

Developing applications with Gambas.

Tutorial and an example made with Gambas.

Summary: We are going to create a simple program with Gambas. We will study how to handle the events and some tips and techniques for working with this wonderful tool.

Thanks to: Daniel Oxley and Dave Sharples. They have reviewed and corrected my poor English translation from Spanish.

The downloadable source code (comments, variable names, etc.) and the screenshots are in Spanish. However, the code showed in this page is translated.

David Asorey Álvarez. February de 2005.

- [Introduction](#)
- [First steps](#)
- [Handling events](#)
- [Designing forms](#)
- [Dive in ...](#)
- ["Clean" action](#)
- ["Add" action](#)
- ["Modify" action](#)
- ["Delete" action](#)
- ["Exit" action](#)
- ["Open" action](#)
- ["Save" action](#)
- [Last adjustments](#)
- [Our program running](#)
- [Deploying our program](#)
- [conclusions](#)
- [About this document and the author](#)
- [Remarks](#)

Introduction

Gambas is an IDE ("Integrated Development Environment") oriented towards RAD ("Rapid Applications Development"), like the popular proprietary programs Microsoft Visual Basic or Borland Delphi.

With Gambas, it is possible to develop GUI programs ("Graphical User Interface") very quickly, because the IDE includes a form designer, a code editor, a code explorer, an integrated help system, etc.

These kind of tools are very common in the Microsoft Windows world, but, on the Linux platform there are only a few, or they are in early stages of development. We find some, like Kdevelop, Kylix or VDK Builder. There is a long tradition and habit in the Linux/Unix world of using a lot of different tools each specialised in a single concrete task (ex. a compiler, an editor, a debugger, each one used separately). For this reason, programming IDEs are relative new to Linux/Unix users.

There are a lot of programmers and developers who are used to these kinds of integrated tools, because they come from other platforms or they feel more comfortable with an IDE.

Gambas tutorial

In the author's words, Benoît Minisini, "*Gambas aims at enabling you to make powerful programs easily and quickly. But clean programs remain your own responsibility...*". The programming language used in Gambas is a version of the "old" BASIC. It is possible that someone will be surprised by this choice as BASIC seems to be very simple and limited. We have to remember that one objective in Gambas is to facilitate programming by non-programmers.

The aim of this tutorial is an introduction to the Gambas IDE and to develop a simple program with it, but we assume that the reader has some familiarity with programming and that terms like *function*, *event* or *variable* are known. Gambas has an integrated help system that provides both introductory and more advanced documentation.

The Gambas version used in this tutorial is 1.0-1. The home page of Gambas is <http://gambas.sourceforge.net>



Download the source code for the example program: [agenda.tar.gz](#)

This tutorial in pdf format: [gambas_tutorial_en.pdf](#)

First steps

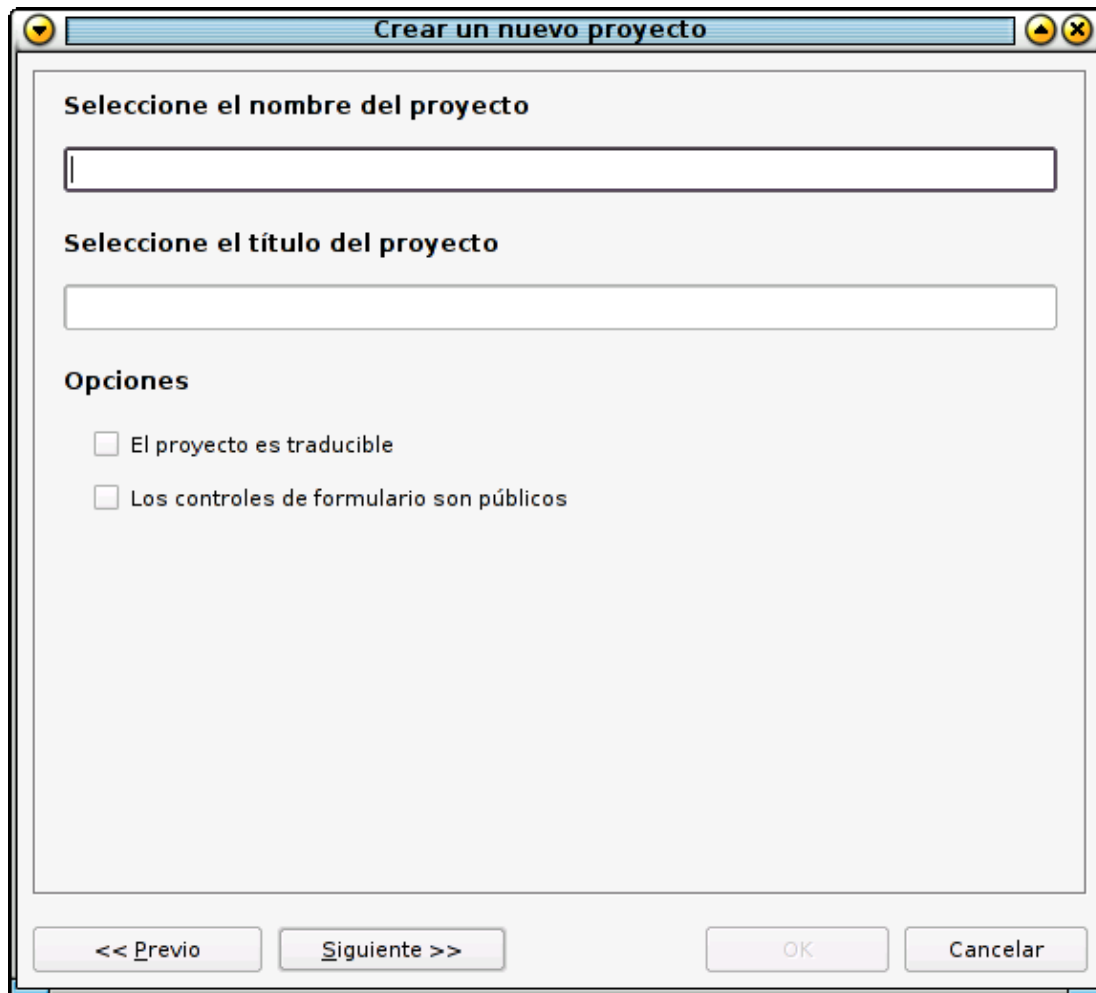
In the Gambas' help there is a document titled "Visual Introduction to Gambas" that explains the program, its different tools, windows and menus. We aren't going to repeat this information in this tutorial.

We will try to develop a complete program with Gambas from the beginning, and we will tackle the requirements and problems as they appear.

Gambas tutorial

Our program will be a notebook or reminder type application. We may add and delete notes and modify the existing entries. The notes may be saved and restored from a file.

Once Gambas has opened, we will select "New project" from the menu. We choose a "graphical project" in the wizard and we then have to provide some information such as the name for the project and the projects title:



The image shows a dialog box titled "Crear un nuevo proyecto". It has a standard window title bar with a yellow minimize button, a yellow maximize button, and a red close button. The dialog content is organized into three sections. The first section, "Seleccione el nombre del proyecto", features a single-line text input field. The second section, "Seleccione el titulo del proyecto", also features a single-line text input field. The third section, "Opciones", contains two checkboxes, both of which are unchecked: "El proyecto es traducible" and "Los controles de formulario son públicos". At the bottom of the dialog, there are four buttons: "<< Previo", "Siguiete >>", "OK", and "Cancelar".

There are two options: "Project is translatable" and "Form controls are public". We leave these options unchecked and choose "Next".

The last task in the wizard is to select a directory where the project will be saved. The main Gambas IDE will then open. In the main project window right-click on "Forms" and we select "New form".

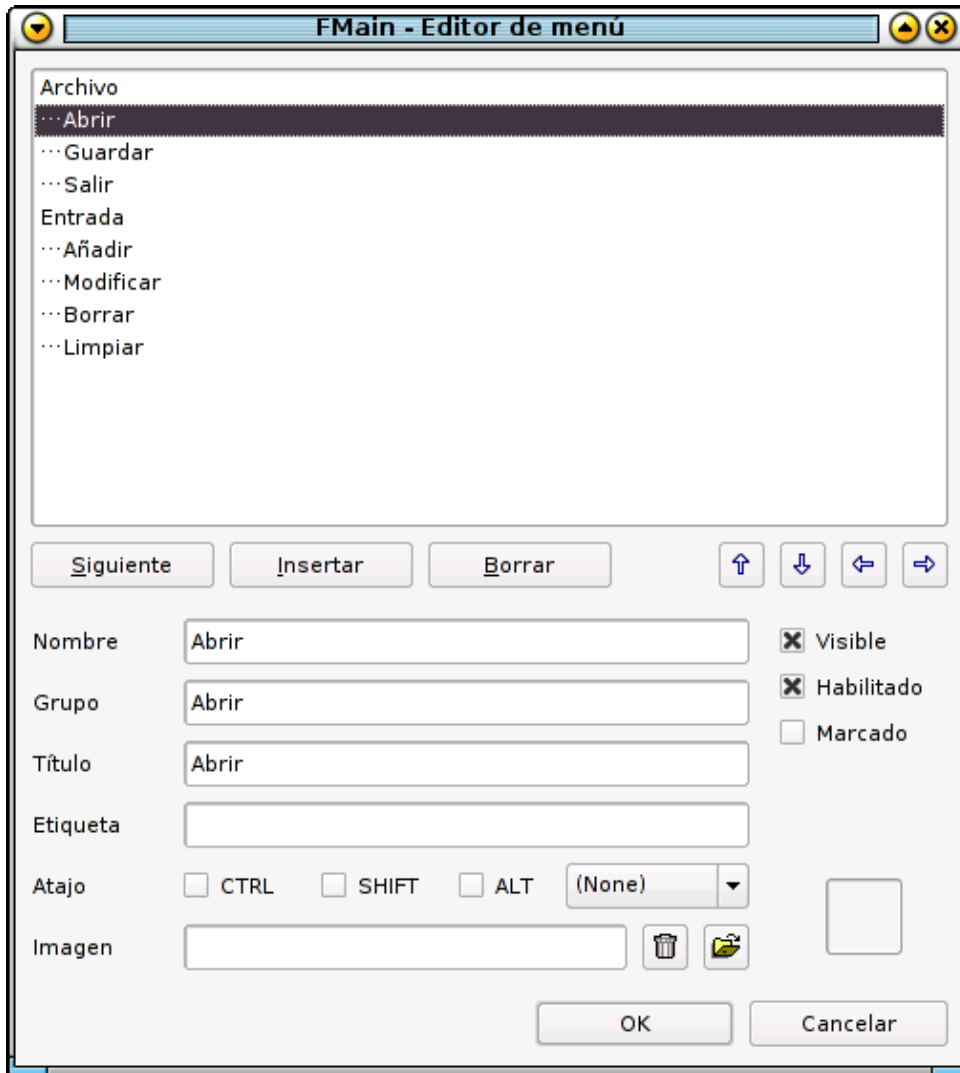
We are going to design the main form, that contains a "ListBox" control and some buttons (open, save, exit, add, modify, delete, ...). Something like this:



We can see some common controls: a Label, a ListBox and some Buttons. We add each controls to the form by selecting them in the Gambas' "Toolbox" and "drawing" the outline of the control on the form. Please note that the buttons "Open", "Save" and "Exit" ("Abrir", "Guardar" and "Salir" in the screenshot) are positioned within a 'Panel' which must be added to the form first and not directly on the main body of the form.

If we want to set keyboard shortcuts, we have to prepend the "ampersand" symbol (&) to the desired letter in the text of the button. example – *A&brir, &Guardar*

We may create and edit our program's menu bar by right-clicking on an empty area of the form and selecting "Menu editor":



When creating the menu entries you will notice that besides the typical properties like name, caption, etc. there is also a property called "Group", form buttons also have this "Group" property as you can see by selecting a button and viewing its properties in the Properties window. This option is very interesting, because we can associate the same code to different controls with the same function, for example, the menu entry "Open" and the button "Open" must do the same task: open a file from disk. Using the "Group" property, we write the code for this task only once.

In our program, we will associate the button and the menu "Open" to a group called "Open", the "Save" menu and button to a group called... "Save", etc.

Now, if we double-click on a button or the equivalent menu entry, the code editor will be opened and the cursor placed, in the header of a subroutine named after the common group chosen for the controls associated with it. For example: `PUBLIC SUB Open_Click()`, where Open is the property of the "Group" named Open, previously defined.

Event handling

Programs with a graphical user interface are usually "event driven". That is, once the user "makes something happen" in the application window, an event is generated. This event can be associated to a function or

subroutine that responds to the user's action.

An example: if the user "clicks" on a control, some events are generated: *MousePress*, when pressing the mouse's button, *MouseRelease* and, finally, *Click* as the global action. If the user double-clicks, then the event generated is *DbClick*. Not all controls respond to or generate events, it is a nonsense to speak about the event *Resize* in a button, because that event is usually generated when we resize a window (a "Form" control).

In Gambas, we will edit the procedure's code for an event (1) like this:

```
PUBLIC SUB Control_Event
```

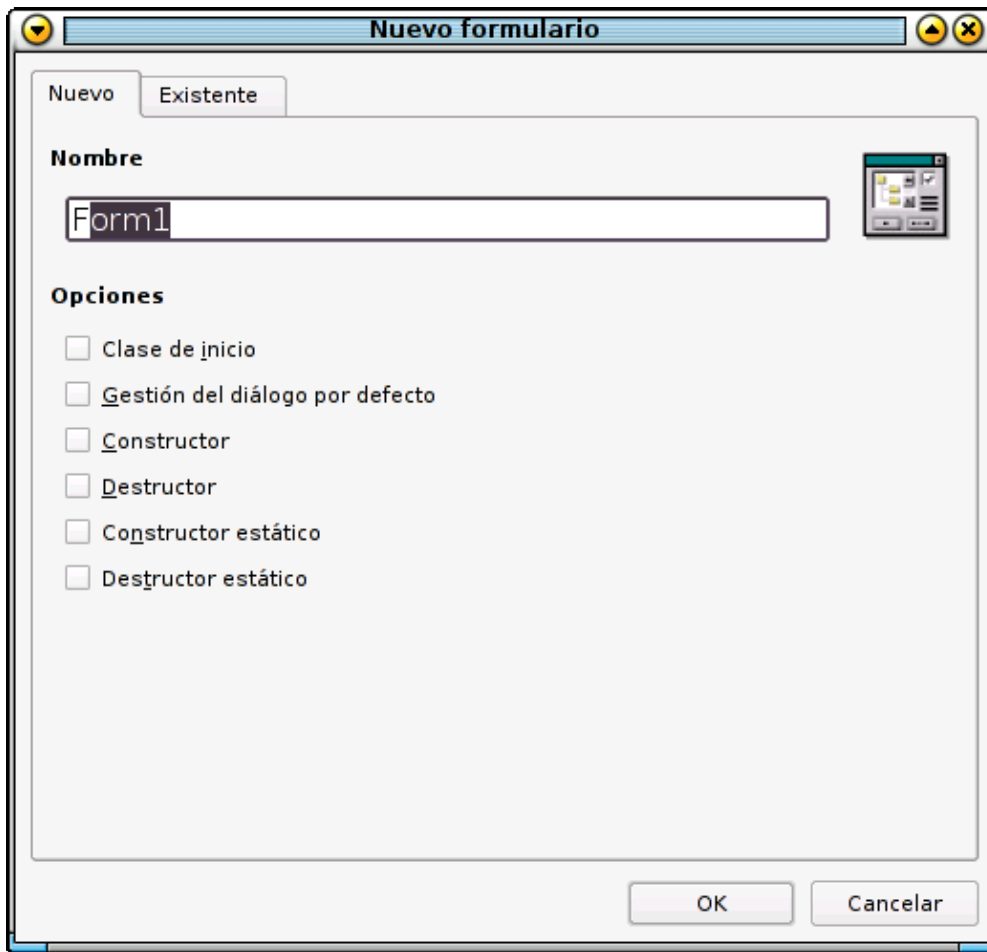
Here, *Control* is the name of the control where the event is generated and *Event* is the event type. Some controls have predetermined events, the most useful predetermined event in a button, for example, is *Click*.

In Gambas, when double-clicking on any control, the code editor is opened and the cursor positioned in the declaration of the subroutine for the predetermined event. There is an exception, if the control is associated to an action group (the "Group" property is defined), the code editor shows the subroutine for the action group, as we said before.

Designing forms

We have to be aware when designing forms:

- The users' screen may be different than our screen: different resolution, window manager and/or fonts size. Don't try to adjust the space too much, the labels, buttons and other controls may be cut or illegible.
- A good practice is to leave the main window of the program resizable. In Gambas, look at the form's property *Border*. Don't set it to *Fixed*.
- When creating a new form, there are some interesting options:



The options related to the constructor and destructor are useful for initializing and finishing a task in a window.

The following declarations are generated:

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

END

PUBLIC SUB _free()

END

PUBLIC SUB Form_Open()

END
```

If we choose "Static constructor" and/or "Static destructor", the declarations are now:

```
' Gambas class file

STATIC PUBLIC SUB _init()

END
```

Gambas tutorial

```
STATIC PUBLIC SUB _exit()  
  
END  
  
PUBLIC SUB _new()  
  
END  
  
PUBLIC SUB _free()  
  
END  
  
PUBLIC SUB Form_Open()  
  
END
```

Using these procedures, we can modify the opening and closing process of the window. We can initialize controls, variables, etc. If the procedure is declared as *STATIC*, the procedure only has access to *STATIC* variables.

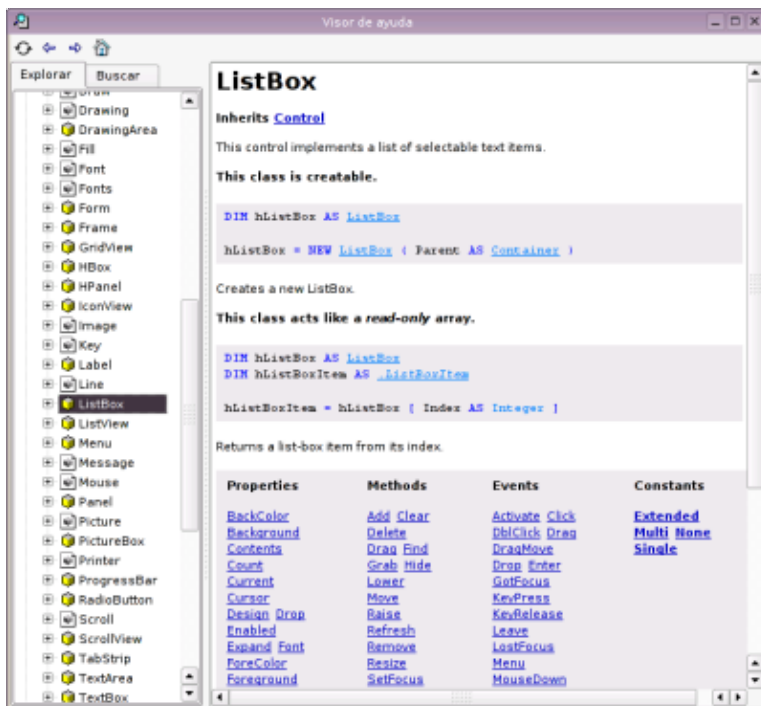
Dive in ...

Our form is fully designed. It is time to write the code for the controls.

The first task is to provide some functionality to the controls. We are going to work with the buttons (and the equivalent menus) "Add", "Modify", "Delete" and "Clean".

"Clean" action

This button must erase all the entries in the *ListBox*. To accomplish this task, we will search in the help system the documents related to a *ListBox*:



Gambas tutorial

The documentation is under the "tree" `gb.qt`, which is the Gambas' component that includes all the "visual" controls (buttons, labels, menus, etc...). We read that the `ListBox` provides a method, "Clean" that clears the all the entries. That's all we want, and we well use this method.

Double clicking on the button "Clean" (or the menu entry "Clean"), the code editor is raised and the cursor is positioned at the corresponding procedure. We write the following code:

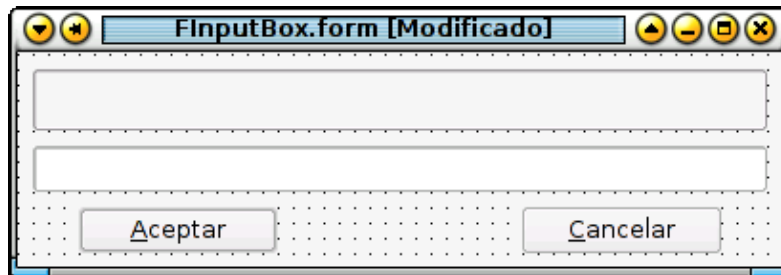
```
PUBLIC SUB Clean_Click()  
    ListBox1.Clean()  
END
```

Very easy :-)

"Add" action

This action is more complex. The users will add a entry (a line of text) to the `ListBox` using this button.

Gambas does not provide a dialog of type "InputDialog", so we are going to create one from scratch. We create a new `Form`, but now we specify that the form has a constructor. Why?. Because, in the creation instant, we will want to set some properties of the form, like the title, the message shown and a default value in the text entry field. This is the proposed design:



The form is very simple. It has a `Label`, a text entry (`TextBox`) and two buttons, "Accept" and "Cancel" ("Aceptar" y "Cancelar" in the screenshot). An intelligent dialog needs to make it convenient for the user to cancel with the `Escape` key and accept with the `Enter` key:

The `Button` controls have properties called "Default" and "Cancel". We set "Default" to `True` for the button "Accept" and "Cancel" property to `True` for the "Cancel" button.

Using these properties, when the user presses the `<ENTER>` key, the form will act as if the user were clicking the "Accept" button, and pressing `<ESC>` will be equivalent to pressing the "Cancel" button.

The next problem is how to return the written text in the dialog to the main window. We have to remember that there are no global variables in Gambas, so we need another solution. In the "Tip of the day" #7, (under the menu "? > Tips of the day") it is suggested that we use a variable declared as `PUBLIC`, so this variable is visible from any point or class within the program.

We create a new module (right click in "Modules > New module") and we name this module `MCommon`, for example. This is the module's implementation:

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

Gambas tutorial

Very simple and easy. Now we have a variable which may be accessed from any point within the program using the following notation: `MComun.my_text`

Now we write the code for our "InputBox" dialog:

```
' Gambas class file

PUBLIC SUB _new(title AS String, message AS String, OPTIONAL text
AS String)
    ME.Caption = title
    Labell.Caption = message
    ' a String is evaluated to FALSE if it is empty:
    IF text THEN TextBox1.Text = text
END

PUBLIC SUB Button1_Click() ' This is the "Accept" button
    MComun.my_text = TextBox1.Text
    ME.Close(0)
END

PUBLIC SUB Button2_Click() ' This is the "Cancel" button
    ME.Close(0)
END
```

The `_new` procedure is the constructor. Using it, we can set a different title, label and text entry contents each time the dialog is used. Furthermore, these properties are set at the creation moment.

The "Accept" button copies the text in the `TextBox` to the variable `my_text` defined in the module `MComun` and closes the dialog. The button "Cancel" simply closes the dialog.

As the variable `MCommon.my_text` is shared, we must remember to "clear" it each time it is used. We will see this now.

The procedure for the "Add" button in the main form is the following code. It is well commented:

```
PUBLIC SUB Add_Click()
    ' Declarating our "Inputbox"
    f AS FInputBox
    ' We create the InputBox, setting the title, message
    ' and a default value: the system date and time
    ' and a small arrow
    f = NEW FInputBox("Write an entry",
        "Please, write here the entry to be added:",
        CStr(Now) & " -> ")
    ' We show the InputBox
    f.ShowModal()
    ' If the user has written some text, it will
    ' be in the shared variable MCommon.my_text
    IF MCommon.my_text THEN ' An empty string is False
        ' The ListBox control has a method for adding entries: Add()
        ListBox1.Add(MCommon.my_text)
        ' We "empty" the shared variable
        MCommon.my_text = ""
    END IF
END
```

"Modify" action

Using this button, the user can change an entry in the ListBox. If there are no entries, the button does nothing, and, if the user does not select an entry, they will be warned. This is the implementation for this action:

```
' "Modify" action
PUBLIC SUB Modify_Click()
  f AS FInputBox
  IF ListBox1.Count > 0 THEN ' If the ListBox is empty, its property
    ' Count is 0
    IF ListBox1.Index = -1 THEN
      ' The Index property returns the index for the selected entry.
      ' It there is not selected line, it returns -1.
      ' We warn the user.
      message.Info("You must select the line to modify.")
    ELSE
      ' The user has selected a valid entry.
      ' We show our InputBox with the text of the selected entry.
      ' The selected text is the property Text of the
      ' selected object ListBoxItem
      ' which is accesible through the property
      ' Selected of the ListBox
      f = NEW FInputBox("Modify entry",
        "Please, modify the selected entry:",
        ListBox1.Current.Text)

      f.ShowModal()
      ' The dialog box FInputBox changes the shared variable
      ' MCommon.my_text
      ' If MCommon.my_text is not empty, we load it in the
      ' selected ListBoxItem
      IF MCommon.my_text THEN ListBox1.Current.Text = MCommon.my_text
      ' We "empty" the shared variable after its use
      MCommon.my_text = ""
    END IF
  END IF
END
```

"Delete" action

As we saw before, the ListBox must contain at least one line and the user must have selected one line. The code is very close to the "Modify" action:

```
PUBLIC SUB Delete_Click()
  i AS Integer
  i = ListBox1.Index
  IF i >= 0 THEN
    ListBox1.Remove(i) ' The Remove method erases the selected line
  ELSE IF ListBox1.Count > 0 AND i = -1 THEN
    ' We check that the ListBox is not empty and
    ' that some entry is selected.
    message.Info("You must select the desired line to delete.")
  END IF
END
```

We can see that the implementation for these four actions is common to the buttons and to their equivalent entries in the menu.

Now, we will implement the actions related to the file management (Open, Save) and closing the program. We will start with the easy stuff:

"Exit" action

The function for this button (and the associated entry in the menu) is to close the program. Very simple:

```
PUBLIC SUB Exit_Click()  
    ME.Close(0) ' ME is a reference to the form itself  
FInputBox  
END
```

We could be more user friendly with this action by adding a dialog box like *"You are going to quit this program. Are your sure?"* and doing the appropriate task. We will let this improvement be home work for the reader.

"Open" action

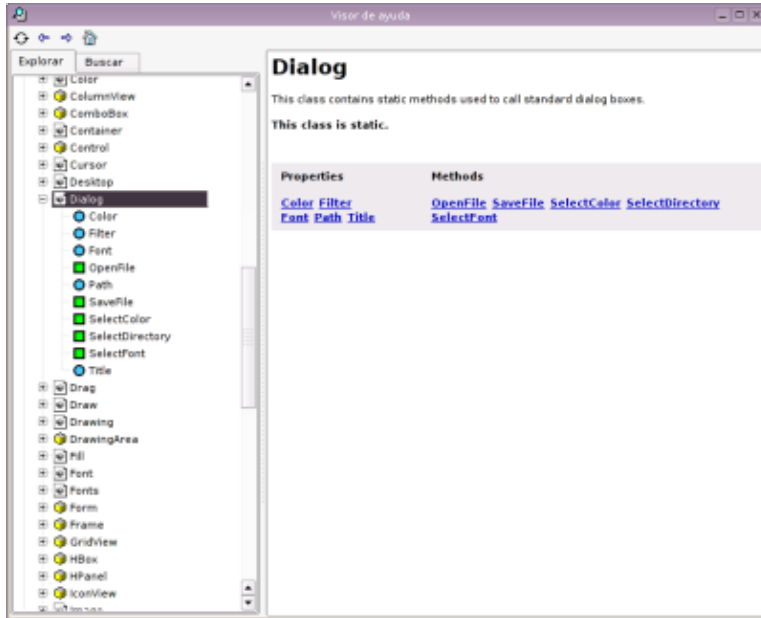
What must this action do? Ask the user the path to a file, read this file and load the data in the ListBox. This is the corresponding code:

```
PUBLIC SUB Open_Click()  
    DIM lin AS String  
    DIM arr_strings AS String[]  
    Dialog.Title = "Please select a file"  
    Dialog.Filter = [ "Minder data (*.data)", "All files (*.*)" ]  
    IF NOT Dialog.OpenFile() THEN  
        arr_strings = Split(File.LOAD(Dialog.Path), "\n")  
        ListBox1.Clean()  
        FOR EACH lin IN arr_strings  
            ListBox1.Add(lin)  
        NEXT  
    END IF  
END
```

This piece of code presents an interesting feature of the Gambas language, the "non instanceable" or static classes ([2](#)). We cannot create objects of these classes, but we may use them directly. In this code we can see two of these classes in action: the class "File" and the class "Dialog".

For example, the class `Dialog` provides access to the typical standard dialog for selecting files, system colors, etc. It is documented in the help node `gb.qt`

Gambas tutorial

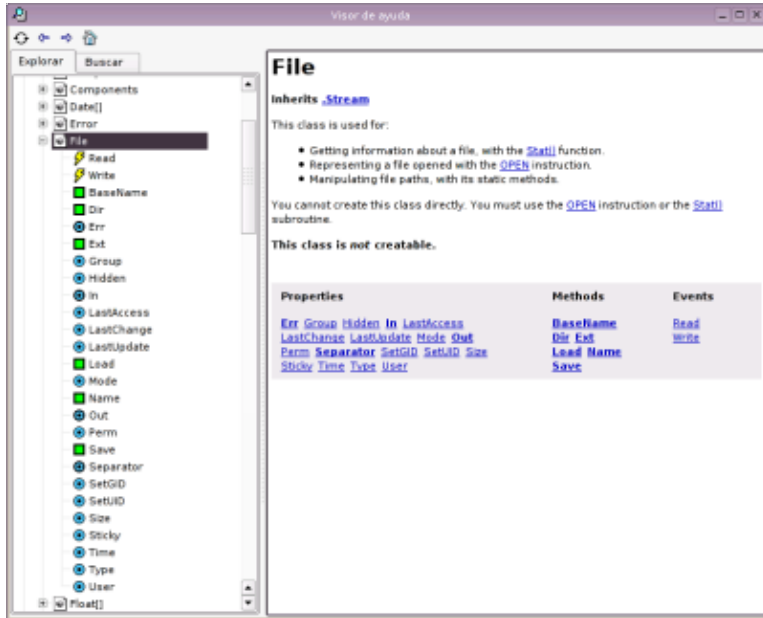


In our program, we want to select a file and load its contents into the `ListBox`. We will use the class `Dialog` in the following manner:

```
Dialog.Title = "Please select a file"
Dialog.Filter = [ "Minder data (*.data)", "All files (*.*)" ]
IF NOT Dialog.OpenFile() THEN
    ' etc ...
```

We set the dialog box's title, provide a filter for selecting the file type by extension (*.data) and, finally, we call the method `OpenFile()` of the class. There is a quirk in the `Dialog` class: if the user does NOT select a file (ie. the user press the cancel button or the ESC key) the return value in the method `OpenFile()` is `True`. Once the user has selected the desired file, we can access the full path through the property `Dialog.Path`

The class `File` (its documentation is under the entry `gb` in the help system) provides several methods to allow us to work with files:



In the Gambas documentation, in the section titled "How do I ..." there are some code examples for reading and writing files. We will use the method "Load()", whose argument is the path to a file and returns a String that contains all the data in that file. We can split the lines of returned data by using the function Split(). Its arguments are the string to be splitted and the separator (the new line character in this case \n) and returns an Array of Strings. For this reason, we have declared the variable arr_strings as String[]:

```
DIM arr_strings AS String[]
```

Once we have all the lines of data contained in the file, we should clear the ListBox and add each line using the Add() method of the ListBox.

"Save" action

When the user presses the "Save" button, the program must output the ListBox contents to a text file. We will show the 'Save File' dialog box asking the user to give a file name for saving the data as. This is the code for this action:

```
PUBLIC SUB Save_Click()
    lines AS String
    destination AS String
    numFile AS Integer
    lines = ListBox1.Contents
    Dialog.Title = "Please, select a file"
    Dialog.Filter = [ "Minder data (*.data)" ]
    IF NOT Dialog.SaveFile() THEN
        IF Right$(Dialog.Path, 5) <> ".data" THEN
            destination = Dialog.Path & ".data"
        ELSE
            destination = Dialog.Path
        END IF
        File.Save(destination, lines)
    END IF
END
```

Gambas tutorial

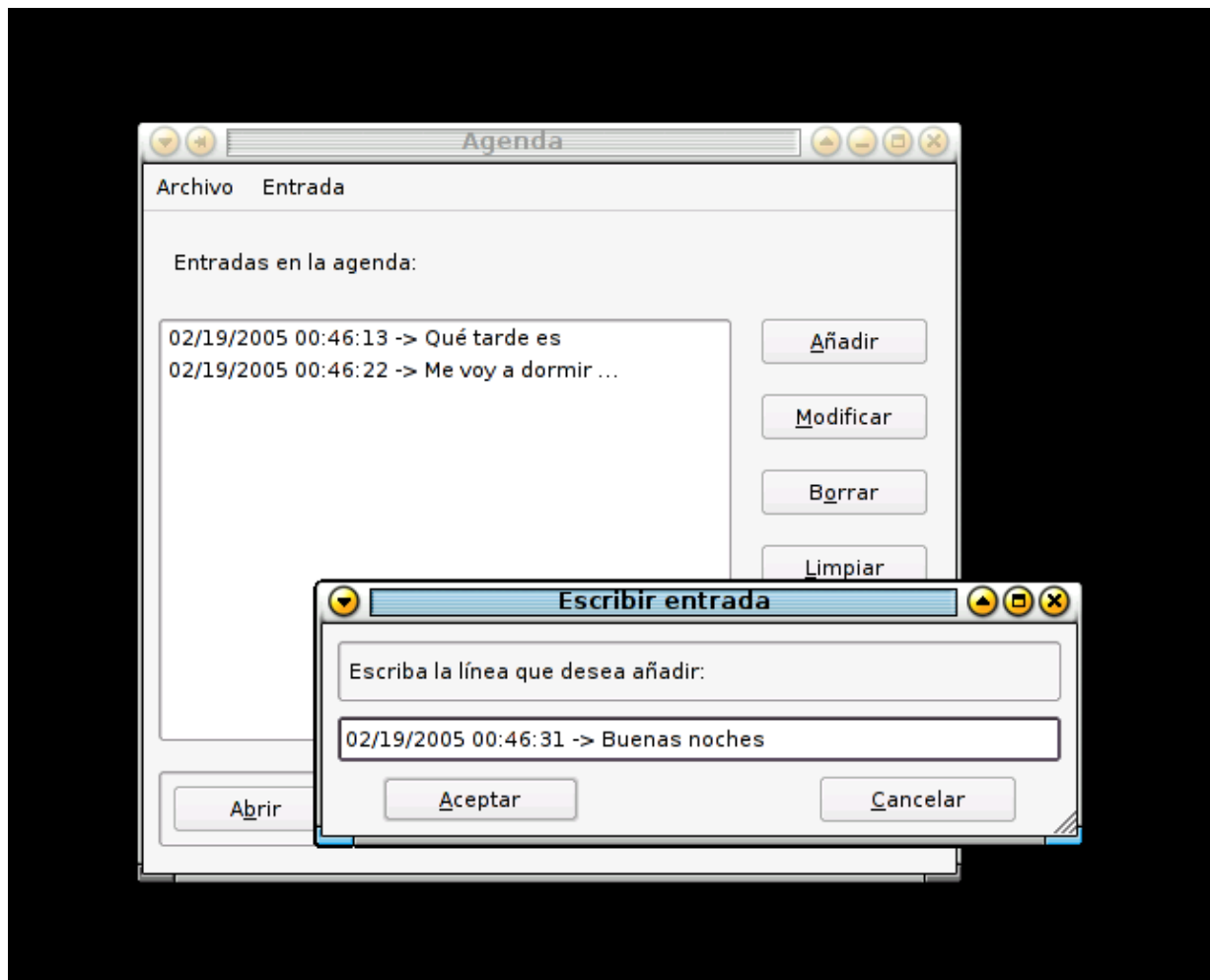
We want save the data in a file with the extension `.data`, so, if the file name provided by the user does not end in `".data"`, we will concatenate the extension. We use the `Save()` method that provides the class `File`. The arguments for this method are the path to the file and the text that we want to save. We access the `ListBox` contents by using the property `Contents`, which returns a `String`, with a "new line" (`\n`) separating each entry in the `ListBox`.

Final adjustment

It would be interesting for the user to see all the text in an entry, because the listbox may not be wide enough to display it if the entries are very long. We implement this feature in this manner: when the user double-clicks an entry, we show all the text in a dialog box:

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

Our program running



Deploying our application

OK. Our program is ready. We may test it at any time from inside the IDE by pressing the key F5.

Now, we want to use the program as a normal application, i. e. without the IDE. To accomplish this, there is an option in the main menu: "Project > Create executable". This will create a "monolithic" executable file, a file that includes all the project resources. This file is not machine code, but bytecode, executable by the Gambas' interpreter, `gbx`. This implies that we need this interpreter installed in the system for executing programs written with Gambas. (This is similar to other languages. For example, we need Java installed for execute programs written in Java).

In all the Linux distros where Gambas is included, the Gambas components are split and there is a "Gambas runtime" package that includes the interpreter but not the complete IDE.

We can create RPM or DEB packages of our program. These packages will have the Gambas interpreter (the `gambas-runtime` program) as a dependency. There is a wizard to help in the package generation. It is very easy and intuitive. You can find it in the menu "Project > Create installation package ...".

Conclusions

We have proved that it is very easy to create a simple but functional application with Gambas. The tool provides several controls and predefined classes. There are also a lot of extensions or components available in order to create client/server applications, database access, multimedia, etc.

IMHO, I think that Gambas is a very promising tool. Luckily, Gambas' development mailing-list is very active, and the bugs are solved very quickly.

Thanks, Benoît (et col.)! Good job!

About this document and its author

As mentioned before, we have used the version 1.0-1 of Gambas in this tutorial (precompiled packages for Debian Sid). At the time of writing this document, the version 1.0.3 has been released, and it is very probable that at the time of reading this, there was a new version. It is very advisable to read the changelog for possible incompatibilities from version to version.

All comments, suggestions, *corrections* and improvements for this document are wellcome.

My mail is [forodejazz \(arroba\) gmail \(punto\) com](mailto:forodejazz@aroba.gmail.punto.com)

Legal stuff: This document is free. You may copy it, link, translate to other languages or sell, but you must preserve this note and mention the document's origin. The author will be thankful if he is notified and even if he is paid for his work ;-)

Notes

1. The events need to be processed by a procedure or subroutine: a function that does not return any value.
2. I am not an expert in Object Oriented Programming terminology. My apologies if I'm using incorrect terms ;-)