Capítulo 2

Fecha Release: Agosto 12, 2006
Versión: 1.0
Ultima Actualización: Agosto 12, 2006

2. Programación de una GUI en Windows

En el capítulo uno dejamos nuestra estación a punto y lista para empezar a crear aplicaciones gráficas que funcionen bien en Windows y en Linux. Antes de construir nuestra primer aplicación es importante tener presente algunos aspectos del desarrollo de entornos gráficos.

Cuando se desarrolla en un sistema operativo, existe "algo" que se encarga de generar la forma que se visualiza en pantalla, de atrapar las interacciones del usuario en pantalla y que permite que nuestro código pueda interactuar con el usuario. Todo lo que vemos comúnmente en una interfaz gráfica, es generada y controlada por ese componente: el administrador de interfaz gráfica.

Para desarrollar en Windows con Microsoft .net se emplea de forma automática el gestor gráfico GDI. Este es el que pinta el formulario, escucha y atrapa los eventos de la interfaz (clics del mouse, manipulación de estados de una ventana, cierre de una aplicación, etc.). Cuando se utiliza el entorno de Microsoft, toda esta ingeniería se esconde y no necesitamos preocuparnos de esa parte y el desarrollador se limitará a construir el resto del código que pone en marcha a la aplicación.

El administrador GDI solo funciona bajo Windows y funciona a la perfección por que viene integrado al sistema Operativo de Microsoft Windows; esa es la razón de esa integración tan perfecta con el IDE de Microsoft.

Pero que pasa si deseamos construir una aplicación con una interfaz gráfica así de bonita como las que hacemos en la plataforma Windows para la plataforma Linux? La respuesta es que debemos buscar un equivalente que nos permita en la plataforma Linux, cumplir las tareas que hace el GDI de Windows.

Se han desarrollado muchos administradores de entorno gráfico, todas en C++ pero sobre sale de todas ellas por su rápido crecimiento el Gtk (www.gtk.org). Este administrador permite desarrollar en la plataforma Windows o Linux, interfaces gráficas que lucen igual en los dos sistemas.

Cuando se desarrolla para Windows se cuenta con los denominados *WinForms*, los cuales son administrados internamente por el GDI. Los desarrolladores de la plataforma mono han portado esta funcionalidad a través de conversiones hacia un administrador gráfico compatible con Windows y Linux, pero no han logrado ese estado de arte que caracteriza a una aplicación desarrollada para GDI.

Gtk cuenta con el poder de que se han desarrollado muchos *drivers* o *wrappers* que permite hacer llamados al gestor gráfico para que administre nuestra GUI y se encargue de atender a los eventos del usuario y comunicarlo a nuestro código para que cumpla con la lógica que tiene el aplicativo.

Aunque existen muchos motores gráficos disponibles, en este documento nos centraremos en el Gtk, especialmente en el *wrapper* existente para C# llamado Gtk# (Gtk Sharp).

Otra situación que se presenta a la hora de querer hacer los diseños gráficos a como nos tiene acostumbrado los IDEs como los de Microsoft o Borland, es que tales entornos cuentan con diseñadores gráficos (lienzos) donde uno selecciona objetos de una barra de herramientas y los coloca sobre una superficie para construir la interfaz gráfica que deseamos. Luego, al hacer doble clic en cada objeto, podemos asociar el código que la lógica de la aplicación requiere. Estos entornos comerciales, oculta por completo esa parte del desarrollo de interfaces gráficas, lo cual resulta muy cómodo para el desarrollador, ya que sabe que tal como se ve en el IDE, se verá cuando el usuario final ejecute nuestra aplicación.

Ahora bien, como el mono apenas está comenzando a dotarse de herramientas, algunos desarrolladores están a la ardua tarea de lograr esa facilidad empleando el XML como gestor de la información que requiere el compilador para construir nuestra interfaz gráfica. El MonoDevelop es un claro ejemplo de los avances y los logros que en esta área del diseño tendrá el mono. A la fecha, ya es posible desarrollar dentro del entorno IDE *MonoDevelop*, la interfaz gráfica, mediante el uso de *Stetic* o de *Gtk*. Pero ahí no paran los conceptos a tener claro a la hora de desarrollar interfaces gráficas.

Como el entorno IDE es el que debe almacenar la información de nuestro diseño GUI, debe existir algo dentro del entorno de desarrollo que reciba esos datos y durante la etapa de compilación, construya los elementos técnicos necesarios para que al momento de ejecutarse, aparezca pintada en la pantalla del computador, el diseño tal cual nosotros lo planeamos. En cualquier IDE existe un componente que se encarga de esta parte: registrar las coordenadas, información de objetos, rutinas asociadas a eventos, imágenes, en fin, cualquier objeto disponible dentro de la paleta de herramientas del entorno gráfico para nuestro diseño. El entorno debe leer cada vez de un fichero, esa configuración y pintarnos la interfaz tal cual la diseñamos desde un principio. El IDE debe ser el puente entre nuestro resto de código y la interfaz gráfica cuando se compile la aplicación.

Ahí es cuando entra a la escena un nuevo componente: Glade (www.glade.org). Esta utilidad nos permitirá diseñar de forma gráfica una GUI. Esta herramienta permite pintar la mayoría de los objetos disponibles para el administrador gráfico Gtk. Pero funciona de forma desconectada a nuestro IDE, lo cual no permite hacer doble clic sobre los objetos y escribir nuestro código. En MonoDevelop se está trabajando con el diseñador Stetic como interfaz que permita diseñar esto. Otro proyecto como Gazpacho, tiene una gran integración con el MonoDevelop pero no será incluido en nuestros ejemplos. En el sitio web de mono-project en la sección de documentación, encontrará una cantidad de revisiones de diseñadores compatibles con mono. Solo el tiempo dirá cual diseñador tiene éxito en el mundo de los desarrolladores de mono.

Volvamos a nuestro diseñador gráfico Glade. Este genera una base de datos XML que el compilador entiende en el momento de generar el código ejecutable. Ahí es cuando entra el componente de Novell Gtk# par Visual Studio .net. Este componente permite que el Glade pueda ser invocado desde el IDE de desarrollo, y que aparezca como un objeto dentro del proyecto que al recibir un doble clic nuestro, cargue al Glade para que podamos diseñar la GUI.

Este componente además, integra dentro de las cajas de diálogo de un proyecto nuevo, la posibilidad de seleccionar proyectos tipo Gtk y Glade. Luego analizaremos estos dos tipos de proyectos. El componente permite integrar en el código, llamado a los objetos de Glade a partir de la base de datos XML con la información de nuestro diseño GUI y a la vez, registrar eventos, código dentro de un evento, invocar mediante código a un objeto de la GUI, programar aspectos de la ventana, es decir, lo que antes nos hacia el entorno IDE en un *winform*.

Perdidos con tanta información? Entonces repacemos lo que hemos dicho hasta el momento. Tenemos el IDE que permite administrar nuestro proyecto de software, un entorno gráfico objetivo para el cual desarrollaremos la interfaz gráfica, y que debe funcionar en cualquier sistema operativo. Como nuestro objetivo es el desarrollo multiplataforma, entonces tenemos el siguiente inventario:

Componente	Producto
IDE proyecto:	Visual Studio .Net 2003
IDE diseño GUI:	Glade
Administrador gráfico objetivo:	Gtk
Puente entre nuestra aplicación y el administrador gráfico:	Gtk#
Requisitos adicionales:	Tres cervezas de la marca preferida (ya era hora de que aparecieran en la etapa de desarrollo)

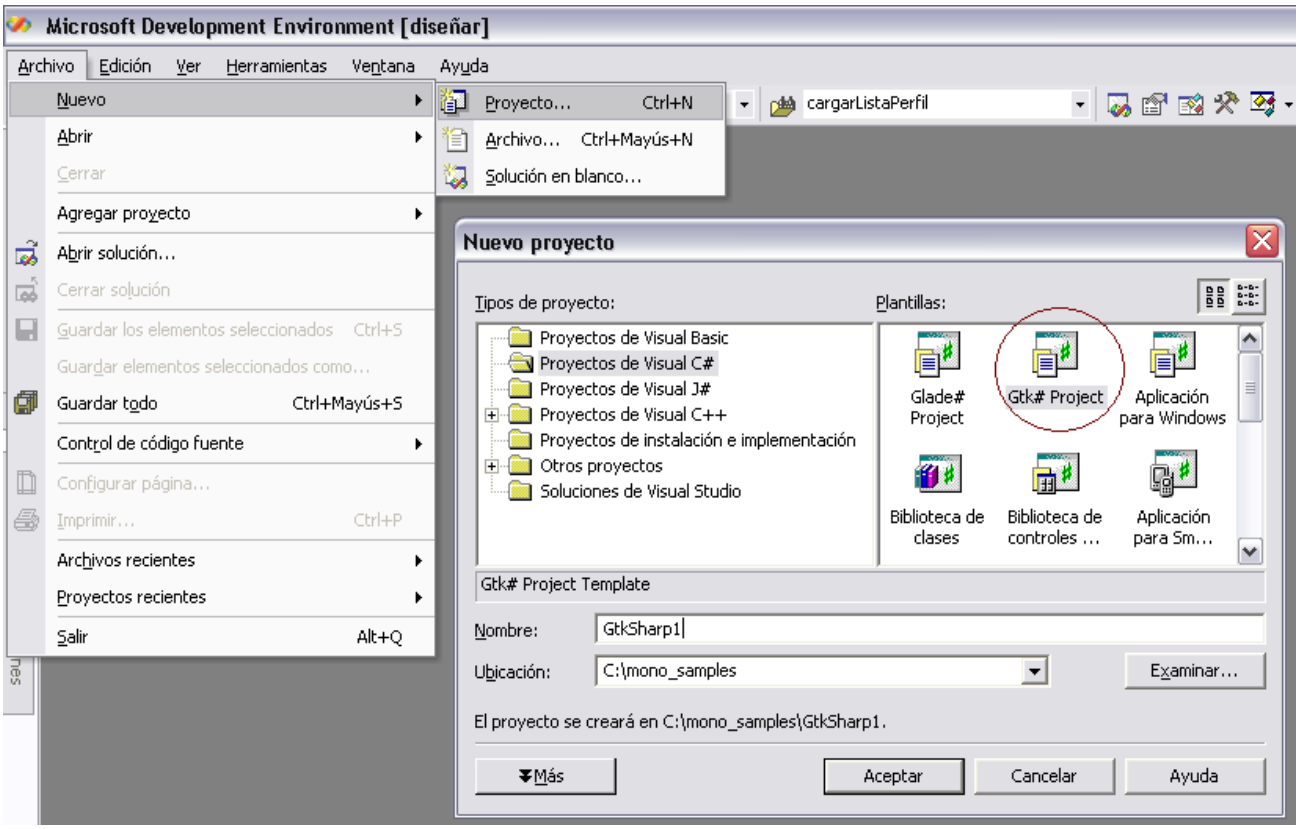
Las siguientes secciones asumen que usted ya ha leído la teoría de Gtk y que conoce los Widgets que este manejador permite definir en una aplicación. Haga especial énfasis en los contenedores de objetos, ya que estos son los que permiten organizar nuestros objetos en la interfaz gráfica. Para eso recomiendo el sitio de mono-project, para que le de una revisión a ejemplos de códigos de Gtk#. Las referencias bibliográficas que relaciono al final del documento, son una buena fuente para experimentar con los widgets del Gtk desde el mono.

Vamos a desarrollar en las siguientes secciones, nuestro primer proyecto con el IDE Visual Studio .Net 2003 y teniendo como meta desarrollar para el administrador Gtk. Tras esta ardua tarea de asimilar los conceptos de Gtk, se ha hecho merecedor de la primera cerveza.

2.1. Nuestro primer proyecto Gtk

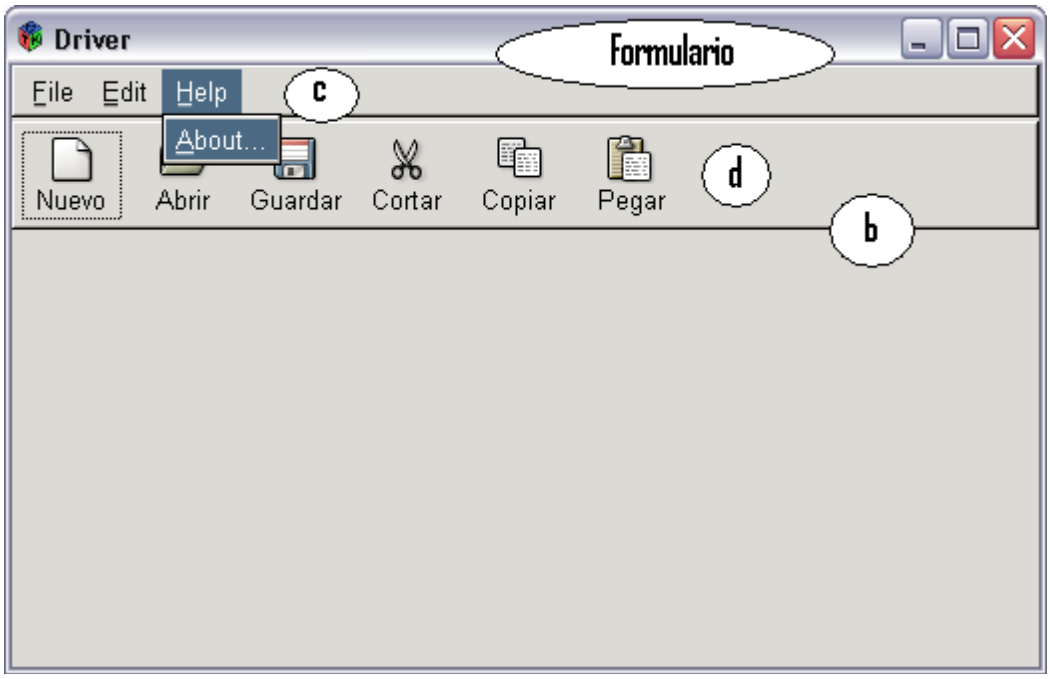
Algo a tener presente a la hora de desarrollar en plataforma cruzada, es no utilizar objetos que hacen gran uso del entorno gráfico propio de una plataforma: por ejemplo, un control grid funcionará bien bajo Windows pero al portarlo a Linux, generará errores, debido a que está diseñado para construir sus propios objetos mediante el GDI en Windows. Otra consideración importante es el juego de caracteres, en Windows funciona un juego pero al llevarlo a Linux se presentan problemas, o viceversa. Siempre que desarrolle una aplicación, lo mejor es contar con todos las plataformas objetivo y probar todas las partes del código para evitarnos trabajo extra de ajustes al código.

Carguemos el IDE de Visual Studio. Haga clic en Archivo, Nuevo, Proyecto. Observe que en la caja de diálogo de proyectos nuevos, aparecen dos nuevos componentes: glade# Project y Gtk# Project. Seleccionemos el segundo. Para los ejemplos de este documento, prepare una carpeta en el disco duro (c:\) llamada mono_samples. Aloje allí todos los proyectos que vamos a ir creando en los capítulos siguientes.



Deje el nombre propuesto por el entorno. Revisemos lo que ha hecho el entorno en el proyecto creado. Note que por ningún lado encontramos el clásico diseñador al que estábamos acostumbrados en los formularios *Winforms*. Ahora solo contamos con puro código para construir nuestra aplicación. Vamos a dar una vista del código resaltando los elementos nuevos y referentes al diseñador gráfico Gtk. En el código haremos referencia a las ventanas como formularios para no perder la costumbre de desarrollo Windows, aunque estemos manipulando widgets de Gtk.

Al ejecutar la aplicación se obtiene la siguiente presentación:



1. Ha agregado un llamado a la librería de Gtk en el código.

```
using Gtk;
```

2. La clase hereda de Gtk.Window. Esta herencia le da la posibilidad a nuestra aplicación de heredar todas las propiedades y métodos de una ventana.

```
class Driver : Gtk.Window
```

3. Ha definido una serie de objetos Gtk.

```
public string m_AppName = "Driver";

private VBox m_FrmPanel = null;
private VBox m_MenuAndToolBarPanel = null;
private MenuBar m_MainMenuBar = null;
private Toolbar m_MainToolbar = null;
private AccelGroup m_AccelGroup = null;
```

a) Ha definido una cadena de texto para el nombre de la solución y la ha bautizado “Driver”

```
public string m_AppName = "Driver";
```

b) Las siguientes dos definiciones crean dos objetos contenedores, que arreglan todos los objetos que se vayan agregando, en sentido vertical, uno debajo del otro. Un contenedor será para alojar todos los objetos componentes de la interfaz del formulario y el otro alojará al menýu y a la barra de herramientas.

```
private VBox m_FrmPanel = null;
private VBox m_MenuAndToolBarPanel = null;
```

c) La siguiente instrucción define un menú principal en el formulario.

```
private MenuBar m_MainMenuBar = null;
```

d) Luego se define una barra de herramientas para agregar al formulario.

```
private Toolbar m_MainToolbar = null;
```

e) La última instrucción define atajos (aceleradores de teclado) para las opciones del menú.

```
private AccelGroup m_AccelGroup = null;
```

4. Ha creado una sección de código llamada #region Wizard generated code. Dentro encontramos los métodos *InitializeComponents*() y dos secciones de código más llamadas *Menus* y Toolbar. Revisemos las instrucciones existentes en esos métodos.

```
#region Wizard generated code

        protected void InitializeComponents()
        {

                //crear una nueva instancia del acelerador de grupo global
                m_AccelGroup = new AccelGroup();

                // Definir las características de la ventana
                // Titulo de la ventana
                this.Title = m_AppName;
                // Tamaño de la ventana
                this.SetDefaultSize (400, 300);
                //Capturar el evento DeleteEvent (Cierre de la ventana) y ejecutar el método
                //onMyWindowDelete()
                this.DeleteEvent += new DeleteEventHandler (OnMyWindowDelete);
                //Agregar el acelerador o atajos de teclado
                this.AddAccelGroup(m_AccelGroup);

                // Contenedor vertical para almacenar todos los otros paneles
                m_FrmPanel = new VBox(false, 3);

                // Contenedor vertical para almacenar el menú principal y la barra de herramientas
                m_MenuAndToolBarPanel = new VBox(false, 2);

        // Esta region de código permite definir todo el sistema de menú en nuestra aplicación
        #region Menus

                // Configuración del menú principal
                m_MainMenuBar = new MenuBar();

                // File menu
                Menu FileMenu = new Menu();  // Nueva opción de menú principal
                MenuItem FileMenuItem = new MenuItem("_File");  // definir el item/opción en menu
                FileMenuItem.Submenu = FileMenu; // definir submenu
                FileMenu.AccelGroup = m_AccelGroup; // Programar atajos del teclado a las opciones
```

```
// Edit menu
Menu EditMenu = new Menu();
MenuItem EditMenuItem = new MenuItem("_Edit");
EditMenuItem.Submenu = EditMenu;
EditMenu.AccelGroup = m_AccelGroup;

// Help menu
Menu HelpMenu = new Menu();
MenuItem HelpMenuItem = new MenuItem("_Help");
HelpMenuItem.Submenu = HelpMenu;
HelpMenu.AccelGroup = m_AccelGroup;

// File New menu item. Se agrega una imagen a la opción.
Gtk.ImageMenuItem NewMenuItem = new ImageMenuItem("gtk-new", m_AccelGroup);
// se agrega un evento a la opción para detectar cuando el usuario activa la opción, se ejecutara el
// metodo NewMenuItem_OnActivate
NewMenuItem.Activated += new EventHandler(NewMenuItem_OnActivate);
FileMenu.Append(NewMenuItem);

// File Open menu item
Gtk.ImageMenuItem OpenMenuItem = new ImageMenuItem("gtk-open", m_AccelGroup);
OpenMenuItem.Activated += new EventHandler(OpenMenuItem_OnActivate);
FileMenu.Append(OpenMenuItem);

// File Save menu item
Gtk.ImageMenuItem SaveMenuItem = new ImageMenuItem("gtk-save", m_AccelGroup);
SaveMenuItem.Activated += new EventHandler(SaveMenuItem_OnActivate);
FileMenu.Append(SaveMenuItem);

// File SaveAs menu item
Gtk.ImageMenuItem SaveAsMenuItem = new ImageMenuItem("gtk-save-as", m_AccelGroup);
SaveAsMenuItem.Activated += new EventHandler(SaveAsMenuItem_OnActivate);
FileMenu.Append(SaveAsMenuItem);

// Se agrega u separador al menu File
SeparatorMenuItem Separator1MenuItem = new SeparatorMenuItem();
FileMenu.Append(Separator1MenuItem);

// File Exit menu item
Gtk.ImageMenuItem ExitMenuItem = new ImageMenuItem("gtk-quit", m_AccelGroup);
ExitMenuItem.Activated += new EventHandler(ExitMenuItem_OnActivate);
FileMenu.Append(ExitMenuItem);

// Edit Cut menu item
Gtk.ImageMenuItem CutMenuItem = new ImageMenuItem("gtk-cut", m_AccelGroup);
CutMenuItem.Activated += new EventHandler(CutMenuItem_OnActivate);
EditMenu.Append(CutMenuItem);

// Edit Copy menu item
Gtk.ImageMenuItem CopyMenuItem = new ImageMenuItem("gtk-copy", m_AccelGroup);
CopyMenuItem.Activated += new EventHandler(CopyMenuItem_OnActivate);
EditMenu.Append(CopyMenuItem);
```

```
            // Edit Paste menu item
            Gtk.ImageMenuItem PasteMenuItem = new ImageMenuItem("gtk-paste", m_AccelGroup);
            PasteMenuItem.Activated += new EventHandler(PasteMenuItem_OnActivate);
            EditMenu.Append(PasteMenuItem);

            // Help About menu item
            MenuItem AboutMenuItem = new MenuItem("_About...");
            AboutMenuItem.Activated += new EventHandler(AboutMenuItem_OnActivate);
            HelpMenu.Append(AboutMenuItem);

            // agregar los submenus al menu principal
            m_MainMenuBar.Append(FileMenuItem);
            m_MainMenuBar.Append(EditMenuItem);
            m_MainMenuBar.Append(HelpMenuItem);

        #endregion

        // Esta región permite definir la barra de herramientas del formulario
        #region Toolbar

            m_MainToolbar = new Toolbar();

            // Botones de la barra de herramientas
            // Se define el botón button1 y se le programa la imagen gtk-new del inventario
            // de iconos (stock) disponibles dentro del Gtk
            Gtk.ToolButton button1 = new ToolButton("gtk-new");
            // Se programa el método button1_Clicked para capturar el evento clic del botón
            button1.Clicked +=new EventHandler(button1_Clicked);

            Gtk.ToolButton button2 = new ToolButton("gtk-open");
            button2.Clicked +=new EventHandler(button2_Clicked);

            Gtk.ToolButton button3 = new ToolButton("gtk-save");
            button3.Clicked +=new EventHandler(button3_Clicked);

            Gtk.ToolButton button4 = new ToolButton("gtk-cut");
            button4.Clicked +=new EventHandler(button4_Clicked);

            Gtk.ToolButton button5 = new ToolButton("gtk-copy");
            button5.Clicked +=new EventHandler(button5_Clicked);

            Gtk.ToolButton button6 = new ToolButton("gtk-paste");
            button5.Clicked +=new EventHandler(button6_Clicked);

            // Agregar los botones a la barra de herramientas
            // Botones para operación de archivos
            m_MainToolbar.Add(button1);
            m_MainToolbar.Add(button2);
            m_MainToolbar.Add(button3);
            // Insertar un separador de botones
            // m_MainToolbar.AppendSpace();
            // Agregar los botones de edición
            m_MainToolbar.Add(button4);
            m_MainToolbar.Add(button5);
            m_MainToolbar.Add(button6);

        #endregion

            // Agregar el menú principal al panel contenedor. Se empaqueta para que llene de forma
```

```
                    // automática (fill)  ni se expanda cuando se cambie detamaño a la ventana.
                    m_MenuAndToolBarPanel.PackStart(m_MainMenuBar, false, false, 0);

                    // Agregar la barra de herramientas al contenedor
                    m_MenuAndToolBarPanel.PackStart(m_MainToolbar, false, false, 0);


                    // Agregar el panel al formulario principal
                    m_FrmPanel.PackStart (m_MenuAndToolBarPanel, false, false, 0);

                    // Agregar el contendor principal con todos los objetos a la ventana de la aplicación
                    this.Add(m_FrmPanel);
    }

    #endregion
```

Parece mucho código complejo, pero si revisa, son instrucciones repetitivas que definen opciones de menú y botones en la barra de herramientas.

5. Aparece un método llamado Driver que es la parte que iniciará toda la interfaz gráfica cuando se ejecute la aplicación. Note que llama a la rutina *InitializeComponents*() y luego invoca al método ShowAll del objeto this, que en este caso es la ventana de la aplicación.

```
public Driver () : base ("Driver")
{
        InitializeComponents();
        this.ShowAll ();
}
```

6. La siguiente rutina es para capturar el evento de cierre de ventana. Observe que se reciben los parámetros *object* o y *DeleteEventArgs* args. Este tipo de declaración es común cuando definimos métodos para atrapar eventos en la interfaz GUI.

```
private void OnMyWindowDelete (object o, DeleteEventArgs args)
{
        // Cerrar la aplicación
        Application.Quit ();
        args.RetVal = true;
}
```

7. Aparece una sección de código llamada menu handlers. Los métodos que aparecen en esta sección de código, permiten atrapar todos los eventos de clic sobre las opciones del menú. Algunos se han dejado en blanco para efectos de plantilla de código.

```
#region Menu handlers

// Rutina a ejecutar cuando hagan clic en la opción Nuevo
private void NewMenuItem_OnActivate(object o, EventArgs args)
{
}
// Rutina a ejecutar cuando hagan clic en la opción Abrir
private void OpenMenuItem_OnActivate(object o, EventArgs args)
{
        // Se invoca al widget FileSelection, que muestra una caja de diálogo de seleccionar archivos
        FileSelection fDlg = new FileSelection("Choose a file");
        int nRc = fDlg.Run();
        fDlg.Hide();
```

```
            if(nRc == (int)ResponseType.Ok) // ResponseType.Ok es un tipo de respuesta definido en el Gtk
            {
            }
}

// Rutina a ejecutar cuando hagan clic en la opción Guardar
private void SaveMenuItem_OnActivate(object o, EventArgs args)
{
}
// Rutina a ejecutar cuando hagan clic en la opción Guardar Como
private void SaveAsMenuItem_OnActivate(object o, EventArgs args)
{
            FileSelection fDlg = new FileSelection("Choose a file");

            int nRc = fDlg.Run();
            fDlg.Hide();

            if(nRc == (int)ResponseType.Ok)
            {
            }
}
// Rutina a ejecutar cuando hagan clic en la opción Salir
private void ExitMenuItem_OnActivate(object o, EventArgs args)
{
            Application.Quit();
}
// Rutina a ejecutar cuando hagan clic en la opción Cortar
private void CutMenuItem_OnActivate(object o, EventArgs args)
{
}
// Rutina a ejecutar cuando hagan clic en la opción Copiar
private void CopyMenuItem_OnActivate(object o, EventArgs args)
{
}
// Rutina a ejecutar cuando hagan clic en la opción Pegar
private void PasteMenuItem_OnActivate(object o, EventArgs args)
{
}
// Rutina a ejecutar cuando hagan clic en la opción Acerca de
private void AboutMenuItem_OnActivate(object o, EventArgs args)
{
            // Este segmento de código define una caja de diálogo para mostrar informacion
            // acerca de la aplicación
            System.Text.StringBuilder AuthorStringBuild = new System.Text.StringBuilder ();
            String []authors = new String[] {"Your Name <your.name@somedomain.com>"};
            AuthorStringBuild.Append ("Driver version 1.0\n\n");
            AuthorStringBuild.Append ("Brief description or your App.\n");
            AuthorStringBuild.Append ("Copyright (c) 2004\n\n");
            AuthorStringBuild.AppendFormat ("Authors:\n\t{0}\n\t",authors[0]);

            MessageDialog md = new MessageDialog(  this, DialogFlags.DestroyWithParent,
                                  MessageType.Info, ButtonsType.Ok, AuthorStringBuild.ToString ());
            md.Modal = true; // Ejecutar en forma modal
            int result = md.Run (); // Ejecutar la caja de diálogo y atrapar el resultado en result
            md.Hide(); // Ocultar la caja de diálogo
            return;
}

#endregion
```

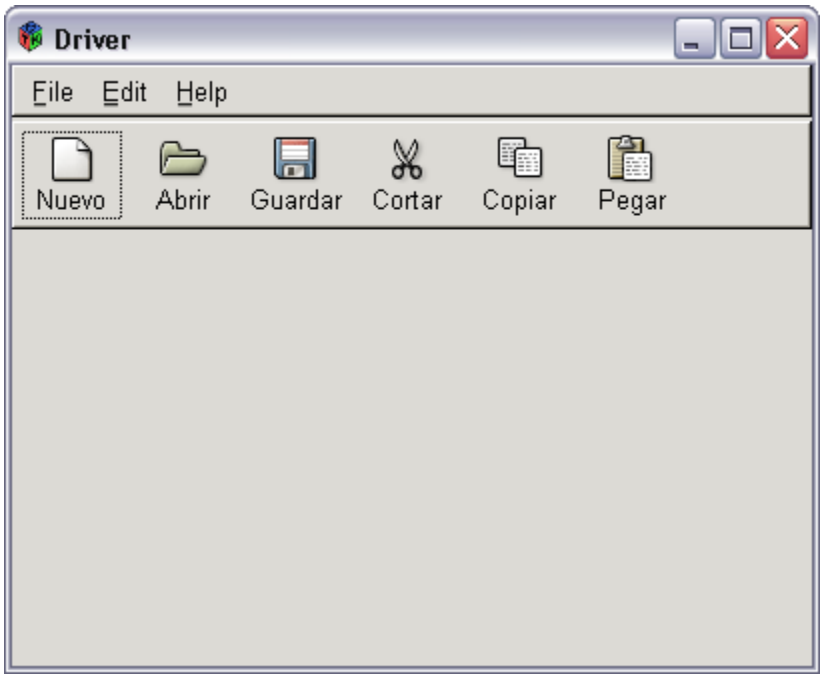
8. La una sección de código llamada #region *Toolbar buttons handlers* permite definir las rutinas asociadas a los botones de la barra de herramientas cuando el usuario haga clic sobre ellas.

```
#region Toolbar buttons handlers

        // Evento clic en File / New
        private void button1_Clicked(object sender, EventArgs e)
        {
                return;
        }
        // Evento clic en File / Open
        private void button2_Clicked(object sender, EventArgs e)
        {
                FileSelection fDlg = new FileSelection("Choose a file");
                int nRc = fDlg.Run();
                fDlg.Hide();
                if(nRc == (int)ResponseType.Ok)
                {
                }
        }
        // Evento clic en File / Save
        private void button3_Clicked(object sender, EventArgs e)
        {
                return;
        }
        // Evento clic en Edit / Cut
        private void button4_Clicked(object sender, EventArgs e)
        {
                return;
        }
        // Evento clic en Edit / Copy
        private void button5_Clicked(object sender, EventArgs e)
        {
                return;
        }
        // Evento clic en Edit / Paste
        private void button6_Clicked(object sender, EventArgs e)
        {
                return;
        }

        #endregion
```

9. La sección main contiene el código necesario para hacer que la interfaz gráfica entre en un ciclo y espere a eventos producidos por el usuario sobre la inetrfaz gráfica.

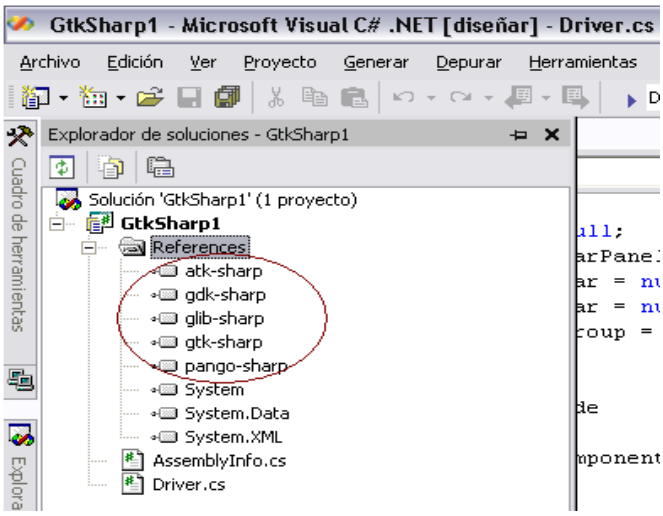
```
[STAThread]
static void Main(string[] args) {
        Application.Init();  // Inicializar el sistema
        new Driver();  // Cargar nuestra GUI
        Application.Run(); // Ejecutar la GUI y esperar a eventos desde el administrador gráfico Gtk
}
```

Cuando la ejecute verá la imagen siguiente en la pantalla.



Observe una de las ventajas que tiene el Gtk cuando se invocan algunas de sus funcionalidades dentro de nuestro código. El código está en inglés, pero los botones de la barra de herramientas fueron creados llamando al inventario de imágenes y botones del Gtk. Cuando ejecuta en mi estación de desarrollo, automáticamente tomó el idioma español para esos botones. Pero el menú, siguió en inglés.

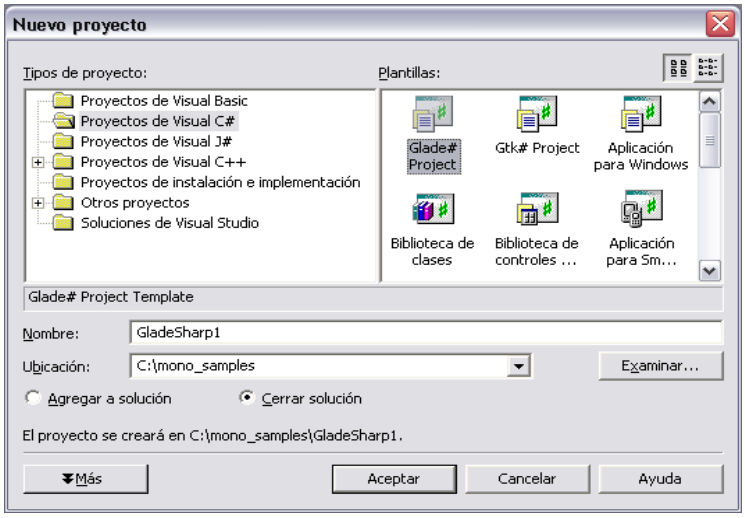
Por último, observe en las referencias del proyecto aparecen referenciadas las librerías atk, gdk, glib, gtk y pango. Todas son necesarias para el desarrollo de interfaces gráficas de multiplataforma. Una gran ventaja es la de poder contar con las mismas librerías en una plataforma diferente a la de Window.



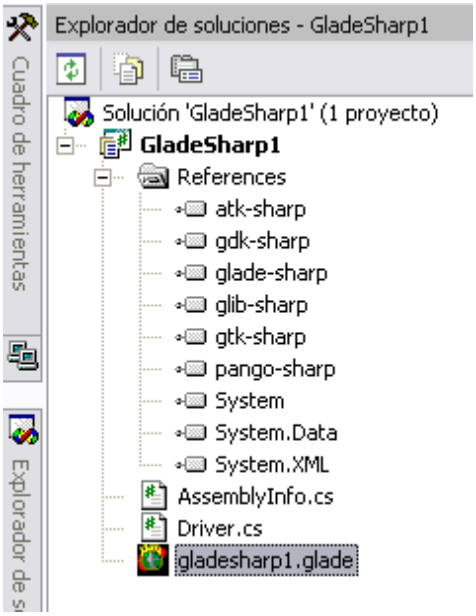
Ahora todo lo que necesita es ir al refrigerador y destapar la segunda cerveza. Luego, experimente con el código y haga que todos los mensajes salgan en español. Trate de agregar nuevos botones y opciones al menú.

2.2. Nuestro segundo proyecto Gtk utilizando Glade

Ahora vamos a empezar un nuevo proyecto pero esta vez utilizaremos Glade como diseñador de la interfaz GUI. Haga clic en Archivo, Nuevo, Proyecto. En la caja de diálogo de Nuevo proyecto, seleccione Glade# Project. Deje el botón de radio de Cerrar Solución activado. Deje además el nombre que propone el IDE de Visual Studio.



Ahora el entorno ha de mostrarnos una ventana con una plantilla de código de la aplicación que utiliza Glade. Note que aparece un archivo dentro del explorador de soluciones llamado *gladesharp1.glade*. Esta es la referencia que necesita el Gtk# para integrar la interfaz diseñada con Glade a nuestra aplicación en el momento de la compilación. Note además que siguen las mismas librerías de gtk referenciadas en el proyecto.



Esta aplicación en teoría es la misma que se creo en el numeral anterior, pero la gran diferencia reside en que incluye menos código, debido a que la definición de objetos se la dejamos al Glade y al Gtk#.

Revisemos ahora la plantilla de código generado en el IDE.

1. Aparece una invocación al espacio de nombres de Gtk y Glade.

```
using Gtk;
using Glade;
```

2. La región de código denominada *Glade Widget* contiene una instrucción que hace disponible el widget ventana (window) dentro de nuestro código. A partir de esa instrucción, nuestro código puede alcanzar los métodos, propiedades y eventos de Gtk.Window.

```
#region Glade Widgets
        [Widget] Gtk.Window window1;
#endregion
```

3. El método main solo incluye una invocación al método Driver, que es el que lanzará toda la interfaz GUI de nuestra aplicación.

```
static void Main(string[] args)
{
        new Driver(args);
}
```

4. El método Driver incluye el código necesario para inicializar el entorno gráfico Gtk, leer las definiciones de diseño desarrollado en Glade y capturar el objeto window1 que contiene nuestra interfaz GUI. La invocación al método Autoconnect(this) conecta nuestro código con las definiciones dentro del archivo Glade.

```
public Driver(string[] args)
        {
           Application.Init();  // Inicializar el entorno gráfico Gtk
          // Cargar una referencia al XML definido por Glade, con información sobre nuestra GUI
           Glade.XML gxml = new Glade.XML (null, "gladesharp1.glade", "window1", null);
           // Conectar la interfaz a nuestro código
           gxml.Autoconnect (this);
           // Ejecutar el sistema
           Application.Run();
        }
```

5. La rutina *on_window1_delete_event* captura el evento cerrar ventana diseñado en Glade. Luego aprenderemos como se hace esta programación. La única función de la rutina es cerrar la aplicación.

```
        // Conectar a  señales definidas en Glade
        public void on_window1_delete_event (object o, DeleteEventArgs args)
        {
           Application.Quit();
           args.RetVal = true;
        }
```

6. La región de código denominada Button Click Events handlers incluyen las rutinas conectadas a las señales de Glade. Su función es capturar los diversos eventos que se indicaron durante el diseño en Glade.

```
#region Button Click Event handlers
        // barra de tareas / botón toolbutton1 / evento click
        protected void on_toolbutton1_clicked(object o, EventArgs args)
        {
                return;
        }
        // barra de tareas / botón toolbutton2 / evento click. En este caso, se hace una programación
        // de seleccionar un archivo del sistema de ficheros, mediante una invocación al widget FileSelection
        protected void on_toolbutton2_clicked(object o, EventArgs args)
        {
                FileSelection fDlg = new FileSelection("Choose a File");
                fDlg.Modal = true;

                int nRc = fDlg.Run();
                fDlg.Hide();

                if(nRc == (int)ResponseType.Ok)
                {
                }
                return;
        }
        // Barra de tareas / botón toolbutton2 / evento click. Otra forma de salir de la aplicación.
        protected void on_toolbutton3_clicked(object o, EventArgs args)
        {
                Application.Quit();
                return;
        }
        #endregion
```

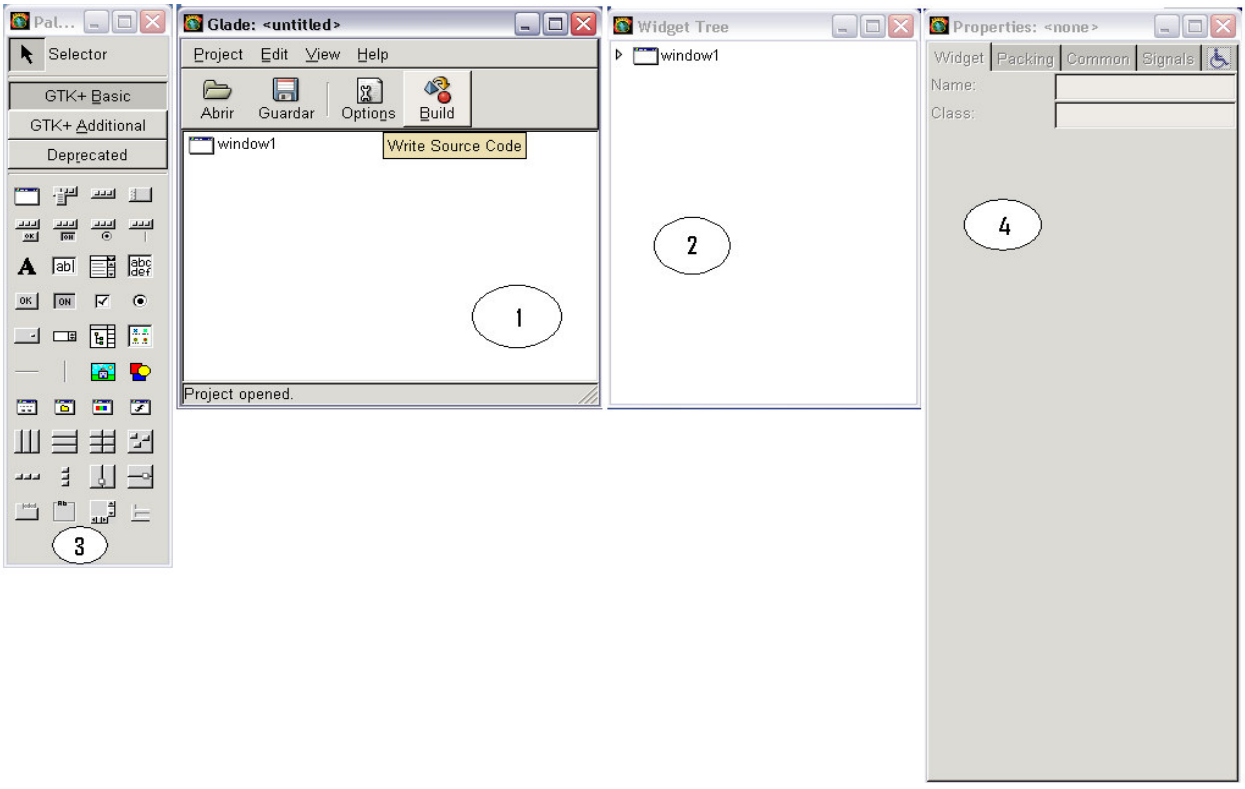
7. La región de código llamada Menu item handlers, incluye las rutinas para capturar las señales asignadas en Glade para el sistema de menú.

```
#region Menu item handlers
        // evento clic en opción new1
        protected void on_new1_activate(object o, EventArgs args)
        {
                return;
        }
        // evento clic en opción open1
        protected void on_open1_activate(object o, EventArgs args)
        {
                FileSelection fDlg = new FileSelection("Choose a File");
                fDlg.Modal = true;
                int nRc = fDlg.Run();
                fDlg.Hide();
                if(nRc == (int)ResponseType.Ok)
                {
                }
                return;
        }
```

```
// evento clic en opción save1
protected void on_save1_activate(object o, EventArgs args)
{
	return;
}
// evento clic en opción save_as1
protected void on_save_as1_activate(object o, EventArgs args)
{
	return;
}
// evento clic en opción quit1
protected void on_quit1_activate(object o, EventArgs args)
{
	Application.Quit();
	return;
}
// evento clic en opción cut1
protected void on_cut1_activate(object o, EventArgs args)
{
	return;
}
// evento clic en opción copy1
protected void on_copy1_activate(object o, EventArgs args)
{
	return;
}
// evento clic en opción delete1
protected void on_delete1_activate(object o, EventArgs args)
{
	return;
}
// evento clic en opción paste1
protected void on_paste1_activate(object o, EventArgs args)
{
	return;
}
// evento clic en opción about1
protected void on_about1_activate(object o, EventArgs args)
{
	System.Text.StringBuilder AuthorStringBuild = new System.Text.StringBuilder ();

	AuthorStringBuild.Append ("gladesharp1 version 1.0\n\n");
	AuthorStringBuild.Append ("Sample Glade Application.\n");
	AuthorStringBuild.Append ("Copyright (c) 2004\n\n");

	Gtk.MessageDialog md = new Gtk.MessageDialog (
		this.window1,
		DialogFlags.DestroyWithParent,
		MessageType.Info,
		ButtonsType.Ok,
		AuthorStringBuild.ToString ()
		);
	int result = md.Run ();
	md.Hide();
	return;
}
#endregion
```

Al ejecutar la aplicación, obtenemos una ventana similar a la siguiente:

Ahora, demos un rápido vistazo al glade y al diseño que de forma automática incluyó en nuestra GUI. Para abrir el diseñador de GUI Glade, haga doble clic en el objeto *gladesharp1.glade* en el *explorador de soluciones*. No se deje intimidar por la cantidad de ventanas flotantes que pueda llegar a mostrar glade. Es algo a lo que debemos acostumbrarnos en el mundo de Linux. Con el tiempo se acostumbrará a este tipo de entorno gráfico y le agarrará cariño al diseñador Glade.



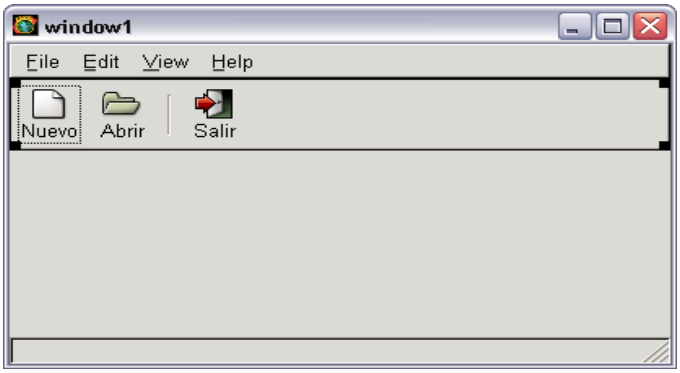
En la gráfica anterior, la ventana marcada con 1 es el explorador del proyecto Glade, e incluye todos los objetos que hemos diseñado en Glade para un proyecto. En este caso, solo aparece Window1.

El área marcada con el 2 es el árbol de los widgtes que tenemos definidos en todos los objetos dentro del proyecto Glade.

El área 3 es la barra de herramientas que incluye los widgest disponibles para nuestros diseños de la GUI.

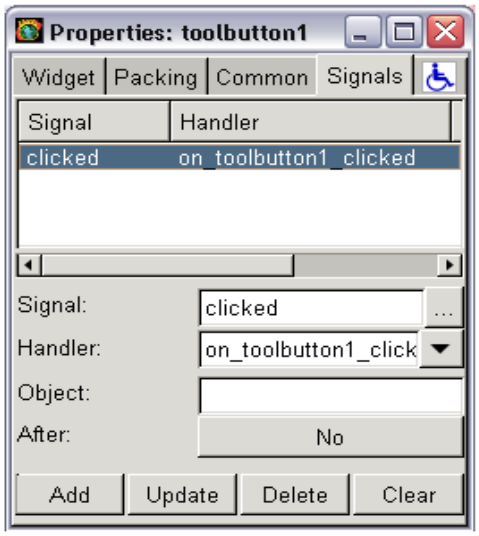
El área 4 muestra las propiedades del objeto seleccionado en el entorno.

Revisemos ahora el diseño que se hizo de forma automática el proyecto Glade. En el área de objetos del proyecto Glade, haga doble clic en el objeto windo1. Debe aparece el diseñador de los objetos GUI.



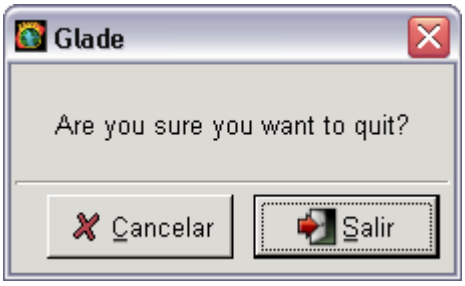
Observe que al ir seleccionado objetos, el explorador de widgets (widget tree) va posicionándose en el objeto definido. Esto será muy útil para ubicar objetos cuando se cuente con una interfaz gráfica con muchos elementos. Note además que la ventana de propiedades va mostrando las propiedades disponibles para cada widget.

Ahora bien, donde le informaron al Glade que eventos y que rutinas tenía que capturar y asociar de forma automática a nuestro código?. Para averiguar esto, seleccione por ejemplo el botón Nuevo en el diseñador Glade. En la ventana de propiedades, haga clic en señales (Signal). Note que ahí aparece una lista con el evento y la rutina que debe existir dentro de nuestro código.

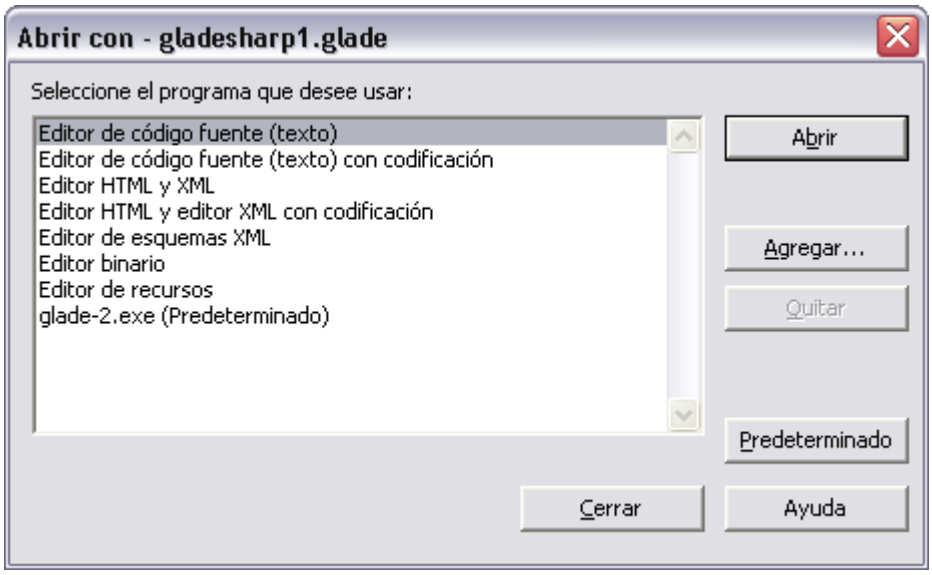


Ahora, desplácese hasta el refrigerador y traiga la tercera cerveza para completar nuestra tercera actividad. Explore las propiedades y widgets disponibles del Glade.

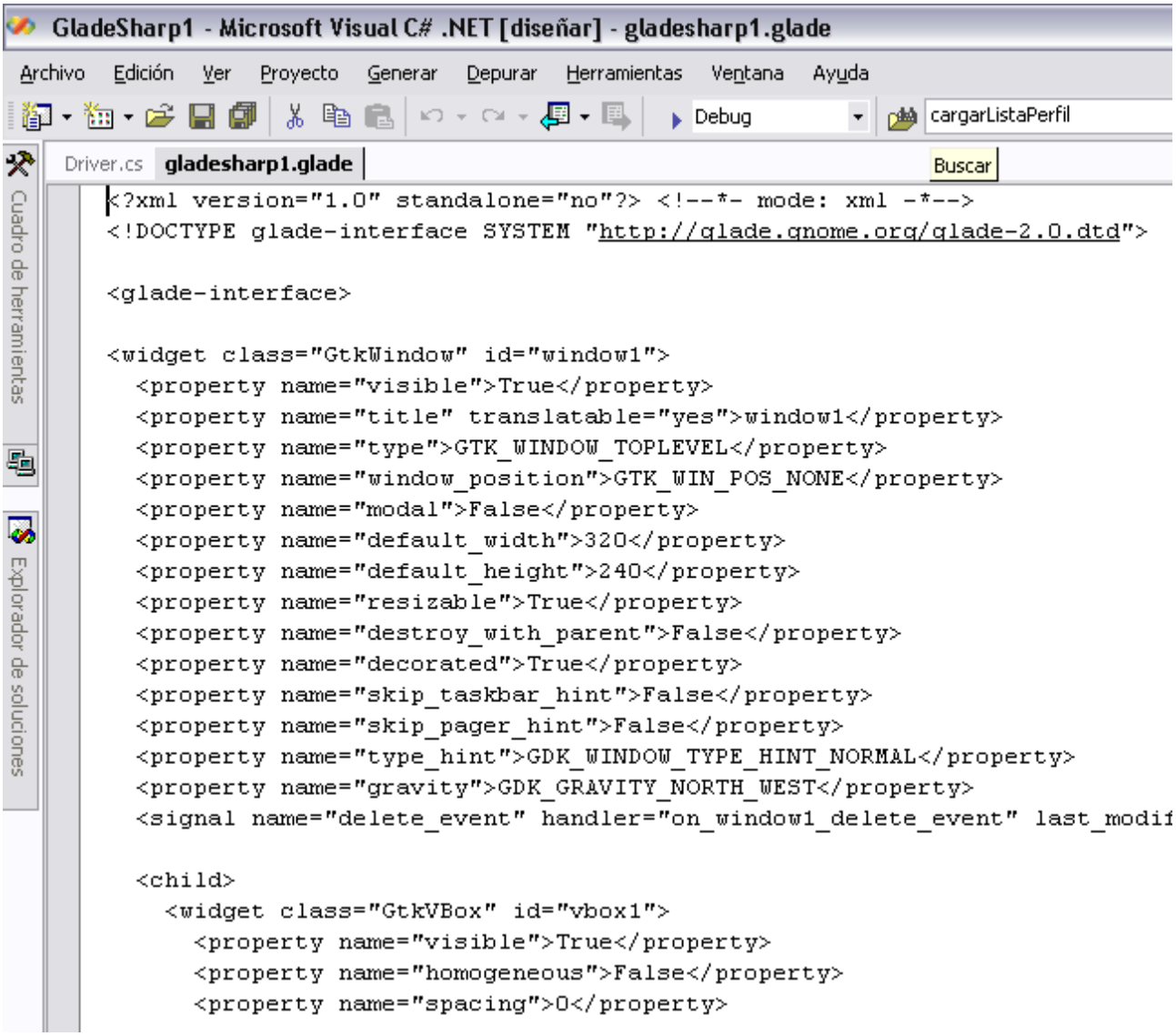
Para salir del Glade haga clic en el botón cerrar de la ventana del proyecto. Glade preguntará si desea salir. Haga clic en el botón Salir. Recuerde guardar los proyectos antes de salir del diseñador.



Por último, demos una rápida mirada al archivo xml que genera Glade. Para lograr esto, en el explorador de soluciones del IDE Visual Studio .Net, haga clic derecho y seleccione *abrir con* y seleccione Editor de código fuente (texto) y clic en el botón [*Abrir*].



El editor cargará el archivo xml con la definición que hace Glade de nuestro proyecto GUI. Realmente no necesitamos saber nada acerca de este contenido, y dejemos que sea Glade el que lo utilice. En los libros que reverenció en la bibliografía, enseñan apartes del contenido de este archivo, pero particularmente, con que nos sirva para la definición y compilación de nuestra interfaz gráfica, es más que suficiente.



Agggg!!! dejemos que los expertos y estudiosos de diseñadores de interfaces se hagan cargo de entender este código. Nosotros, nos dedicaremos a explotar las herramientas que ellos de buena fe nos construyen. Y que reciban nuestro más sincero agradecimiento por darnos esas herramientas.

2.3. Resumen

En este capítulo dimos un rápido vistazo a la técnica de desarrollo empleando la herramienta de Microsoft Visual Studio .Net. Revisamos un proyecto Gtk a puro código y otro proyecto Gtk que utiliza Glade como facilitador del diseño de la GUI.

Una desventaja de Glade es que no tiene reversa (deshacer) para muchas de las acciones que hagamos. Así que debemos ser muy cuidadosos a la hora de diseñar nuestro entorno. Lo que hago en mi caso, es grabar constantemente los cambios que voy ejecutando sobre la GUI y cuando daño algo, salgo de Glade sin guardar los cambios y lo vuelvo a cargar. Tedioso pero funciona.

Al principio cuesta un poco acostumbrarse a este tipo de situación, pero les garantizo que es mucho más práctico y rápido que desarrollar aplicaciones Gtk a puro código. La experiencia les enseñará que tener combinación de los dos mundos (Código y Glade) es una buena técnica, debido a que muchas operaciones con widgets más avanzados no las podemos hacer desde el Glade.