la imagen al organizador horizontal
i.SetAlignment( 0.5F, 0);
h.PackStart( i, false, false, 0);
```

Este código define un objeto de tipo imagen:

```
Image i = new Image();
```

Luego, con base en el parámetro tipo asignaremos una imagen desde el inventario (Stock) de Gtk:

```
i.SetFromStock( Stock.DialogInfo, IconSize.Dialog)
```

Después se agrega la imagen al organizador horizontal:

```
i.SetAlignment( 0.5F, 0);
h.PackStart( i, false, false, 0);
```

Se agrega un organizador vertical (Vertical Box) para mostrar el mensaje. Se agrega al contenedor vertical el mensaje:

```
VBox v = new VBox();
Label l = new Label( mensaje);
l.LineWrap = true;
l.UseMarkup = true;
l.Selectable = true;
l.Xalign = 0; l.Yalign = 0;
v.PackStart( l);
```

Después agregamos el contenedor vertical al horizontal principal, pero empaquetado al final. Se le indica al gtk que muestre todos los objetos empaquetados en el horizontal. Al final, se agrega el organizador horizontal a un organizador vertical dentro de la ventana actual:

```
h.PackEnd( v);
h.ShowAll();
this.VBox.Add( h);
```

Por último, se define el botón Aceptar en la caja de diálogo. El botón se crea a partir del modelo predefinido de Aceptar que reside en el inventario del Gtk. Este automáticamente se ubicará en la posición ideal de un botón en una caja de diálogo: ajustado a la derecha y al fondo de la ventana.

```
this.AddButton( Stock.Ok, ResponseType.Ok);
```

A continuación se lista el código completo de la clase utilidad:

```
using System;
using Gtk;

namespace gtkmodelo1
{
	/// <summary>
	/// Descripción breve de clsDialogoAceptar.
	/// </summary>
	public class clsDialogoAceptar : Gtk.Dialog
	{
		public clsDialogoAceptar( string titulo, string mensaje, string tipo) : base()
		{
			//presentación en pantalla
			this.Title = titulo; //titulo en la ventana
			this.HasSeparator = true; //incluye un separador entre el cuerpo y los botones?
			this.BorderWidth = 6; //distancia del borde de la ventana y los objetos que contiene
			this.Resizable = false; //se puede cambiar su tamaño?
			this.WindowPosition = Gtk.WindowPosition.Center; //caja de dialogo centrado cuando cargue

			//icono de la ventana
			this.IconName = Stock.DialogInfo;
			if ( tipo == "ATENCION"){ this.IconName = Stock.DialogWarning;};
			if ( tipo == "INFO"){ this.IconName = Stock.DialogInfo;};
			if ( tipo == "PREGUNTA"){ this.IconName = Stock.DialogQuestion;};
			if ( tipo == "ERROR"){ this.IconName = Stock.DialogError;};

			//contenedor principal
			HBox h = new HBox();
			h.BorderWidth = 6;
			h.Spacing = 12;

			//Imagen del mensaje
			Image i = new Image();
			i.SetFromStock( Stock.DialogInfo, IconSize.Dialog);
			if ( tipo == "ATENCION"){ i.SetFromStock( Stock.DialogWarning, IconSize.Dialog);};
			if ( tipo == "INFO"){ i.SetFromStock( Stock.DialogInfo, IconSize.Dialog);};
			if ( tipo == "PREGUNTA"){ i.SetFromStock( Stock.DialogQuestion, IconSize.Dialog);};
			if ( tipo == "ERROR"){ i.SetFromStock( Stock.DialogError, IconSize.Dialog);};

			//agregar la imagen al organizador horizontal
			i.SetAlignment( 0.5F, 0);
			h.PackStart( i, false, false, 0);

			//agregar un organizador vertical para el mensaje
			VBox v = new VBox();
			Label l = new Label( mensaje);
			l.LineWrap = true;
```

```
                        l.UseMarkup = true;
                        l.Selectable = true;
                        l.Xalign = 0; l.Yalign = 0;
                        v.PackStart( l);

                        //agregar el contenedor vertical al horizontal
                        h.PackEnd( v);
                        h.ShowAll();
                        this.VBox.Add( h);

                        //agregar el botón Aceptar desde el inventario Gtk
                        this.AddButton( Stock.Ok, ResponseType.Ok);

                }
        }
}
```

La forma correcta de invocar a esta utilidad desde otras partes de la aplicación es de la siguiente forma:

```
clsDialogoAceptar miVariable = new clsDialogoAceptar( "mi_titulo", "mi_mensaje", "mi_parametro");
ResponseType resp = (ResponseType) miVariable.Run();
miVariable.Hide();
```

Este diálogo suspenderá la ejecución de la aplicación, hasta que el usuario haga clic en el botón aceptar. Los parámetros del tipo de diálogo aceptado son los siguientes: *ATENCION*, *INFO*, *PREGUNTA* y *ERROR*. Claro que usted puede implementar más tipos dependiendo de sus necesidades de desarrollo. Con estas cuatro se cubre a la mayoría de tipos de diálogo.

La siguiente es una muestra del tipo de mensaje que aparece cuando se define una variable de la forma

```
clsDialogoAceptar miVariable = new clsDialogoAceptar( "mi titulo", "mi mensaje", "parametro");
```

| *ATENCION* | *INFO* | *PREGUNTA* | *ERROR* |
|---|---|---|---|
|  |  |  |  |



Con esta utilidad de forma rápida podemos pasarle mensajes al usuario desde nuestra aplicación.

## 4.2. La caja de diálogo con botones [*SI*] y [*NO*].

Esta caja de diálogo le hará al usuario una pregunta y tomará la respuesta. Desde el código de donde se invoque esta utilidad, podremos verificar que respondió el usuario. En realidad es bastante útil para obtener afirmaciones o negaciones por parte del usuario cuando necesitamos ejecutar algunas tareas en nuestras aplicaciones.

Cree una nueva clase y defina como nombre: clsDialogoSN.

En realidad esta clase es igual en la lógica a la anterior, pero se debe ajustar el siguiente código que aparece en negrita. El resto del código será el mismo. Aquí está el listado completo del código:

```
using System;
using Gtk;

namespace gtkmodelo1
{
	/// <summary>
	/// Descripción breve de clsDialogoSN.
	/// </summary>
	public class clsDialogoSN : Gtk.Dialog
	{
		public clsDialogoSN( string titulo, string mensaje) : base()
		{
			//presentacion en pantalla
			this.Title = titulo; //titulo en la ventana
			this.HasSeparator = true; //incluye un separador entre el cuerpo y los botones?
			this.BorderWidth = 6; //distancia del borde de la ventana y los objetos que contiene
			this.Resizable = false; //se puede cambiar su tamaño?
			this.WindowPosition = Gtk.WindowPosition.Center; //caja de dialogo centrado cuando cargue

			//icono de la ventana
			this.IconName = Stock.DialogQuestion;

			//contenedor principal
			HBox h = new HBox();
			h.BorderWidth = 6;
			h.Spacing = 12;

			//Imagen del mensaje
			Image i = new Image();
			i.SetFromStock( Stock.DialogInfo, IconSize.Dialog);
			i.SetFromStock( Stock.DialogQuestion, IconSize.Dialog);

			//agregar la imagen al organizador horizontal
			i.SetAlignment( 0.5F, 0);
			h.PackStart( i, false, false, 0);

			//agregar un organizador vertical para el mensaje
			VBox v = new VBox();
			Label l = new Label( mensaje);
			l.LineWrap = true;
```

```
                        l.UseMarkup = true;
                        l.Selectable = true;
                        l.Xalign = 0; l.Yalign = 0;
                        v.PackStart( l);

                        //agregar el contenedor vertical al horizontal
                        h.PackEnd( v);
                        h.ShowAll();
                        this.VBox.Add( h);

                        //agregar el botones SI y NO desde el inventario Gtk
                        this.AddButton( Stock.Yes, ResponseType.Yes);
                        this.AddButton( Stock.No, ResponseType.No);
                }
        }
}
```

La forma correcta de invocar a esta utilidad desde otras partes de la aplicación es de la siguiente forma:

```
clsDialogoSN miVariable = new clsDialogoSN( "mi_titulo", "mi_mensaje");
ResponseType resp = (ResponseType) miVariable.Run();
if ( resp == ResponseType.Yes)   // ResponseType.No
{
        //hacer algo
}
miVariable.Hide();
```

Este diálogo suspenderá la ejecución de la aplicación, hasta que el usuario haga clic en el botón aceptar. Observe que este diálogo define de forma fija su imagen de tipo pregunta, debido a que será utilizado para obtener respuestas del usuario.

La siguiente es una muestra del tipo de mensaje que aparece cuando se define una variable de la forma

```
clsDialogoSN miVariable = new clsDialogoSN( "mi titulo", "mi mensaje");
```


Con esta utilidad de forma rápida podemos pedirle confirmación al usuario sobre algo desde nuestra aplicación. Como ya la tenemos funcionando, es hora de ir por la segunda cerveza.

## 4.3. La caja de diálogo para pedir información al usuario

Esta caja de diálogo servirá para pedir de forma rápida, un dato al usuario. Incluirá una petición, una caja de texto para recibir la respuesta y dos botones de [*Aceptar*] y [*Cancelar*]. Su código es muy similar al de las utilidades anteriores, pero incluye algunos elementos que ayudan a obtener el dato que el usuario suministró.

Aquí está el listado del código completo. Se resalta en negrita el código que cambia respecto a la utilidad anterior.

```
using System;
using Gtk;

namespace gtkmodelo1
{
	/// <summary>
	/// Descripción breve de clsDialogEntrada.
	/// </summary>
	public class clsDialogEntrada : Gtk.Dialog
	{
		string strTexto;
		Entry txtDato;

		//propiedad para almacenar la respuesta del usuario
		//estará disponible para nuestro codigo, una vez que el usuario
		//cierre la caja de dialogo conel botón Aceptar
		public string txtTexto
		{
			get { return strTexto;}
			set { strTexto = value;}
		}

		public clsDialogEntrada( string titulo, string pregunta, string tipo) : base()
		{
			//presentacion en pantalla
			this.Title = titulo; //titulo en la ventana
			this.HasSeparator = true; //incluye un separador entre el cuerpo y los botones?
			this.BorderWidth = 6; //distancia del borde de la ventana y los objetos que contiene
			this.Resizable = false; //se puede cambiar su tamaño?
			this.WindowPosition = Gtk.WindowPosition.Center; //caja de dialogo centrado cuando cargue

			//se inicializa la propiedad del texto a retornar en blanco
			this.txtTexto = "";

			//icono de la ventana
			this.IconName = Stock.DialogInfo;
			if ( tipo == "ATENCION"){ this.IconName = Stock.DialogWarning;};
			if ( tipo == "INFO"){ this.IconName = Stock.DialogInfo;};
			if ( tipo == "PREGUNTA"){ this.IconName = Stock.DialogQuestion;};
			if ( tipo == "ERROR"){ this.IconName = Stock.DialogError;};
			if ( tipo == "LOGIN"){ this.IconName = Stock.Network;};
```

```
                //contenedor principal
                HBox h = new HBox();
                h.BorderWidth = 6;
                h.Spacing = 12;

                //Imagen del mensaje
                Image i = new Image();
                i.SetFromStock( Stock.DialogInfo, IconSize.Dialog);
                if ( tipo == "ATENCION"){ i.SetFromStock( Stock.DialogWarning, IconSize.Dialog);};
                if ( tipo == "INFO"){ i.SetFromStock( Stock.DialogInfo, IconSize.Dialog);};
                if ( tipo == "PREGUNTA"){ i.SetFromStock( Stock.DialogQuestion, IconSize.Dialog);};
                if ( tipo == "ERROR"){ i.SetFromStock( Stock.DialogError, IconSize.Dialog);};
                if ( tipo == "LOGIN"){ i.SetFromStock( Stock.Network, IconSize.Dialog);};

                //agregar la imagen al organizador horizontal
                i.SetAlignment( 0.5F, 0);
                h.PackStart( i, false, false, 0);
                //agregar un organizador vertical para el mensaje
                VBox v = new VBox();
                Label l = new Label( pregunta);
                l.LineWrap = true;
                l.UseMarkup = true;
                l.Selectable = true;
                l.Xalign = 0; l.Yalign = 0;
                v.PackStart( l);
                //en el mismo organizador vertical, agregar el campo texto de entrada
                txtDato = new Entry();
                txtDato.Text = "";
                txtDato.Xalign = 0;
                v.PackEnd( txtDato);
                //agregar un capturador del evento cuando el usuario escriba texto
                txtDato.Changed += new EventHandler( txtDato_Changed);

                //agregar el contenedor vertical al horizontal
                h.PackEnd( v);
                h.ShowAll();
                this.VBox.Add( h);
                //agregar el botón Aceptar y Cancelar desde el inventario Gtk
                this.AddButton( Stock.Cancel, ResponseType.Cancel);
                this.AddButton( Stock.Ok, ResponseType.Ok);
        }

        //capturador del evento escribir en el campo texto
        public void txtDato_Changed(object o, EventArgs args)
        {
                //guardar en la propiedad txtTexto de la clase
                //la respuesta que dio el usuario. Desde el codigo se leera luego
                //este valor una vez se cierre la caja de dialogo con Aceptar/Cancelar
                this.txtTexto = txtDato.Text.Trim();
                return;
        }


    }
}
```

En este código aparecen algunos elementos nuevos no tratados en las dos utilidades anteriores.

3. Aparece una propiedad para almacenar la respuesta del usuario en la clase. Una vez se cierre la caja de diálogo, estará disponible en la clase para poder capturar el texto que entró el usuario.

```
string strTexto;
Entry txtDato;

public string txtTexto
{
        get { return strTexto;}
        set { strTexto = value;}
}
```

4. Se define un nuevo tipo de icono o imagen a mostrar en la caja de diálogo.

```
if ( tipo == "LOGIN"){ this.IconName = Stock.Network;};
if ( tipo == "LOGIN"){ i.SetFromStock( Stock.Network, IconSize.Dialog);};
```

5. Se crea un objeto de tipo Entrada (Entry) para leer la respuesta del usuario

```
txtDato = new Entry();
```

6. Se agrega un capturador del evento texto cambiado (text Changed) al objeto de tipo Entry.

```
txtDato.Changed += new EventHandler( txtDato_Changed);
```

Esta rutina se ejecuta cuando el usuario modifica el texto y pasa el control a otro objeto del formulario con el uso del mouse o del teclado. Inmediatamente almacenaremos el valor entrado en la propiedad del formulario

```
public void txtDato_Changed(object o, EventArgs args)
{
        this.txtTexto = txtDato.Text.Trim();
        return;
}
```

.

7. Se agregan dos botones de Aceptar y Cancelar a la caja de diálogo.

```
this.AddButton( Stock.Cancel, ResponseType.Cancel);
this.AddButton( Stock.Ok, ResponseType.Ok);
```

La forma correcta de invocar esta utilidad dentro de nuestro código es de la siguiente manera:

```
clsDialogoEntrada miVariable = new clsDialogoEntrada( "Titulo", "Pregunta", "LOGIN");
ResponseType resp = (ResponseType) miVariable.Run();
miVariable.Hide();
if (resp == ResponseType.Cancel || resp == ResponseType.DeleteEvent)
{
        //hacer algo
}
if (resp == ResponseType.Ok)
{
        //hacer algo
}
return;
```

Con ResponseType se puede validar que botón pulsó el usuario. La lógica le dirá con cual de las dos respuestas juega en el código. El valor a verificar lo podemos obtener de *miVariable.txtTexto*.

Esta es una muestra de la caja de diálogo que se obtiene cuando se invoque a la rutina de la siguiente forma:

```
clsDialogoEntrada miVariable = new clsDialogoEntrada( "Mi Titulo", "Mi Pregunta", "LOGIN");
```


Ahora sus aplicaciones podrán solicitar de forma bastante sencilla información. A su vez, el código podrá saber que datos suministró el usuario y hacer algo con ese valor.

## 4.4. Empleando la primera utilidad en nuestra GUI principal

Vamos a utilizar la clase clsDialogoSN para preguntarle al usuario si desea cerrar la aplicación cuando hace clic en el botón Cerrar de la barra de herramientas, o cuando hace selecciona del submenú Catálogo la opción Salir.

Identifique dentro del código de la clase principal (Driver.cs) la línea que tiene el método *on_tbSalir_clicked* y modifique la rutina según el siguiente código:

```
protected void on_tbSalir_clicked(object o, EventArgs args)
{
	clsDialogoSN d = new clsDialogoSN( "Cerrar Aplicacion", "Desea cerrar la aplicación?");
	ResponseType resp = (ResponseType) d.Run();
	d.Hide();
	if (resp == ResponseType.Yes)
	{
		Application.Quit();
	}
	return;
}
```

Identifique ahora dentro del código la línea que tiene el método *on_opcSalir_activate* y modifique la rutina según el siguiente código:

```
protected void on_opcSalir_activate(object o, EventArgs args)
{
	clsDialogoSN d = new clsDialogoSN( "Cerrar Aplicacion", "Desea cerrar la aplicación?");
	ResponseType resp = (ResponseType) d.Run();
	d.Hide();
	if (resp == ResponseType.Yes)
	{
		Application.Quit();
	}
	return;
}
```

Compile y verifique lo que hace la rutina. Ahora ya no se cerrará hasta que el usuario responda en forma afirmativa a la pregunta.

## 4.5. Resumen

Hemos desarrollado tres utilidades que nos pueden ayudar mucho a definir de forma rápida cajas de diálogo para interactuar con el usuario. Se han diseñado las tres típicas cajas de diálogo. Estas cajas de diálogo las iremos utilizando en los proyectos que definiremos más adelante.

En el próximo capítulo diseñaremos la clase para gestionar los países de nuestra aplicación ejemplo. Servirá de paso para mostrar como adicionar nuevos elementos a la GUI principal, diseñados a su vez en Glade.

Cordialmente,

Mauricio Cano Ossa
Un discípulo más del dios de los monos...