Desenvolvendo aplicações com Gambas.

Este tutorial é um exemplo de um programa feito com Gambas

Sumario: Vamos criar uma aplicação simples com gambas. Veremos como se programa os eventos e alguns truques e técnicas de trabalho com este magnífico ambiente de desenvolvimento.

David Asorey Álvarez. Fevereiro de 2005.

- Introdução
- Primeiros passos
- Gestão e eventos
- Cnsiderações relativa ao desenho de formulários
- Direto ao assunto...
- Ação "Limpar"
- Ação "Adicionar"
- Ação "Modificar"
- Ação "Apagar"
- Ação "Sair"
- Ação "Abrir"
- Ação "Salvar"
- Ultimos ajustes
- Nosso programa funcionando
- Distribuindo nossa aplicação
- Conclusões
- Sobre este documento e o autor
- Notas

Introdução

Gambas é uma ferramenta visual para desenvolvimento de aplicações muito similar aos conhecidos programas comerciais Microsoft **Visual Basic** ou **Borland Delphi**.

Com o Gambas podemos fazer aplicações ou programas com interface gráfica de forma muito rápida, pois integra um criador de formulários ou janelas, um editor de código, um explorador de classes, um visualizador de ajuda, etc.

Este tipo de ferramenta é muito comum na plataforma Microsoft Windows, mas, para Linux não existia muitas, ou não estavam tão apuradas. Podemos encontrar **Kdevelop**, **Kylix** ou **VDR Builder**. Temos que destacar que no desenvolvimento de aplicações em **Linux** existe uma grande tradição e costume de empregar muitas ferramentas diferentes, cada uma especializada em uma determinada tarefa (por exemplo, um compilador, um editor, um depurador, cada um em separado), por isso é que este tipo de ferramentas integradas (**IDE**) não tinham aparecido até a pouco tempo.

Existe um grupo de programadores e desenvolvedores que estão acostumado com estas ferramentas integradas, seja porque acostumaram a trabalhar com elas em outras plataformas ou porque sãomais comodas ou fácil.

Gambas é uma ferramenta, que, na palavra de seu autor, Benoît Minisini, permite a criação de prorgramas poderosos, de forma fácil e simples. A linguagem de programação que se utiliza é uma versão do "velho" BASIC. Pode nos surpreender a escolha de uma linguagem tão básica e inclusive tão limitada como é o BASIC, mas, não podemos esquecer que um dos objetivos dessa ferramenta é sobre o desenvolvimento de aplicações por pessoas com pouca experiência em programação.

O objetivo deste tutorial é apresentar um pouco desta ferramenta,mas, vamos pressupor que o leitor já sabe programar um puco, e que termos como ***funções, evento, variável*** e similares lhe são familiares. Há excelentes tutoriais disponíveis na internet (1) e o próprio programa incorporaum navegador de documentação bem completo.

A versão atualizada do Gambas ao escrever este tutorial é a 1.0-1. A página web do Gambas está em http://gambas.sourceforge.net

Download do programa de exemplo: agenda.tar.gz
Este tutorial em pdf: gambas_tutorial.pdf

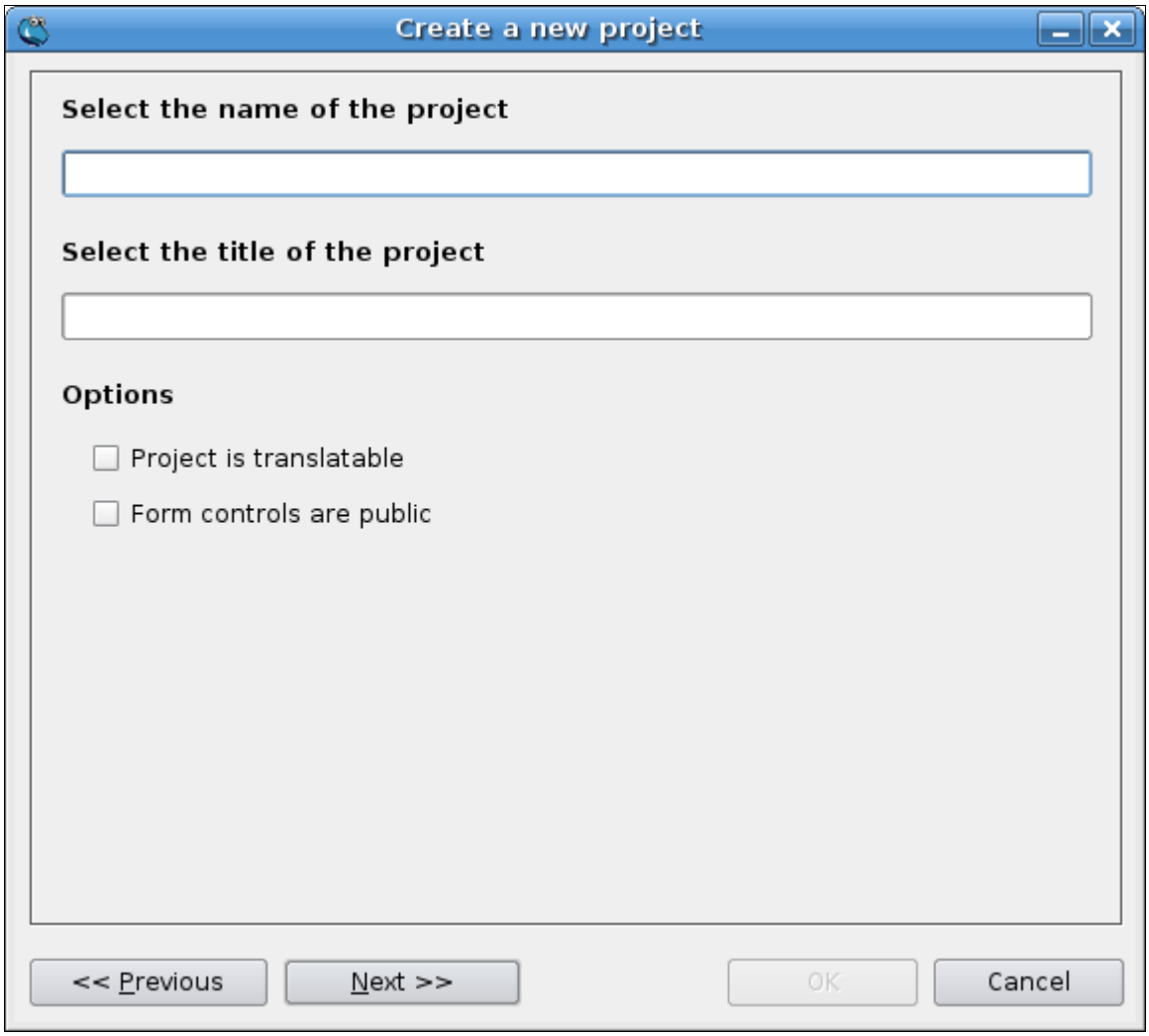
Primeiros passos

Para não nos repetimos e contribuir com algo mais aoque já foi escrito, não vamos descrever como é o ambiente de desenvolvimento, nem para que serve cad ferramenta, etc. Na própria documentação do Gambas vem alguns tutoriais introdutório e um paragrafo chamado "**Visual introduction to Gambas**".

Neste tutorial tentaremos fazer um programa completo e funcional a partir do inicio, e solucionaremos as necessidades segundo vão surgindo.

Vamos criar um programa que seja uma espécie de caderneta ou agenda para tomar notas. Onde podemos adicionar ou apagar notas, alem de modificar as existentes. A qualquer momento podemos salvar as notas em um arquivo ou recuperaroutras de um arquivo.

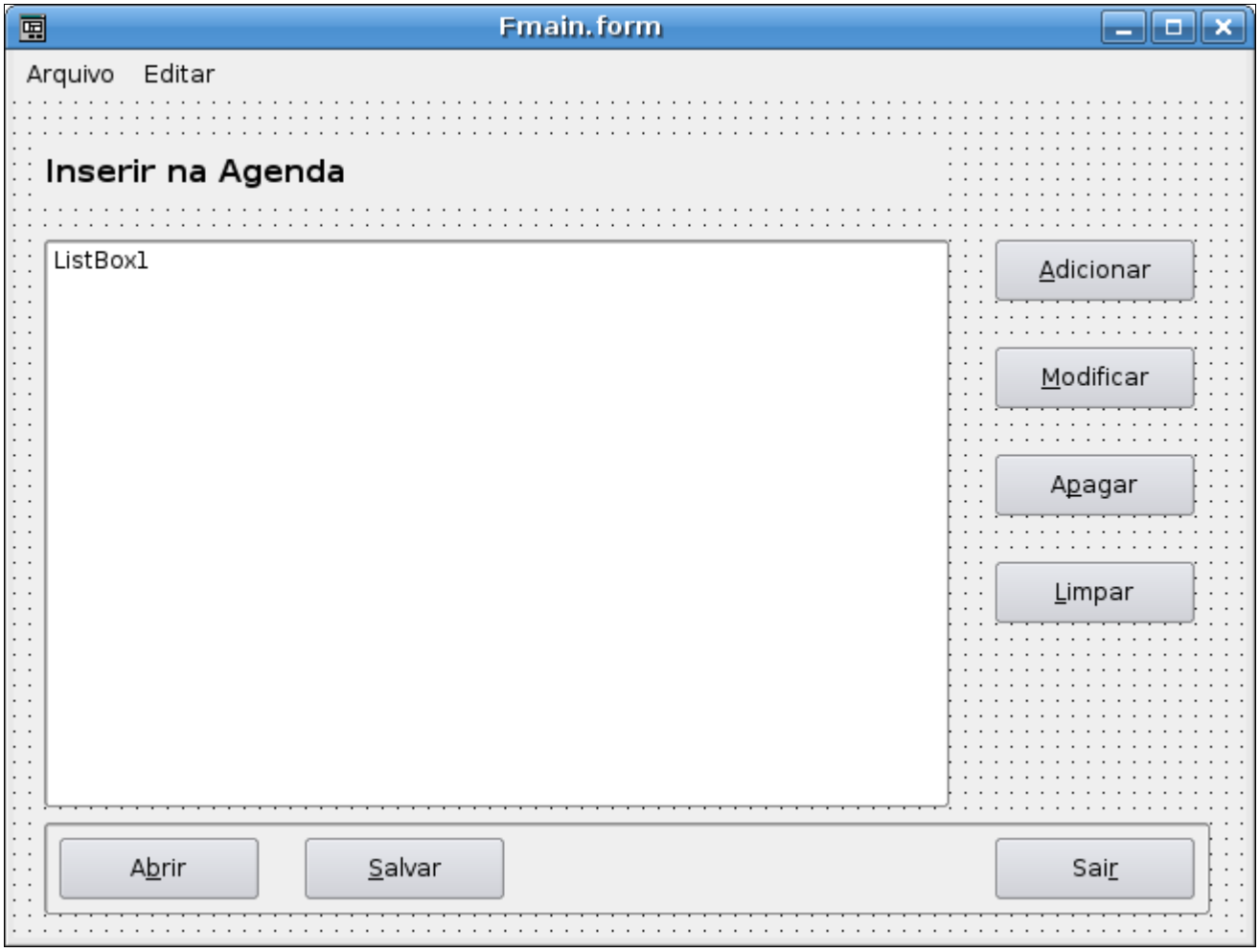
No Gambas, selecionemos a opção "**Novo projeto**" (New project...). Selecionemos "**Criar um projeto gráfico**" (Creat a graphical project) e o programa nos pede alguns dados como o nome e título do projeto:



Também nos deixa escolher duas opções adicionais: "**O projeto é Traduzível**" e os controles do formulário são "públicos". Os deixemos sem marcar e seguimos.

Em seguida selecionaremos o diretório onde queremos salvaro projeto e finalizaremos o assistente para a criação de projetos. Com o botão direito pressionaremos sobre o ícone "**Formulários**" (Forms) e selecionamos a opção "**Novo formulário**" (New | Form)

Vamos desenhar o formulário com um "ListBox" e vários botões para adicionar, modificar, apagar, etc. O desenho que propomos seria igual a esse:

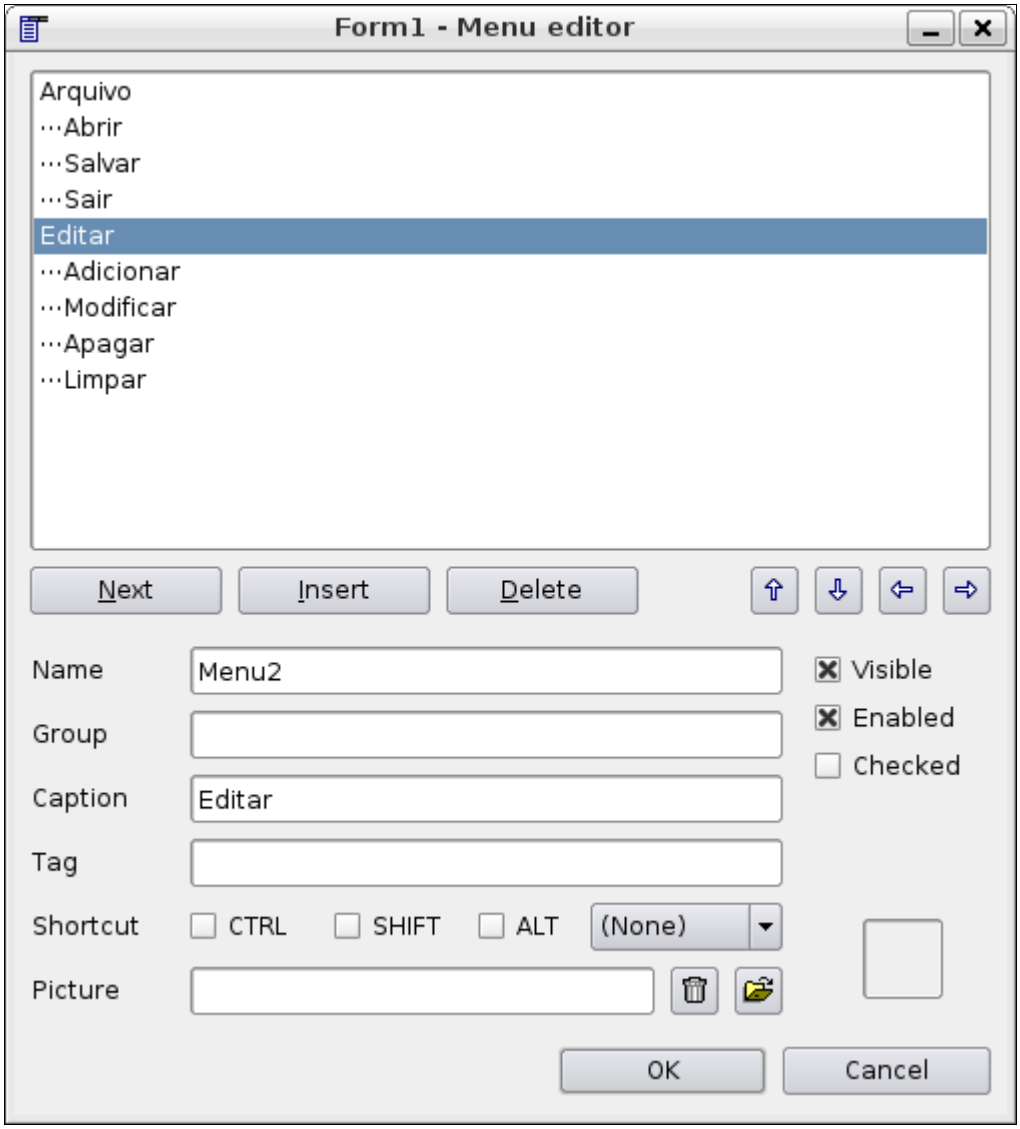


Temos um "Label", um "ListBox" e vários botões, que inserimos no formulário selecionando-os na caixa de ferramentas e "desenhando-os" sobre o formulário. Em destaque neste caso são os botões "**Abrir**", "**Salvar**" e "**Sair**", que os colocamos sobre um "**Panel**" em vez de sobre o formulário diretamente.

Para fazer que os botões responda ao "atalhos do teclado", temos que por um "ampersand" **(&)** Antes da letra que servirá como "atalho"

Para criarmos o menu, pressionamos com o botão direito em qualquer ponto vazio do formulário e selecionamos a opção

"**Editor de menu**" (Menu editor):



Ao criarmos os botões e as diversas entradas no menu podemos observar na janela de propriedades que há, à parte das opções típicas (nome, texto a mostrar, etc.) uma opção chamada "**Grupo**". Esta opção é muito interessante, já que se tivermos vários controles (por exemplo, o menu "**Abrir**" e o botão "**Abrir**") que deve fazer o mesmo, associando-os ao mesmo grupo só temos que escrever o código correspondente ao grupo de ações a que pertence cada controle.

Assim, em nosso programa de exemplo, vamos associar ao grupo "**Abrir**" o menu e o botão "**Abrir**", ao grupo "**Salvar**" o botão e o menu "**Salvar**", etc.

Se agora dermos um click em um botão ao em um menu correspondente, abrira-se o editor de código posicionando-se o cursor na declaração de um procedimento que terá o mesmo nome que o grupo de associações.

Gestão de eventos

Os programas com interfacegráficas baseia seu funcionamento em eventos. Isto é, cada vez que o usuário "faz algo" na aplicação, gera-se um evento e este evento pode estar associado a uma função ou procedimento que corresponda a ação do usuário.

Se, por exemplo, o usuário da um click em um determinado controle, geram-se vários eventos:

MousePress, ao pressionar o botão do mouse, ***MouseRelease,*** ao liberar o botão do mouse, ***Click*** como resultado desta ação. Se o usuário da um duplo click, o evento gerado é um ***DblClick***. Nem todos os controles são capazes de responder a todos os eventos. Não faz sentido haver um evento ***Resize*** em um botão, já que este evento gera-se ao redimensionar uma janela.

Em Gambas, para introduzir o código do procedimento (2) correspondente a um evento, declara-se da seguinte maneira:

```
PUBLIC SUB   Control_Evento   ()
```

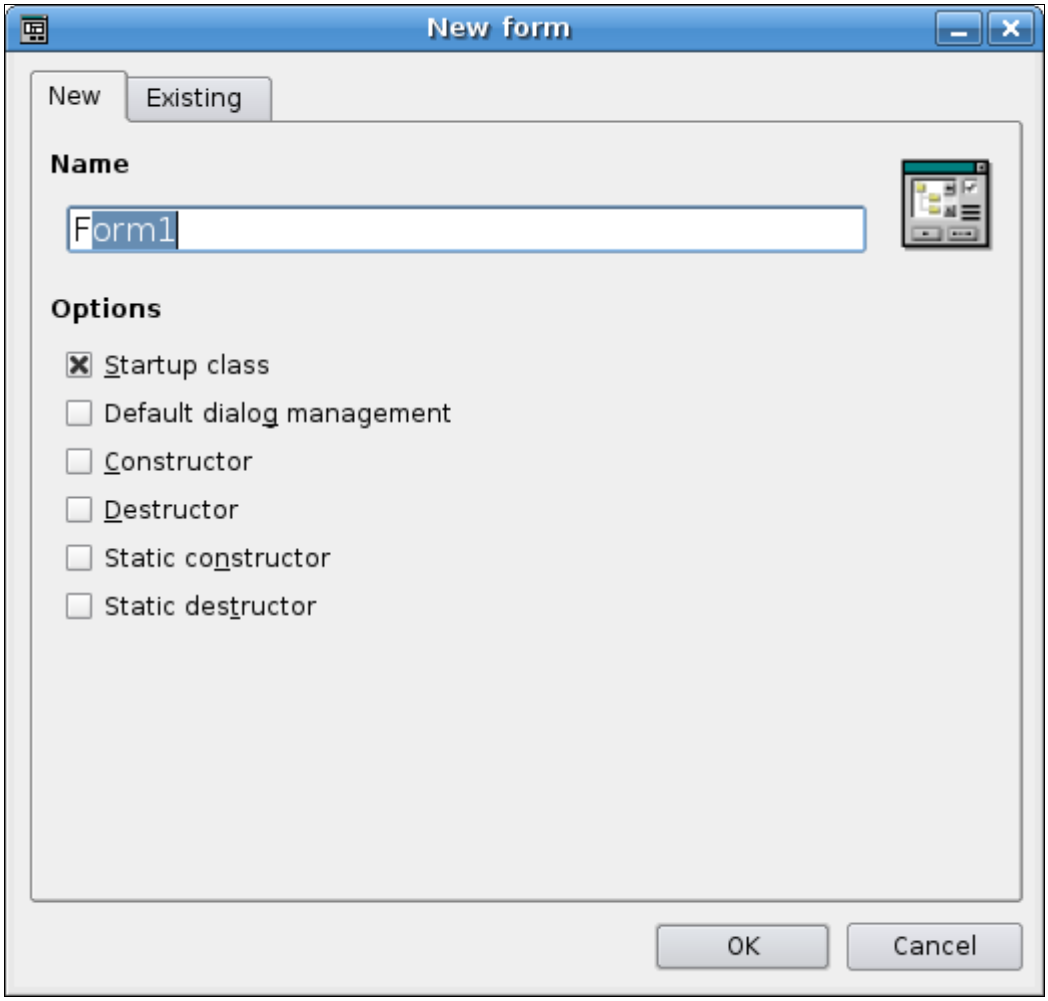
Onde **Control** é o nome do controle que responde ao evento e **Evento** é o evento que produziu-se. Alguns controles tem um evento predeterminado, que é mais usual: um botão tem como evento evento predeterminado o **Click,** etc.

Em Gambas ao darmos um click sobre qualquer controle, abre-se o editor de código na declaração do evento predeterminado, com uma exceção. Como comentávamos antes, se o controle está associado a um grupo de ações, o editor abre-se na declaração do procedimento correspondente ao grupo de ações.

Considerações relativas ao desenho de formularios

A desenharmos o formulário da aplicação, devemos levar em conta várias questões:

- Nem todos os usuários utilizam a mesma resolução da tela, gestor de janelas e tipo de fontes. Temos de ter cuidado e não "**aproveitar**" demais o espaço. Podemos acabar com etiquetas de texto (**Label**) ilegíveis, botões com o texto cortado, etc.
- Pela mesma razão, convém que a janela principal da aplicação seja redimensionável para o usuário (em Gambas é a propriedade ***Border*** do formulário). Não é recomendado deixar esta propriedade como ***Fixed.***
- Ao criar um formulário temos varias opções que nos parece interessantes:



As opções relativas ao construtor e destrutor nos serve no caso de querermos fazer alguma operação sobre o formulário antes de visualiza-lo e ao encerra-lo, respectivamente.

Aparecem as seguintes declarações:

```
' Gambas class

file  PUBLIC  SUB  _new ()

END

PUBLIC  SUB  _free ()

END

PUBLIC  SUB  Form_Open  ()

END
```

- Se selecionamos as opções "Construtor Estático (Constructor Static)" e "Destrutor Estático

(Destructor Static)" as declarações que nos parecem agora no editor de código são:

```
' Gambas class file

STATIC PUBLIC SUB _init ()

END

STATIC PUBLIC SUB _exit ()

END

PUBLIC SUB _new ()

END

PUBLIC SUB _free ()

END

PUBLIC SUB Form_Open ()

END
```

Podemos assim alterar o comportamento de nossa aplicação ao abrir ou ao encerrar o formulário. Quando o procedimento está declarado como STATIC significa que só poderá acessar a variáveis declaradas também como STATIC.

Direto ao assunto ...

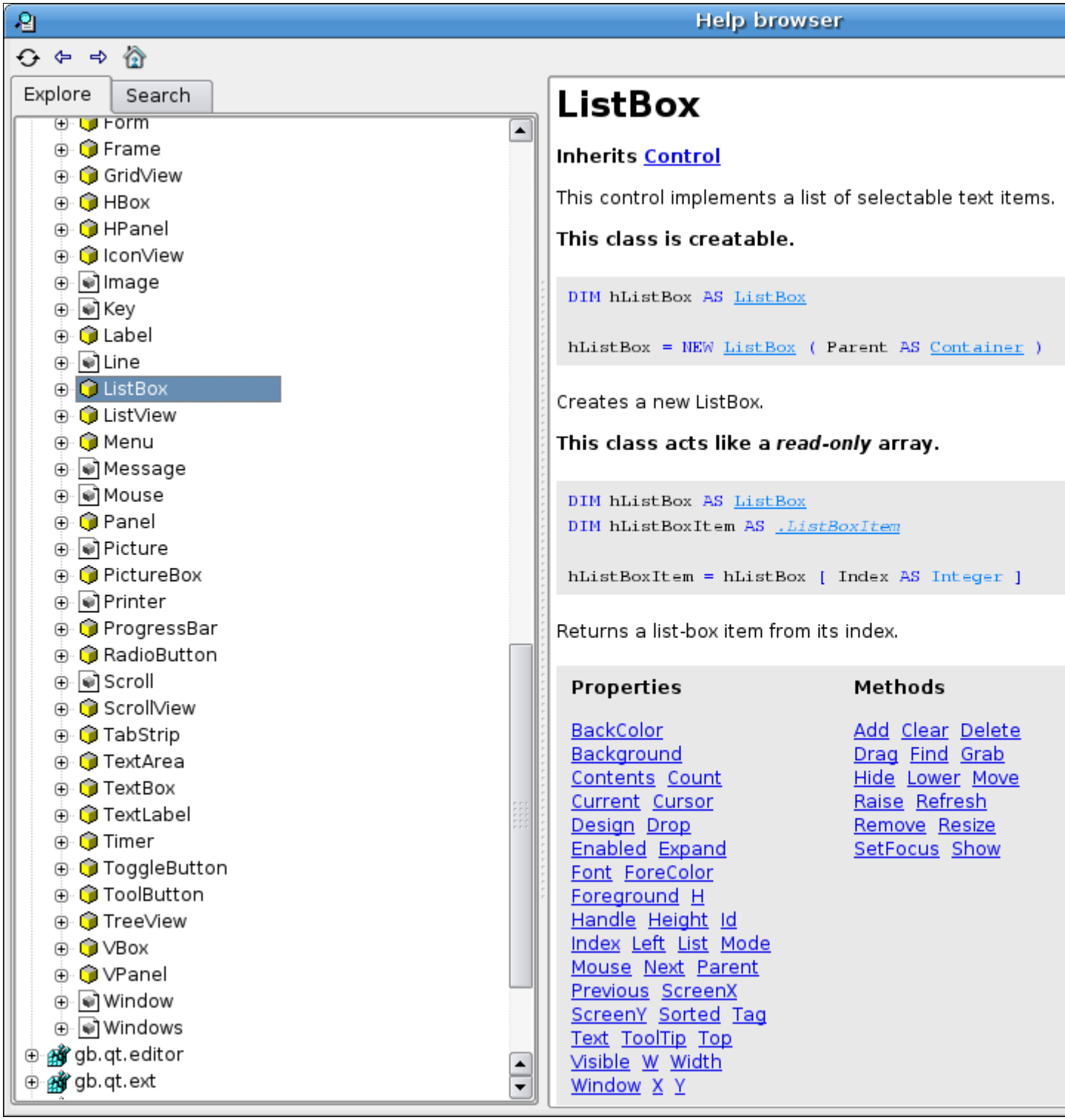
Já temos nosso formulário desenhado. Vamos implementar funcionalidades aos controles.

O primeira coisa que vamos fazer para que os botões "Adicionar", "Modificar", "Apagar" e "Limpar" (e as entradas correspondentes nos menus) funcionem:

Ação "Limpar"

Este botão encarrega-se de apagar todas as entradas que exista no **ListBox**. Para saber como fazer isto,

procuramos no navegadorde ajuda a documentação relativa ao controle **ListBox:**



A documentação encontra-se abaixo na "Arvore" **gb.qt**, que é onde encontra-se a documentação de todos os controles do tipo "visual" (botões, label, menus, etc....). Vemos que o ListBox possue um método "**clear**", que precisamente faz o que queremos: apagar todo o conteúdo do controle.

Damos um click no botão "**Limpar**", abre-se o editor de código no procedimento correspondente. Adicionemos o seguinte código:

```
PUBLIC SUB Limpiar_Click ()
```

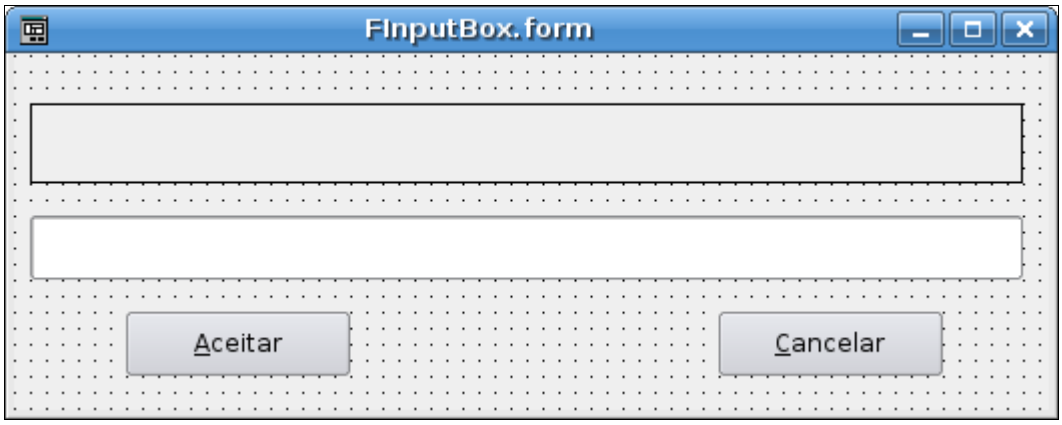
```
    ListBox1.Clear ()

END
```

Fácil, não é?.

Ação "Adicionar"

Este já é um pouco mais complicado. Queremos que o usuário, ao pressionar o botão, possa escrever uma linha de texto que carregue-se no ListBox.

Gambas não proporciona por padrão um diálogo do tipo "**InputBox**", sendo assim vamos criar o nosso próprio InputBox. Criamos um novo formulário, mas agora queremos dispor de um construtor. Por que?. Porque no momento de cria-lo trocaremos algumas propriedades como título, alguma mensagem de texto e um valor por padrão para a entrada de texto. Este é o desenho que propomos:



O formulário não tem muita complicação. Dispõe de um **label** e uma caixa de texto (**TextBox**) e dois botões. Como qualquer caixa de diálogo que se presa, é conveniente que possamos cancelar com a tecla Escap (**Esc**) e aceitar com a tecla **Enter**:

Os Controles **Button** tem duas opções adequadas para esta situação. São "***Defoult***" e "***Cancel***". Para o botão "Aceitar", colocamos "***Defoult***" a **True** e "***Cancel***" a "**False**" para o botão "Cancelar", colocamos ao contrario.

Desta aneira, quando abre-se o formulário, ao pressionarmos a tecla <ENTER> será equivalente a pressionar o botão "Aceitar" se pressionarmos a tecla <ESC> simulará o botão "Cancelar".

Temos o seguinte problema, como retornar o valor que o usuário introduziu na caixa de texto para a janela principal. Temos que destacar que em Gambas não há variável global, sendo assim teremos que procurar outra solução. Criamos então um módulo no qual ponhamos uma variável PUBLIC, assim podemos acessa-la de qualquer ponto da aplicação.

Criamos um módulo (botão direito em "**Modules | New | Modules...**") o chamamos de **Mcomum,** por exemplo. Esta seria a implementação do módulo:

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

Assim sem mais. Agora temos uma variável visível a partir de qualquer ponto do programa que pode ser acessada com: **Mcomum.texto**

A seguir é implementar o formulário que faráas vezes do "**InputBox**". Esta seria sua implementação:

```
' Gambas class file
PUBLIC SUB _new(Titulo AS String, Mensagem AS String, OPTIONAL Texto AS String)

 ME.Caption = Titulo
 Label1.Caption = Mensagem

 ' uma String é avaliada como False si estiver "vazia"
 IF Texto THEN TextBox1.Text = Texto

END

PUBLIC SUB Button1_Click () ' Este e o botão Aceitar

 MComun.texto = TextBox1.Text
 ME.Close (0)

END

PUBLIC SUB Button2_Click () ' Este é o botão Cancelar

 ME.Close (0)

END
```

O procedimento _new éo construtor. Como nos interessa que o texto do label, o título e o texto a editar sejam diferentes acada vez, os ajustamos ao criar a janela.

O botão "Modificar" insere o texto do **TextBox** na variável **Text** do módulo **Mcomum** e fecha o

formulário. O Botão "Cancelar" simplesmente fecha a janela

Como a variável **Mcomum.Texto** é comum, temos que lembrar de "limpa-la" cada vez que a utilizemos. Vamos ver agora mesmo.

O procedimento para o botão "Adicionar" do formulário principal é o seguinte. É bastante explicativo:

```
PUBLIC SUB  Adicionar_Click  ()
   ' Declaramos nosso "Inputbox"
   f AS FInputBox

   ' Criamos el InputBox, passando-lhe o
título, mensagem a mostrar
   ' e un valor por padrão: a data e a
hora momento  e uma flexa
   f = NEW FInputBox( "Editar",
"Escreva a linha que deseja
adicionar:" , CStr(Now)  & " -> " )
   ' O mostremos
   f.ShowModal ()

   ' Se poressionarmos agora Aceitar
introduzimos o texto, ficará na variável
MComun.texto
   IF MComum.texto  THEN    'Uma cadeia
vazia e False
      ' El controle listBox tem um método
para adicionar texto: .Add
      ListBox1.Add (MComum.texto )
      ' "Esvaziamos" a variável comum
      MComum.texto  = ""

 END IF
END
```

Ação "Modificar"

Ao pressionarmos este botão, o usuário modificará algumas das entradas que exista no ListBox. Se não tiver nenhuma, o botão não deve fazer nada, e se não for selecionada nenhuma linha, mostrará uma mensagem de aviso. Vamos a implementação do procedimento associado.

```
' Ação "Modificar"
PUBLIC SUB  Modificar_Click  ()
  f AS FInputBox
  IF ListBox1.Count  > 0 THEN  ' Se não houver nada no formulário,
                ' sua propriedade count é 0. Neste caso, ' não fazemos nada.
    IF ListBox1.Index  = -1 THEN

      ' a propiedade Index nos devuelve o índice da linha selecionada.
      ' Se não selecionarmos nenhuma, devolve -1. Nste caso, avisamos
      ' o usuário e nã fazemos nada mais.
       message.Info ("Deve selecionar a linha que deseja modificar!"  )
    ELSE

      ' O usuário selecionou uma linha no ListBox.
      ' Mostraremos nosso InputBox, passando também o texto selecionado.
      ' O texto selecionado é a propiedade Text do objeto ListBoxItem
      ' selecionado, o que temos acesso com a propiedade Selected' do ListBox
       f =  NEW FInputBox ("Modificar", " Modifique a linha selecionada:", ListBox1.Current.Text  )
       f.ShowModal ()

      ' O quadro de diálogo FInputBox modifica a variável compartilhada
      ' no módulo MComun.
      ' Se nã estiver vazia, a
```

```
        designamos ao ListBoxItem selecionado.
            IF  MComum.texto    THEN
        ListBox1.Current.Text      = MComum.texto
            ' Como antes, "esvaziamos" a
        variável compartilhada depois de usa-la.
            MComum.texto    = ""
          END IF
        END IF
        END
```

Ação "Apagar"

Como no caso anterior, o ListBox deve ter alguma linha, e o usuário deve ter selecionado ao menos uma. O código é similar ao do botão "Modificar":

```
        PUBLIC SUB   Apagar_Click   ()
          i  AS  Integer
          i  =  ListBox1.Index
          IF i   >=   0   THEN
            ListBox1.Remove(i)       ' El método
        Remove tira uma linha, justo o que
        queremos
          ELSE IF   ListBox1.Count     >  0  AND  i  =
        -1  THEN
            ' Comprovamos que o ListBox não
        está vazia e que haja algo
        selecionado.
            message.Info  ("Deve selecionar a
        linha que deseja apagar."    )
          END IF
         END
```

Podemos observar que a implementação destas quatros ações é comum para os botões e as entradas equivalentes no menu.

Agora passamos a implementar as ações relativa a manipulação de arquivos (Abrir, Salvar) e sair da aplicação. Começaremos pelo mais fácil:

Ação "Sair"

A função deste botão (e a correspondente entrada no menu) é encerrar a aplicação. Nada mais simples:

```
PUBLIC SUB  Sair_Click  ()
ME.Close (0)  ' ME é uma referência ao
própio formulário
'  FInputBox
END
```

Poderíamos fazer está ação um pouco mais amigável agregando um diálogo do tipo *"**Realmente gostaria de sair da aplicação**"* e atuar em consequência. Deixamos esta melhora como exercício para o leitor.

Ação "Abrir"

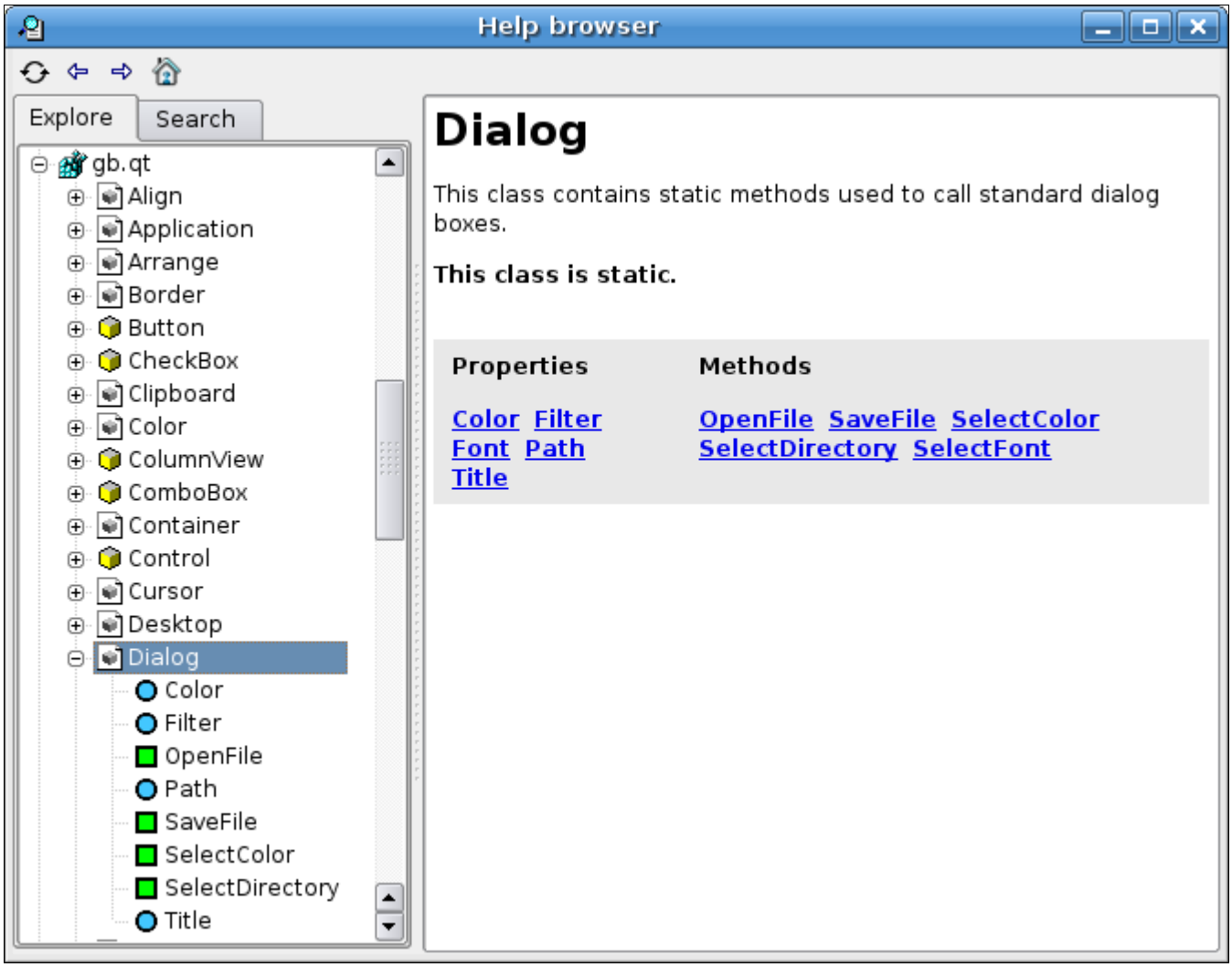
O que deve fazer?. Pedir ao usuário por um arquivo, lê-lo e carregar o conteúdo no ListBox. Vejamos diretamente a ação correspondente:

```
PUBLIC SUB  Abrir_Click  ()
  DIM  c  AS  String
  DIM  arr_cadeias  AS String  []

  Dialog.Title  = "Selecione un
arquivo"
  Dialog.Filter =  [ "Dados da agenda
(*.data)" , "Todos os Arquivos (*.*)"
]

  IF NOT  Dialog.OpenFile  ()  THEN
    arr_cadeias  =
Split(File .LOAD (Dialog.Path ), "\n" )
    ListBox1.Clear  ()
   FOR EACH  c  IN arr_cadeias
     ListBox1.Add  (c)
    NEXT

  END IF
END
```

Esta parte do código apresenta uma característicamuito interessante do Gambas, as classes "não

instanciáveis" ou estáticas (3) são classes que não pode estanciar-se mas podemos utiliza-la diretamente. Nesta ação veremos duas destas classes: a classe "**File**" e "**Dialog**".

Por exemplo, a classe **Dialog** proporciona acesso aos típicos Quadro de diálogo de seleção de arquivo, cores, etc. Está documentada em `gb.qt`.



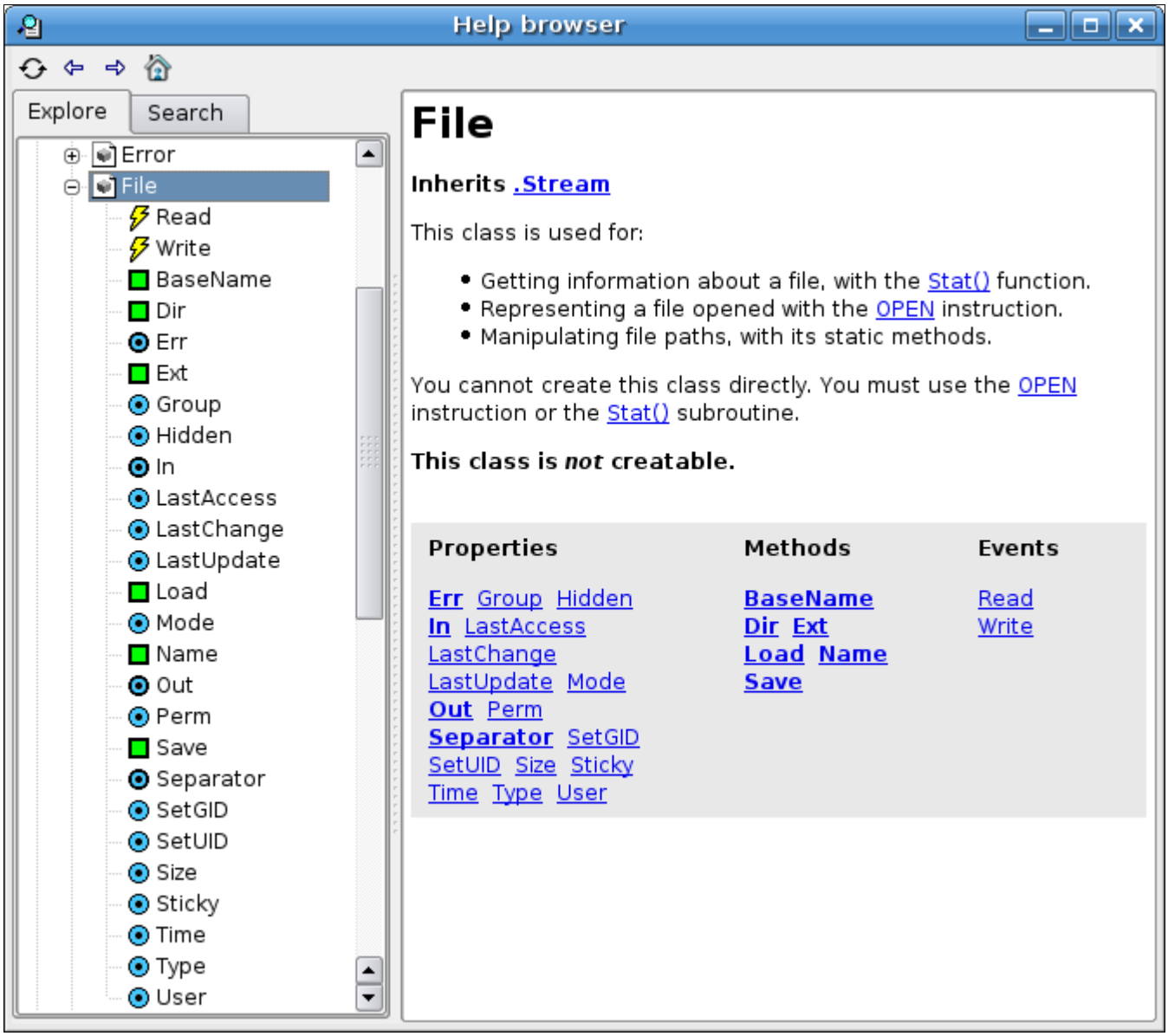
Em nossa aplicação, queremos selecionar um arquivo e carrega-lo. Para fazer isto utilizamos a classe **Dialog** da seguinte forma:

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

Ajustamos o título do quadro de diálogo, proporcionamos um filtro para a seleção do tipo de arquivo por extensão e finalmente invocamos o método **OpenFile()** da classe. Curiosamente se não selecionamos um arquivo (o usuário pressionar "Cancelar", etc. ...) o valor de retorno do método **OpenFile()** é **True**. Uma

vez selecionado o arquivo pelo usuário, podemos acessar o endereço completo com a propriedade **Dialog.Path.**

A Classe **File** (Sua documentação encontra-se na entrada **gb**) proporciona vários métodos para trabalhar com arquivos.



Na documentação do Gambas, na seção "How do I ... " existem vários exemplos para ler e escrever arquivos, nós vamos utilizar em nossa aplicação o método **Load()** que recebe como argumento o endereço de um arquivo e devolve uma **String** com todo o conteúdo do arquivo. Para separar as linhas que contém o arquivo utilizamos a função **Split()** que toma como argumentos a cadeia que queremos "separar", o caracter a utilizar como separador (um fim de linha em nosso caso, "\n") e devolve uma **Array** de **String.** Por isso vamos declarar a variável **arr_cadeias** como **String[]:**

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

Uma vez que temos a lista de cadeias contidas no arquivo, limpamos o ListBox e vamos adicionando uma a uma cada cadeia utilizando o método **Add()** do ListBox.

Ação "Salvar"

Ao pressionar o botão "Salvar" ou a entrada equivalente no menu, o programa deve salvar o conteúdo em um arquivo de texto. Mostraremos um quadro de diálogo ao usuário para que nos proporcione o nome do arquivo a utilizar. Este é o código corrrespondente.

```
PUBLIC SUB  Salvar_Click  ()
 DIM  linhas  AS  String
 DIM destino  AS  String
 DIM numArquivo  AS  Integer
  linhas  =  ListBox1.Contents
   Dialog.Title  =  "Selecione um
arquivo"
   Dialog.Filter  =  [  "Dados da agenda
(*.data)"  ]

    IF NOT  Dialog.SaveFile  ()  THEN
     IF  Right$ (Dialog.Path  ,  5) <>
".data"  THEN

        destino  =  Dialog.Path  &
".data"

      ELSE
        destino  =  Dialog.Path
      END IF
        File.Save  (destino ,  linhas )
   END IF
END
```

Queremos que os dados sejam salvos em um arquivo com a extensão **data**, assim como o nome do arquivo do usuário pode não terminar em ".data", concatenamos manualmente a extensão. Para salvar o conteúdo em um arquivo usamos o método **Save()** da classe **File,** que utiliza como argumento o endereço do arquivo e o texto que queremos salvar. Acessemos o conteúdo do ListBox mediante sua propriedade **Contents**, que devolve uma **String,** com um salto de linha "**\n**" separando cada entrada no ListBox.

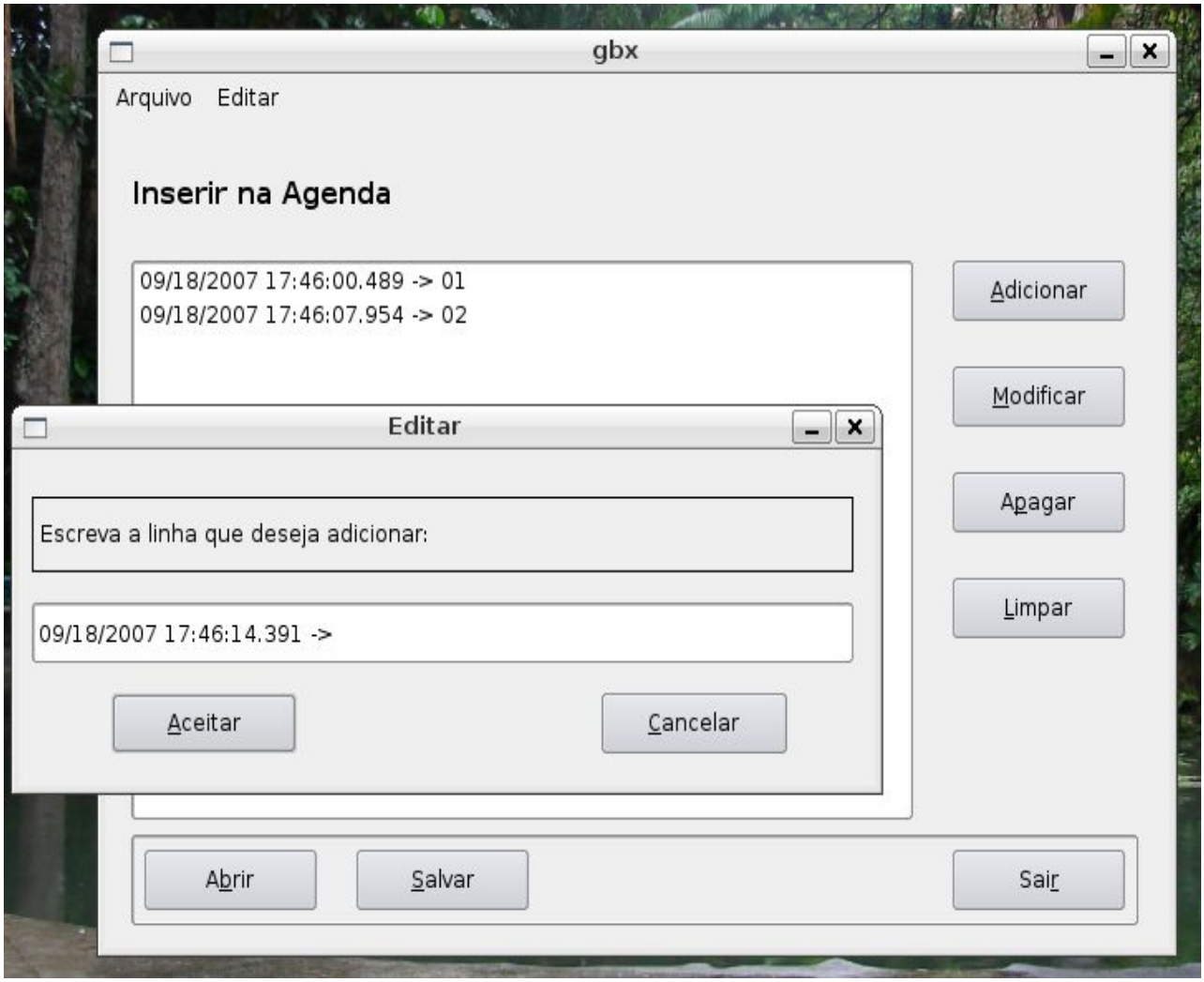
Um último ajuste

Nos ocorre que seria interessante que quando o usuário selecione uma das linhas do ListBox pudesse visualizar o conteúdo completo da linha, já que pode aparecer cortada caso sejam demasiadamente grandes.

Vamos fazer da seguinte forma: quando o usuário der um duplo click em uma linha, mostraremos o conteúdo da linha em um quadro de diálogo:

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

Nosso programa funcionando



Distribuindo nossa aplicação

Já temos a aplicação criada. Podemos testa-la a qualquer momento do desenvolvimento utilizando a tecla **F5**

Agora queremos utiliza-la como um programa normal, sem ter que ter o Gambas funcionando. Para isso existe uma opção no menu principal do Gambas ('**Projeto | Criar executável**"). Em inglês ("**Project |**

Make Executable")**.** Isto nos gera um arquivo executável "Mondítico", ou seja, inclui todos os formulários, implementações e arquivos adicionais do projeto. Este executável não é código de maquina, é um "bytecode" executável pelo interpretador do Gambas, **gbx.** Isto implica que devemos ter o Gambas instalado para podermos executar o programas escrito com Gambas (igual a outras linguagens: é necessário ter Java para executar um programa escrito em Java).

Por sorte, na maioria das distribuições incluem Gambas os componentes estão separados e há um "Gambas runtime", que inclui o interpretador, mas não o ambiente de desenvolvimento completo.

Também podemos criar pacotes RPM ou DEB para nosso programa. Estes pacotes terão como dependência o interpretador Gambas (o g**ambas-runtime**). Há um assistente muito fácil de usar para criar os pacotes (**"Projeto | Criar pacotes de instalação..."**). Em inglês (**"Make installation packag..."**).

Conclusões

Vimos como é fácil criar uma aplicação minimamente funcional com Gambas. Proporciona bastante controles e classes predefinidas. Há também extensões para criar aplicações cliente/servidor, acesso a base de dados, multimídia, etc.

Pessoalmente me parece que é uma ferramenta com muitíssimo futuro, e, afortunadamente, o desenvolvimento do Gambas é muito ativo, corrigindo os erros que vão surgindo com muita rapidez.

Graças, Benoît (et col.)! ¡Excelente Trabalho!

Sobre este documento e o autor

Como mencionado anteriormente, a aplicação foi desenvolvida utilizando a versão 1.0-1 do Gambas (utilizando os pacotes pré compilados para Debian "Sid"). No momento que escrevo este documento a versão 1.0.3 acaba de ser publicada, e no momento que ler este documento Provavelmente há uma versão mais moderna. Convém ler a lista de troca de uma versão para outra pois pode haver alguma incompatibilidade.

Qualquer comentário sugerindo a melhora deste documento é bem vinda meu E-Mail é forodejazz (arroba) gmail (punto) com

Nota legal: Este documento é livre, pode copia-lo, vende-lo, distribui-lo, modifica-lo, traduzi-lo para outras linguagens e inclusive vende-lo, porem sempre conservando esta nota e citando a procedência do documento. Em qualquer caso, o autor agradeceria que o notificasse, em um dado momento, ser retribuído economicamente por seu esforço (se colar, colou ;-)

Notas

1. Há um bom tutorial para iniciantes e documentação em Casterliano em http://gambas.gnulinex.org
2. Os eventos devem ser tratados como procedimentos, isto é, uma função que não retorna valor

algum.

3. Não sou expert na terminologia usada em programação orientada a objetos, sendo que, provavelmente estarei algum termino incorreto. Minhas desculpas ;-)