by Jonás Alvarez
<jalvarez(at)eitb.com>

About the author:

Jonás Alvarez hat mehrere Jahre als Entwickler für Anwendungen in Unix– und Windowsumgebungen gearbeitet. Unter anderem gab er Kurse über Betriebssysteme, Netzwerke und Entwicklung.

Gambas: Basic für Linux

Abstract:

Gambas ist eines der heute zur Verfügung stehenden Basics für Linux. In diesem Artikel werden wir anhand eines Beispiels zeigen, wie einfach und wirkungsvoll Gambas für die tägliche Anwendung sein kann.

Translated to English by:
Miguel Alfageme Sánchez,
Samuel Landete Benavente.
<mas20(at)tid.es>

Einführung

Basic ist eine der am meisten verbreiteten und einfachsten Programmiersprachen, besonders für Anfänger. Microsoft Visual Basic IDE war bisher die bekannteste Umgebung für die Entwicklung von Basicanwendungen. Linux verbreitet sich neuerdings auf den Benutzer–Desktops. Bisher begrenzt auf Severanwendungen und Guru–Benutzer, entwickelt es sich zum Betriebssystem auf Clientmaschinen für E–mail, Surfen im Web und Bearbeiten von Texten. Diesem Trend folgend stehen jetzt mehrere Basic–Entwicklungsumgebungen zur Verfügung. Gambas ist die, die wir in diesem Artikel vorstellen, eine graphische Entwicklungsumgebung für Basic. Der Programmierstil ähnelt dem von Visual Basic. Wie wir später sehen werden gibt es jedoch auch Unterschiede. Ich werde Version 0.64a benutzen, die Teil meiner SuSE 9.0–Distribution ist. Auf der Projektseite von Gambas können wir feststellen, dass 0.81 die neueste Version ist, unser Artikel wird davon nicht beeinflusst.

Wen könnte Gambas interessieren?

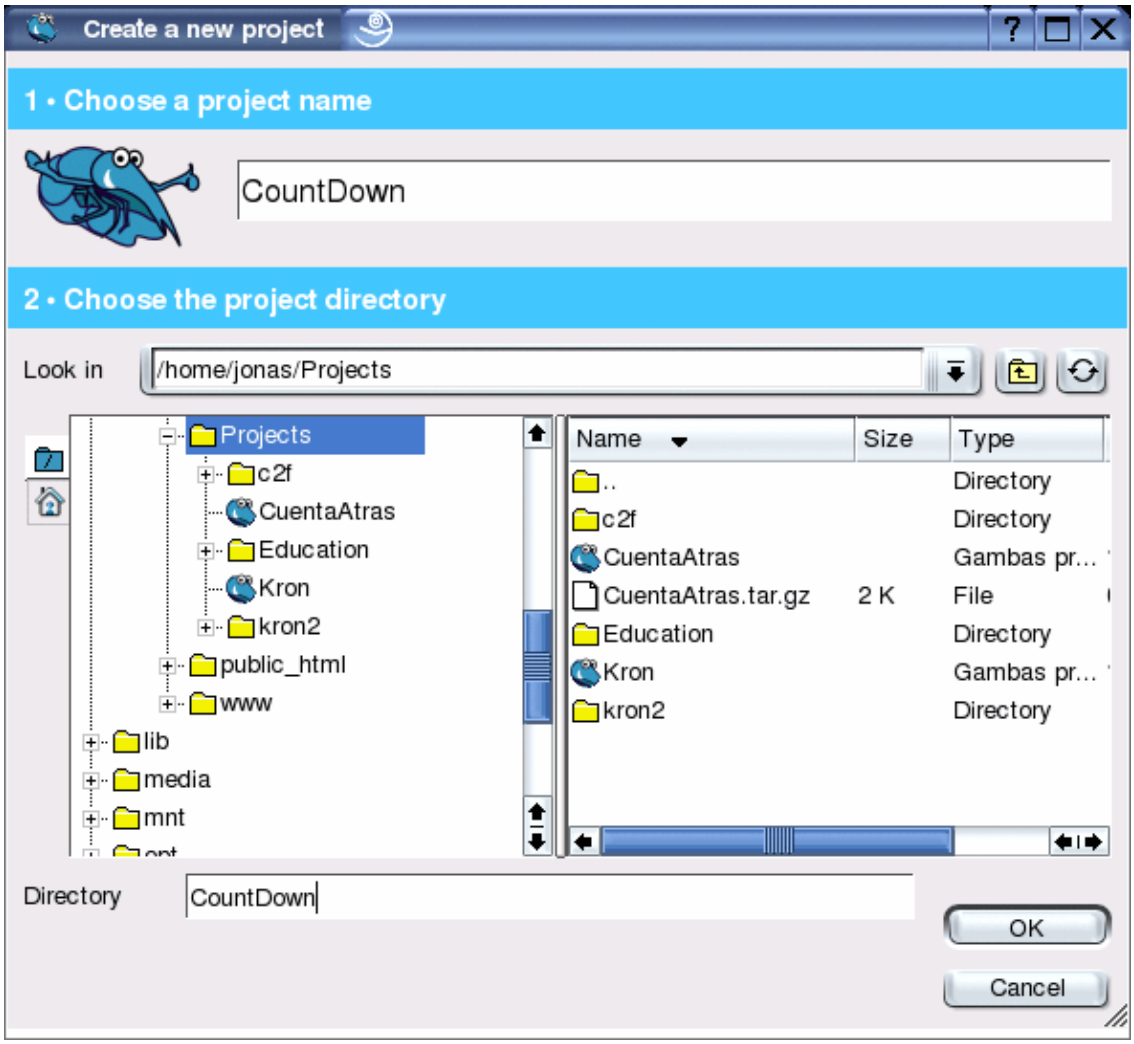
Als langzeitiger Entwickler von Basic−Anwendungen war es für mich einfach, mit diesem Beispiel zu beginnen. Ich wende Gambas hier zum erstem Mal an, was beweist, dass es für Visual Basic−Kenner einfach zu beherrschen ist. Für andere sollte es eine Demonstration sein, wie einfach Basic für viele Dinge nützlich sein kann.

Das Beispiel

Weil ich am liebsten durch die Praxis lerne, fangen wir mit einem Beispiel an. Es ist die sehr einfache Anwendung einer laufenden Stoppuhr auf dem Bildschirm. Wir können die Zeit einstellen, stoppen und starten, ganz nach Wunsch.

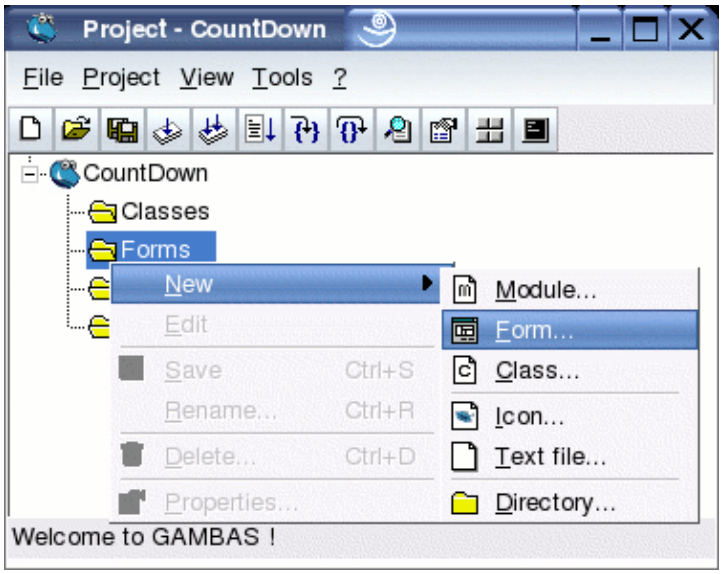
Kurz nachdem wir Gambas starten, treffen wir den Assistenten:

Wir wählen **New Project**. Im nächsten Fenster werden wir nach dem **Project Name** gefragt. Wir nennen unser Projekt *CountDown*. Im **Zweiten Dialog** müssen wir unser **project directory** wählen. Wir geben dort unser Arbeitsverzeichnis an. In der Textbox unten geben wir den Namen für das neue Verzeichnis ein.

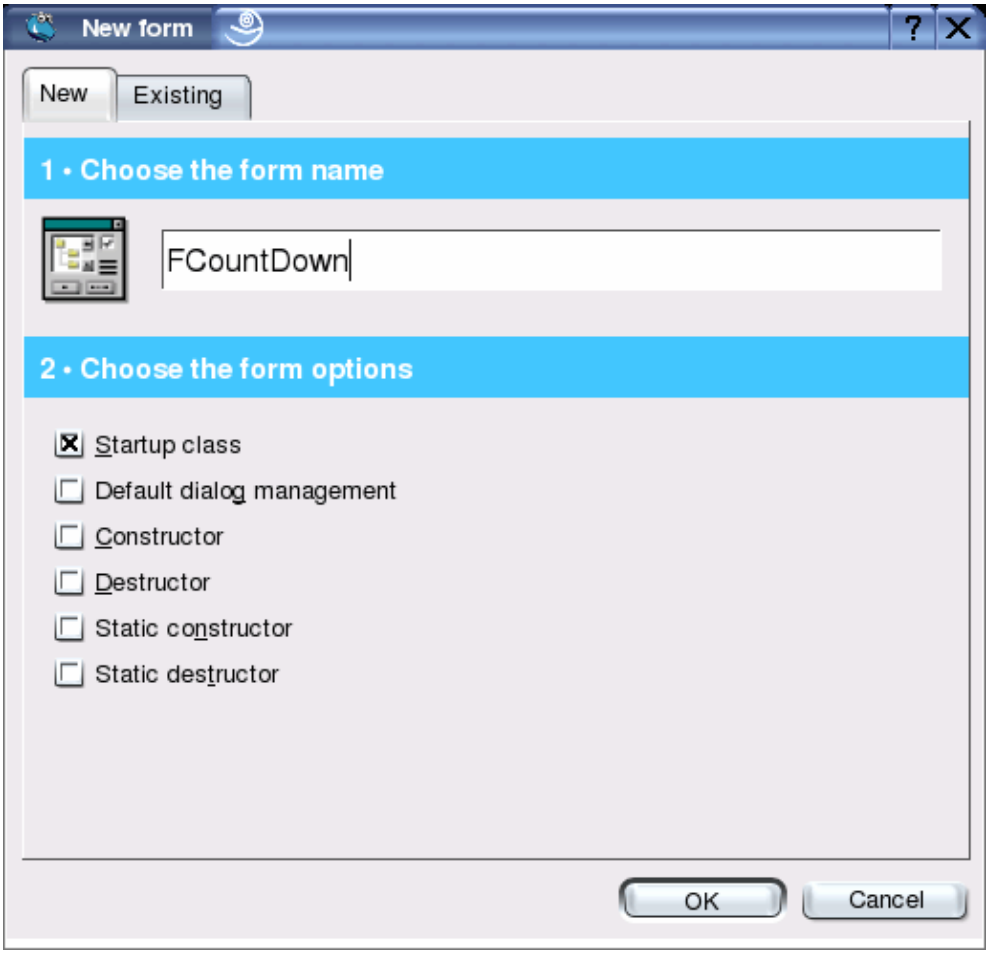


Beim ersten Öffnen von Gambas, oder wenn wir die Option nicht abgestellt haben, erscheint der *'Ratschlag des Tages*'. Wir lesen, was uns interessiert und schliessen dann das Fenster. Damit sind wir bereits in der Arbeitsumgebung. Auf unserem Desktop sehen wir mehrere Fenster. Benutzen wir eine Umgebung, wie z.B. KDE, mit mehreren Desktops, wäre es am besten, einen der Desktops Gambas zuzuordnen, um alle seine Fenster unter Kontrolle zu haben. Eine der ersten Optionen, die ich in KDE aktiviere, ist, daß jeder Desktop nur seine eigenen Icons anzeigt.

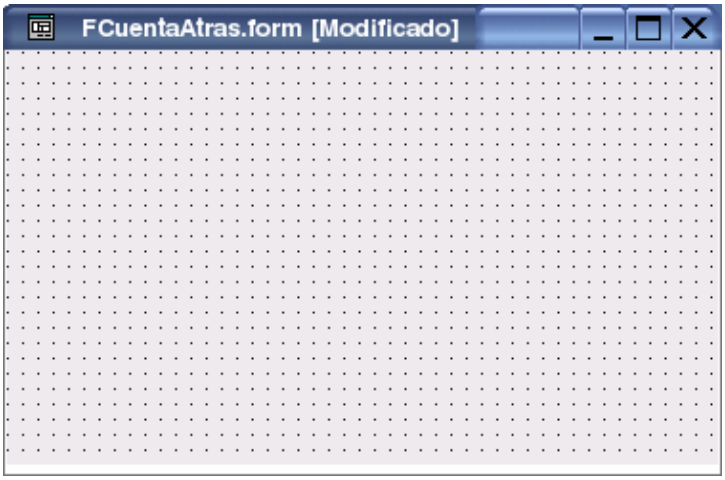
Wir werden jetzt die Hauptform unserer Applikation erstellen. Dafür klicken wir mit der rechten Maustaste irgendwo in das Projektfenster und erzeugen eine neue Form.



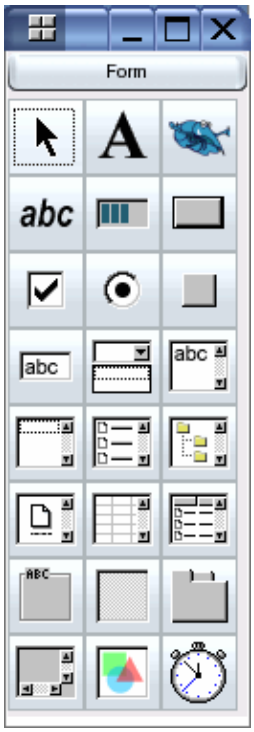
Im Dialogfenster geben wir der Eingabemaske (*'New form'*) einen Namen, in unserem Fall *FCountDown*, alle seine Werte lassen wir in der Grundeinstellung.



Damit haben wir unse erste Eingabemaske, die zur Zeit leer ist.



Hier werden wir die Bedienelemente für unsere Stoppuhr zusammenstellen. Wir klicken auf die Schaltflächen der Werkzeugleiste, die wir in unserer Eingabemaske haben wollen. Fahren wir mit dem Mauszeiger über eine Schaltfläche, erscheint deren Name. Mit einem Doppelklick kopieren wir das entsprechende Werkzeug in die obere linke Ecke unserer Eingabemaske. Mit einem einfachen Klick können wir es in der Eingabemaske an die gewünschte Stelle bringen und die Grösse ändern. Für unser Programm benötigen wir ein Label, eine Textbox, einen Timer, zwei Schaltflächen und einen Tastschalter.



Sobald die Schaltflächen an ihrem Platz sind, sieht das ungefähr so aus (mehr oder weniger, je nach persönlichem Geschmack):

Nachdem wir die Schaltflächen in unserer Eingabemaske haben, ändern wir deren Namen, so dass sie die gewünschten Funktionen ausdrücken. Dafür bearbeiten wir **Name** im **Property Sheet**. Falls wir das **Property Sheet** nicht auf dem Bildschirm sehen, können wir es mit der Eigenschaften−Dialogfläche aktivieren. Diese erscheint, wenn wir mit dem Mauszeiger über die entsprechende Schaltfläche fahren.

Ich nenne die *Label1*−Schaltfläche *lblContador:* Ich klicke auf die Schaltfläche, ändere deren Namen im **Property Sheet**. Das geschieht mit der Änderung der **Name**−Eigenschaft und der Eingabe von *lblContador* als Wert. Danach ändere ich auf einen grösseren Fontwert. Dafür wähle ich *Fonttyp Courier Bold 72* als **font**−Eigenschaft der Schaltfläche und klicke **OK**. Auf die gleiche Weise ändere ich den *ToggleButton1* auf *Courier Bold 72* und den Namen auf *tglFuncionando*. Die *TextBox1* Schaltfläche wird *txtSegundos*, die *Timer1*−Schaltfläche wird *clkMiReloj*, *Button1* wird *cmdPonerSegundos* und zuletzt ändere ich *Button2* zu *cmdSalir*. Ausserdem ändere ich das **Alignment** von *txtSegundos* auf *Right*.

Jetzt beginnen wir mit dem Basiccode. Es ist sehr einfach und nicht sehr strikt mit der Syntax. Zuerst ändern wir den Text auf der Eingabemaske zu realistischeren Werten. Obwohl viele Optionen mittels Basic geändert werden können, hätten wir das auch bei den Eigenschaften der Schaltflächen durchführen können, beide Methoden führen zum gleichen Ergebnis.

Sobald die Eingabemaske offen ist, geben wir den Titel für jede Schaltfläche ein. Mit dem Ausdruck *'sobald die Eingabemaske offen ist*' meinen wir die Ausführung eines Vorgangs: die Eingabemaske öffnen. Um das auszuführen, doppelklicken wir auf einen Teil der Eingabemaske, der keine Schaltflächen enthält. Ein Bearbeitungsfenster wird geöffnet und der Cursor befindet sich innerhalb eines neuen Vorgangs: **Public Sub Form_Open()** (falls wir schon mal mit Visual Basic programmiert haben, benutzten wir den Form_Load−Vorgang). Wir bezeichnen die *lblContador*−Schaltfläche zur Anzeige der verbleibenden Sekunden des Countdowns. Die ersten Zeilen des Codes der Control−Klasse sehen so aus:

```
' Gambas class file
CONST fSegundosPorDefecto AS Float=120.0
fSegundos AS Float

PRIVATE SUB VerValores()
  DIM nMinutos AS Integer

  nMinutos = Int(Int(fSegundos) / 60)
  lblContador.Caption = nMinutos & ":" & Format (fSegundos -
                                                 nMinutos * 60, "00.0")
END

PRIVATE SUB VerActivarDesactivar()
  IF tglFuncionando.Value THEN
```

```
    tglFuncionando.Text = ("&Detener")
  ELSE
    tglFuncionando.Text = ("&Arrancar")
  ENDIF
END

PUBLIC SUB Form_Open()
  fSegundos = fSegundosPorDefecto
  VerValores
  tglFuncionando.Value = FALSE
  VerActivarDesactivar
  txtSegundos.Text = fSegundos
  cmdPonerSegundos.Text = ("&Reiniciar")
  cmdSalir.Text = ("&Salir")
END
```

Direkt nach dem von Gambas generierten Kommentar *'Gambas class file* haben wir eine Konstante mit der Grundeinstellung der Anzahl der Sekunden für den Countdown *fSegundosPorDefecto* hinzugefügt, mit dem Wert 120 Sekunden (zwei Minuten) und eine Variable, *fSegundos*, die den Countdown durchführt. Wir haben auch zwei Vorgänge eingeführt: *VerValores*, welcher die Countdownwerte anzeigt und *VerActivarDesactivar*, der den Text der Start/Stop−Schalter ändert.

Jetzt haben wir schon eine Eingabemaske, die funktioniert. Sie hatte bis jetzt keinen praktischen Wert, ausser uns zu zeigen, was wir durchgeführt haben, probieren wir's also aus. Wir speichern unsere Änderungen vom Hauptfenster unseres Projekts, *Project CountDown* und starten die Anwendung mit **F5** oder mit der ***Execute***−Schaltfläche im gleichen Fenster. Wir sollten folgendes sehen:

Ist das nicht der Fall oder wir erhalten eine Fehlermeldung, müssen wir unser bisheriges Beispiel überprüfen. Selbst, wenn wir **Start**, **Reset** oder **Exit** betätigen, passiert nichts. Das wird also unsere nächste Aufgabe: den Schaltflächen derart Funktionen zuteilen, dass etwas geschieht, wenn der Benutzer irgendetwas betätigt. Bevor wir weitermachen, sollten wir etwas mit unserer Anwendung spielen, um herauszufinden, was möglich ist. Um sie zu schliessen betätigen wir den X−Schalter oben rechts. Ich bin in KDE mit dem SuSE−Motiv, wie man an den Eingebemasken sehen kann. Es ist möglich, dass Sie das Fenster (der Anwendung) auf eine andere Weise schliessen müssen.

Probieren wir die einfachste Schaltfläche: was muss geschehen, wenn der Benutzer **Exit** betätigt? Die Anwendung sollte geschlossen werden. Um den Basiccode vorzustellen, der ausgeführt wird, wenn der Benutzer diesen Schalter betätigt, doppelklicken wir auf den Schalter mit der Aufschrift **Exit** (*cmbExit*). Wir stellen fest, dass Gambas einige Zeilen Code erzeugt hat, mit dem Cursor zwischendrin. Hier muss der Code eingefügt werden. Dieser wird ausgeführt, wenn der Benutzer auf diese Schaltfläche klickt. Um die

Anwendung zu schliessen, müssen wie Me.close ausführen, der Code für diesen Vorgang sieht so aus:

```
PUBLIC SUB cmdSalir_Click()

  ME.Close

END
```

Die nächste Schaltfläche, die wir prüfen wollen, ist **Reset**. Wie schon gehabt: doppelklicken auf die Schaltfläche und im Codefenster von Gambas müssen wir folgendes einfügen:

```
 PUBLIC SUB cmdPonerSegundos_Click()

  fSegundos = txtSegundos.Text

  VerValores

END
```

Wie wir feststellen, passiert immer noch nichts. Wir müssen unserer Anwendung etwas auf die Beine helfen. Wir werden das *Timer*–Objekt aktiv machen, es war schon von Anfang an in der Eingabemaske. Das geschieht mit der Einstellung des Intervals der Zeitaufnahme. Wir können das entweder mit Code – wie im vorangegangenen Ablauf über *Form_Open* – oder über die Eingabemaske durchführen. In der Eingabemaske klicken wir auf das Objekt *Timer* und sein **Property Sheet**, dort ändern wir den **Delay**–Wert von 1000ms auf 100, um einen Vorgang alle Zehntel einer Sekunde zu empfangen – das wird die Genauigkeit unserer Stoppuhr sein.

Wir haben immer noch keinen ausführbaren Code und nichts, um ihn zu aktivieren, wenn die Uhr betätigt wird. Um den Code für die Uhr zu erzeugen, doppelklicken wir ganz einfach auf die Uhr in der Eingabemaske. Das führt uns zum Codefenster. Nachdem wir unseren Code einfügen, sollte das so aussehen:

```
PUBLIC SUB clkMiReloj_Timer()
  IF fSegundos < 0.1 THEN
    tglFuncionando.Value = FALSE
    tglFuncionando_Click
  ELSE
    fSegundos = fSegundos - 0.1
    VerValores
  END IF
END
```

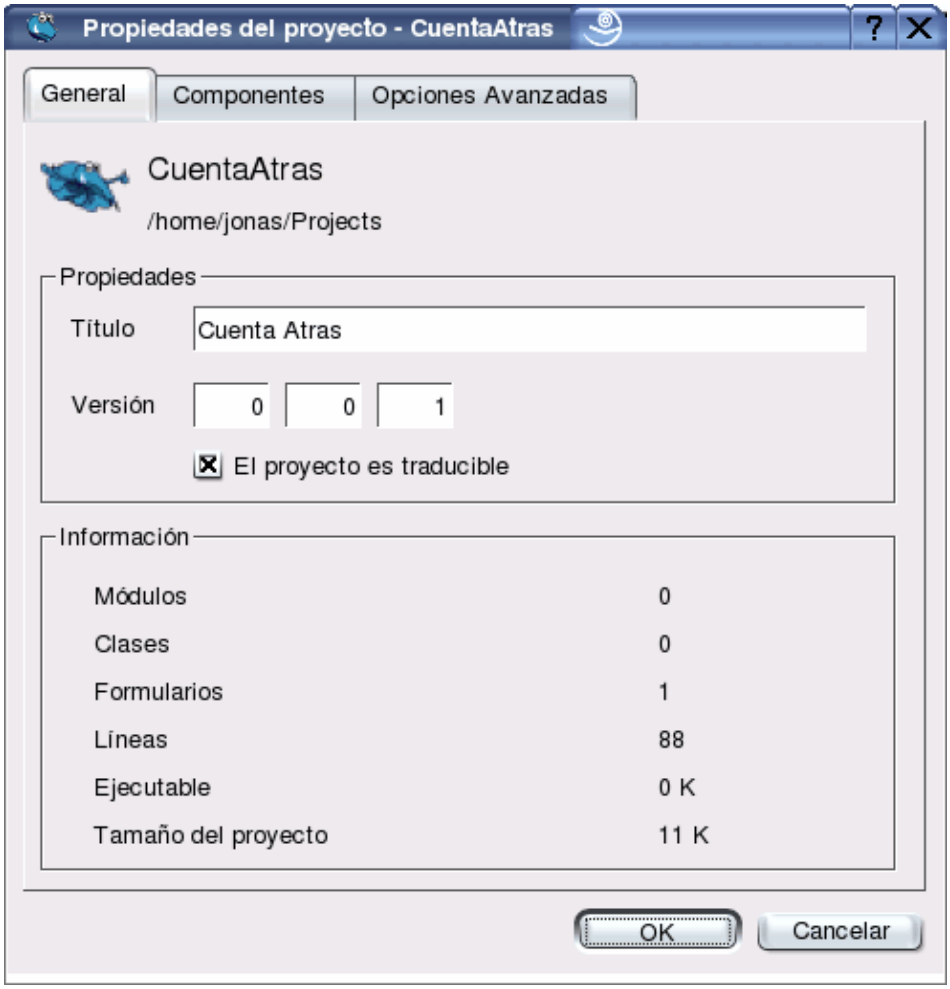
Und zum Schluss aktivieren wir den übriggebliebenen Tastschalter der Stoppuhr. Mit einem Doppelklick auf den Schalter fügen wir den Code für diesen Vorgang ein:

```
PUBLIC SUB tglFuncionando_Click()
  clkMiReloj.Enabled = tglFuncionando.Value
  VerActivarDesactivar
END
```

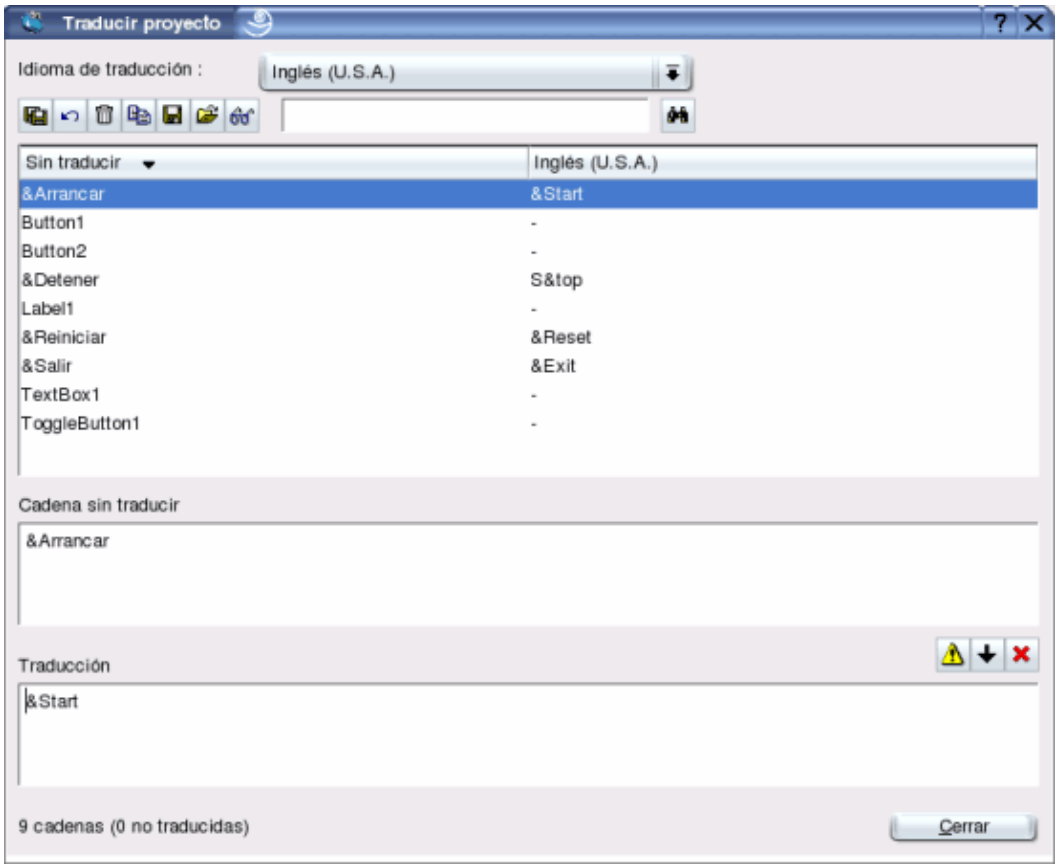
Und jetzt können wir unser Beispiel testen.

Zu guter Letzt: Gambas ist multilingual, wie es sich gehört

Eine weitere Eigenschaft von Gambas ist die Unterstützung mehrerer Sprachen. Ein Blick auf den Code zeigt uns, dass die Strings in Klammern eingeschlossen sind. Diese zeigen Gambas an, dass eine Übersetzung erfolgen soll. Der Text für die Einträge in der Eingabemaske benötigt keine Klammern. Unser Projekt hat sich als etwas sehr Brauchbares herausgestellt und die Anwender möchten die Dialoge in ihrer Sprache sehen. Nichts ist einfacher. Wir begeben uns zum **Project / Properties**–Menü des Projektfensters.



Hier geben wir unserem Projekt einen **Title** und aktivieren die **Project is translatable**– Option, welche uns die Übersetzung der Dialoge ermöglicht. Im Menü erscheint eine neue Option: Project/Translate. Wenn wir den Dialog öffnen, sehen wir, dass die Übersetzung sehr intuitiv erfolgt:



Im oberen Teil der Combo–Box wählen wir zuerst die Zielsprache in der Dropdown–Liste. In der oberen Box, direkt unter der Werkzeugleiste, finden wir eine zweisprachige Liste der übersetzten Strings. Wir tragen die gewünschte Übersetzung in die unterste Box ein. Sobald alle übersetzten Strings dort eingetragen sind, führen wir von einen Terminal einen Probelauf mit der Anwendung durch, die Variable LANG muss vorher auf die Übersetzungssprache geändert werden. Wenn ich das in meinem Beispiel mit Englisch durchführen will, schliesse ich Gambas und führe folgendes aus:

```
$ LANG=en_US; gambas
```

Dann starte ich Gambas aus dem KDE–Menü, weil die Variable hier bereits in der gewünschten Form existiert.

Fazit

Obwohl Gambas eine interpretierte Sprache ist und vollständig installiert sein muss, bietet sie doch eine gute Möglichkeit, um mit der Entwicklung von Anwendungen für den Linux–Desktop zu beginnen. Das ist sehr einfach und schnell – wie wir gesehen haben – es ist ausreichend für viele der täglichen Anwendungen.

Die auf den Bildschirm aufrufbare Hilfe ist recht vollständig, daneben haben wir Zugang zu Beispielen über das **File/Open example** –Menü. Wir können auch zum project web gehen, in der Linksammlung finden wir viele interessante Basicprojekte. Das ist erst der Anfang eines Projekts, dem ich eine gute Zukunft voraussage.

Webpages maintained by the LinuxFocus Editor team
© Jonás Alvarez
"some rights reserved" see linuxfocus.org/license/
http://www.LinuxFocus.org

Translation information:
es −−> −− : Jonás Alvarez <jalvarez(at)eitb.com>
es −−> en: Miguel Alfageme Sánchez, Samuel Landete Benavente. <mas20(at)tid.es>
en −−> de: Jürgen Pohl <sept.sapins/at/verizon.net>

2005−01−11, generated by lfparser_pdf version 2.51