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

L´auteur:

Jonás Alvarez a travaillé comme développeur d'application pour les environnements UNIX et Windows pendant de nombreuses années. Il donne, en autres, différents cours sur les systèmes d'exploitation, les réseaux et le développement.

Gambas: le Basic pour Linux

Résumé:

Gambas est l'un des programmes de Basic, disponibles actuellement pour Linux. Dans cet article, nous allons développer un exemple dans lequel nous constaterons que Gambas est simple et puissant pour les tâches quotidiennes.

Traduit en Français par:
Laurent RICHARD.
<kouran(at)linuxmail.org>

Introduction

Un des langages de programmation le plus simple et le plus complet, destiné principalement aux débutants, est sans conteste le Basic. Jusqu'à présent, l'environnement le plus courant pour le développement d'application en Basic est l'IDE Visual Basic de Microsoft. Actuellement, Linux se propage de plus en plus sur le bureau de l'utilisateur. Avant limité aux seuls serveurs et utilisé par des gourous, il est devenu un système d'exploitation pour les ordinateurs clients donnant une réponse aux besoins tels la consultation des courriers électroniques, la navigation sur Internet ainsi que l'édition de texte. En suivant cette tendance, de nombreux environnements de développement en Basic sont actuellement accessibles. Gambas, que nous allons étudier dans cette article, est un des ces environnements graphique pour le BASIC. Avec un style de programmation similaire à celui du Visual Basic, que nous verrons plus tard, il possède également ces propres différences. La version que je vais utiliser est la 0.64a, incluse dans ma distribution SuSE 9.0. En lisant ces lignes, vous pourrez voir sur la page du projet de Gambas que la dernière version est la 0.81. Mais cela ne devrait pas affecter notre article.

Qui est intéressé par Gambas?

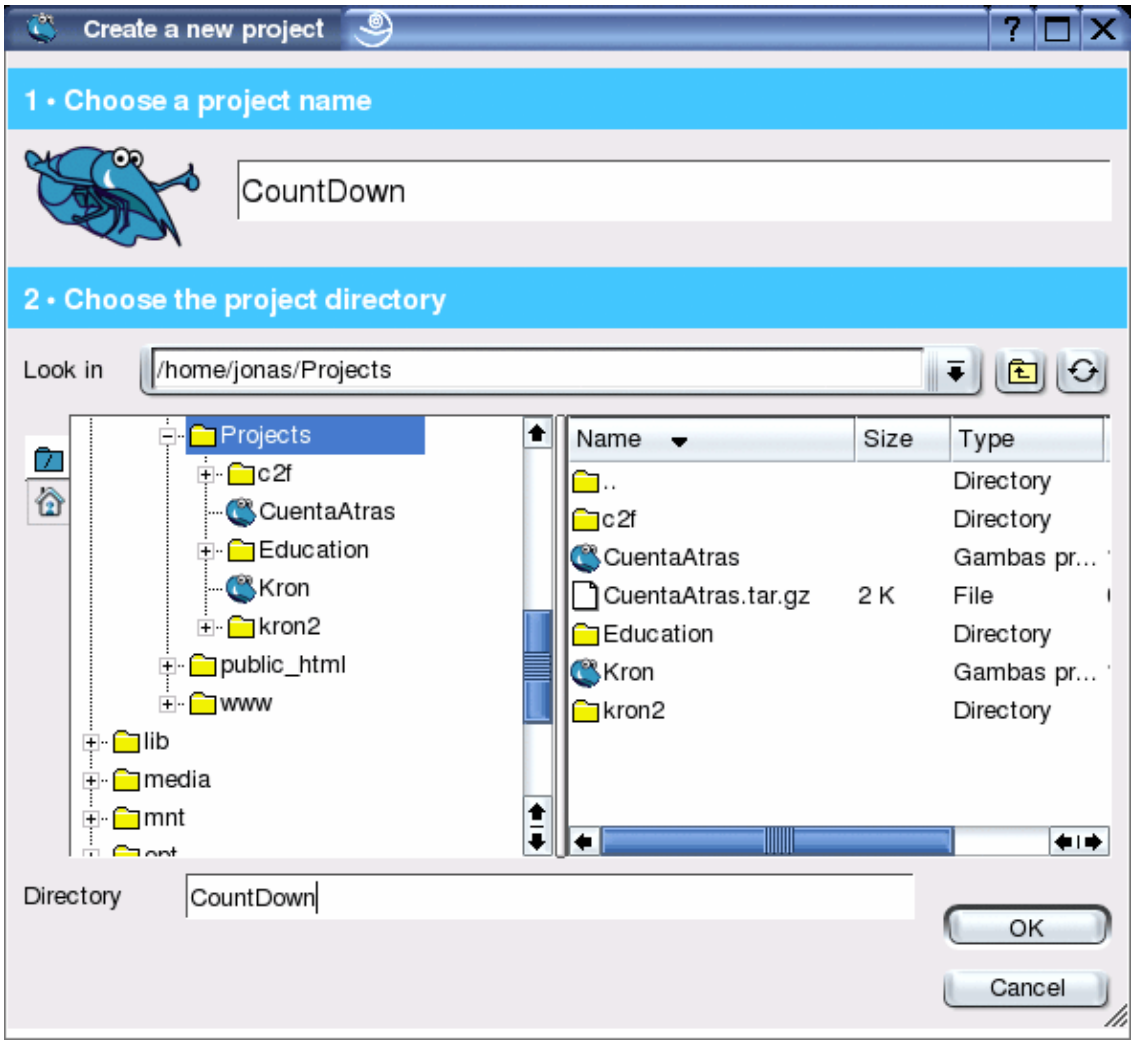
En tant que développeur d'applications en Visual Basic pendant un moment, je n'ai eu besoin que de lancer l'application pour m'y retrouver et développer l'exemple de cet article. De plus, c'est la première fois que j'utilise Gambas. Cela prouve que quiconque ayant déjà travaillé avec Visual Basic peut l'utiliser. Pour le reste, il est un exemple de la simplicité avec laquelle le Basic peut être simple et utile pour pas mal de choses.

L'exemple

Vu que j'aime apprendre en pratiquant, nous allons débuter avec un exemple. C'est une application très simple qui affiche un chronomètre avec un décompte à l'écran. Nous pouvons modifier le temps, l'arrêter et le lancer quand nous le désirons.

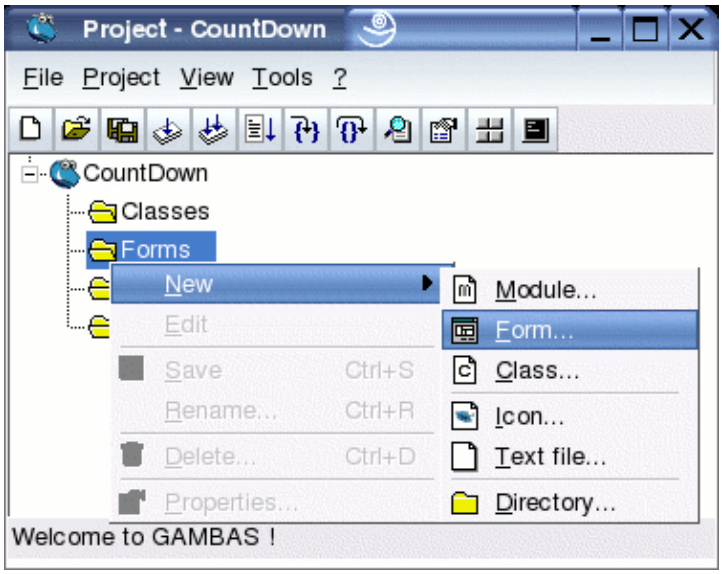
Juste après avoir lancé Gambas, nous faisons la connaissance de son assistant:

Nous choisissons **New Project**. Dans la fenêtre suivante, on nous demande le **nom du projet**. Notre projet s'appellera *CountDown*. Sur la **deuxième boîte de dialogue**, nous devons choisir le **répertoire du projet**. Nous sélectionnons notre répertoire de travail et nous écrivons le nom du répertoire dans lequel nous allons travailler dans la zone de texte du bas.

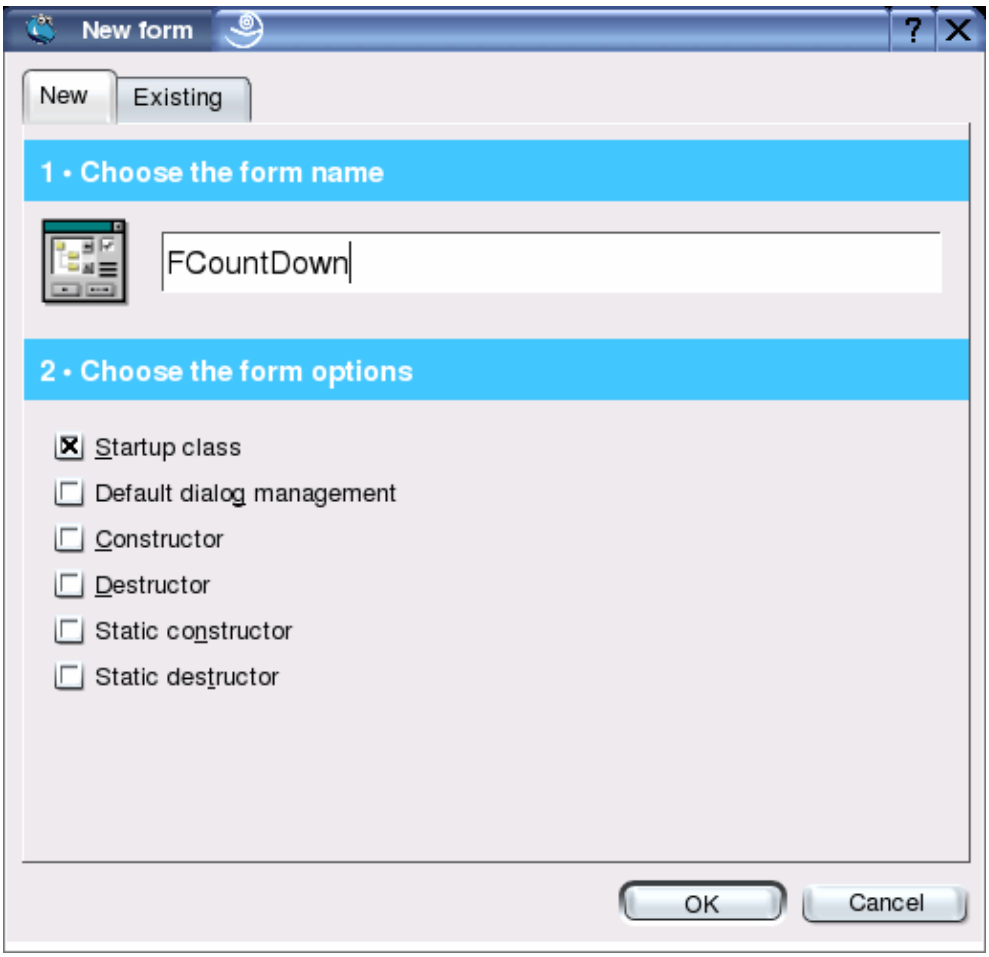


Si c'est la première fois que nous démarrons Gambas ou si nous n'avons pas désactivé l'option, nous verrons l'astuce du jour. Nous lisons ce qui nous intéresse et nous fermons la fenêtre. Nous sommes déjà dans l'environnement, prêt à travailler. Nous pouvons voir plusieurs fenêtres sur notre bureau. Si nous sommes dans un environnement comme KDE avec plusieurs fenêtres, nous pourrions être intéressés à en allouer une à Gambas et donc avoir toutes ses fenêtres sous contrôle. Personnellement, une des premières options que j'active dans KDE est celle permettant à chaque bureau de n'afficher que ces propres icônes.

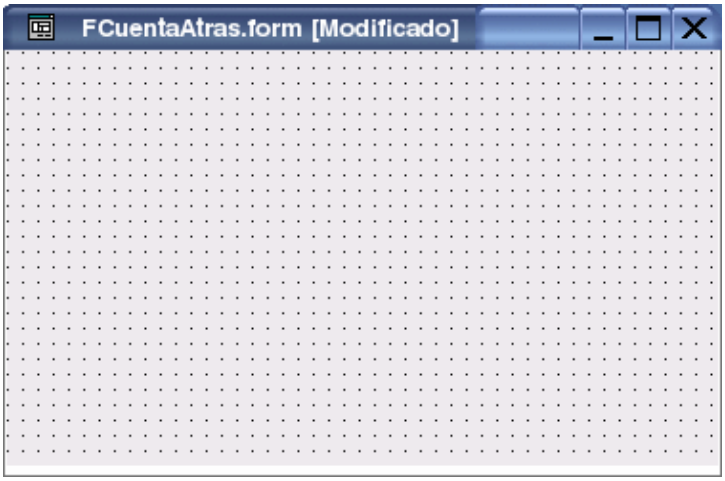
Nous allons créer le formulaire principal de l'application. Pour cela, nous allons faire un clic−droit sur n'importe quelle partie de la fenêtre du projet et créons un nouveau formulaire.



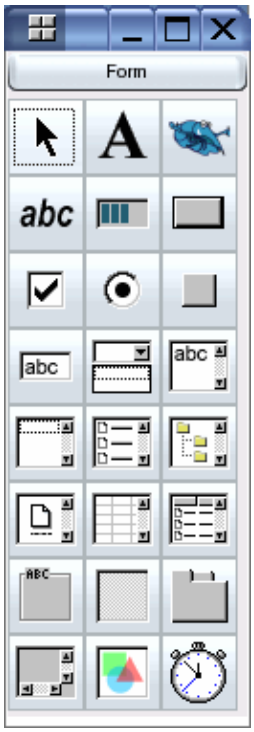
Dans la boîte de dialogue, nous indiquons le nom du formulaire (dans notre cas *FCountDown*) avec toutes les valeurs qui sont laissées par défaut.



Nous avons déjà notre premier formulaire qui est vide jusqu'à présent.



Nous allons inclure ici les contrôles pour notre chronomètre. Nous allons cliquer sur les éléments de la barre d'outils que nous allons inclure dans notre formulaire. Nous pouvons voir leur nom si nous déplaçons la souris au dessus de chaque contrôle. Au moyen d'un double clic, le contrôle sera positionné à l'extrémité gauche de notre formulaire. Avec un simple clic, nous allons changer sa taille et le positionner dans la partie du formulaire vers laquelle nous le destinons. Pour notre programme, nous allons avoir besoin d'une étiquette, une zone de texte, une minuterie, trois boutons dont un bouton à bascule.



Une fois que tous les contrôles sont en place, nous devrions avoir quelque chose qui ressemble à ceci (plus ou

moins, chacun des contrôles pouvant être positionné n'importe où) :

Une fois que nous avons les contrôles dans notre formulaire, nous allons changer les noms de ceux−ci pour avoir quelque chose qui a du sens pour nous. Pour cela, nous éditons la propriété **Name** sur la **feuille des propriétés**. Si nous ne voyons pas la **feuille des propriétés** sur l'écran, nous pouvons l'activer depuis la fenêtre du projet avec le bouton de propriétés. Pour le trouver nous pouvons bouger la souris au dessus des boutons afin de localiser celui qui nous intéresse.

Je nomme le contrôle *Label1* en *lblContador:*. Je clique sur le contrôle et ensuite, je change son nom dans la **feuille des propriétés**. Pour cela, j'édite la proporiété **Nom** et je lui donne *lblContador* comme valeur. Après cela, je change son type de police pour en avoir une plus grande. Pour cela, dans le bouton ... dans sa propriété **police**, je choisis le type de police *Courier Bold 72* et j'accepte (**OK**). De la même façon, je change le nom *ToggleButton1* en *tglFuncionando*. Le contrôle *TextBox1* devient *txtSegundos*, le contrôle *Timer1* devient *clkMiReloj*, *Button1* devient *cmdPonerSegundos* et pour finir, je renomme *Button2* en *cmdSalir*. A coté, je change l'**Alignement** de *txtSegundos* en *Droit*.

Et nous commençons avec le code Basic. C'est très simple et pas très strict du point de vue syntaxique. Ce que nous ferons d'abord, c'est changer les textes que nous voyons dans le formulaire en des valeurs plus réelles. Même si beaucoup des options sont changées en Basic, nous pourrions faire cela dans chacune des feuilles de propriétés des contrôles. L'une ou l'autre des options donnera le même résultat.

Dès que le formulaire s'ouvre, nous remplissons les titres que nous désirons pour chaque contrôles que nous avons. Lorsque nous disons *dès que le formulaire s'ouvre*, nous parlons de la gestion d'un évènement : l'ouverture du formulaire. Pour cela, nous faisons un double−clic dans une partie du formulaire qui n'a pas de contrôle. Une fenêtre d'édition s'ouvre et le curseur est situé dans une nouvelle procédure : **Public Sub Form_Open()** (si vous avez déjà programmé auparavant dans Visual Basic, nous devrions utiliser l'évènement Form_Load). Nous allons faire que le contrôle *lblContador* nous montre les secondes restantes du décompte. Les première lignes de code de la classe formulaire ressemble à cela :

```
' 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
```

Nous avons ajouté juste à la suite du commentaire que Gambas a généré *Gambas class file* qui est une constante qui retient le nombre de secondes par défaut du chronomètre, *fSegundosPorDefecto*, avec une valeur de 120 secondes (deux minutes), et une variable, *fSegundos* laquelle va gérer le décompte. Nous avons créé également deux procédures : *VerValores*, qui affiche la valeur du décompte et *VerActivarDesactivar*, qui change le texte du bouton Start/Stop.

En ce moment, nous avons déjà une formulaire qui fonctionne. Il ne fait rien d'utile à part de nous faire comprendre ce que nous avons déjà fait jusque là. Cela en vaut déjà la peine. Sauvegardons les modifications depuis la fenêtre principale du projet *Project CountDown*, et lançon l'application avec **F5** ou avec le bouton ***Exécuter*** ou le bouton de la barre de la même fenêtre. Voici ce que vous devriez avoir :

Si cela n'apparaît pas ou que vous recevez une erreur, vous devez revoir ce que nous avons fait jusqu'alors. Même si vous appuyez sur **Start**, **Reset** ou **Exit** et que rien ne se passe. Cela sera notre prochaine tâche : assigner les évènements à ces boutons dans la mesure où lorsque l'utilisateur appuie sur l'un d'eux, cela bouge. Avant d'aller plus loin, jouons un peu avec notre application et découvrons tout ce qu'elle contient. Pour la fermer, nous pouvons appuyer sur le X en haut à droite. Je suis sous KDE avec le thème Suse comme vous

pouvez le remarquer dans les formulaires et il est possible que vous puissiez fermer votre fenêtre d'une autre manière.

En avant pour les boutons les plus simples : Que se passe−t−il lorsque l'utilisateur appuie sur **Exit**? Nous devons fermer l'application. Pour saisir le code Basic qui sera exécuté quand l'utilisateur appuye sur le bouton, nous allons double−cliquer sur le bouton avec le texte **Exit** (*cmbExit*). Nous voyons que Gambas gère quelques lignes de code et que le curseur est situé entre elles. C'est ici que le code doit être saisi. Cette procédure sera exécutée lorsque l'utilisateur clique sur ce bouton. Pour fermer l'application, nous devons exécuter Me.Close ainsi le code de cet évènement sera :

```
PUBLIC SUB cmdSalir_Click()

  ME.Close

END
```

La bouton suivant que nous allons contrôler est le **Reset**. De la même manière, nous double−cliquons sur le bouton et dans la fenêtre de code que Gambas nous présente, nous insérons :

```
PUBLIC SUB cmdPonerSegundos_Click()

  fSegundos = txtSegundos.Text

  VerValores

END
```

Jusqu'ici, on dirait toujours que rien ne se passe. Notre application doit être plus vivante. Nous allons activer l'objet *Timer* situé dans le formulaire depuis le début. Pour le faire, nous devons paramétrer l'intervalle pour recevoir les évènements de l'horloge. Soit nous le faisons à partir du code, dans notre évènement précédent *Form_Open*, soit nous le mettons dans le formulaire. Nous le ferons de cette manière. Dans le formulaire, nous cliquons sur l'objet *Timer* et sa **fuille de propriétés**. Nous changeons la valeur de son **Delay** de 1000ms à 100 pour recevoir un évènement chaque dixième de seconde ce qui sera la précision de notre chronomètre.

Nous n'avons toujours pas de code qui sera exécuté chaque fois que l'horloge sonne et la manière de l'activer. Pour générer le code de l'horloge, rien n'est plus simple que, comme toujours, double−cliquer sur le formulaire de l'horloge. Cela nous amènera à la fenêtre de code à l'endroit adéquat. Après l'insertion de notre code, il devrait ressembler à ceci :

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

Et finalement, nous activons le chronomètre à la demande de l'utilisateur via le bouton à bascule qui est celui que nous n'avons pas encore géré. Avec un double−clic sur le bouton, nous pouvons insérer le code pour cet évènement :

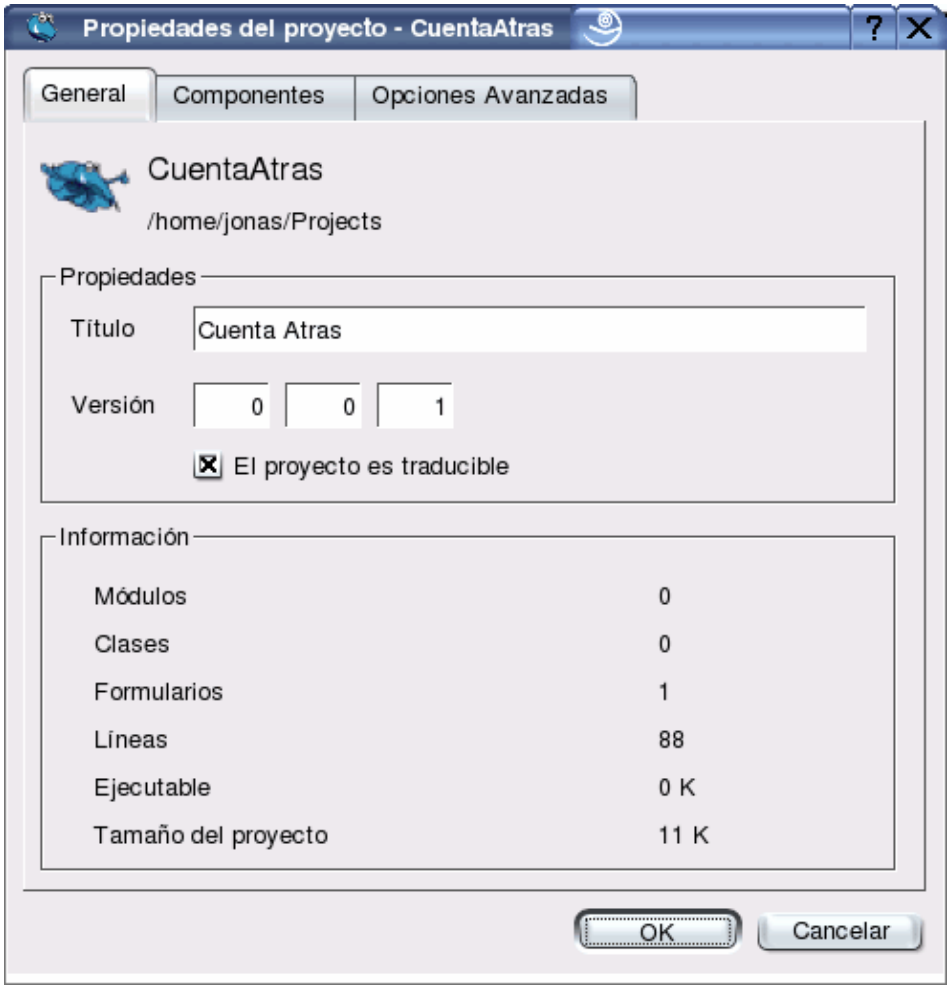
```
PUBLIC SUB tglFuncionando_Click()
  clkMiReloj.Enabled = tglFuncionando.Value
```

```
  VerActivarDesactivar
END
```

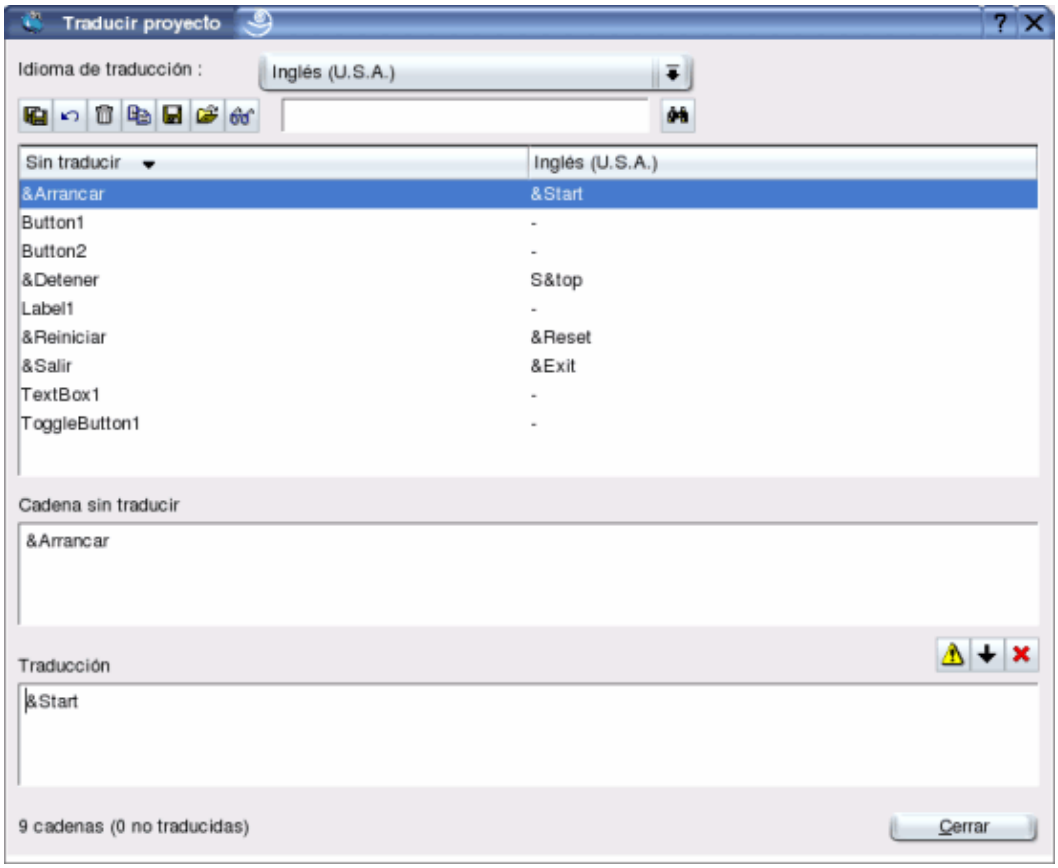
Et maintenant, nous pouvons enfin tester le résultat de notre travail.

Et la touche finale : Gambas est multilingue comme il se doit

Une autre des caractéristique de Gambas est son support de différentes langues. Si vous regarder bien le code, nous remarquerez que les chaînes sont entourées de parenthèses. C'est pour montrer que Gambas va être traduit. Le texte des contrôles du formulaire n'ont pas besoin de parenthèses. Notre projet va devenir quelque chose de très utile et les utilisateurs vont demander à ce que les boîtes de dialogue apparaissent dans leur langue. Rien de plus simple. Nous allons dans le menu **Project / Properties** de la fenêtre du projet.



Nous mettons un **titre** à notre projet et activons l'option **Project is translatable** qui permettra de traduire les dialogues. Nous avons maintenant une nouvelle option active dans les menus : Project / Translate. Si nous ouvrons le dialogue, nous pouvons voir que la traduction est maintenant très intuitive :



D'abord, nous sélectionnons la langue cible dans la partie supérieure. Lorsque nous voulons traduire une chaîne, nous la sélectionnons et remplissons la partie inférieure. Une fois toutes les chaînes traduites, nous pouvons la tester en lançant l'application depuis un terminal si nous avons réglé la variable LANG avec la langue de la traduction. Si vous voulez voir ce que donne la traduction en anglais, je ferme Gambas et j'exécute

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

Pour revenir à la situation précédente, je lance Gambas et je le lance depuis le menu KDE juste parce que la variable d'environnement n'est pas définie. Elle n'est active que dans sa console.

Conclusion

Même si c'est un langage interprété et que nous avons besoin que tout Gambas soit installé, c'est une bonne option de se lancer dans le développement d'applications pour le bureau Linux. Comme nous l'avons vu, c'est très simple et le développement se fait très rapidement. C'est suffisant pour la plupart des aplications quotidiennes..

L'aide à l'écran est assez complète avec les exemples disponibles dans le menu **File/Open example**. Nous pouvons nous rendre également sur la page d'accueil du projet. Dans la section liens, il y a pas mal de projets en Basic qui peuvent être intéressant. Ce n'est que le commencement d'un projet pour lequel je prévois un futur prometteur.

Translation information:
es −−> −− : Jonás Alvarez <jalvarez(at)eitb.com>
es −−> en: Miguel Alfageme Sánchez, Samuel Landete Benavente. <mas20(at)tid.es>
en −−> fr: Laurent RICHARD. <kouran(at)linuxmail.org>

2005−06−02, generated by lfparser_pdf version 2.51