

Programa de ejemplo: Contactos

Uso de Clases, Colecciones y Ficheros Planos

First Name	John	Initial	Q
Last Name	Public	Suffix	II
Street1	12345 Gambas Drive		
Street2	Apt 101		
City	Dancer		
State	TX	Zipcode	77929 - 0101
Phone	888 - 666 - 4444		

Autores: John W.Rittinghose y Jon Nicholson

Traducido por Julio Sánchez Berro
<http://jsbsan.blogspot.com/>

Diciembre 2011

Índice del usuario

Introduccion.....	3
Inicio.....	4
Metodo Contact.GetData	4
Metodo Contact.PutData	7
Fichero Fmain.class.....	9
Subrutina Form_Open	11
Añadiendo controles a Fmain.Form.....	11
Los ToolButtons.....	12
El botón Salir. (Quit).....	12
Añadiendo Label and Textbox Controls.....	13
Subrutina UpdateForm()	15
Codificando los botones: Primero , Previo, Siguiente y Ultimo. (First, Prev, Next, and Last).....	16
Codificando boton: Añadir datos.....	18
Codificando Boton: Limpiar datos.....	20
Validar la información del usuario	21
Agregar una característica de búsqueda.....	23
La subrutina Buscar (DoFind).....	25
Editando un dato existente:.....	26
Borrando un dato de Contacto.....	28
Guardando los datos:.....	30
Creación de un archivo ejecutable independiente.....	31
El resultado final.....	32
Fuentes y enlaces interesantes:.....	32

Introduccion

Este manual es una traducción de parte del capítulo nº 11 Conceptos de Programación Orientada a Objetos del libro “A Beginner's Guide to Gambas – Revised Edition”, cuyos autores son John W.Rittinghose y Jon Nicholson.

Este libro se puede descargar de manera gratuita en la pagina web:
<http://beginnersguidetogambas.com/>

Inicio

Para ilustrar el uso de una clase de Gambas, vamos a crear un programa de ejemplo que implementa una libreta de contactos. Va a ser bastante sencilla, pero será útil para mantener contactos.

Para empezar, cree un nuevo proyecto llamado “contactos” y cuando llegas al IDE, crear una clase (no una clase de inicio) denominada “Contact”.

Aquí es donde vamos a empezar con nuestro programa, construyendo la definición de clase para un contacto.

```
' Gambas Contact.class file
PUBLIC FirstName AS String
PUBLIC LastName AS String
PUBLIC Initial AS String
PUBLIC Suffix AS String
PUBLIC Street1 AS String
PUBLIC Street2 AS String
PUBLIC City AS String
PUBLIC State AS String
PUBLIC Zip5 AS String
PUBLIC Zip4 AS String
PUBLIC Areacode AS String
PUBLIC Prefix AS String
PUBLIC Last4 AS String
PUBLIC RecordID AS Integer
```

Queremos que nuestra clase proporcione dos métodos, uno para recuperar nuestros datos de contacto desde el archivo en que se almacena y otro para poner los datos en ese archivo. Los dos métodos se denominará Contact.GetData y Contact.PutData.

Metodo Contact.GetData

Vamos a comenzar con el método GetData, que es declarado como una función porque tomará una variable de la colección como un parámetro y devuelve una colección llena de datos desde el archivo contacts.data si existen datos.

Declaramos variables locales dentro de nuestros métodos utilizando la palabra clave DIM. Tenga en cuenta que nosotros podríamos declarar estas variables como privados fuera de las definiciones de método y lo que les harían visible sólo para los métodos de la clase de contacto.

Sin embargo, al declararlas locales al método, además restringe el alcance y hace que toda la clase sea mas limpia en apariencia desde la perspectiva del usuario. Ellos verán todos los métodos públicos y variables y nada más.

La variable de la colección se pasa como un parámetro de la llamada de constructor en FMain.class por lo que no es necesario declarar por separado. La función devolverá una variable de la colección lo agregamos como colección hasta el final de la declaración de función.

Vamos a mostrar cómo la variable “ cTheData” de la colección se pasa a esta función cuando se

llama desde el archivo FMain.class.

PUBLIC FUNCTION GetData (cTheData AS Collection) AS Collection

'esta variable de cadena, se llevará a cabo cada línea de datos desde el archivo.

DIM sInputLine AS String

'Nosotros necesitamos un identificador de archivos y dos variables de cadena para la construcción del nombre del archivo.

DIM hFileIn AS File

DIM filename AS String

DIM fullpath AS String

'La siguiente variable clase "contact" se utiliza para convertir los datos de cadena que se leen del archivo en un registro de contacto.

Declaramos la variable aquí y luego (mas abajo) sera instanciada

DIM MyContactRecord AS Contact

'Variable local que es un entero que nos servira para realizar un seguimiento de cuantos registros se añaden.

DIM RecordNum AS Integer

'Esta variable es una matriz de cadenas, que se llenará con los datos leídos. Se usara la orden Split, que separa los datos separados por comas. (ver mas abajo)

DIM arsInputFields AS String[]

'Este contador se utiliza para recorrer la matriz y asignar valores a los registro de contacto.

DIM counter AS Integer

filename = "contacts.data" 'El nombre de nuestro fichero de datos es "contacts.data"

'Usaremos el directorio actual donde este la aplicacion ejecutandose . Esta es la ruta completa:

fullpath = Application.Path &/ filename

'Ahora comprueba si existe el archivo , en caso de existir , procede a su lectura

IF Exist(fullpath) THEN

OPEN fullpath FOR READ AS #hFileIn

'comienza con el registro 1

RecordNum = 1

'Vamos a recorrer todas las líneas de datos hasta llegar al final del archivo.

WHILE NOT Eof(hFileIn)

'cada contacto agregado a la colección debe crear una instancia cada vez que un registro se agrega a la colección. Aunque se puede reutilizar el misma var de clase para cada re-instancia.

MyContactRecord = NEW Contact

'aquí es donde se renueva la línea del archivo y se almacena la variable de cadena "sInputLine"

LINE INPUT #hFileIn, sInputLine

'Usamos la consola de depuración. (nos muestra lo que hemos leído del fichero de datos)

PRINT "Reading: " & sInputLine

'Ahora, llena nuestra matriz de cadenas mediante Split, para convertir la línea de lectura en campos de una matriz de datos

```
    arsInputFields = Split(sInputLine, ",")
```

'Mostramos en la consola de depuración, los datos leídos,

```
    FOR counter = 0 TO 14
```

```
        PRINT "Loading: " & arsInputFields[counter]
```

```
    NEXT
```

'Ahora, ponemos los datos del campo en nuestra clase de variable local de contacto y agregará ese contacto registro a la colección de contactos

```
    MyContactRecord.FirstName = arsInputFields[0]
```

```
    MyContactRecord.Initial = arsInputFields[1]
```

```
    MyContactRecord.LastName = arsInputFields[2]
```

```
    MyContactRecord.Suffix = arsInputFields[3]
```

```
    MyContactRecord.Street1 = arsInputFields[4]
```

```
    MyContactRecord.Street2 = arsInputFields[5]
```

```
    MyContactRecord.City = arsInputFields[6]
```

```
    MyContactRecord.State = arsInputFields[7]
```

```
    MyContactRecord.Zip5 = arsInputFields[8]
```

```
    MyContactRecord.Zip4 = arsInputFields[9]
```

```
    MyContactRecord.Areacode = arsInputFields[10]
```

```
    MyContactRecord.Prefix = arsInputFields[11]
```

```
    MyContactRecord.Last4 = arsInputFields[12]
```

```
    MyContactRecord.RecordID = CInt(arsInputFields[13])
```

```
    MyContactRecord.Deleted = CBool(arsInputFields[14])
```

'Mostramos en la consola de depuración, los datos de la clase:

```
    PRINT "Adding data for record: " & RecordNum & " ";
```

```
    PRINT MyContactRecord.FirstName & " " & MyContactRecord.LastName
```

'todos los campos de registros se asignan datos, por lo que añadimos a la colección
cTheData.Add(MyContactRecord, CStr(RecordNum))

'para la salida de depuración a la consola

```
    PRINT cTheData[CStr(RecordNum)].LastName & " stored."
```

'incrementar nuestro contador de registros "RecordNum y borramos la variable sInputLine

```
    INC RecordNum
```

```
    sInputLine = ""
```

```
WEND 'bucle para más registros
```

'Cuando se sale del bucle es porque se ha llegado al final del fichero, ahora cerramos el archivo de datos.

```
    CLOSE # hFileIn
```

ELSE 'No encuentra ningún archivo de datos, muestra un mensaje al usuario.

```
    Message.Info("No contacts.data file present. ", "OK")
```

```
ENDIF
```

'En la consola de depuración, nos mostrará todos los registros de la colección:

```
PRINT "Here is what is stored in cTheData:"
```

```
FOR EACH MyContactRecord IN cTheData
```

```
PRINT MyContactRecord.FirstName & " " & MyContactRecord.LastName  
NEXT  
RETURN cTheData 'devolver la colección (ahora llenada) con datos  
END 'de GetData
```

Metodo Contact.PutData

La rutina recíproca a GetData es PutData, donde almacenaremos la recopilación de datos que se consideran más actuales.

Sea cual fuere las modificaciones o cambios realizados se reflejará en esta operación de guardar. Salvaremos la fecha en un archivo, guardar el registro como una cadena delimitada por comas. Cada línea de salida representará un registro de contacto.

El método PutData se declara como una subrutina ya no devuelve una variable. Toma un solo parámetro, el objeto de colección que queremos almacenar. En este caso, cTheData es la colección pasada a esta subrutina.

PUBLIC SUB PutData (cTheData AS Collection)

'esta variable de cadena, se llevará a cabo cada linea de datos desde el archivo.

DIM sInputLine AS String

'necesitamos un identificador de archivo y dos variables de cadena para la construcción de nombres de archivo

DIM hFileOut AS File

DIM filename AS String

DIM fullpath AS String

'esta variable de clase se utiliza para convertir los datos de cadena en un registro de contacto. Declaramos la variable de aquí y será instanciada mas abajo

DIM MyContactRecord AS Contact

'RecordNum: es un entero local que nos servira para llevar el seguimiento de cuantos registros se agregarón.

DIM RecordNum AS Integer

'creamos una instancia de un nuevo registro en blanco. Tenga en cuenta que ya no estamos agregando este registro a una colección, no es necesario reinstanciarlo cada vez.

MyContactRecord = NEW Contact

'empezamos con el registro 1

RecordNum = 1

'Tenemos un nombre de archivo codificado por lo que sólo necesitamos la ruta al programa porque nos almacenará los datos donde reside el programa.

filename = "contacts.data"

fullpath = Application.Path &/ filename

'Tenemos que especificar: WRITE CREATE cuando abrimos el archivo. Si el archivo existe, se abrirá para la escritura, si no existe, se creara un nuevo archivo

OPEN fullpath FOR WRITE CREATE AS #hFileOut

'Ahora, recorreremos cada registro en la colección. Tenga en cuenta que utilizamos la construcción FOR EACH para ello.

FOR EACH MyContactRecord IN cTheData

'Vamos a concatenar cada campo con el ultimo, insertando una coma como delimitador. (Este delimitador, es el que hemos usado para la lectura)

sInputLine = sInputLine & MyContactRecord.FirstName & ","

```
sInputLine = sInputLine & MyContactRecord.Initial & ","
sInputLine = sInputLine & MyContactRecord.LastName & ","
sInputLine = sInputLine & MyContactRecord.Suffix & ","
sInputLine = sInputLine & MyContactRecord.Street1 & ","
sInputLine = sInputLine & MyContactRecord.Street2 & ","
sInputLine = sInputLine & MyContactRecord.City & ","
sInputLine = sInputLine & MyContactRecord.State & ","
sInputLine = sInputLine & MyContactRecord.Zip5 & ","
sInputLine = sInputLine & MyContactRecord.Zip4 & ","
sInputLine = sInputLine & MyContactRecord.Areacode & ","
sInputLine = sInputLine & MyContactRecord.Prefix & ","
sInputLine = sInputLine & MyContactRecord.Last4 & ","
sInputLine = sInputLine & MyContactRecord.RecordID & ","
sInputLine = sInputLine & CStr(MyContactRecord.Deleted) & ","
'para que la depuración mostramos lo que se está almacenando
PRINT "Storing: " & sInputLine
'Con la siguiente expresión, escribimos en el archivo la cadena de texto que hemos creado
anteriormente
PRINT #hFileOut, sInputLine
'Limpiamos nuestra cadena, para reutilizarla en la siguiente iteración
sInputLine = ""
NEXT 'iteración del bucle para enumerar objetos
'Llegados a este punto, ya hemos escrito todos los datos, ahora nos toca cerrar el archivo:
CLOSE # hFileOut
RETURN 'No devolvemos nada ya que es una subrutina y no una función
END
```

Guarde el archivo Contact.class el archivo y ciérrelo.

Nuestro programa permitirá a un usuario agregar, cambiar o eliminar un contacto y almacenar los datos en un archivo de datos.

Los datos de contacto se gestiona en nuestro programa como una colección de objetos que son del tipo de contacto, que hemos definido en el archivo Contact.class anterior.

Vamos a permitir al usuario desplazarse al primer elemento en la colección, desplazarse a elementos siguientes o anteriores, ir al último elemento, actualizar los datos en el formulario actual, guardar los datos actuales en archivo y salir. Todo ello, es lo que hace la mayoría de los conceptos básicos necesarios para la gestión de contactos.

Añadiremos también una función de búsqueda (mediante un módulo) que encontrará la primera instancia de un contacto mediante la búsqueda de un apellido.

Sorprendentemente para lograr todo esto, es necesario muy poco código en Gambas!

Fichero Fmain.class

Nuestra definición de clase contacto es completa por lo que ahora tendremos que crear un formulario denominado Fmain.

FMain servirá como el principal interfaz entre el usuario y el programa y sus datos. Aquí es cómo será nuestro formulario terminado:



The screenshot shows a window titled "Contacts Manager" with a standard Mac OS-style title bar. The window contains a contact form with the following fields and values:

- Navigation icons: Home, Previous, Next, End, and a list icon.
- Buttons: Add (green plus), Confirm (green checkmark), Cancel (red X), and Save (floppy disk).
- Form fields:
 - First Name: John
 - Last Name: Public
 - Street1: 12345 Gambas Drive
 - Street2: Apt 101
 - City: Dancer
 - State: TX
 - Zipcode: 77929 - 0101
 - Phone: 888 - 666 - 4444
 - Initial: Q
 - Suffix: II
- Bottom left: "Record: 1 of 2"
- Bottom right: "Quit" button and a search icon.

Ahora, analicemos el programa que va a utilizar la clase contacto.

El programa requerirá cuatro variables públicas, como se muestra a continuación. Nuestra rutina de constructor automáticamente buscar y cargar el archivo predeterminado de contacto, que hemos llamado contacts.data. No es necesario que el usuario conozca o usar este nombre de archivo, por lo que es codificado en los métodos que se utiliza. Si el `_new()` de rutina del constructor no se puede cargar un archivo, se creará un registro ficticio por lo que el usuario nunca ve una pantalla de inicio en blanco.

' Gambas FMain class file

""Contacts"" representará a la colección de registros para el libro de dirección
PUBLIC Contacts AS Collection

'Ahora definimos una variable del tipo Contact, que es la que usaremos en todas las subrutinas

PUBLIC ContactRecord AS Contact

'índice que señala el registro actual

PUBLIC RecordPointer AS Integer

'necesitaremos un variable de cadena a utilizar cuando el usuario busca por apellido

PUBLIC SearchKey AS String

FMain Constructor

'la rutina de constructor es llamada cuando se llama a FMain_open()

PUBLIC SUB _new()

'crear una instancia de la colección que se rellenara con los registros de datos

Contacts = NEW Collection

'crea una instancia de un nuevo contacto (registro) que está en blanco

ContactRecord = NEW Contact

'Escribimos en el depurador...

PRINT "Calling GetData..."

'Llame al método GetData definido en la definición de clase contacto

ContactRecord.GetData(Contacts)

'establece la variable "RecordPointer", en el último registro que se encuentra en el conjunto de datos

RecordPointer = Contacts.Count

'Escribimos en el depurador...

PRINT "In _new(), rtn fm GetData with: " & RecordPointer & " records."

'el constructor predeterminado garantizará al menos un registro existe

' si la propiedad count de la colección de contactos muestra < 1

IF RecordPointer < 1 THEN

'Agregar nuestros datos ficticios

ContactRecord.FirstName = "John"

ContactRecord.Initial = "Q"

ContactRecord.LastName = "Public"

ContactRecord.Suffix = "II"

ContactRecord.Street1 = "12345 Gambas Drive"

ContactRecord.Street2 = "Apt 101"

ContactRecord.City = "Dancer"

ContactRecord.State = "TX"

ContactRecord.Zip5 = "77929"

ContactRecord.Zip4 = "0101"

ContactRecord.Areacode = "888"

ContactRecord.Prefix = "666"

ContactRecord.Last4 = "4444"

ContactRecord.RecordID = 1

ContactRecord.Deleted = FALSE

RecordPointer = ContactRecord.RecordID

Contacts.Add (ContactRecord, CStr(ContactRecord.RecordID))

ENDIF

```
'Compruebe nuestra colección para asegurarse de que existen registros
IF Contacts.Exist(CStr(RecordPointer)) THEN
'Solamente para depuración
PRINT "Record data exists for " & CStr(RecordPointer) & " records."
'este bucle for es para depurar sólo
FOR EACH ContactRecord IN Contacts
    PRINT ContactRecord.FirstName & " " & ContactRecord.LastName
NEXT
ELSE
'si no hay registros, poner un cuadro de mensaje para informar al usuario
Message.Info("Records do not exist!","Ok")
ENDIF
END
```

Subrutina Form_Open

Nuestro programa abrirá automáticamente el formulario utilizando la subrutina Form_Open() a continuación. Desde FMain.form se define como un formulario de inicio, no es necesario llamar al método FMain.Show. Lo que vamos a hacer aquí es establecer el título para mostrar en la parte superior de la ventana cuando se ejecuta el programa y obtener el registro apuntado por el puntero del registro actual. En este caso, el puntero debe apuntar al último registro de carga desde el archivo de contactos después de flujo del programa se devuelve desde el constructor.

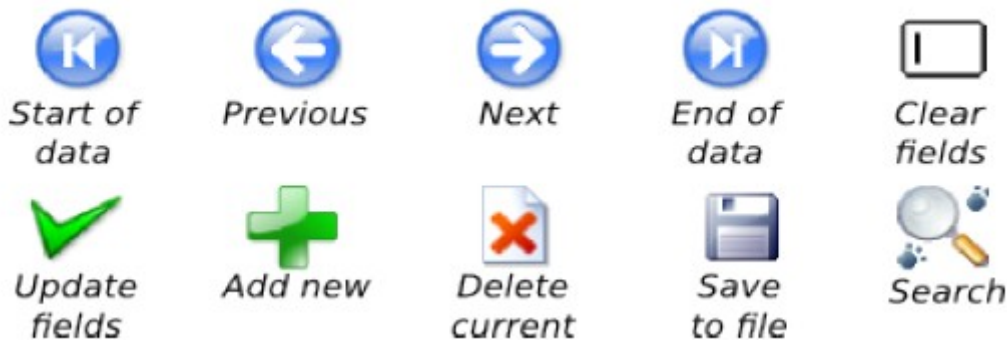
```
PUBLIC SUB Form_Open()
'establecer el título del programa
FMain.Caption = " Contacts Manager "
'iniciar con el registro señalado después de la llamada de constructor, que debería de ser
el ultimo de la coleccion
ContactRecord = Contacts[CStr(RecordPointer)]
'Esta subrutina actualizará todos los campos con datos del registro actuales
UpdateForm
END
```

Añadiendo controles a Fmain.Form.

Es hora de poner todos los controles en el formulario y darles nombre. Estaremos usando nueve ToolButtons, un botón regular, 14 etiquetas y cuadros de 13 texto. También tendremos que utilizar nueve iconos, que podemos obtener de los iconos de cuota de usr/predeterminado-kde/32 x 32 carpeta

(Nota: el sistema puede ser diferente así examinar para encontrar dónde se almacenan los iconos si es necesario).

Los iconos que se utilizan para este programa son los siguientes:



Los ToolButtons

Las nueve ToolButtons que utilizan estos iconos en su propiedad Picture se denominan:

FirstRecordBtn PrevRecordBtn NextRecordBtn

LastRecordBtn ClearBtn Update

AddBtn DeleteRecordBtn SaveAllBtn

Crear nueve ToolButtons y organizarlos en la forma como se muestra al principio de este capítulo. Asignar los nombres mostrados anteriormente a la ToolButtons y establecer el correspondiente botón Propiedades de imagen a los nombres de los iconos que ha elegido para su programa, basado en lo que prevé la distribución de su sistema. Peor de los escenarios es que puede utilizar el editor de icono incorporado en Gambas para crear sus propios iconos.

El botón Salir. (Quit)

Una vez concluida la ToolButtons, agregaremos el botón regular como nuestro botón "Salir". Seleccione un botón de la barra de herramientas y colóquelo en el formulario en la esquina inferior derecha, lo más cerca posible a la figura anterior. Establezca la propiedad Text para que este botón "Salir". Mientras estamos aquí, nos podemos código rutina de salida del programa, que se llamará cuando se hace clic en este botón. Haga doble clic en el botón salir y se colocará en el editor de código. Agregar este código:

PUBLIC SUB QuitBtn_Click()

'preguntar si se desea para guardar los datos antes de que salga

SELECT Message.Question("Save before exit?","Yes", "No", "Cancel")

CASE 1

'Si el usuario dice que sí, salvamos los datos

SaveAllBtn_Click

CASE 2 'no guardar, cerrar sólo

CASE 3 'el usuario a cancelado, retornamos al programa

```
RETURN  
CASE ELSE 'no hacer nada, pero cerrar  
END SELECT  
ME.Close 'aquí cerramos la ventana  
END 'programa parado.
```

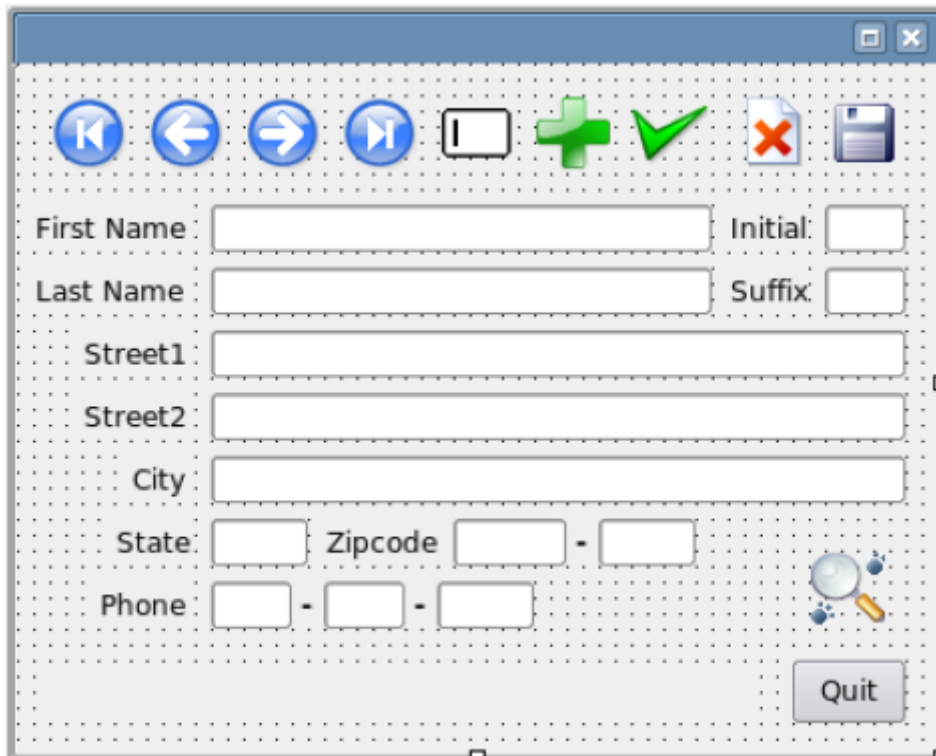
Añadiendo Label and Textbox Controls

Probablemente la parte más tediosa del desarrollo de este programa es a continuación, agregar los controles Label y TextBox al formulario. Intentar organizarlos como se muestra a continuación. Para los controles Label, puede tomar el valor por defecto los nombres de variables previstos por Gambas todas las etiquetas excepto uno en la parte inferior del formulario donde exhibimos nuestro n de registro de datos n. Esa etiqueta debería llamarse StatusLabel. Los campos del cuadro de texto, como se muestra en la parte superior izquierda a la inferior derecha en el formulario, deben ser nombrados como sigue:

```
FirstName  
MI  
LastName  
Suffix  
Street1  
Street2  
City  
State  
Zip5  
Zip4  
AreaCode  
DialPrefix  
DialLast4
```

Es de esperar que los nombres deberían ser fáciles de entender (es decir, el código autoexplicativo) y tiene problemas de creación del formulario. Tenga cuidado al utilizar el caso correcto y escriba los nombres exactamente como se muestra arriba, de lo contrario será ser rastrear variables undeclared cuando se intenta ejecutar el programa. En este punto, su forma, en modo de diseño, quedará así:

A Beginner's Guide to Gambas - Edición Revisada 2011
El programa Contactos



The image shows a screenshot of the Gambas IDE window titled "El programa Contactos". The window contains a contact form with the following fields and controls:

- Navigation and Action Icons:** A row of icons at the top includes a back arrow, a left arrow, a right arrow, a forward arrow, a stop icon, a green plus sign, a green checkmark, a red X, and a floppy disk icon.
- Form Fields:**
 - First Name:
 - Initial:
 - Last Name:
 - Suffix:
 - Street1:
 - Street2:
 - City:
 - State: Zipcode: -
 - Phone: - -
- Search and Quit:** A magnifying glass icon is located to the right of the form fields. A "Quit" button is located in the bottom right corner of the window.

Subrutina UpdateForm()

La rutina UpdateForm() se llamará cada vez que los datos en el formulario ha cambiado y necesita actualizarse por lo que el usuario verá las actualizaciones que se produzcan. Esta rutina puestos de cada campo del formulario con los datos actuales que se encuentran en la colección basada en el puntero del registro actual. Además, la etiqueta StatusLabel se actualiza con el número de registro actual de n registros en la colección desde aquí. Vamos a recorrer el código:

PUBLIC SUB UpdateForm()

'Compruebe que hay registros antes de hacer nada

IF Contacts.Exist(CStr(RecordPointer)) THEN

'hemos encontrado registros, y ahora nos movemos al registro actual puntero

ContactRecord = Contacts[CStr(RecordPointer)]

'para depuración solamente

PRINT "In UpdateForm with RecordPointer of: " & RecordPointer

'asignamos los valores de los datos de contactos a cada Textbox.Text

FirstName.Text = ContactRecord.FirstName

MI.Text = ContactRecord.Initial

LastName.Text = ContactRecord.LastName

Suffix.Text = ContactRecord.Suffix

Street1.Text = ContactRecord.Street1

Street2.Text = ContactRecord.Street2

City.Text = ContactRecord.City

State.Text = ContactRecord.State

Zip5.Text = ContactRecord.Zip5

Zip4.Text = ContactRecord.Zip4

AreaCode.Text = ContactRecord.Areacode

DialPrefix.Text = ContactRecord.Prefix

DialLast4.Text = ContactRecord.Last4

'actualizar el campo de registro de la situación actual con los datos actuales

StatusLabel.Text = "Record: " & CStr(RecordPointer)

StatusLabel.Text = StatusLabel.Text & " of " & CStr(Contacts.Count)

'llamar al método Refresh para repintar el contenido del formulario

FMain.Refresh

'seleccionar el contenido del campo Nombre

FirstName.Select

'establecer el foco del cursor en el campo Nombre

FirstName.SetFocus

ELSE ' i no existen datos !

'para depuración...

PRINT "In UpdateForm, Record: " & RecordPointer & " not found."

ENDIF

'para depuración...

PRINT "Leaving UpdateForm..."

END



Codificando los botones: Primero , Previo, Siguiente y Ultimo. (First, Prev, Next, and Last)

El primero es el que nos llevará al principio de la de datos. Haga doble clic en el ToolButton primero y debe ser llevado a la ventana de código, donde la FirstRecordBtn_Click () rutina espera. Añadir el siguiente código:

```
PUBLIC SUB FirstRecordBtn_Click()  
'mueve al primer datos  
RecordPointer = 1  
'recupera el dato de la colección  
ContactRecord = Contacts[CStr(RecordPointer)]  
'actualizar la presentación de formulario con los datos nuevos  
UpdateForm  
END
```

Como puede ver, moviéndose en el primer registro fue muy sencillo. Ahora, vamos a codificar la Anterior y Siguiente. Haga doble clic en el segundo ToolButton (←) y se le llevará a la PrevRecordBtn_Click () subrutina. Añadir el siguiente código:

```
PUBLIC SUB PrevRecordBtn_Click()  
'solamente para depuracion...  
PRINT "in Prev: current pointer is: " & RecordPointer  
'nos estamos moviendo a un datos inferior, por lo disminuimos el puntero de registro.  
DEC RecordPointer  
'Si llegamos a 0, tenemos que poner el 1º registro.  
IF RecordPointer = 0 THEN  
RecordPointer = 1  
ENDIF  
'actualizar nuestra varaible contacto con los datos del registro nuevo (prev)  
ContactRecord = Contacts[CStr(RecordPointer)]  
'si ya estamos en el comienzo de los datos, se lo notificamos al depurador  
IF RecordPointer = 1 THEN  
'solamente para el depurador:  
PRINT "At first contact already!"  
ELSE  
'solamente para el depurador:  
PRINT "in Prev - now current pointer is: " & RecordPointer  
ENDIF
```


'actualizar nuestro formulario con los datos nuevos

```
UpdateForm  
END
```

Ahora, vamos a código en el botón Siguiente. Haga doble clic en el tercer ToolButton (→) y se le llevará a la subrutina NextRecordBtn_Click (). Añadir el siguiente código:

```
PUBLIC SUB NextRecordBtn_Click()
```

```
'escribimos en el depurador...
```

```
PRINT "in Next: current pointer is: " & RecordPointer
```

```
'Nosotros vamos a avanzar una posición en la colección, incrementamos el puntero.
```

```
INC RecordPointer
```

```
'si nos movemos mas alla del ultimo registro, nos quedamos en el último
```

```
IF RecordPointer > Contacts.Count THEN
```

```
RecordPointer = Contacts.Count
```

```
ENDIF
```

```
'actualizar el registro de contacto con los nuevos datos
```

```
ContactRecord = Contacts[CStr(RecordPointer)]
```

```
'si ya estamos en el último registro, avisar al usuario
```

```
IF RecordPointer = Contacts.Count THEN
```

```
'escribimos en el depurador...
```

```
PRINT "At Last contact already!"
```

```
ELSE
```

```
'escribimos en el depurador...
```

```
PRINT "in Next, the current pointer is: " & RecordPointer
```

```
ENDIF
```

```
'actualizar nuestra forma de nuevo con los nuevos datos
```

```
UpdateForm
```

```
END 'final del mover siguiente
```

El ToolButton último que necesitamos en el código para moverse por los datos de la subrutina LastRecordBtn_Click (). Esta subrutina nos colocará al final de la recolección de datos, en el último registro. Haga doble clic en el ToolButton cuarto y agregue este código:

```
PUBLIC SUB LastRecordBtn_Click()
```

```
'asignamos el valor del ultimo registro a RecordPointer usando la propiedad  
collection.count
```

```
RecordPointer = Contacts.Count
```

```
'actualizar nuestro registro var con los nuevos datos para el último registro
```

```
ContactRecord = Contacts[CStr(RecordPointer)]
```

```
'actualizar el formulario con los datos nuevos
```

```
UpdateForm
```

```
END
```



Codificando boton: Añadir datos.

Ahora, vamos a ver la forma de agregar un nuevo contacto a la colección. Haga doble clic en el quinto ToolButton, el icono de persona poco, y se le puso en la ventana de código en el subrutina AddBtn_Click () . Vamos a pasar por el código:

PUBLIC SUB AddBtn_Click()

'Declaramos una variable tipo "contact", de forma local para ser solo usada en esta subrutina.

DIM MyContactRecord AS Contact

'instanciamos el registro para poderlo usar:

MyContactRecord = NEW Contact

'si no hay un nombre, no vamos a añadir un nuevo registro

IF FirstName.Text <> "" THEN

'escribimos en el depurador...

PRINT "Adding " & FirstName.Text & " to collection."

'Como hay un nombre, asignamos los textos contenidos en los distintos textbox a los campos de la clase:

MyContactRecord.FirstName = FirstName.Text

MyContactRecord.LastName = LastName.Text

MyContactRecord.Initial = MI.Text

MyContactRecord.Suffix = Suffix.Text

MyContactRecord.Street1 = Street1.Text

MyContactRecord.Street2 = Street2.Text

MyContactRecord.City = City.Text

MyContactRecord.State = State.Text

MyContactRecord.Zip5 = Zip5.Text

MyContactRecord.Zip4 = Zip4.Text

MyContactRecord.Areacode = AreaCode.Text

MyContactRecord.Prefix = DialPrefix.Text

MyContactRecord.Last4 = DialLast4.Text

' la clave de registro de una colección debe de ser única. Para ello vamos a incrementar en 1 el contador. Gambas exige que la variable "clave" debe de ser una cadena, por lo tanto, lo que hacemos es convertir el numero en cadena con la expresión CStr(RecordPointer)

MyContactRecord.RecordID = Contacts.Count + 1

'si añadimos un registro, no lo marcamos como borrado.

MyContactRecord.Deleted = FALSE

'asignamos a la variable global RecordPointer, el valor del incremento.

RecordPointer = MyContactRecord.RecordID

```
'ahora, llamamos al método Collection.Add para añadir el dato de la clase MyContactRecord.
Contacts.Add (MyContactRecord, CStr(RecordPointer) )
'las siguientes líneas son para escribir en el depurador...
IF Contacts.Exist(CStr(RecordPointer)) THEN
'escribimos en el depurador...
PRINT "Record " & CStr(RecordPointer) & " added!"
ELSE
'escribimos en el depurador...
PRINT "Record does not exist!"
ENDIF
ELSE 'No se puede agregar un contacto sin un nombre, le digo al usuario
Message.Info("Cannot add without a first name", "Ok")
ENDIF
'actualizar el formulario con los datos nuevos
UpdateForm
'esto es para la depuración, mostrará todos los registros de la colección.
FOR EACH MyContactRecord IN Contacts
    PRINT "Record " & CStr(RecordPointer) & ": ";
    PRINT MyContactRecord.FirstName & " " & MyContactRecord.LastName
NEXT
END 'fin de la subrutina add
```



*Clear
fields*

Codificando Boton: Limpiar datos.

El sexto ToolButton se utiliza para borrar los datos en el formulario para permitir a los usuarios comenzar con una forma limpia, (en blanco) para introducir datos. Simplemente deja en blanco cada campo de texto. Haga doble clic en el ClearBtn y escriba este código:

```
PUBLIC SUB ClearBtn_Click()  
FirstName.Text = ""  
MI.Text = ""  
LastName.Text = ""  
Suffix.Text = ""  
Street1.Text = ""  
Street2.Text = ""  
City.Text = ""  
State.Text = ""  
Zip5.Text = ""  
Zip4.Text = ""  
AreaCode.Text = ""  
DialPrefix.Text = ""  
DialLast4.Text = ""  
END
```



Validar la información del usuario

Cuando el usuario introduce los datos en el campo inicial, y también en el sufijo, Estado, Zip5, Zip4, AreaCode, DialPrefix y campos DialLast4, queremos asegurarnos de que los datos se ajusta y no se puede escribir letras o caracteres extraños.

Cada una de las rutinas a continuación se activan en un Evento de cambio (`_Change`). En otras palabras, si el texto en el campo cambia `TextBox.Text`, irá a la `_Change` respectivos (`MI`) siguiente proceso.

Cada rutina funciona casi igual al límite de la número de caracteres que el usuario puede escribir. Si se escribe más, los caracteres adicionales se truncan en la longitud especificada por el programa. Por ejemplo, un código de área no puede ser más de tres dígitos. También se podría añadir el código para asegurarse de dígitos sólo se introducen en los campos numéricos, etc .

Por cada chequeo a continuación, vamos a necesitar una variable de cadena temporal, `sTemp`. Si la longitud de los datos `TextBox.Text` excede la longitud que permiten, usamos `MID $` copiamos las primeras `n` letras a `sTemp` y luego copiamos ese valor de nuevo en la propiedad `TextBox.Text`.

```
PUBLIC SUB MI_Change()  
DIM sTemp AS String  
IF Len(MI.Text) > 1 THEN  
    sTemp = Mid$( MI.Text, 1, 1)  
    MI.Text = sTemp  
ENDIF  
END  
PUBLIC SUB Suffix_Change()  
DIM sTemp AS String  
IF Len(Suffix.Text) > 3 THEN  
    sTemp = Mid$( Suffix.Text, 1, 3)  
    Suffix.Text = sTemp  
ENDIF  
END  
PUBLIC SUB State_Change()  
DIM sTemp AS String  
IF Len(State.Text) > 2 THEN  
    sTemp = Mid$( MI.Text, 1, 2)  
    State.Text = sTemp  
ENDIF
```

```
END
PUBLIC SUB Zip5_Change()
DIM sTemp AS String
IF Len(Zip5.Text) > 5 THEN
    sTemp = Mid$( MI.Text, 1, 5)
    Zip5.Text = sTemp
ENDIF
END
PUBLIC SUB Zip4_Change()
DIM sTemp AS String
IF Len(Zip4.Text) > 4 THEN
    sTemp = Mid$( MI.Text, 1, 4)
    Zip4.Text = sTemp
ENDIF
END
PUBLIC SUB AreaCode_Change()
DIM sTemp AS String
IF Len(AreaCode.Text) > 3 THEN
    sTemp = Mid$( AreaCode.Text, 1, 3)
    AreaCode.Text = sTemp
ENDIF
END
PUBLIC SUB DialPrefix_Change()
DIM sTemp AS String
IF Len(DialPrefix.Text) > 3 THEN
    sTemp = Mid$( DialPrefix.Text, 1, 3)
    DialPrefix.Text = sTemp
ENDIF
END
PUBLIC SUB DialLast4_Change()
DIM sTemp AS String
IF Len(DialLast4.Text) > 4 THEN
    sTemp = Mid$( DialLast4.Text, 1, 4)
    DialLast4.Text = sTemp
ENDIF
END
```

Como puede ver, gran cantidad de código repetitivo, pero es necesario.

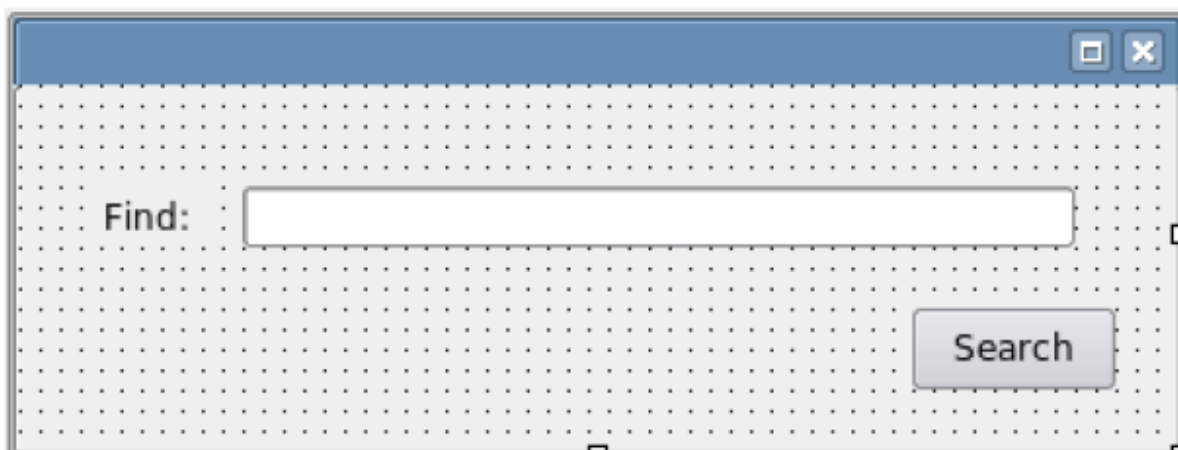


Agregar una característica de búsqueda

Queremos que el usuario sea capaz de buscar un contacto mediante la búsqueda de un **apellido**. Hay muchos controles en el formulario ya que la adición de otro iría en detrimento de la estética en general y hacer que parezca menos que elegante. Una mejor solución es utilizar un solo clic del ratón en cualquier parte de la misma forma. Se genera un evento y nos permitirá capturar ese evento e iniciar un proceso de búsqueda. Cuando esto ocurre, vamos a aparecer un formulario de búsqueda personalizado y pedir al usuario que escriba el apellido a buscar. Aquí está el código para el evento del ratón:

```
PUBLIC SUB Form_MouseDown()  
SearchForm.ShowModal  
Message.Info("Back to main form with: " & SearchKey, "Ok")  
DoFind  
END
```

Tenga en cuenta que se muestra el formulario mediante el método ShowModal en lugar de llamar a Show. Esto se debe a que queremos que el usuario introduzca algo y haga clic en el botón Buscar. Esto es lo que la misma forma se parece en modo de diseño:
evento por:



Es bastante básico, sólo una etiqueta, un cuadro de texto y un botón. Desde el IDE de crear un nuevo formulario, llamado SearchForm y añadir la etiqueta y el botón como se muestra a continuación. El nombre del botón SearchBtn y el cuadro de texto debe ser nombrado

A Beginner's Guide to Gambas – Edición Revisada 2011
El programa Contactos

SearchInput. Aquí está todo el código que se necesita para el SearchForm:

```
' Gambas SearchForm class file
```

```
PUBLIC SUB SearchBtn_Click()
```

```
'Este es un llamado a nuestro módulo de búsqueda personalizada, el cual tiene un solo método, SearchOn, que se explica a continuación
```

```
Search.SearchOn( SearchInput.Text)
```

```
'escribimos en el depurador...
```

```
PRINT "returning: " & SearchInput.Text
```

```
'asignar la clave de búsqueda a la var global, nombrado en FMain
```

```
FMain.SearchKey = SearchInput.Text
```

```
'Ahora cerramos el formulario de búsqueda
```

```
SearchForm.Close
```

```
END
```

```
PUBLIC SUB Form_Open()
```

```
'establecemos el título de la ventana de búsqueda
```

```
SearchForm.Caption = " Find by last name "
```

```
'resaltar y seleccionar el cuadro de texto SearchInput ya que sólo hay un campo de entrada, que tiene el foco
```

```
SearchInput.Select
```

```
END
```

En este punto, puede que se pregunte qué está pasando. Básicamente, en Gambas, las variables públicas

son públicas a un archivo de clase. Cuando queremos pasar parámetros entre las clases, podemos utilizar el método de paso de parámetros con funciones como lo hicimos cuando empezamos el programa y se pasa una colección vacía variable al método GetData.

Otra forma de conseguir esto es mediante el uso de módulos. El punto clave a recordar es que las variables públicas en módulos son pública a todas las clases.

En esencia, todo lo que hemos hecho es crear un canal para pasar la palabra de la búsqueda de la parte posterior formulario de búsqueda para el programa principal a través del módulo. Tenga en cuenta que el único código necesarios en el módulo es una declaración que toma el parámetro que se pasa desde el formulario de búsqueda y vuelve al programa principal. Tan pronto como regrese del módulo, se le asigna a la clase SearchKey FMain variable pública. Tenga en cuenta que se completa el nombre de la variable, especificando FMain.SearchKey para asegurarse de que se resuelve correctamente. Esto es necesario porque la SearchButton_Click () subrutina que invoca el módulo está en un archivo de clase diferente a la FMain archivo de clase que lo llamó.

```
' Gambas module file
```

```
'PUBLIC SearchKey AS String
```

```
PUBLIC FUNCTION SearchOn(sKey AS String) AS String
```

```
RETURN sKey
```

```
END
```


La subrutina Buscar (DoFind)

At this point, we have popped up the form, obtained the user input of a last name to search for, and have passed that variable back to our program using the module conduit method described above. Now, we are ready to actually search the collection and find the entry. Here is the code to do that:

```
PUBLIC SUB DoFind()  
'declaramos un objeto contact para uso local  
DIM MyContactRecord AS Contact  
'lo instanciamos  
MyContactRecord = NEW Contact  
'asignamos el RecordPointer al primer dato de la collection  
RecordPointer = 1  
'escribimos en el depurador...  
PRINT "In DoFind with: " & SearchKey  
'usamos FOR EACH para cada irretaccion de la colección Contacts  
FOR EACH MyContactRecord IN Contacts  
'asignamos cada dato temporal a nuestro objeto global contact (contactRecord)  
    ContactRecord = Contacts[CStr(RecordPointer)]  
'si el apellido (ultimo nombre, lasName) coincide con la busqueda  
    IF ContactRecord.LastName = SearchKey THEN  
'escribimos en el depurador...  
        PRINT "Found: " & MyContactRecord.LastName  
'ahora hacemos update al formulario  
        UpdateForm  
        BREAK 'fuerza el retorno. force return with this statement  
    ENDIF  
'no encontrado match asi que nosotros incrementamos el puntero de datos y movemos al siguiente.  
    INC RecordPointer  
NEXT  
END
```

Editando un dato existente:

Si un usuario ha editado un dato que ya existe, nosotros tendremos que salvar ese dato....

Updating an Existing Record

If the user happens to edit data on an existing record when it is displayed, we want the **data to be saved to the same record. To accomplish this using a collection object is** complicated by the fact that each object in the collection must have a unique key. To get around this, we will delete the object with the current key using the built-in method Remove, and re-add the object using the same key but with updated data. Sort of sneaky, but it gets the job done without much effort. Here is how it works:

PUBLIC SUB Update_Click()

'declaramos una variable local conct para trabajar con ella

DIM MyContactRecord AS Contact

'la instanciamos

MyContactRecord = NEW Contact

'ahora, borramo el dato existente de la colección.

Contacts.Remove(CStr(RecordPointer))

'emtonces, con los datos del formulario actualizamos nuestra clase contact
(MyContactRecord)

then we populate our work record with the form data

MyContactRecord.FirstName = FirstName.Text

MyContactRecord.LastName = LastName.Text

MyContactRecord.Initial = MI.Text

MyContactRecord.Suffix = Suffix.Text

MyContactRecord.Street1 = Street1.Text

MyContactRecord.Street2 = Street2.Text

MyContactRecord.City = City.Text

MyContactRecord.State = State.Text

MyContactRecord.Zip5 = Zip5.Text

MyContactRecord.Zip4 = Zip4.Text

MyContactRecord.Areacode = AreaCode.Text

MyContactRecord.Prefix = DialPrefix.Text

MyContactRecord.Last4 = DialLast4.Text

'añadimos a la colección, usando el que que anteriormente borramos

'add the work copy to the collection with the same

'key that belonged to the record we deleted above

Contacts.Add (MyContactRecord, CStr(RecordPointer))

IF Contacts.Exist(CStr(RecordPointer)) THEN

'escribimos en el depurador...

PRINT "Record " & CStr(RecordPointer) & " updated!"

ELSE

PRINT "Record not updated!" 'escribimos que ha fallado

ENDIF

'update el formulario con el nuevo dato.

A Beginner's Guide to Gambas - Edición Revisada 2011
El programa Contactos

UpdateForm
END



Borrando un dato de Contacto.

Si el usuario quiere borrar un registro dentro de la colección, también tenemos que pensar en cómo manejar eso.

Administrar qué teclas están disponibles y cuales no, podría ser pesadilla del programador. Para un enfoque más sencillo es simplemente marcar el registro como eliminado y lo limpio. De este modo, está disponible para su reutilización con la misma clave. Limpiar los datos y, cuando se produce el guardar, no almacena los datos antiguos, eliminarlo eficazmente. La única diferencia es que nosotros podemos reutilizar la clave por la que el usuario simplemente introduzca datos en un registro vacío y haga clic en el botón Actualizar.

Así es cómo esto se logra:

```
PUBLIC SUB DeleteRecordBtn_Click()  
'declaramos un objeto contact par usarlo de modo local  
DIM MyContactRecord AS Contact  
' lo instanciamos  
MyContactRecord = NEW Contact  
' borramos el dato actual usando el metodo Remove.  
Contacts.Remove(CStr(RecordPointer))  
'limpia todos los campos del formulario  
ClearBtn_Click  
'el primer campo le asignamo un indicador que el dato (registro) esta borrado.  
MyContactRecord.FirstName = "<Deleted Record>"  
'dejamos el resto de campos en blanco  
MyContactRecord.LastName = LastName.Text  
MyContactRecord.Initial = MI.Text  
MyContactRecord.Suffix = Suffix.Text  
MyContactRecord.Street1 = Street1.Text  
MyContactRecord.Street2 = Street2.Text  
MyContactRecord.City = City.Text  
MyContactRecord.State = State.Text  
MyContactRecord.Zip5 = Zip5.Text  
MyContactRecord.Zip4 = Zip4.Text  
MyContactRecord.Areacode = AreaCode.Text  
MyContactRecord.Prefix = DialPrefix.Text  
MyContactRecord.Last4 = DialLast4.Text
```

A Beginner's Guide to Gambas - Edición Revisada 2011
El programa Contactos

' marcamos el dato como borrado

MycontactRecord.Deleted = TRUE

'añadimos el registro, borrando el datos como existiera dentro de la colección.

Contacts.Add (MyContactRecord, CStr(RecordPointer))

IF Contacts.Exist(CStr(RecordPointer)) THEN

'para depuracion solamente...

PRINT "Record " & CStr(RecordPointer) & " marked deleted!"

ELSE

'si hay error, entonces muestra un mensaje al usuario

Message.Info("Record not marked deleted!","Ok")

ENDIF

'update el formulario

UpdateForm

END



Guardando los datos:

La última cosa que necesitamos es el botón de guardar. Es muy simple porque tenemos el método necesario en la clase Contact. Aquí nosotros llamaremos al método PutData.:

```
PUBLIC SUB SaveAllBtn_Click()  
'llamamos al metodo PutData  
ContactRecord.PutData(Contacts)  
'escribimos en el depurador...  
PRINT "Contacts saved to contact.data file."  
END
```

Esto es todo lo que tenemos que hacer. Correr el programa e introducir datos.

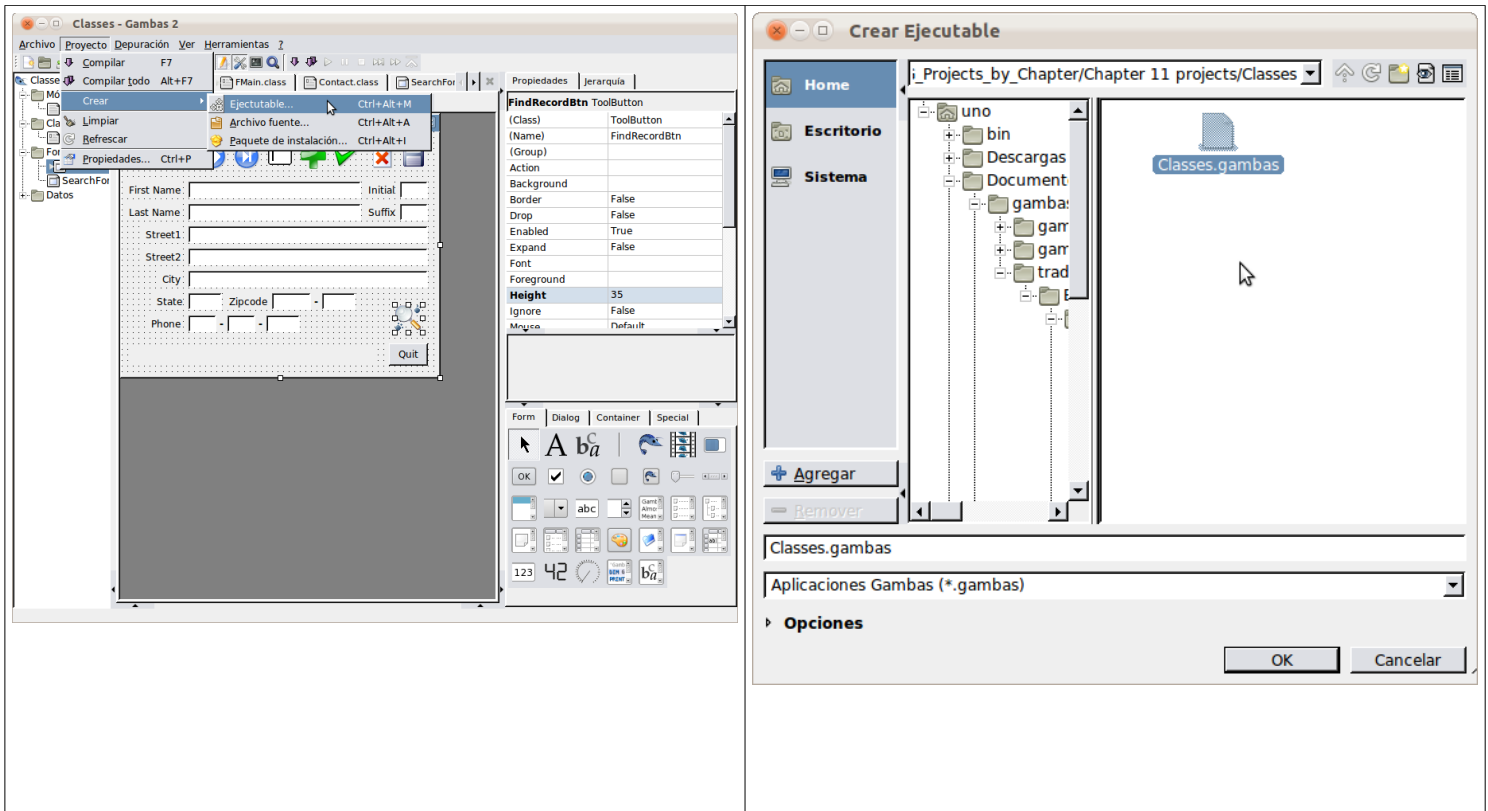
Asegúrese de probar las opciones de salas y crear unos registros. Guardar los datos de archivo y cargarlo. Cuando esté satisfecho que funciona como se planeó, está bien volver y comentario a todos la consola de salida (las declaraciones de impresión) por lo que no se mostrarán en tiempo de ejecución.

Ahoramismo, solamente lo podemos ejecutar en el IDE de gambas. Puesto que este programa puede ser una adición útil a tu escritorio, ahora le mostraremos cómo crear un archivo ejecutable independiente y permitir que se ejecute independiente del entorno de programación (IDE) de Gambas.

Creación de un archivo ejecutable independiente

Es muy fácil crear el ejecutable. Simplemente asegúrese de que su programa está listo para el lanzamiento (este probado y sin errores!).

Y luego ir al menú Proyecto y elija la opción de ejecutables. El IDE de Gambas va a crear una secuencia de comandos de ejecución que se invoque el intérprete de Gambas y le permitirá ejecutar su código.



Usted tendrá que crear el enlace a la aplicación en de escritorio, que señale donde este el archivo ejecutable donde lo haya creado.

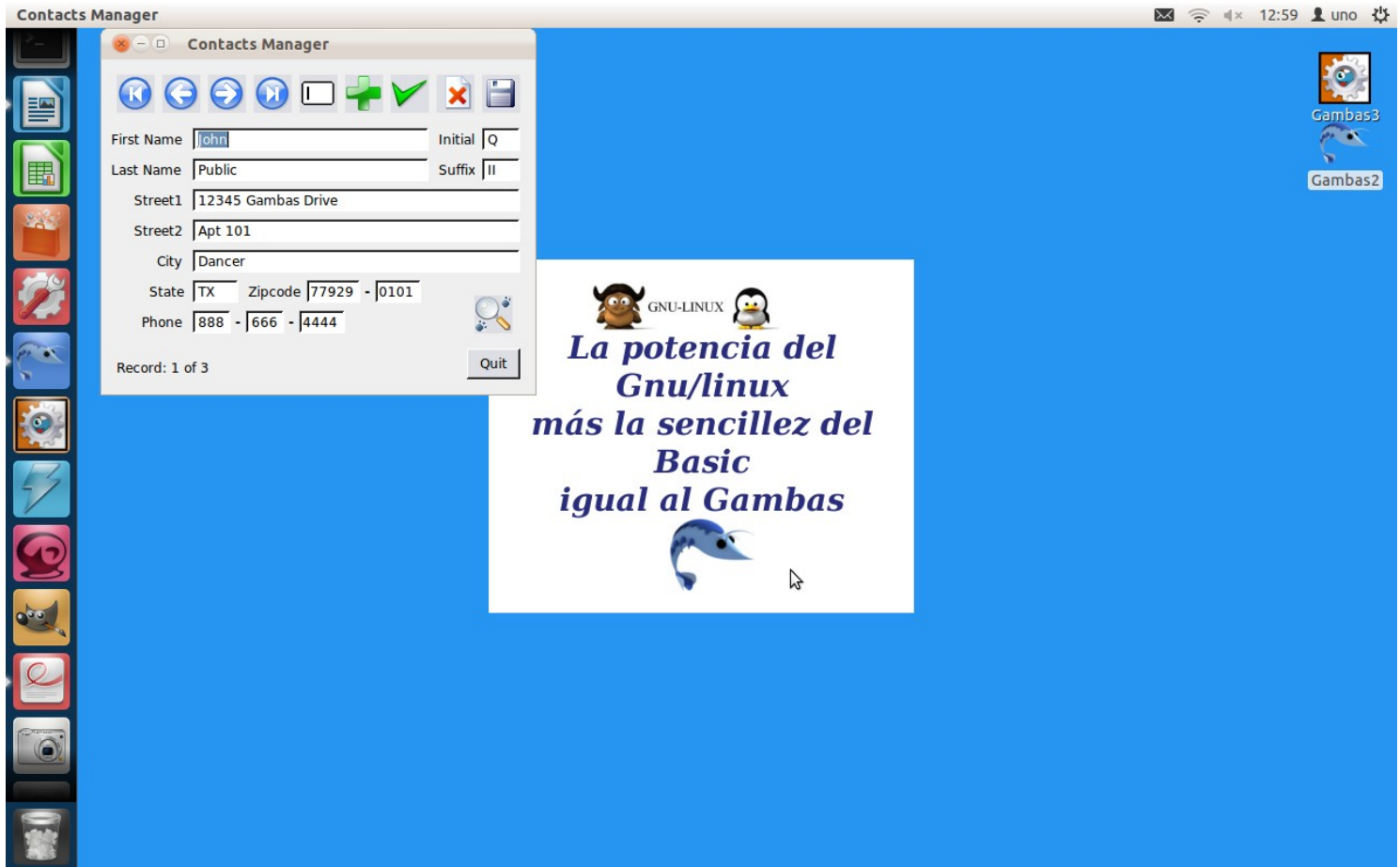
Para ejecutar el programa en la terminal (o consola), ejecute la orden.

```
$/Classes.gambas
```

en el directorio donde este el archivo Classes.gambas.

El resultado final

Aquí está el resultado final, una vez creado el ejecutable:



Fuentes y enlaces interesantes:

<http://beginnersguidetogambas.com/>

<http://gambasdoc.org/help/?es&v3>

<http://gambas.sourceforge.net/en/main.html>

<http://www.gambas-es.org/>

<http://jsbsan.blogspot.com>