Desarrollo de aplicaciones con Gambas.

Tutorial y ejemplo de un programa hecho con Gambas: *Agenda de Notas*

Sumario: Vamos a crear una aplicación sencilla con Gambas. Veremos cómo se programan los eventos y algunos trucos y técnicas de trabajo con este magnífico entorno de desarrollo.

Auto Originalr: David Asorey Álvarez. Febrero de 2005.

Revision por Julio Sánchez, Octube 2010. Adaptado a la versión Gambas2.21

Introducción

Gambas es una herramienta de desarrollo visual de aplicaciones muy similar a los conocidos programas comerciales Microsoft Visual Basic o Borland Delphi.

Con Gambas se pueden hacer aplicaciones o programas con interfaz gráfica de forma muy rápida, pues integran un diseñador de formularios o ventanas, un editor de código, un explorador de clases, un visor de ayuda, etc.

Este tipo de herramientas han sido siempre muy habituales en la plataforma Microsoft Windows, pero para Linux no existían tantas, o bien no estaban tan depuradas. Podemos encontrar Kdevelop, Kylix o VDK Builder. Hay que destacar que en el desarrollo de aplicaciones en Linux hay una larga tradición y costumbre de emplear muchas herramientas diferentes, cada una especializada en una tarea en concreto (p. ej., un compilador, un editor, un depurador, cada uno por separado), por lo que este tipo de herramientas integradas (IDE) no han aparecido hasta hace poco.

Existe un grupo de programadores y desarrolladores que están acostumbrados a estas herramientas integradas, ya sea porque suelen trabajar con ellas en otras plataformas o porque les resulta más cómodo o fácil.

Gambas es una herramienta, que, en palabras de su autor, Benoît Minisini, permite la creación de programas potentes, de forma fácil y sencilla. El lenguaje de programación que se utiliza es una versión del "viejo" BASIC. Puede sorprender que se haya escogido un lenguaje tan básico e incluso limitado como es el BASIC, pero no hay que olvidar que uno de los objetivos de la herramienta es acercar el desarrollo de aplicaciones a personas no expertas en la programación.

El objetivo de este tutorial es presentar un poco por encima la herramienta, pero vamos a presuponer que el lector ya sabe programar un poco, y que términos como *función*, *evento*, *variable* y similares le son familiares. Hay excelentes tutoriales disponibles en Internet (ver nota 1), y el propio programa incorpora un navegador de documentación bastante completo.

La versión de Gambas utilizada al redactar este tutorial es la 2.21.

La página web de Gambas está en http://gambas.sourceforge.net

Puedes encontrar documentación en varios idiomas en http://gambasdoc.org/help/

Descargar el programa de ejemplo: descarga del programa completo agenda de notas
Este tutorial en pdf: dirección de descarga del manual en pdf.

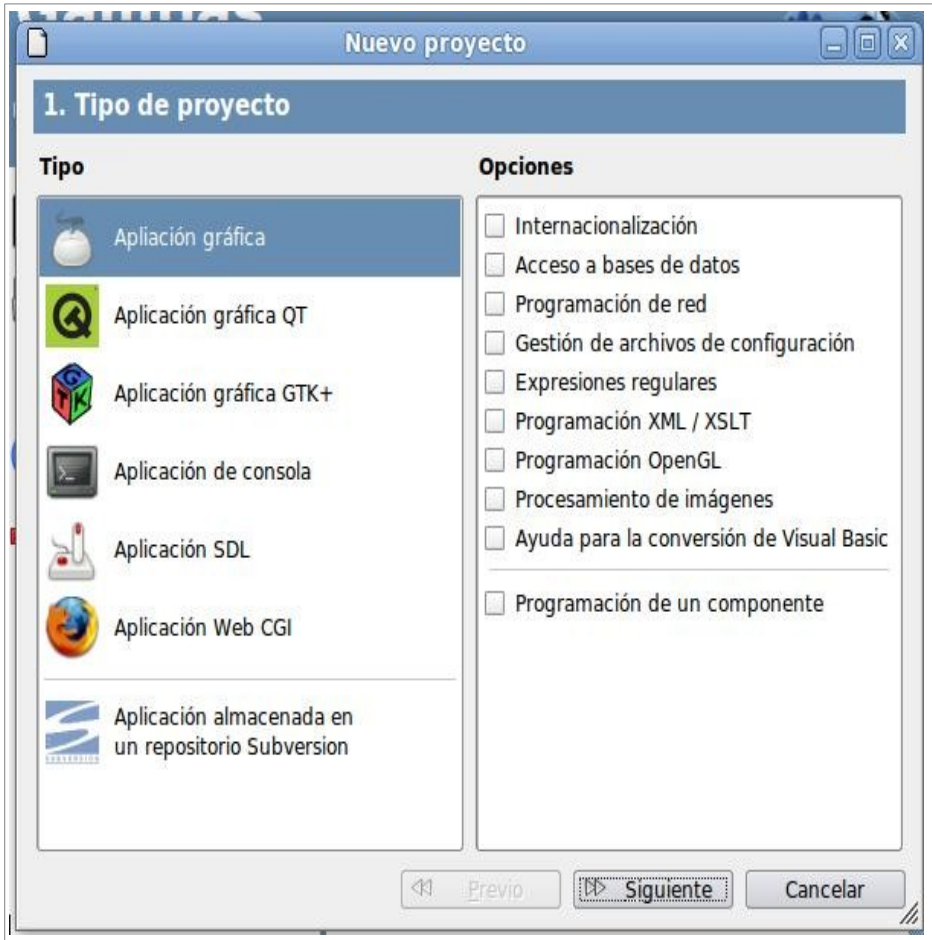
Primeros pasos

Para no repetirnos y aportar algo más a lo que ya hay escrito, no vamos a entrar en cómo es el entorno de desarrollo, ni para qué sirve cada herramienta, etc. En la propia documentación de Gambas vienen algunos tutoriales introductorios y un apartado llamado "Visual Introduction to Gambas".

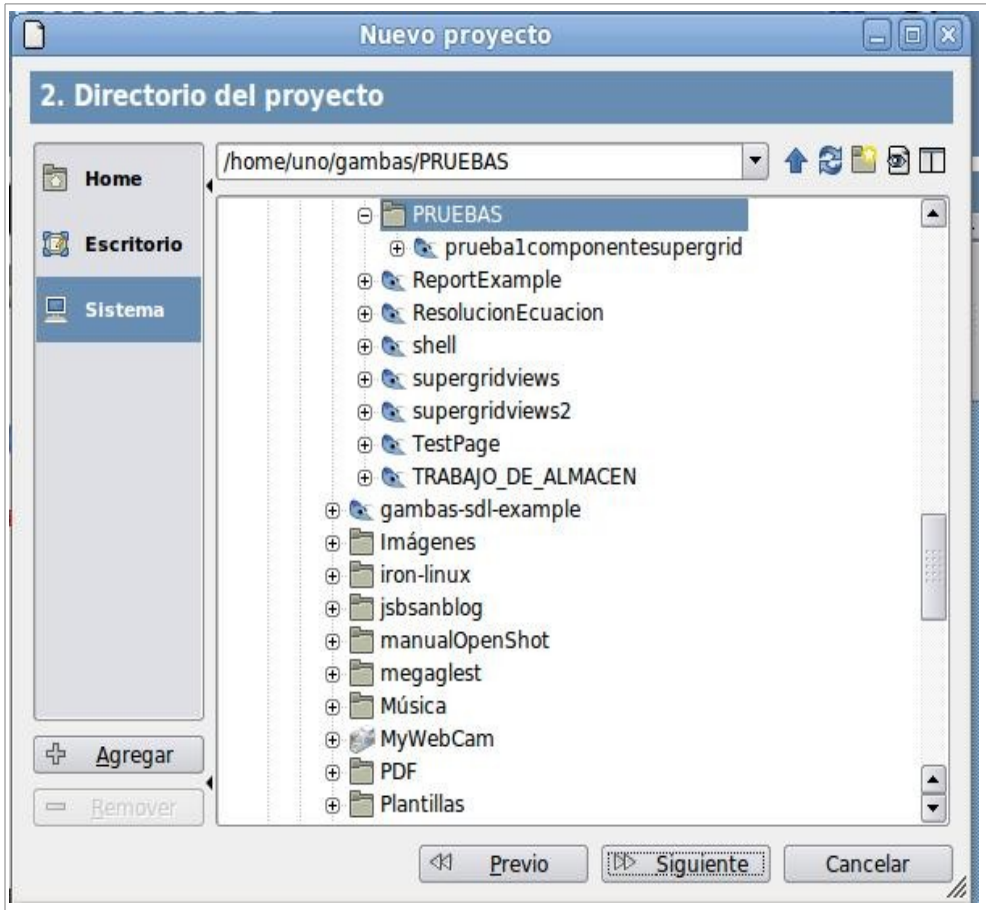
En este tutorial intentaremos hacer un programa completo y funcional desde el principio, y solucionaremos las necesidades según vayan surgiendo.

Vamos a crear un programa que sea una especie de cuaderno o agenda para tomar notas. Se podrán añadir o borrar notas, además de modificar las existentes. En cualquier momento se pueden guardar las notas a un fichero o recuperar otras de un fichero.

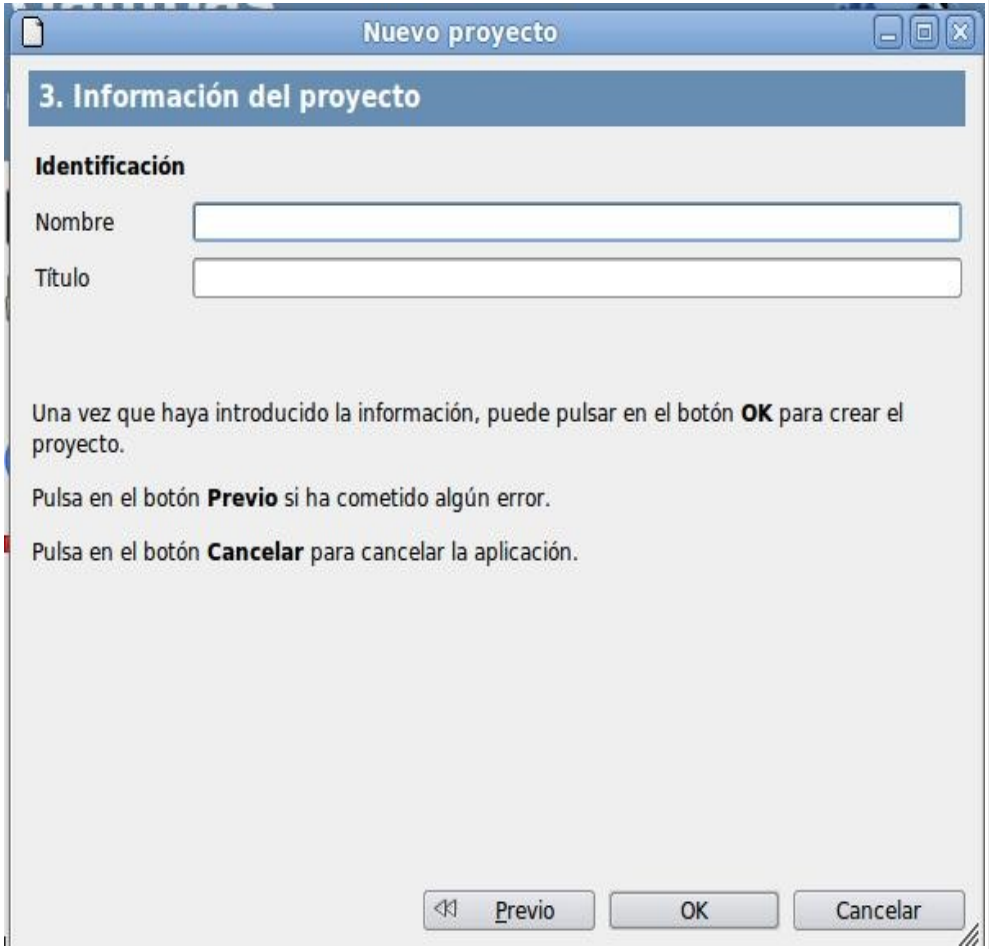
En Gambas, seleccionamos la opción "Nuevo proyecto". Seleccionamos crear un proyecto gráfico y el programa nos pide algunos datos como el nombre y título del proyecto:



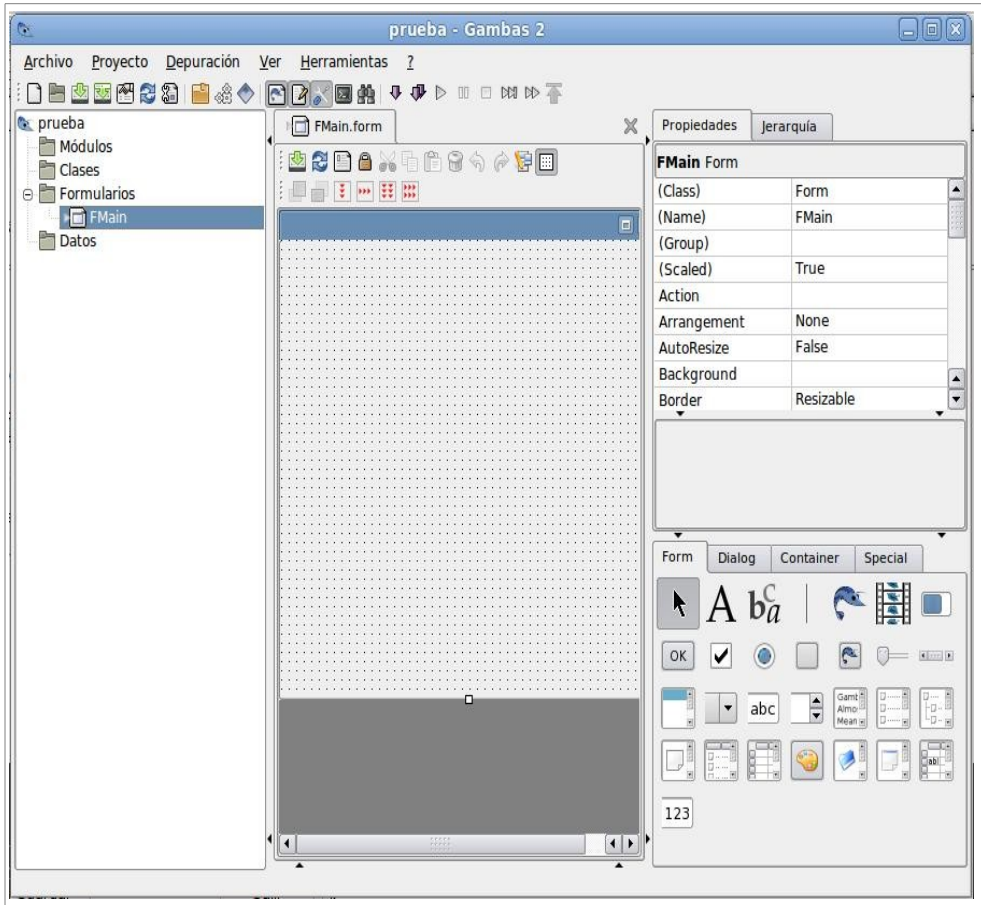
Si pulsamos en Internacionalización, nos permite que podamos traducir el programa a otros idiomas



Elegimos la carpeta donde guardar el proyecto.

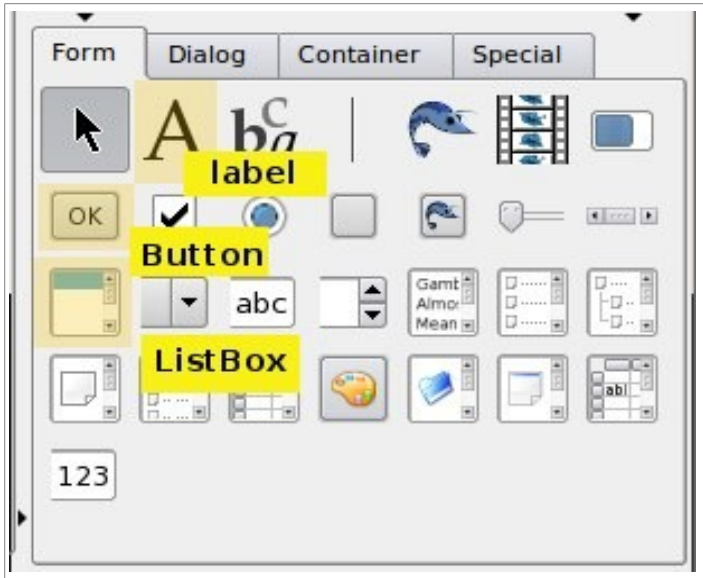


Escribimos el nombre y el titulo del programa.

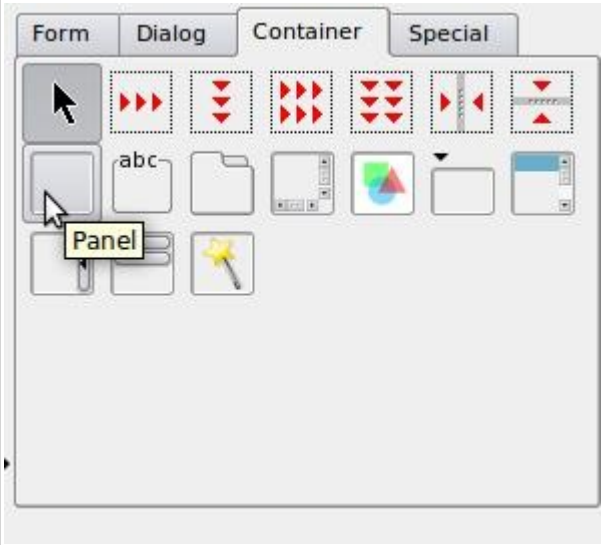


Con el botón derecho pulsamos en el apartado "Formularios", y vemos el que crea automáticamente el programa: Fmain

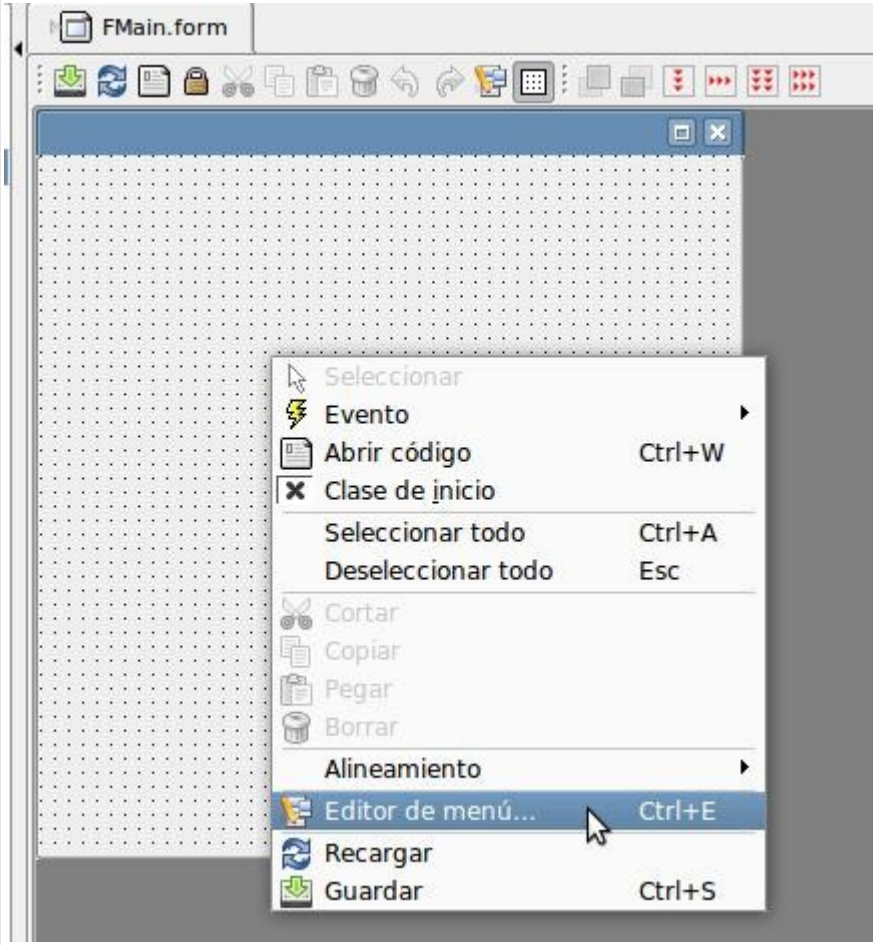
Vamos a diseñar el formulario con un "ListBox" y varios botones para añadir, modificar, borrar, etc.



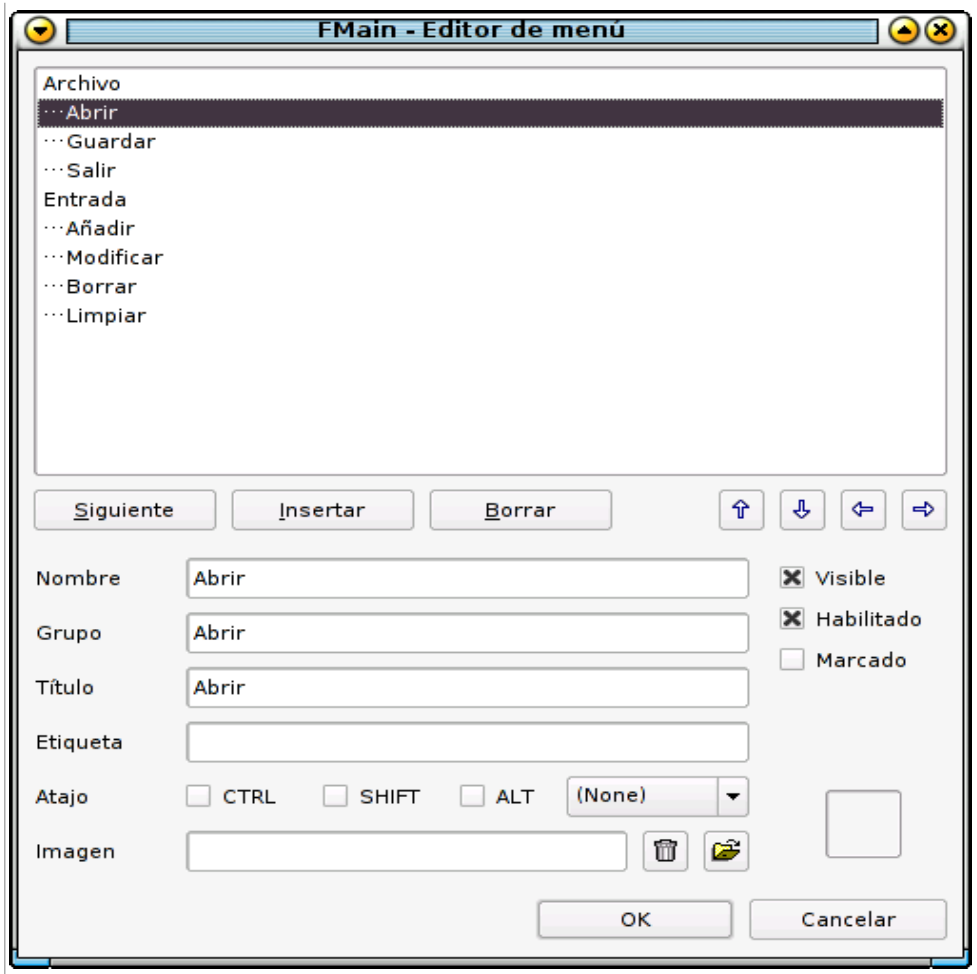
Tenemos un "Label", un "ListBox" y varios botones, que se insertan en el formulario seleccionándolos en la caja de herramientas y "dibujándolos" sobre el formulario.



Lo más destacable en este caso son los botones "Abrir", "Guardar" y "Salir", que los hemos situado encima de un "Panel" en vez de sobre el formulario directamente



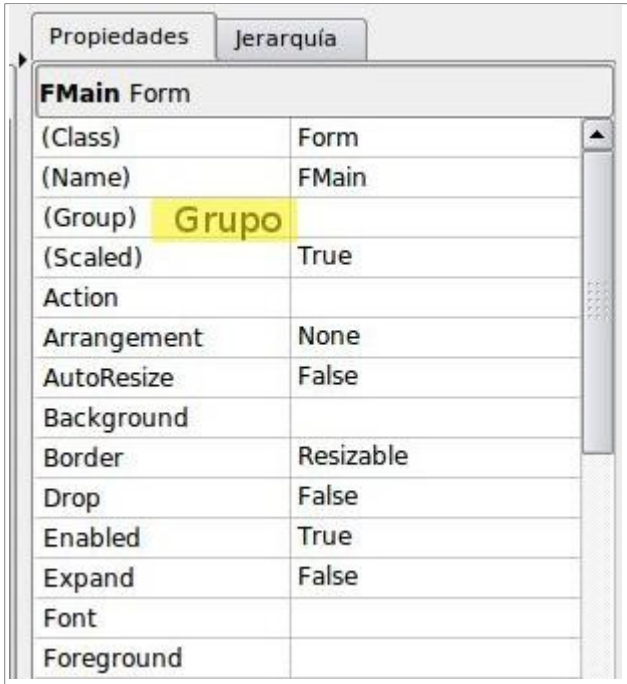
Para crear el menú, pulsamos con el botón derecho en cualquier punto vacío del formulario y seleccionamos la opción "Editor de menú" o Crtl+E



Menus:

Para conseguir que los botones respondan a "atajos de teclado", hay que poner un "ampersand" (&) delante de la letra que servirá como "atajo".

Para la opción de “añadir”, el nombre y grupo le tenemos que poner “annadir”, ya que la “ñ”, no lo acepta . Para el título si nos acepta poner “añadir”



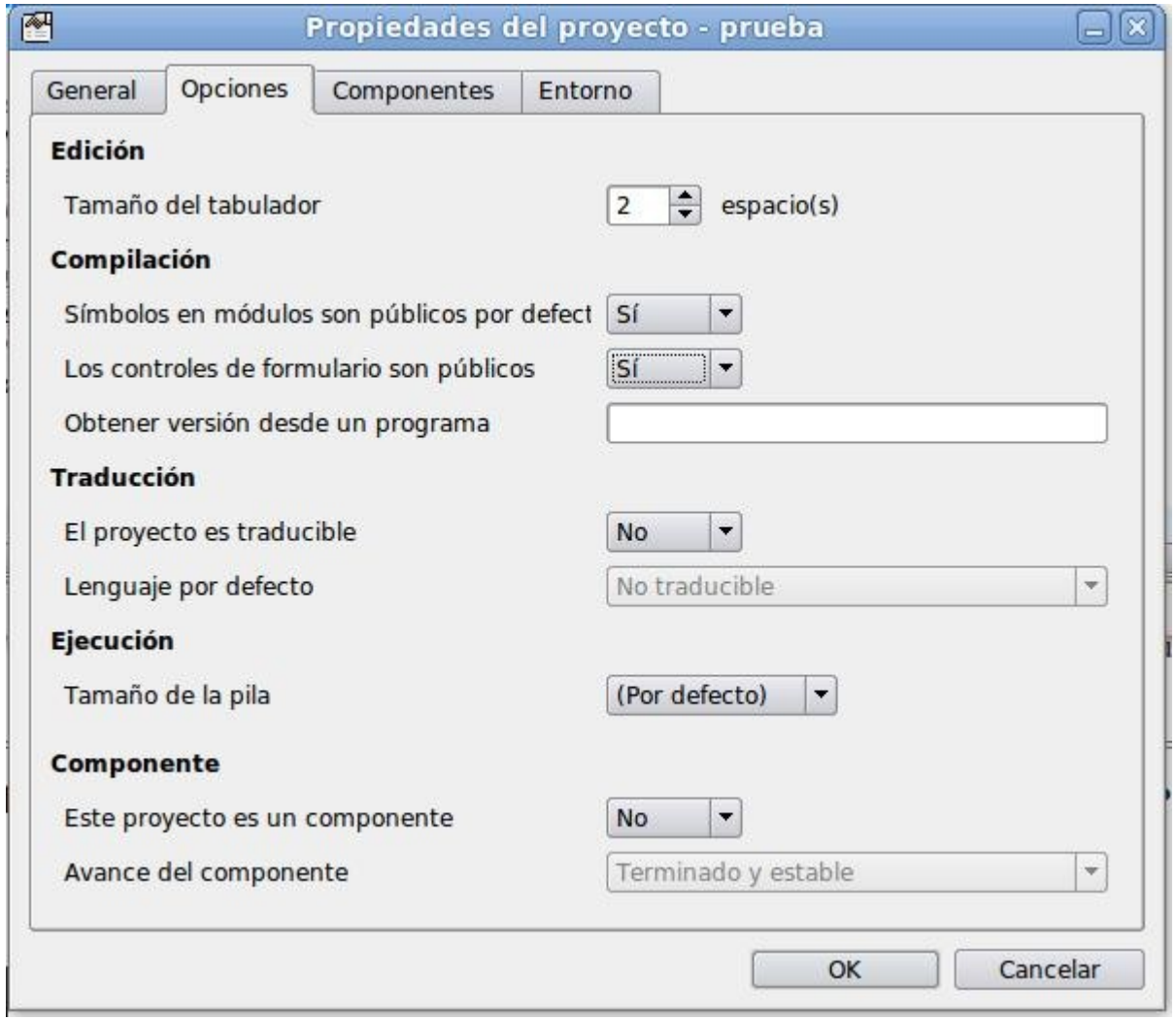
Al crear los botones y las distintas entradas en el menú podemos observar en la ventana de propiedades que hay, aparte de las opciones típicas (nombre, clase, texto a mostrar, etc.), una opción llamada “(Group)” ("Grupo"). Esta opción es muy interesante, puesto que si tenemos varios controles (p. ej., el menú "Abrir" y el botón "Abrir") que deben hacer lo mismo, asociándolos al mismo grupo sólo tenemos que escribir el código correspondiente al grupo de acciones al que pertenece cada control. Así pues, en nuestro programa de ejemplo, hemos asociado al grupo "Abrir" el menú y el botón "Abrir", al grupo "Guardar" el botón y el menú "Guardar", etc.

El diseño que proponemos sería algo así:

Si ahora hacemos click en un botón o en el menú correspondiente, se nos abrirá el editor de código posicionándose el cursor en la declaración de un procedimiento que se llama igual que el grupo de acciones.

Importante:

Vamos a hacer que los formularios y variables sean accesibles desde cualquier sitio del programa, esto se hace en el menú Proyecto/Propiedades/ pestaña Opciones, y decimos “Si” a “Símbolos en módulos son publicos por defecto” y “los controles de formularios son públicos”



Gestión de eventos

Los programas con interfaz gráfica de usuario suelen basar su funcionamiento en eventos. Esto es, cada vez que el usuario "hace algo" en la aplicación, se genera un evento y éste evento puede tener asociado una función o procedimiento que responda a la acción del usuario.

Si, por ejemplo, el usuario hace click en un control determinado, se generan varios eventos: *MousePress*, al presionar el botón del ratón, *MouseRelease*, al liberar el botón del ratón, *Click* como resultado de esta acción. Si el usuario hace doble click, el evento generado es un *DblClick.* Por supuesto, no todos los

controles son capaces de responder a todos los eventos. No tiene sentido hablar del evento *Resize* en un botón, puesto que este evento se genera al redimensionar una ventana.

En Gambas, para introducir el código del procedimiento (ver nota 2) correspondiente a un evento, se declara de la siguiente manera:

```
PUBLIC SUB Control_Evento
```

Donde `Control` es el nombre del control que está respondiendo al evento y `Evento` es el evento que se produce. Algunos controles tienen un evento predeterminado, que es el más usual: un botón tiene como evento predeterminado el `Click`, etc.

En Gambas, al hacer click sobre cualquier control, se abre el editor de código en la declaración del evento predeterminado, con una excepción. Como comentábamos antes, si el control está asociado a un grupo de acciones, el editor se abre en la declaración del procedimiento correspondiente al grupo de acciones.

Consideraciones relativas al diseño de formularios

Al diseñar el formulario de la aplicación, debemos tener en cuenta varias cuestiones:

- No todos los usuarios utilizan la misma resolución de pantalla, gestor de ventanas y tipo de fuentes. Hay que tener cuidado y no tratar de "aprovechar" demasiado el espacio. Podemos acabar con etiquetas de texto (Label) ilegibles, botones con el texto cortado, etc
- Por la misma razón, conviene que la ventana principal de la aplicación sea redimensionable por el usuario (en Gambas es la propiedad *Border* del formulario. No es recomendable fijar esta propiedad a *Fixed*.
- Un formulario tiene varias opciones que pueden ser interesantes:
  Tenemos varias opciones que nos sirven en el caso de que queramos hacer alguna operación sobre el formulario antes de visualizarlo y al cerrarlo, respectivamente.

```
' Gambas class
file PUBLIC SUB _new()

END

PUBLIC SUB _free()

END

PUBLIC SUB Form_Open()

END
' Gambas class file
```

```
STATIC PUBLIC SUB _init()

END

STATIC PUBLIC SUB _exit()

END

PUBLIC SUB _new()

END

PUBLIC SUB _free()

END

PUBLIC SUB Form_Open()

END
```

Podemos así alterar el comportamiento de nuestra aplicación al abrirse y/o cerrarse el formulario. Que el procedimiento esté declarado como STATIC significa que sólo podrá acceder a variables declaradas también como STATIC.

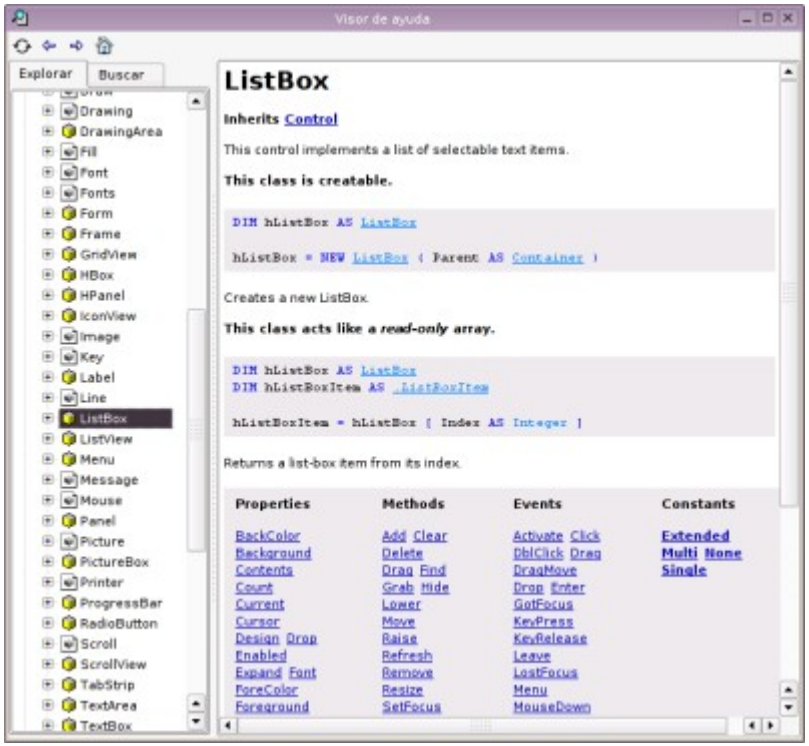
Al grano ...

Ya tenemos nuestro formulario diseñado. Ahora se trata de implementar funcionalidad a los controles.

Lo primero que vamos a hacer es que los botones "Añadir", "Modificar", "Borrar" y "Limpiar" funcionen.

Acción "Limpiar"

Este botón se encarga de borrar todas las entradas que haya en el ListBox. Para saber cómo hacer ésto, buscamos en el navegador de ayuda la documentación relativa al control ListBox:



La documentación se encuentra bajo el "árbol" `gb.qt`, que es donde se encuentra la documentación de todos los controles del tipo "visual" (botones, etiquetas, menús, etc...). Vemos que el ListBox proporciona un método " `Clear`", que precisamente hace lo que queremos: borrar todo el contenido del control.

Haciendo click en el botón "Limpiar", se abre el editor de código en el procedimiento correspondiente (el botón Limpiar y el menú limpiar, están asociado al grupo limpiar). Añadimos el siguiente código:

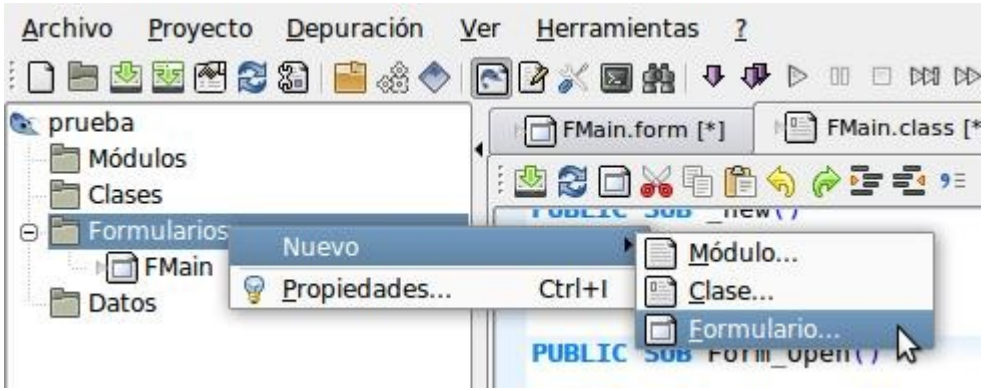
```
PUBLIC SUB Limpiar_Click()
  ListBox1.Clear()
END
```

Fácil, ¿verdad?.

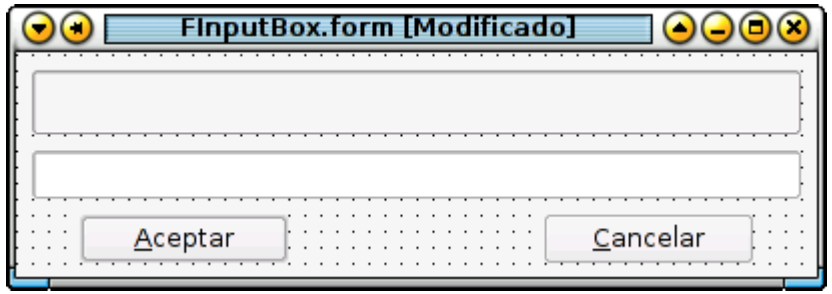
Acción "Añadir"

Esto ya es un poco más complicado. Queremos que el usuario, al pulsar el botón, pueda escribir una línea de texto que se cargue en el ListBox.

Vamos a crear un nuevo formulario nosotros mismos para ello:



Este es el diseño que proponemos:



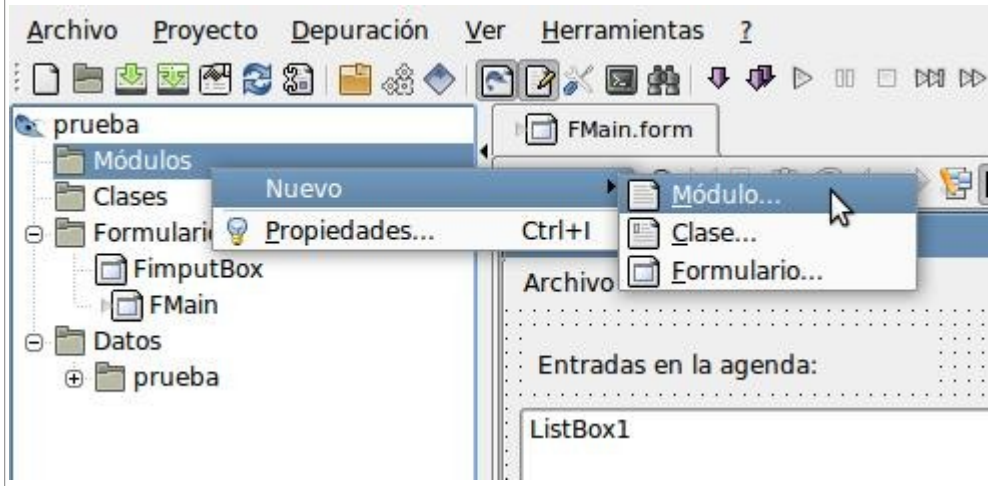
El formulario no tiene mucha complicación. Dispone de una etiqueta o `Label`, una entrada de texto (`TextBox`) y dos botones. Como buen cuadro de diálogo que se precie, es conveniente que se pueda cancelar con la tecla `Escape` y aceptar con la tecla `Enter`:

Los controles `Button` tienen dos propiedades adecuadas para este cometido. Son " *Default*" y " *Cancel*". Para el botón "Aceptar", ponemos " *Default*" a `True` y " *Cancel*" a `False`. Para el botón "Cancelar", al contrario.

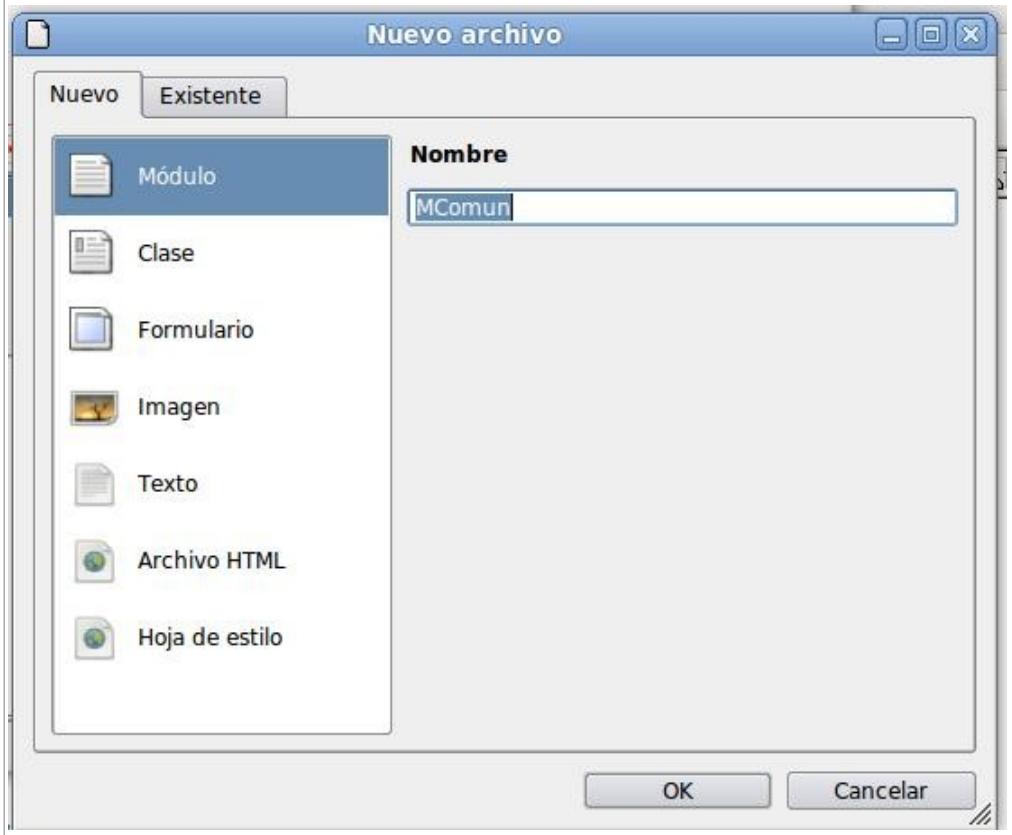
De esta manera, cuando se abra el formulario, una pulsación en la tecla < `ENTER`> será equivalente a pulsar el botón "Aceptar" y una pulsación en la tecla < `ESC`> simulará el botón "Cancelar".

El siguiente problema que se nos plantea es cómo retornar el valor que introduce el usuario en la entrada de texto a la ventana principal. Hay que destacar que en Gambas no hay variables globales, así que tendremos que buscar otra solución. En el "Consejo del día nº 7", (menú "? > Consejos del día") se nos sugiere que creemos un módulo en el cual ponemos una variable PUBLIC, así puede ser accedida desde cualquier punto de la aplicación.

Creamos un nuevo módulo



y lo llamamos `MComun`, por ejemplo.



Esta sería la implementación del módulo:

```
' Gambas module file
PUBLIC texto AS String
```

Así, sin más. Ahora tenemos una variable visible desde cualquier punto del programa que puede ser accedida con la notación `MComun.texto`

Lo siguiente es implementar el formulario "FInputBox". Esta sería su implementación:

```
' Gambas class file
```

```
PUBLIC SUB _new(titulo AS String, mensaje AS String, OPTIONAL texto AS String)
  ME.Caption = titulo
  Label1.Caption = mensaje
  ' un String se evalúa como False si está "vacío"
  IF texto THEN TextBox1.Text = texto
END

PUBLIC SUB Button1_Click() ' Este es el botón Aceptar
  MComun.texto = TextBox1.Text
  ME.Close(0)
END

PUBLIC SUB Button2_Click() ' Este es el botón Cancelar
  ME.Close()
END
```

El procedimiento `_new` es el constructor. Como nos interesa que el texto de la etiqueta, el título y el texto a editar sean distintos cada vez, los ajustamos al crear la ventana.

El botón "Aceptar" asigna el texto en el `TextBox` en la variable `texto` del módulo `MComun` y cierra el formulario. El botón "Cancelar" simplemente cierra la ventana.

Como la variable `MComun.texto` es común, tenemos que acordarnos de "limpiarla" cada vez que la utilicemos. Vamos a verlo ahora mismo.

Seguimos ahora con el formulario Fmain()

El procedimiento para el botón "Añadir" del formulario principal es el siguiente. Es bastante autoexplicativo:

```
PUBLIC SUB Annadir_Click()
    ' Declaramos nuestro "FInputbox"
    ' Creamos el FInputBox, pasándole el título, mensaje a mostrar
    ' y un valor por defecto: la fecha y hora del momento y una flechita
    Dim f AS new FInputBox("Escribir entrada","Escriba la línea que desea añadir:",CStr(Now) & " -> ")
    ' Lo mostramos
    f.ShowModal()
    ' Si han pulsado aceptar y han metido texto,
    ' estará en la variable MComun.texto
    IF MComun.texto THEN   'Una cadena vacía es False
        ' El control ListBox tiene un método para añadir texto: .Add
        ListBox1.Add(MComun.texto)
        ' "Vaciamos" la variable común
        MComun.texto = ""
```

```
    END IF
END
```

Acción "Modificar"

Al pulsar este botón, el usuario modificará alguna de las entradas que haya en el ListBox. Si no hay ninguna, el botón no debe hacer nada, y si no han seleccionado ninguna línea, mostrará un mensaje de aviso. Veamos la implementación del procedimiento asociado.

```
' Acción "Modificar"
PUBLIC SUB Modificar_Click()
    Dim f AS FInputBox
    IF ListBox1.Count > 0 THEN ' Si no hay nada en el formulario,
                               ' su propiedad Count es 0. En este caso,
                               ' no hacemos nada.
        IF ListBox1.Index = -1 THEN
            ' La propiedad Index nos devuelve el índice de la línea seleccionada.
            ' Si no hay seleccionada ninguna, devuelve -1. En este caso, avisamos
            ' al usuario y no hacemos más.
            message.Info("Debe seleccionar la línea que desea modificar.")
        ELSE
            ' El usuario ha seleccionado una línea en el ListBox.
            ' Mostramos nuestro InputBox, pasándole también el texto seleccionado.
            ' El texto seleccionado es la propiedad Text del objeto ListBoxItem
            ' seleccionado, al que se accede a su vez con la propiedad Selected
            ' del ListBox
            f = NEW FInputBox("Modificar entrada",
                              "Modifique la línea seleccionada:",
                              ListBox1.Current.Text)
            f.ShowModal()
            ' El cuadro de diálogo FInputBox modifica la variable compartida
            ' en el módulo MComun.
            ' Si no está vacía, la asignamos al ListBoxItem seleccionado.
            IF MComun.texto THEN ListBox1.Current.Text = MComun.texto
            ' Como antes, "vaciamos" la variable compartida después de usarla.
            MComun.texto = ""
        END IF
    END IF
END
```

Acción "Borrar"

Como en el caso anterior, el ListBox debe tener alguna línea, y el usuario debe haber seleccionado una al menos. El código es similar al del botón "Modificar":

```
PUBLIC SUB Borrar_Click()
    Dim i AS Integer
    i = ListBox1.Index
    IF i >= 0 THEN
        ListBox1.Remove(i)  ' El método Remove quita una línea, justo
                            ' lo que queremos
    ELSE IF ListBox1.Count > 0 AND i = -1 THEN
        ' Comprobamos que el ListBox no esté vacío y que
        ' haya algo seleccionado.
        message.Info("Debe seleccionar la línea que desea borrar.")
    END IF
END
```

Podemos observar que la implementación de estas cuatro acciones es común para los botones y las entradas equivalentes en el menú, porque están cada una en su **grupo**.

Ahora pasamos a implementar las acciones relativas al manejo de ficheros (Abrir, Guardar) y salir de la aplicación. Empezaremos por lo fácil:

Acción "Salir"

La función de este botón (y la correspondiente entrada en el menú) es cerrar la aplicación. Nada más sencillo:

```
PUBLIC SUB Salir_Click()
    ME.Close() ' ME es una referencia al propio formulario

END
```

Se podría hacer un poco más amigable esta acción agregando un diálogo del tipo " *¿Está Ud. seguro de que quiere salir de la aplicación?*" y actuar en consecuencia. Dejamos esta mejora como ejercicio para el lector.

Acción "Guardar"

Al pulsar el botón "Guardar" o la entrada equivalente en el menú, el programa debe volcar los contenidos del listbox1 a un fichero de texto. Mostraremos un cuadro de diálogo al usuario para que nos proporcione el nombre del fichero a utilizar. Este es el código correspondiente:

```
PUBLIC SUB Guardar_Click()
    dim lineas AS String
    dim destino AS String
    dim numArchivo AS Integer
    lineas = ListBox1.List.Join("\n")
    Dialog.Title = "Seleccione un archivo"
    Dialog.Filter = ["*.data", "Datos de agenda"]
    IF NOT Dialog.SaveFile() THEN
        IF Right$(Dialog.Path, 5) <> ".data" THEN
            destino = Dialog.Path & ".data"
        ELSE
            destino = Dialog.Path
        END IF
            File.Save(destino, lineas)
    END IF
END
```

Queremos que los datos se guarden en un fichero con la extensión `.data`, así que si el nombre del fichero que proporciona el usuario no termina en ".data", concatenamos manualmente la extensión. Para guardar el contenido en un fichero, utilizamos el método `Save()` de la clase `File`, que toma como argumentos la ruta al fichero y el texto que queremos volcar. Accedemos al contenido del ListBox mediante su propiedad .`List.Join`, que devuelve un `String`, añadiendo un salto de línea "`\n`" para separar cada entrada en el ListBox.

Este trozo de código se nos presenta una característica muy interesante de Gambas, las clases "no instanciables" o estáticas (3). Son clases que no pueden instanciarse pero pueden utilizarse directamente. En esta acción vemos en acción dos de estas clases: la clase "`File`" y "`Dialog`", ver mas explicaciones en la siguiente entrada (Acción “Abrir”)

Acción "Abrir"

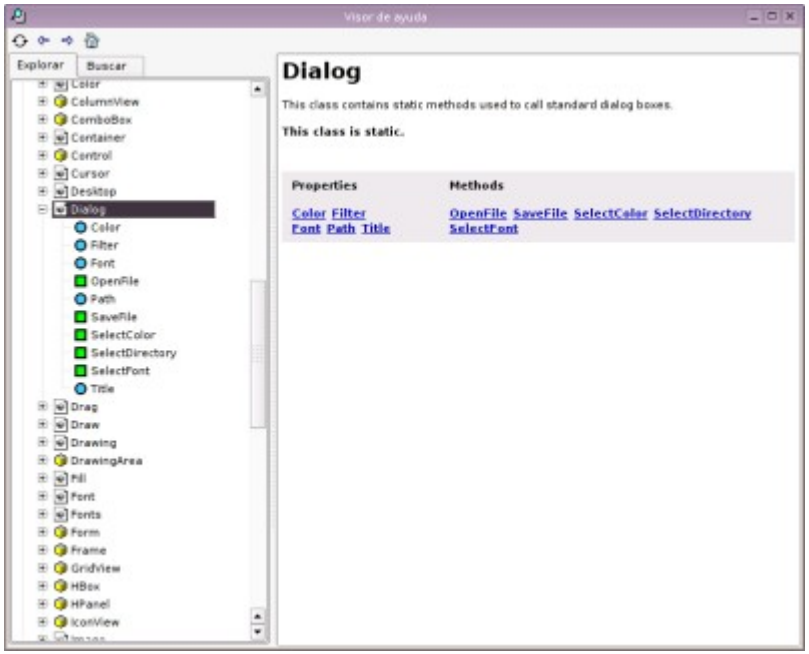
¿Qué se supone que debe hacer?. Pues preguntarle al usuario por un archivo, leerlo y cargar el contenido en el ListBox. Veamos directamente la acción correspondiente:

```
PUBLIC SUB ButtonAbrir_Click()
    DIM c AS String
    DIM arr_cadenas AS String[]
    Dialog.Title = "Seleccione un archivo"
    Dialog.Filter = ["*.data", "Datos de agenda", "*.*", "Todos los ficheros"]
    IF NOT Dialog.OpenFile() THEN
        arr_cadenas = Split(File.LOAD(Dialog.Path), "\n")
        ListBox1.Clear()
```

```
        FOR EACH c IN arr_cadenas
            ListBox1.Add(c)
        NEXT
    END IF
END
```

La clase "`File`" y "`Dialog`".

Por ejemplo, la clase `Dialog` proporciona acceso a los típicos cuadros de diálogo de selección de ficheros, colores, etc. Está documentada en `gb.qt`

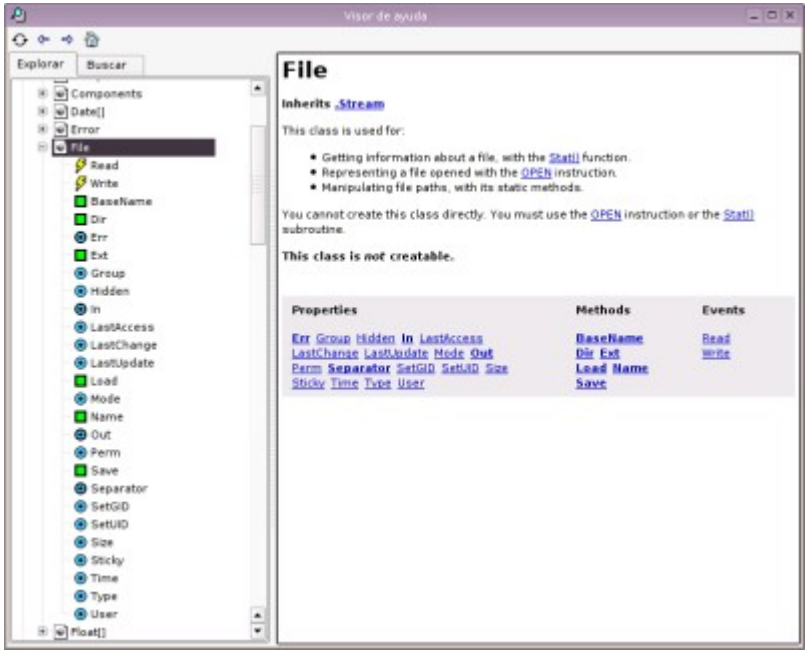


En nuestra aplicación, queremos seleccionar un fichero y cargarlo. Para hacer ésto, utilizaremos la clase `Dialog` de la siguiente forma:

```
Dialog.Title = "Seleccione un archivo"
Dialog.Filter = ["*.data", "Datos de agenda", "*.*", "Todos los ficheros"]
IF NOT Dialog.OpenFile() THEN
    ' etc ...
```

Ajustamos el título del cuadro de diálogo, proporcionamos un filtro para la selección del tipo de fichero por extensión y finalmente invocamos el método `OpenFile()` de la clase. Curiosamente, si NO se selecciona un fichero (el usuario pulsa "Cancelar", etc ...), el valor de retorno del método `OpenFile()` es `True`. Una vez seleccionado el fichero por parte del usuario, podemos acceder a la ruta completa con la propiedad `Dialog.Path`

La clase `File` (su documentación se encuentra "colgando" de la entrada `gb`) proporciona varios métodos para trabajar con ficheros.



En la documentación de Gambas, en la sección "How do I ..." se muestran varios ejemplos para leer y escribir ficheros. Nosotros vamos a utilizar en nuestra aplicación el método `Load()`, que recibe como argumento la ruta de un fichero y devuelve un `String` con todo el contenido del fichero. Para separar las líneas que contiene el fichero, utilizamos la función `Split()`, que toma como argumentos la cadena que queremos "partir", el carácter a utilizar como separador (un salto de línea en nuestro caso, "`\n`") y devuelve un `Array` de `Strings`. Por ello hemos declarado la variable `arr_cadenas` como `String[]`:

```
DIM arr_cadenas AS String[]
```

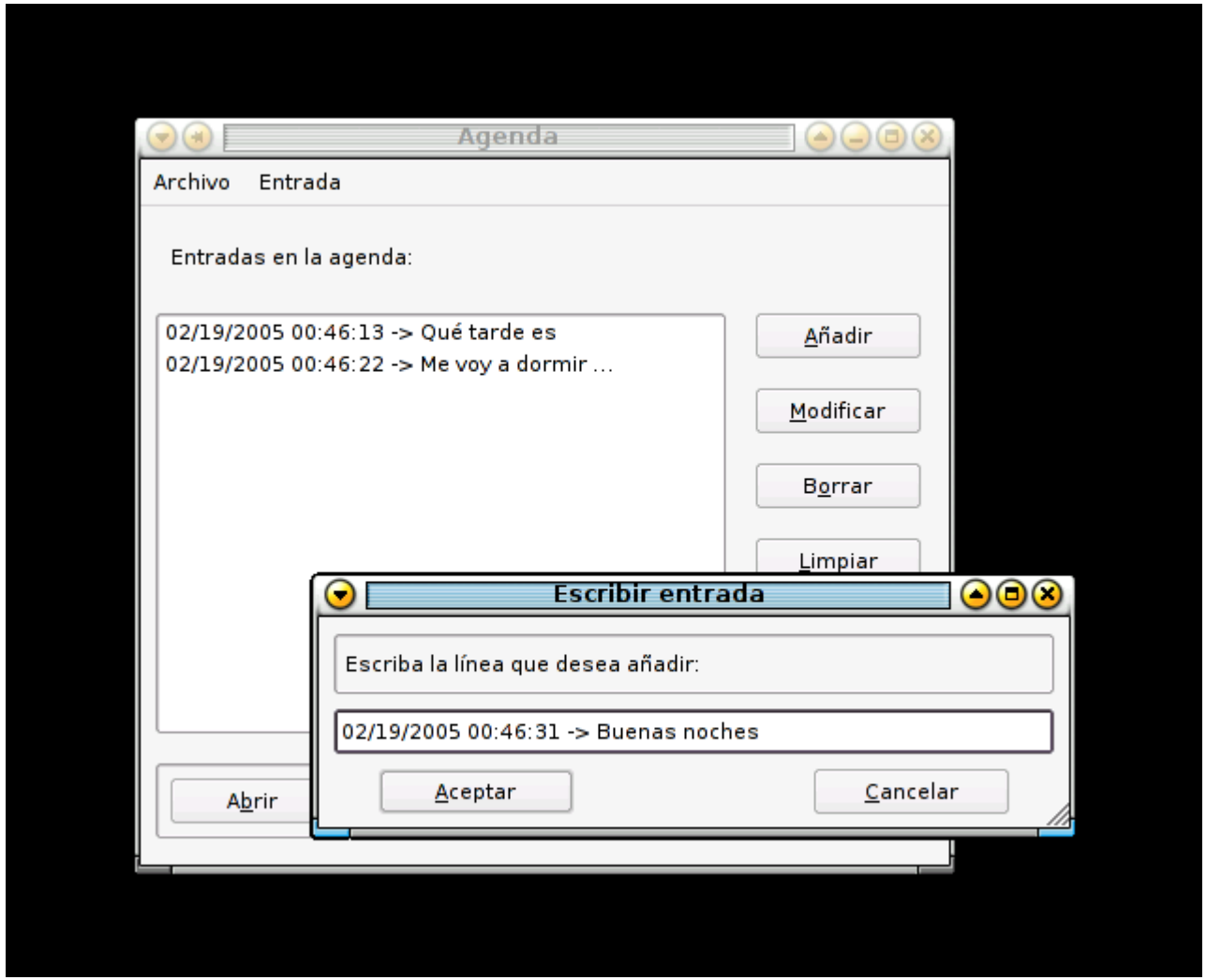
Una vez que tenemos la lista de cadenas contenidas en el fichero, limpiamos el ListBox y vamos añadiendo una a una cada cadena utilizando el método `Add()` del ListBox.

Un último ajuste

Se nos ocurre que sería interesante que cuando el usuario se posicione en una de las líneas del ListBox pudiese visualizar el contenido completo de la línea, ya que pueden aparecer cortadas si son muy largas. Lo vamos a hacer de la siguiente forma: cuando el usuario hace doble click en una entrada, el contenido de la línea lo mostraremos en un cuadro de diálogo:

```
PUBLIC SUB ListBox1_DblClick()
    IF ListBox1.Index >= 0 THEN
        message.Info(ListBox1.Current.Text)
    END IF
END
```

Nuestro programa funcionando



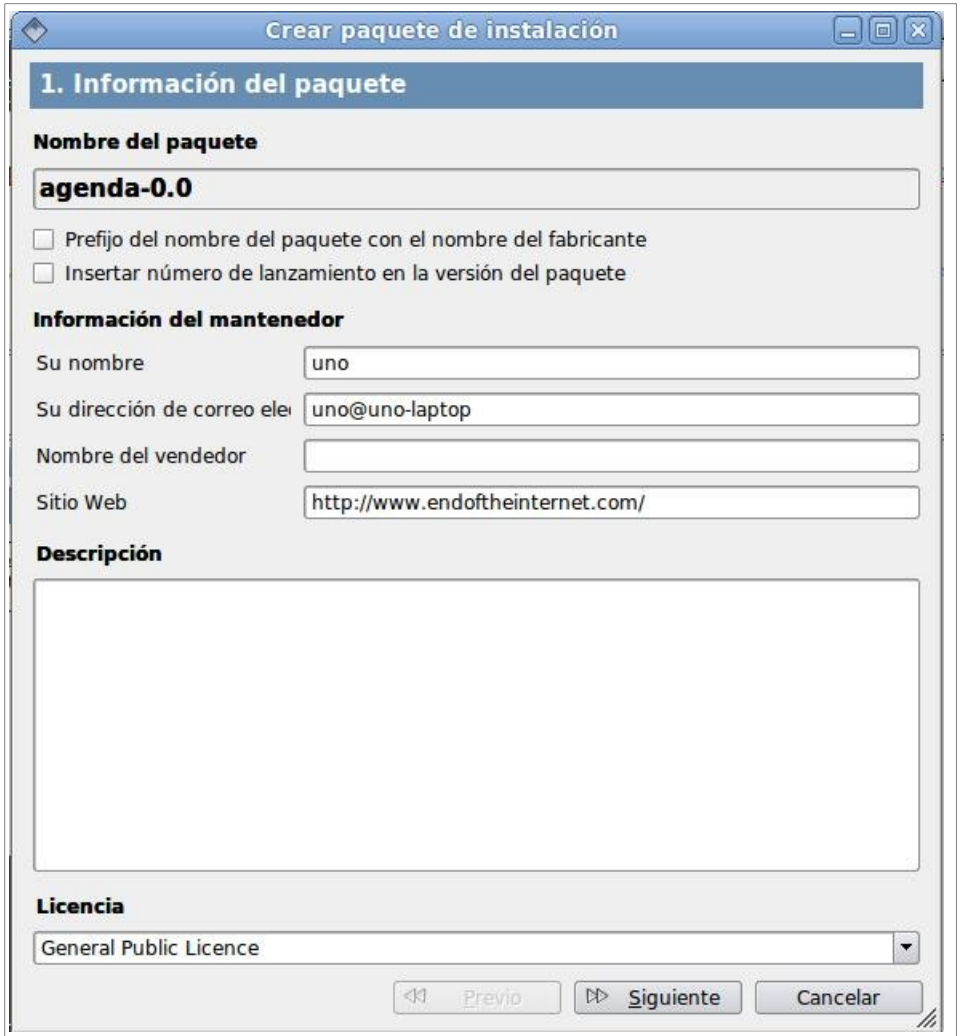
Distribuyendo nuestra aplicación

Ya tenemos la aplicación creada. Podemos probarla en cualquier momento del desarrollo utilizando la tecla F5 o pulsando sobre el botón:

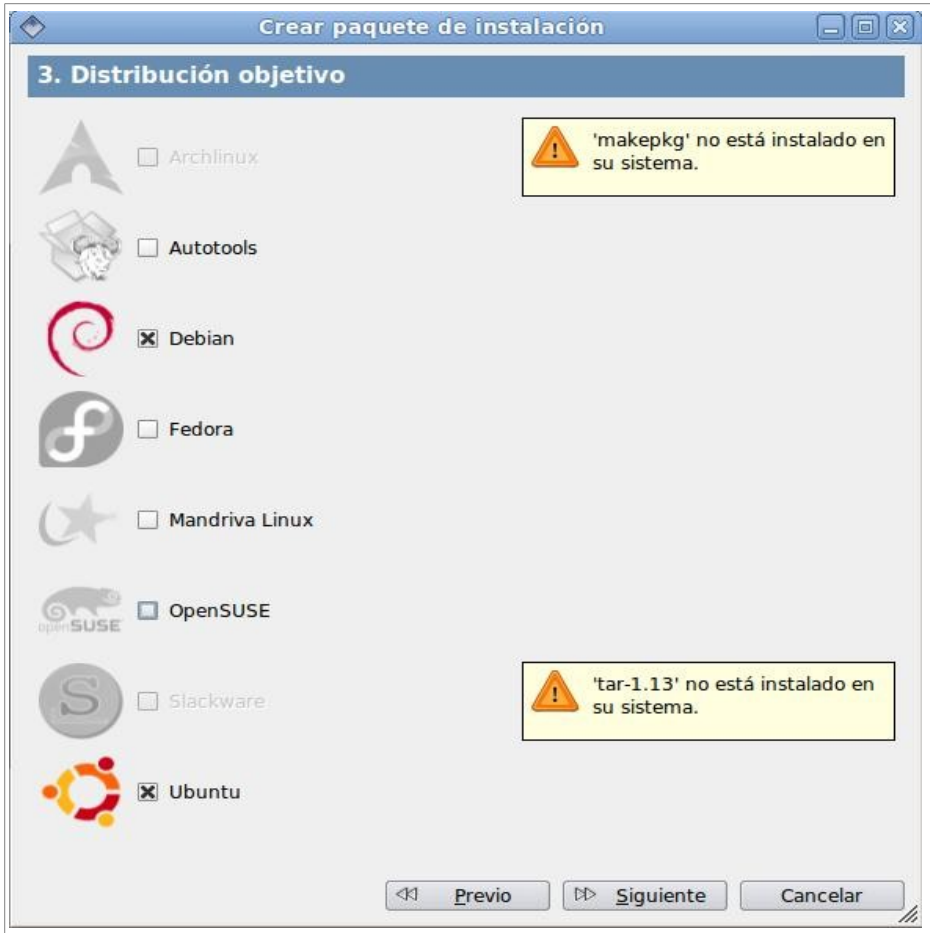
Ahora queremos utilizarla como un programa normal, sin tener que tener Gambas funcionando. Para ello hay una opción en el menú principal de Gambas ("Proyecto > Crear > ejecutable"). Esto nos genera un archivo ejecutable "monolítico", esto es, incluye todos los formularios, implementación y ficheros adicionales del proyecto. Este ejecutable no es código máquina, es "bytecode" ejecutable por el intérprete de Gambas, `gbx`. Esto implica que necesitamos tener instalado Gambas para ejecutar programas escritos con Gambas (al igual que otros lenguajes: se necesita tener Java para ejecutara un programa escrito en Java).

Por fortuna, en la mayoría de las distribuciones que incluyen Gambas se han separado los componentes y hay un "Gambas runtime", que incluye el intérprete, pero no el entorno de desarrollo completo.

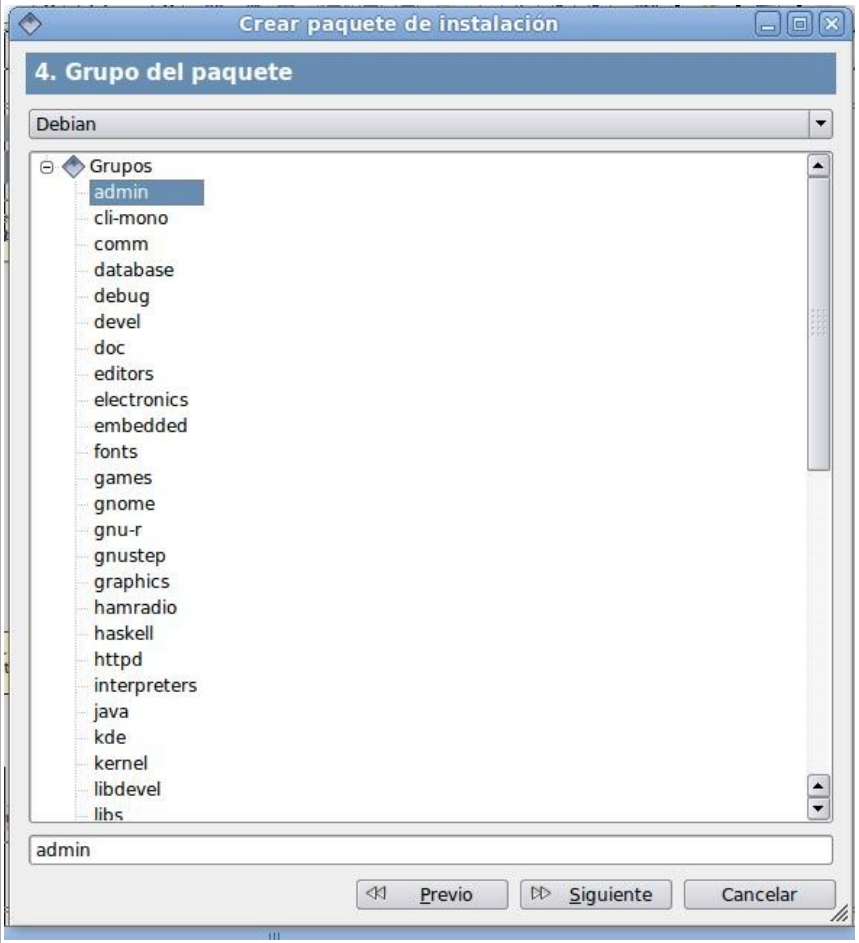
También podemos crear paquetes RPM o DEB para nuestro programa. Estos paquetes tendrán como dependencia el intérprete de Gambas (el `gambas-runtime`). Hay un asistente muy fácil de usar para crear los paquetes ("Proyecto > Crear > paquete de instalación ...").


Crear paquete de instalación
1. Información del paquete
Nombre del paquete
agenda-0.0
Prefijo del nombre del paquete con el nombre del fabricante
Insertar número de lanzamiento en la versión del paquete
Información del mantenedor
Su nombre
uno
Su dirección de correo ele
uno@uno-laptop
Nombre del vendedor
Sitio Web
http://www.endoftheinternet.com/
Descripción
Licencia
General Public Licence
Previo
Siguiente
Cancelar


Crear paquete de instalación
2. Changelog
Por favor introduzca los cambios de su proyecto.
Wed Oct 20 2010 uno <uno@uno-laptop> 0.0
Lanzamiento inicial
Previo
Siguiente
Cancelar

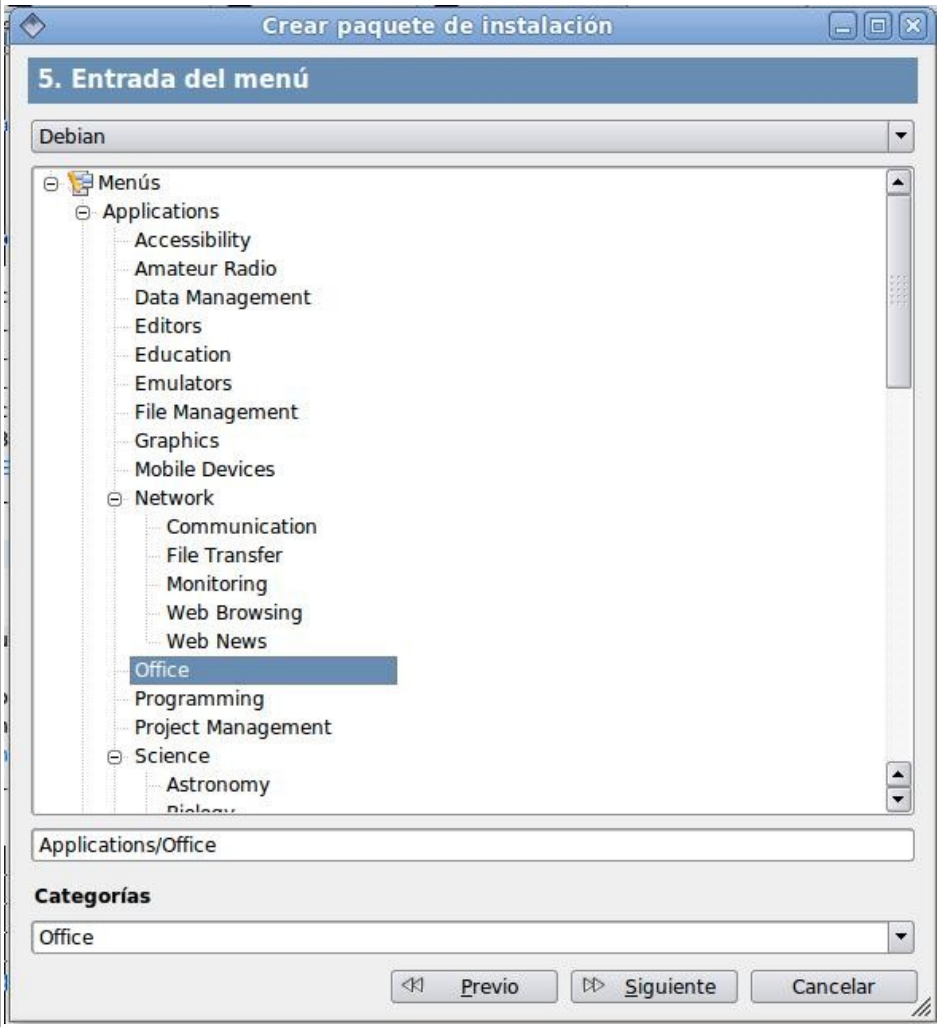


Elegimos Debian y Ubuntu. (pon dos ya que sino te dara problemas)

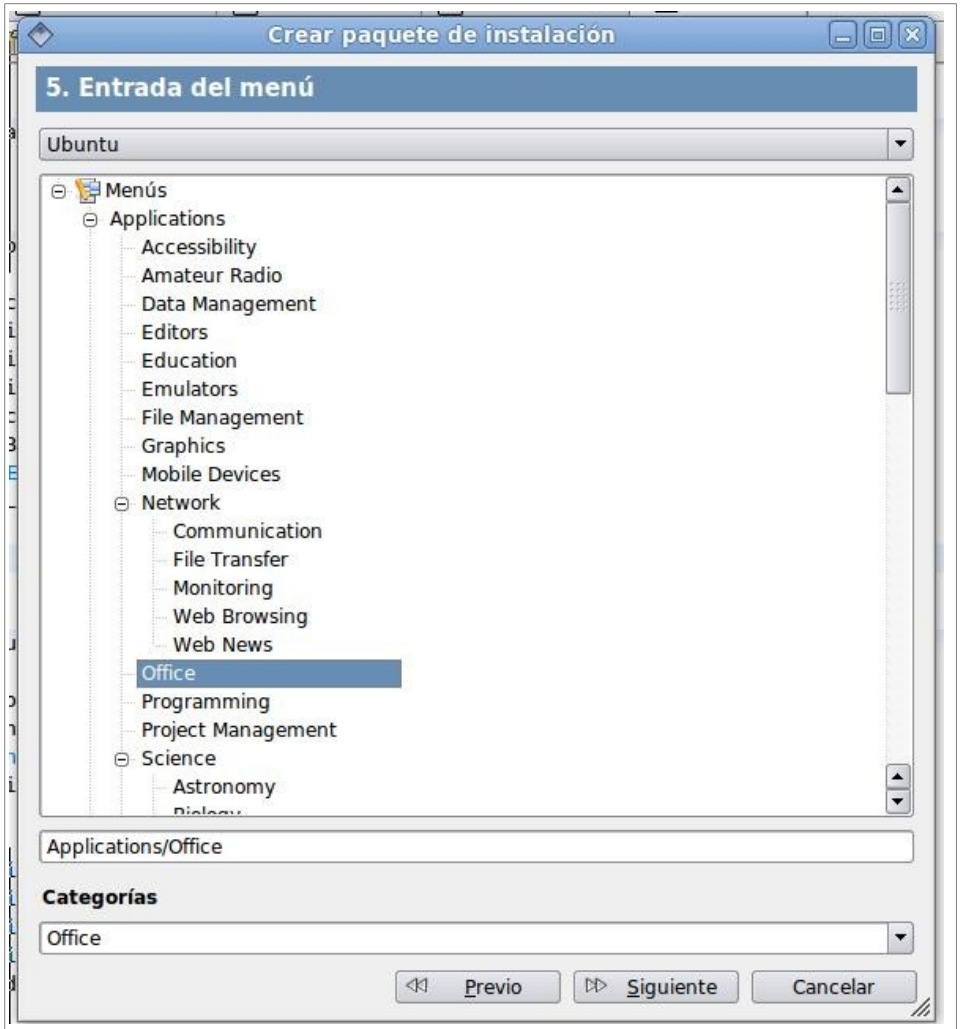


Aquí elegimos el grupo para DEBIAN

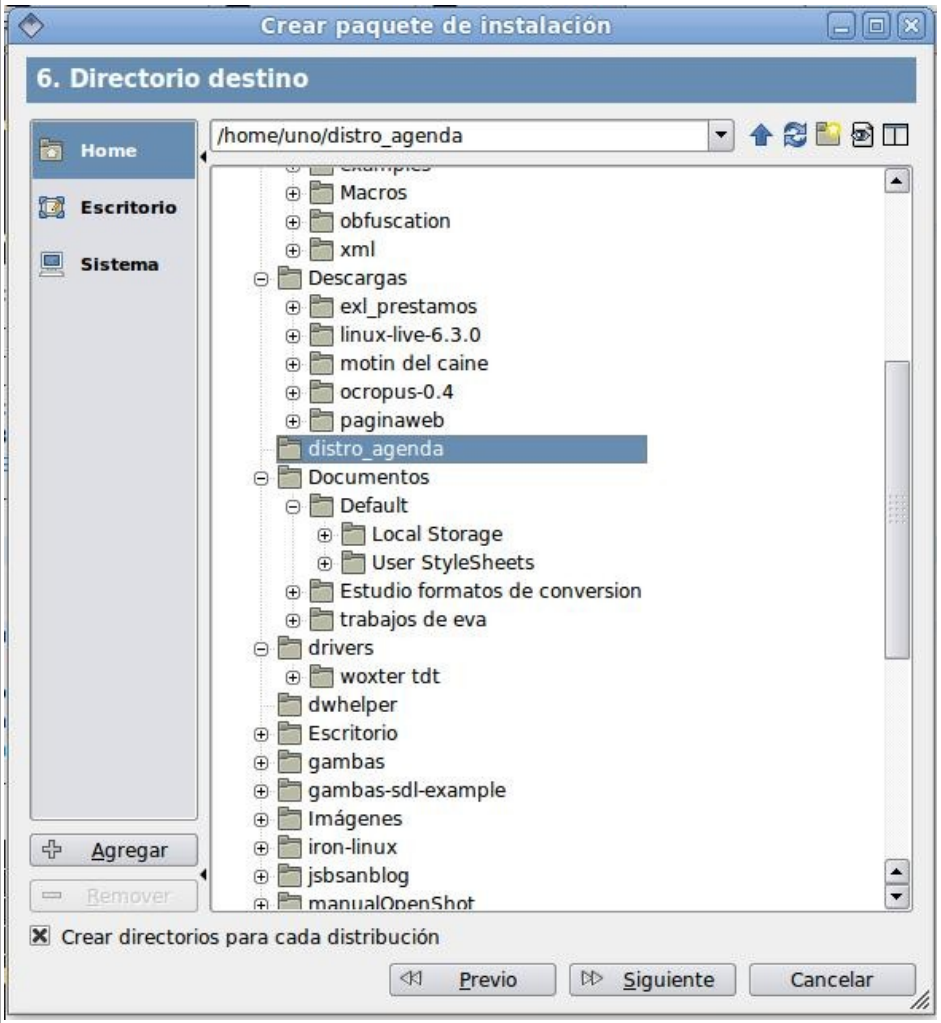
Y aquí el grupo para UBUNTU



La entrada del Menu para DEBIAN

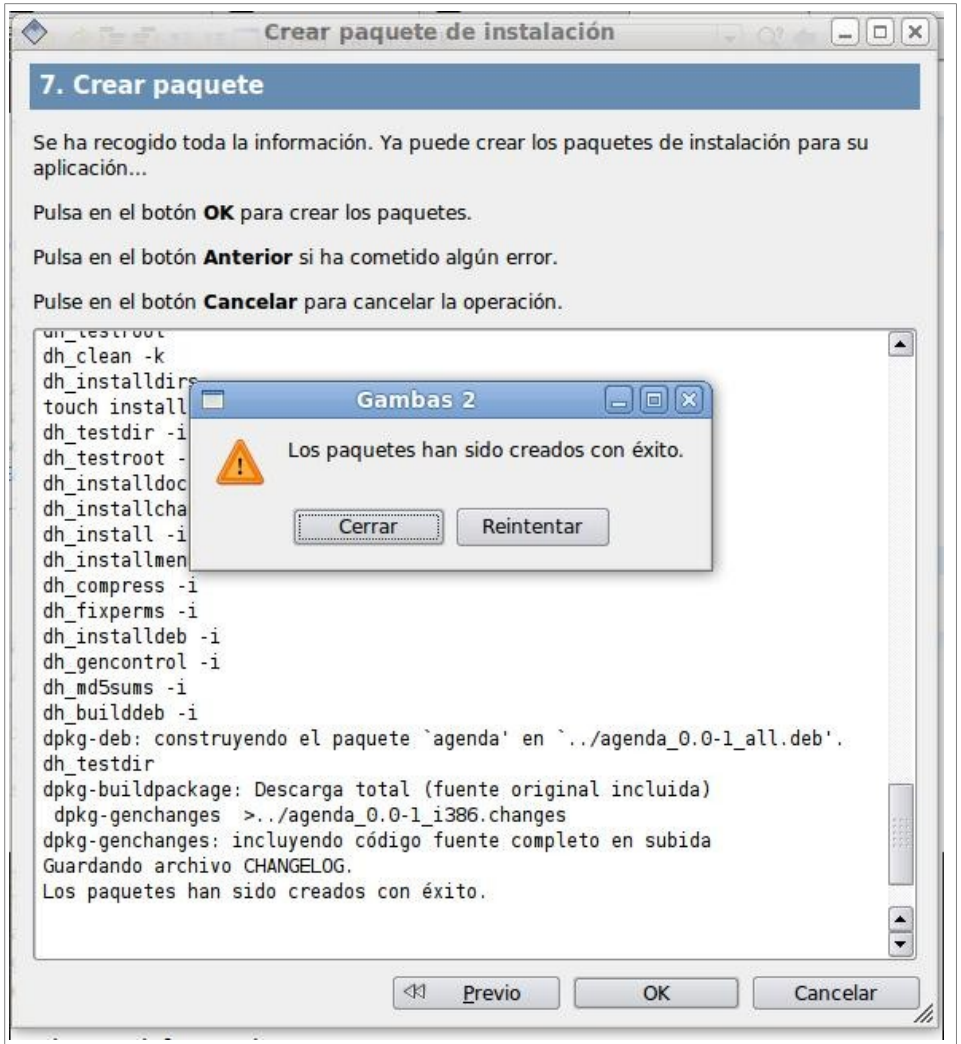


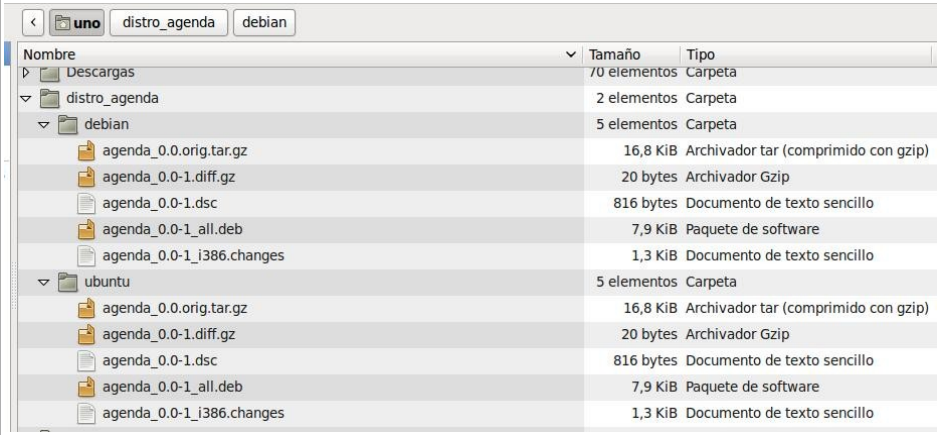
La entrada del Menu para UBUNTU



Creamos una carpeta (que no tenga en el nombre espacios o caracteres especiales, ya que nos dara error al crear los paquetes de instalación)

Hacemos click en “Crear directorios para cada distribución” y nos creara un directorio para Ubuntu y otro para Debian

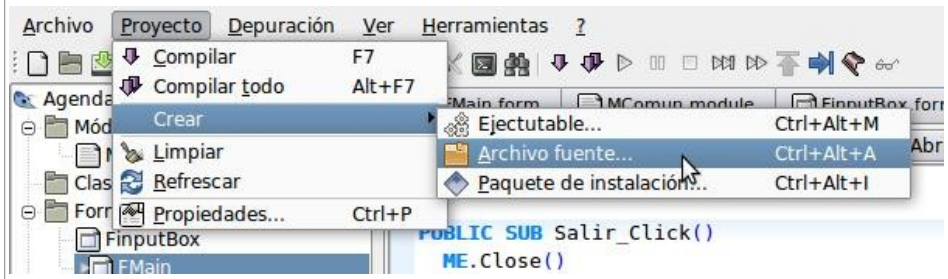


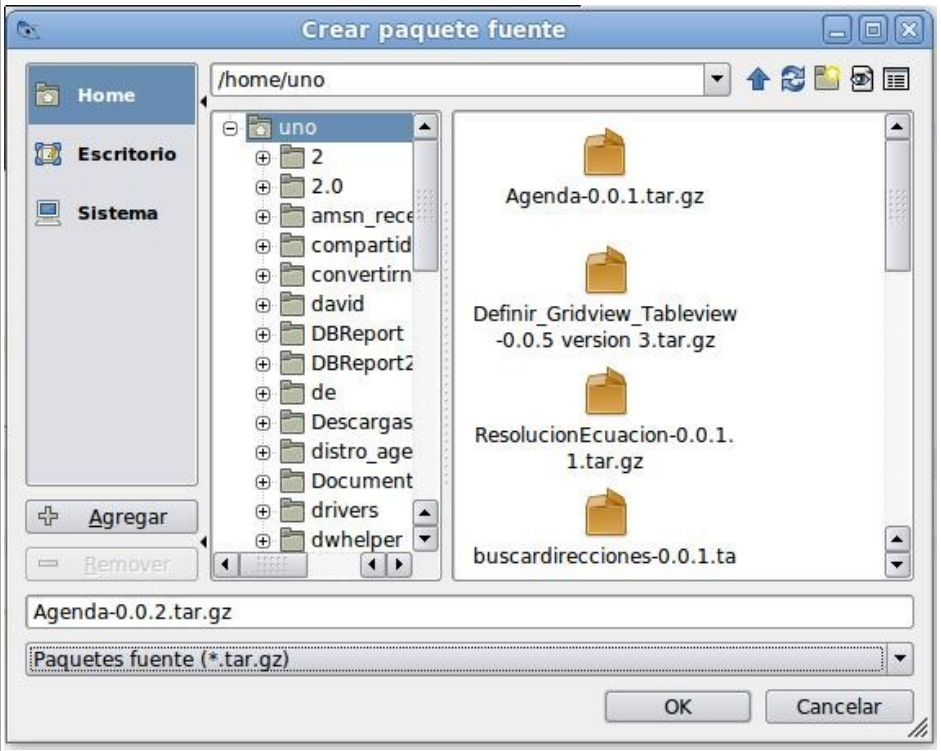


Podemos ver con Nautilus (el explorador de archivos de Gnome), las carpetas y los archivos que se crean.

Estos son los que distribuiremos para la instalación de nuestro en cualquier ordenador que tenga Ubuntu o Debian, sin tener el IDE de gambas2

Para distribuir el código fuente tenemos que ir a: Proyecto/Crear/Archivo Fuente





Elegir el nombre (el programa nos da uno automaticamente) y al darle ok, creara un archivo comprimido tar.gz, el cual si contiene todo el código fuente (formularios, modulos, etc) de nuestro programa.

Conclusiones

Hemos visto lo fácil que es crear una aplicación mínimamente funcional con Gambas. Proporciona bastantes controles y clases predefinidas. Hay también extensiones para crear aplicaciones cliente/servidor, acceso a bases de datos, multimedia, etc.

Personalmente me parece que es una herramienta con muchísimo futuro, y, afortunadamente, el desarrollo de Gambas es muy activo, corrigiéndose los errores que van surgiendo con mucha rapidez.

¡Gracias, Benoît (et col.)! ¡Excelente trabajo!

Acerca de este documento y del autor

Como mencionábamos antes, la aplicación se ha desarrollado utilizando la versión 2.21 de Gambas . En el momento en que leas este documento, probablemente haya una versión más moderna. Conviene leer la lista de cambios de una versión a otra por si se produce alguna incompatibilidad.

Cualquier comentario, sugerencia o mejora de este documento es bienvenida.
Mi correo es forodejazz (arroba) gmail (punto) com (autor original, 2005)

Actualizado: jusabejusabe (arroba) hotmail (punto) com (reversión 2010)

Rollo legal: Este documento es libre, puedes copiarlo, distribuirlo, modificarlo, enlazarlo, traducirlo a otras lenguas e incluso venderlo, pero siempre conservando esta nota y citando la procedencia del documento. En cualquier caso, el autor agradecería que se le notificase, y en un momento dado, ser retribuido económicamente por su esfuerzo.

Notas

1. Hay un buen tutorial de iniciación y documentación de Gambas en castellano en http://gambasdoc.org/help/?es , http://jsbsan.wordpress.com/2010/09/06/manual-paso-a-paso-de-como-hacer-un-programa-gambas2-listin-2-2/ , y en el foro: http://www.gambas-es.org/

2. Los eventos deben tratarse con un procedimiento, esto es, una función que no retorna valor alguno.

3. No soy experto en la terminología usada en la programación orientada a objetos, así que, probablemente estaré usando algún término incorrectamente.

   Mis disculpas ;-)