Разработка приложений в Gambas

Руководство и пример выполнены в Gambas

Резюме: Мы собираемся создать простую программу в Gambas. Мы научимся поддерживать события и некоторые типы и техники для работы с этим удивительным инструментом.

Моя благодарность: Daniel Oxley и Dave Sharples. Они просмотрели и поправили мой бедный английский перевод с испанского.

Загружаемый исходный код (комментарии, имена переменных и т.д.) сделан на испанском. Однако код, показанный на этих страницах, транслировался.

David Asorey Alvarez. February de 2005.

Оглавление

Введение

Gambas - это IDE (Integrated Development Environment - интегрированное окружение разработки), ориентированное на RAD (Rapid Applications Development - быстрая разработка приложений) подобное популярным патентованным программам Microsoft Visual Basic или Borland Delphi.

В Gambas возможно разрабатывать GUI программы (Graphical User Interface - графический интерфейс пользователя) очень быстро, поскольку IDE включает конструктор форм, редактор кода, обозреватель кода, интегрированную систему помощи и т.д.

Такого рода инструменты весьма характерны для мира Microsoft Windows, на платформе Linux есть только несколько, или они на ранних стадиях разработки. Мы находим несколько, подобно Kdevelop, Kylix или VDK Builder.

Есть добрая традиция и обыкновение в мире Linux/Unix использовать множество различных инструментов, каждый из которых специализируется на решении единственной конкретной задачи (например, компилятор, редактор, отладчик - каждый из них используется отдельно). По этой причине программные IDE относительно внове для пользователей Linux/Unix.

Есть множество программистов и разработчиков, кто пользуется такого рода интегрированными инструментами, поскольку они пришли с других платформ или они чувствуют себя комфортнее с IDE.

По словам автора, Benoit Minisini: "*Gambas нацелен на то, чтобы дать вам возможность создавать мощные программы легко и быстро. Но чистота программирования остается на вашей совести...*". Язык программирования, используемый в Gambas - это версия "старого" BASIC. Возможно, что кто-нибудь будет удивлен этим выбором, поскольку BASIC кажется слишком простым и ограниченным. Должны напомнить, что одной из целей Gambas является содействие программированию не программистов.

Цель данного руководства - дать введение в Gambas IDE, и разработать простую программу с его помощью, но мы подразумеваем, что читатель в какой-то мере знаком с программированием, и что термины подобные функции, событию или переменной ему знакомы. Gambas имеет встроенную систему помощи, предоставляющую и вводную, и более детальную документацию.

Версия Gambas, использованная в данном руководстве - это 1.0−1. Домашняя страница Gambas - http://gambas.sourceforge.net

Загрузите исходный код примера программы: agenda.tar.gz
Это руководство в pdf формате: gambas_tutorial_en.pdf (gambas_tutorial_ru.pdf)

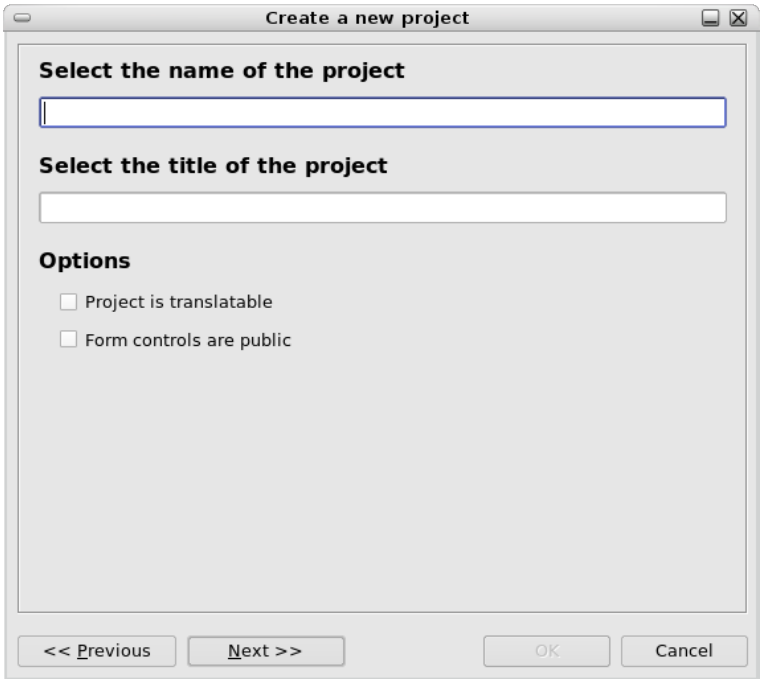
Первые шаги

В Gambas "help" есть документ озаглавленный "Наглядное введение в Gambas - Visual Introduction to Gambas", который рассказывает о программе, ее различных инструментах, окнах и меню. Мы не собираемся повторять эту информацию в данном руководстве.

Мы постараемся разработать законченную программу в Gambas с самого начала, и разберемся с требованиями и проблемами по мере их появления.

Наша программа будет приложением типа записной книжки или памятки. Мы сможем добавлять и удалять записи, модифицировать существующие заметки. Они будут сохраняться и читаться из файла.

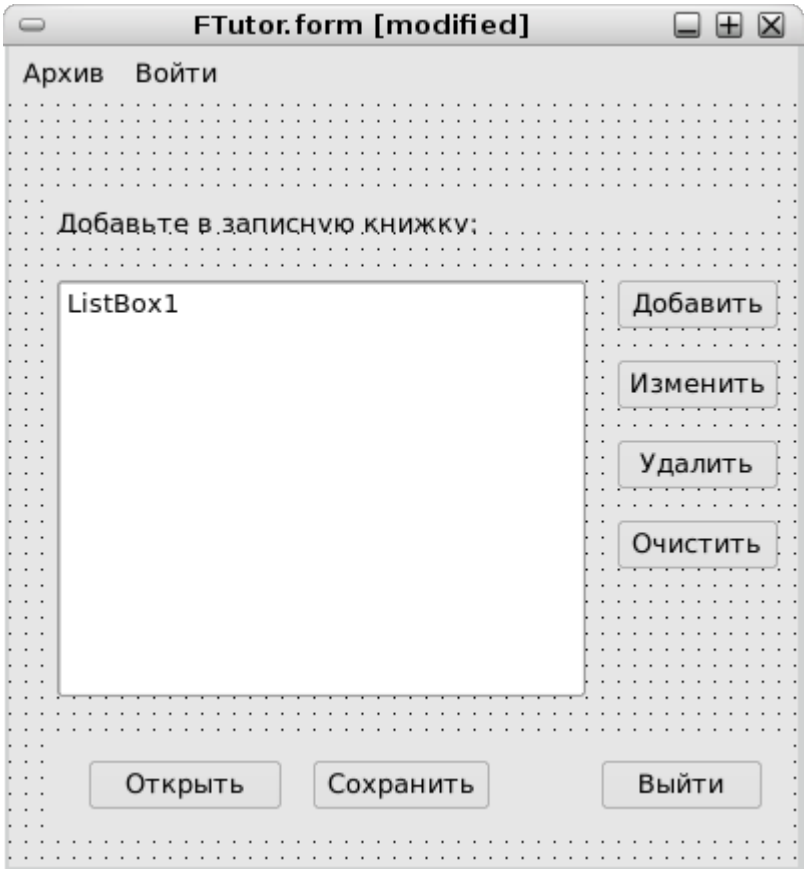
Как только Gambas откроется, мы выберем "New project - новый проект" из меню. Определим его, как "graphical project - графический проект" в мастере проектов и мы затем должны обеспечить некоторую информацию, такую как имя проекта и заголовок проекта:



Есть две опции: "Project is translatable - проект транслируемый" и "Form controls are public - управление формами общее". Мы оставим эти опции не выбранными и нажмем клавишу "Next".

Последняя задача в мастере проектов - выбрать директорию, где будет сохраняться проект. После чего откроется основное Gambas IDE. В главном окне проекта следует щелкнуть правой клавишей мышки на "Forms" и выбрать "New form - новая форма".

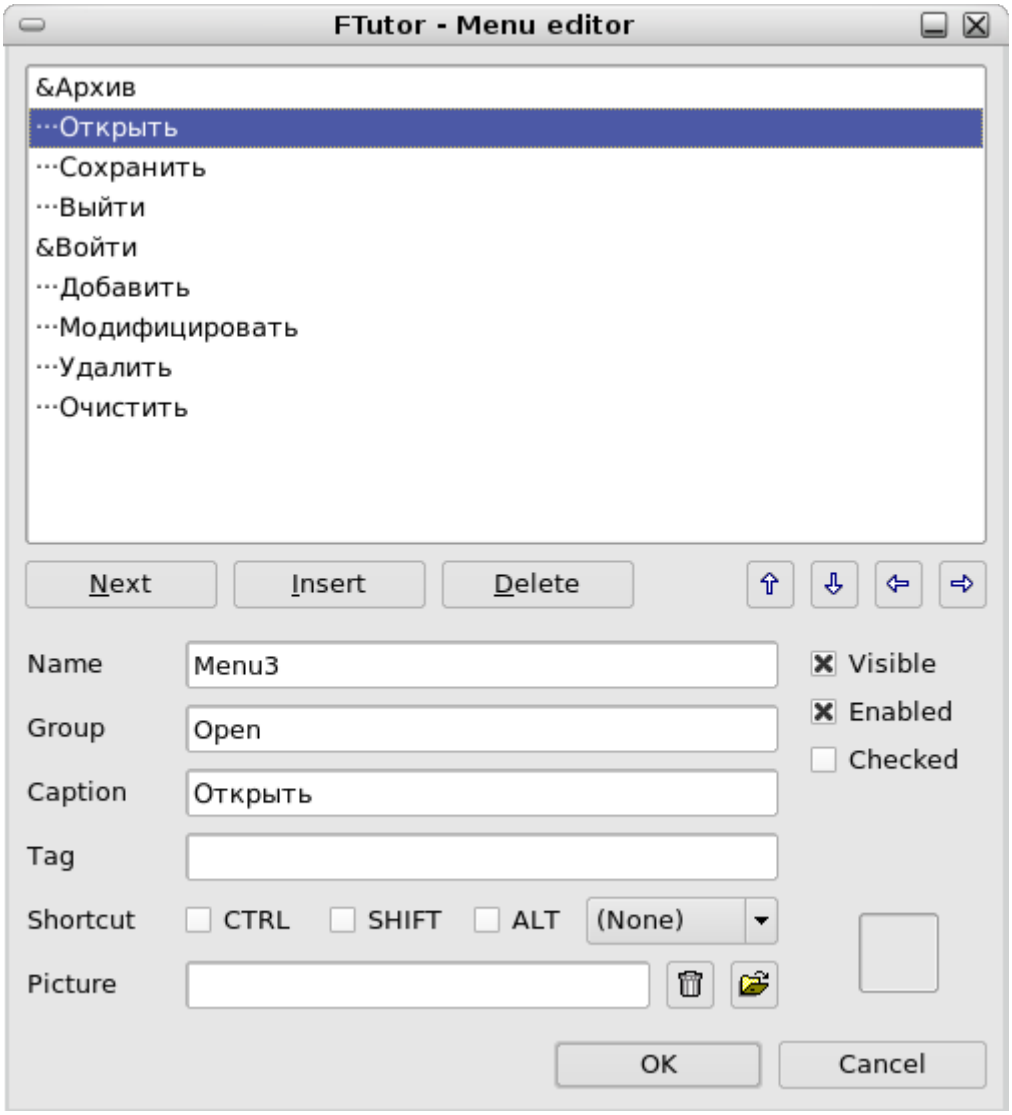
Мы собираемся разработать основную форму, которая содержит управление "ListBox" и несколько клавиш (open - открыть, save - сохранить, exit - выйти, add - добавить, modify - изменить, delete - удалить, ...). Нечто, похожее на это:



Можно видеть несколько общих средств управления: Label - надпись, ListBox - область списка и несколько клавиш. Мы добавляем каждое средство управления в форму, выбирая их на инструментальной панели (Toolbox) Gambas и "рисуя" контур управления на форме. Пожалуйста, отметьте, что клавиши "Open - окткрыть", "Save - сохранить" и "Exit - выйти" расположены на "Panel - панель", которая должна быть добавлена на форму первой и не непосредственно в основное тело формы.

Если мы хотим задать горячие клавиши, мы должны предпослать символ "амперсанд" (&) выбранной букве в тексте клавиши. Например, O&ткрыть, &Сохранить

Мы можем создать и отредактировать основное меню нашей программы, щелкнув правой клавишей мышки на свободном пространстве формы и выбрав в выпадающем меню "Menu editor - редактор меню":



После создания пунктов меню вы можете заметить, что среди типичных свойств, как имя, надпись, и т.д., есть также свойство, названное "Group - группа", клавиши формы имеют это свойство "Group", как вы можете увидеть, выбрав клавишу и просмотрев ее свойства в окне "Properties - свойства". Эта опция очень интересна, поскольку мы можем ассоциировать один и тот же код с разными средствами управления, имеющими одинаковые функции. Например, пункт меню "Open - открыть" и клавиша "Open" должны выполнить одну и ту же задачу: открыть файл на диске. Используя свойство "Group", мы напишем код для этой задачи только один раз.

В нашей программе мы объединим клавишу и пункт меню "Open" в группу, названную "Open", "Save - сохранить" меню и клавишу в группу, названную... "Save", и т.д.

Теперь, если мы дважды щелкнем по клавише или одноименному пункту меню, редактор кода откроется и курсор окажется на заголовке подпрограммы, названной общим именем группы, выбранном для средств управления, ассоциированных с ней. Например: `PUBLIC SUB Open_Click()` , где `Open` - это свойство "Group", названной Open, определенное ранее.

Поддержка событий

Программы с графическим пользовательским интерфейсом обычно "event driven - движимые событиями". Что означает, когда пользователь "заставляет что-либо произойти" в окне приложения, генерируется событие. Это событие может быть связано с функцией или подпрограммой, которые отвечают на действие пользователя.

Пример: если пользователь "щелкнул" по средству управления, генерируются некоторые события - MousePress, при нажатии клавиши мышки, MouseRelease, при отпускании, и, наконец, Click (щелчок), как глобальное действие. Если пользователь сделал двойной щелчок, тогда генерируется событие DblClick. Не все средства управления отвечают на события или генерируют события, нелепо говорить о событии Resize in a button (уменьшить размер клавиши), поскольку это событие обычно генерируется, когда мы изменяем размер окна (управление "Form").

В Gambas мы будем редактировать код процедуры для события (1), подобный этому:

```
PUBLIC SUB Control_Event
```

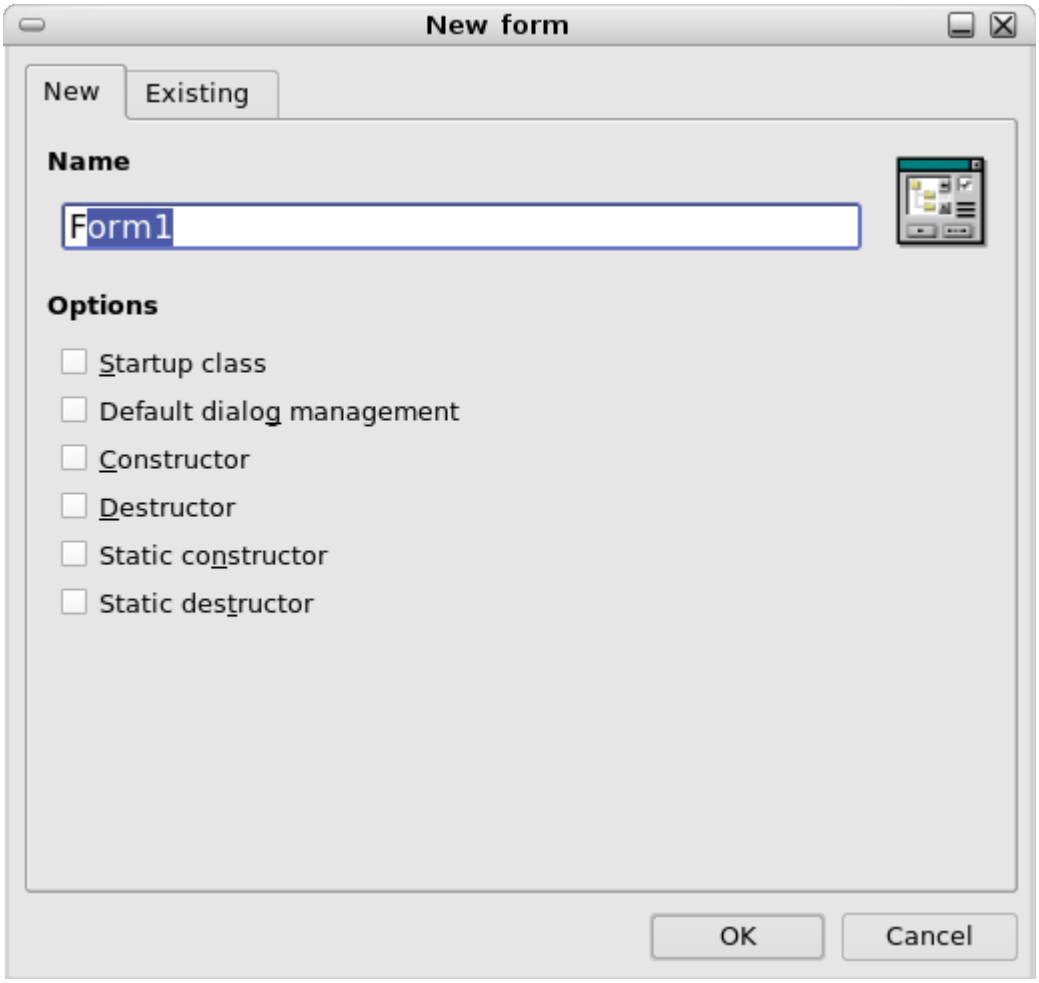
Здесь `Control` - это имя средства управления, где генерируется событие, а `Event` тип события. Некоторые средства управления имеют предопределенные события. Наиболее полезное предопределенное событие для клавиши, например, это `Click`.

В Gambas при двойном щелчке по любому средству управления редактор кода открывается и курсор позиционируется на объявлении подпрограммы для предопределенного события. Есть исключение, если управление ассоциировано с группой действия (свойство "Group" определено), редактор кода показывает подпрограмму для группы действия, как упоминалось выше.

Конструирование форм

Следует понимать при конструировании форм:

- Экран пользователя может отличаться от нашего: другое разрешение, оконный менеджер и/или размер шрифта. Не пытайтесь предельно использовать все пространство, этикетки. клавиши и другие средства управления могут обрезаться или быть неразборчивы.
- Хорошая практика оставлять основное окно программы "растягиваемым". В Gambas обратите внимание на свойство формы Border - рамка. Не устанавливайте его в Fixed - фиксировано.
- При создании новой формы есть несколько интересных опций:



Опции, относящиеся к конструктору и деструктуру (constructor и destructor), полезны для инициализации и завершения задачи в окне.

Генерируются следующие определения:

```
' Gambas class
file PUBLIC SUB _new()
END
PUBLIC SUB _free()
END
PUBLIC SUB Form_Open()
END
```

Если мы выберем "Static constructor - статический конструктор" и/или "Static destructor - статический деструктор", определения станут:

```
' 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
```

Используя эти процедуры, мы можем изменить процесс открывания закрывания окна. Мы можем инициализировать средства управления, переменные и т.д. В процедуре, объявленной как STATIC, процедура имеет доступ только к STATIC переменным.

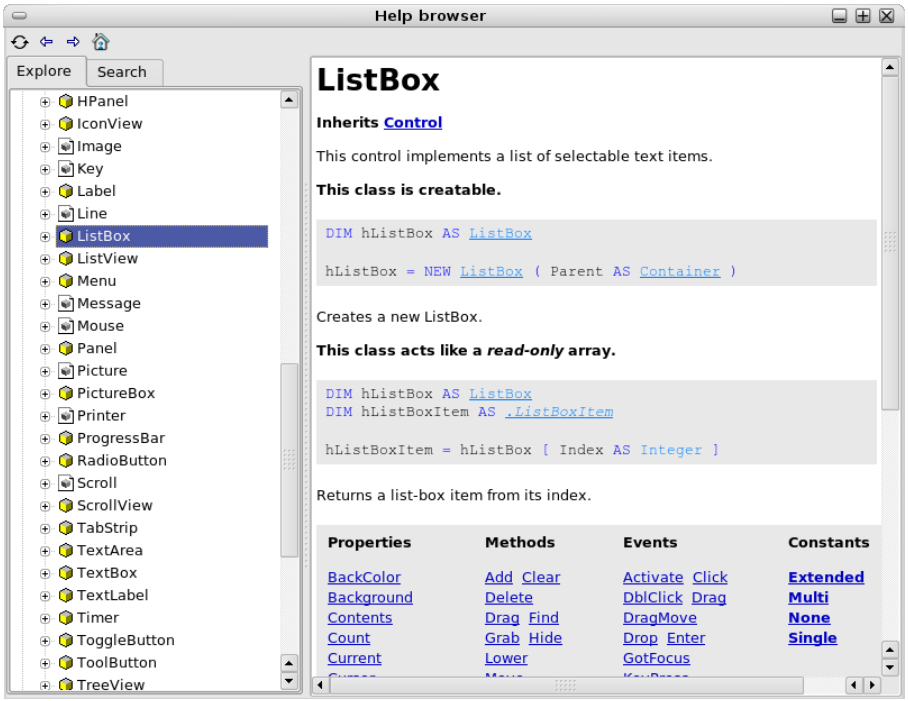
Прыжок в ...

Наша форма полностью разработана. Пришло время написать код для управления.

Первая задача - обеспечить некоторую функциональность управлению. Мы собираемся поработать с клавишами (и равнозначными пунктами меню) "Add - добавить", Modify - изменить", "Delete - удалить" и "Clean - очистить".

Действие "Clean"

Эта клавиша должна удалять все записи в ListBox. Для выполнения этой задачи мы поищем в системе помощи (help system) документацию, относящуюся к ListBox:



Эта документация в "дереве" под пунктом `gb.qt`, компонент Gambas, который включает все "видимые" средства управления (клавиши, этикетки, меню и т.д.). Прочитаем, что ListBox предоставляет метод "`Clean`", который очищает все вводы. Это все, что мы хотели, и мы с удовольствием воспользуемся этим методом.

Двойной щелчок по клавише "Очистить" (или пункту меню "Очистить"), открывается редактор кода и курсор позиционируется на соответствующей процедуре. Мы запишем следующий код:

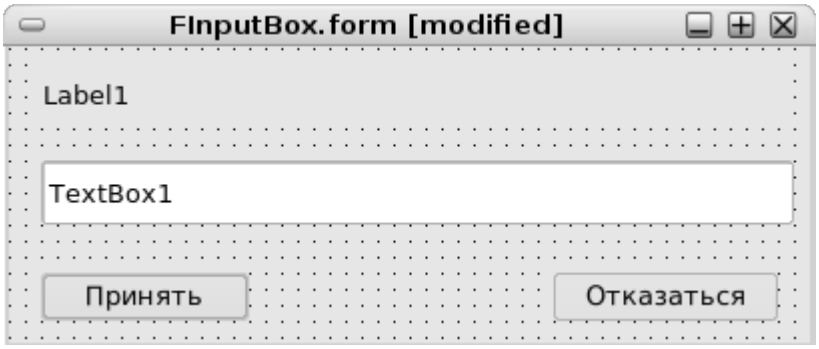
```
PUBLIC SUB Clean_Click()
ListBox1.Clean()
END
```

Очень легко :−)

Действие "Add"

Это действие более сложное. Пользователи будут добавлять ввод (строку текста) в ListBox, используя эту клавишу.

Gambas не предоставляет диалог тип "InputBox", так что мы собираемся создать нечто в этом роде. Мы создадим новую `Form`, но теперь мы зададим, что форма имеет конструктор. Почему? Потому, что в создаваемом образце мы хотим задать некоторые свойства формы, такие как заголовок, показ сообщений и определенное значение в поле текстового ввода. Вот предлагаемая конструкция:



Форма очень проста. Она имеет `Label`, текстовый ввод (`TextBox`) и две клавиши, "Accept - принять" and "Cancel - отказаться". Разумный диалог предполагает наличие удобной для пользователя возможности отказаться с помощью клавиши `Escape` и принять с помощью клавиши `Enter`:

> `Button` управление имеет свойства, названные "*Default*" и "*Cancel*". Мы установим "Default" в `True` для клавиши "Принять" и "Cancel" свойство в `True` для клавиши "Отказаться".

Применение этих свойств приведет к тому, что когда пользователь нажмет клавишу <ENTER>, форма поведет себя также, как если бы пользователь щелкнул по клавише "Принять", а нажатие на <ESC> будет эквивалентно нажатию на клавишу "Отказаться" button.

Следующая проблема, как вернуть написанный текст в диалоге в основное окно. Мы должны помнить, что в Gambas нет глобальных переменных, так что нам следует поискать другое решение. В "Tip of the day" #7, (пункт меню "? > Tips of the day - подсказки дня") есть предложение использовать переменную, объявленную как PUBLIC, так что эта переменная видима из любой точки или

класса внутри программы,

Создадим новый модуль (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
```

Очень просто и легко. Теперь мы имеем переменную, которая может быть доступна из любой точки внутри программы при использовании следующей нотации: `MCommon.my_text`

Теперь мы запишем код для нашего диалога "InputBox":

```
' Gambas class file
PUBLIC SUB _new(title AS String, message AS String, OPTIONAL text
AS String)
      ME.Caption = title
      Label1.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
      MCommon.my_text = TextBox1.Text
      ME.Close(0)
END
PUBLIC SUB Button2_Click() ' This is the "Cancel" button
      ME.Close(0)
END
```

Процедура `_new` - это конструктор. Используя его, мы можем устанавливать разные заголовки, этикетки и содержание текстовых вводов каждый раз, когда используется диалог. Более того, эти свойства устанавливаются в момент создания.

Клавиша "Принять" копирует текст в `TextBox` в переменную `my_text`, определенную в модуле `MCommon` и закрывает диалог. Клавиша "Отказаться" просто закрывает диалог.

Поскольку переменная `MCommon.my_text` в общем пользовании, мы должны помнить о необходимости "clear - очистить" каждый раз при ее использовании. Мы увидим это сейчас.

Процедура для клавиши "Добавить - Add" в основной форме снабжена следующим кодом. Он хорошо прокомментирован:

```
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
```

Действие "Modify"

Применяя эту клавишу пользователь может изменить ввод в ListBox. Если записей нет, клавиша ничего не делает, а, если пользователь не выбрал запись, он будет предупрежден об этом. Вот реализация для этого действия:

```
' "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
```

Действие "Delete"

Как мы видели выше, ListBox должен содержать хотя бы одну строку, а пользователь должен выделить одну строку.

Код очень похож на тот, что для действия "Modify - изменить":

```
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
```

Можно видеть, что реализация для этих четырех действий общая для клавиш и их эквивалентов в меню.

Теперь мы реализуем действие, относящееся к управлению файлами (Open - открыть, Save - сохранить) и закрытию программы. Мы начнем с того, что попроще.

Действие "Exit"

Функция для этой клавиши (и ассоциированного пункта меню) - закрыть программу. Очень просто:

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

```
      FInputBox
END
```

Мы можем проявить больше дружелюбия к пользователю, добавив диалог наподобие: "Вы собираетесь выйти из программы. Вы уверены?", - и создав подходящее задание. Но, давайте, оставим эту реализацию в качестве домашнего задания для читателей.

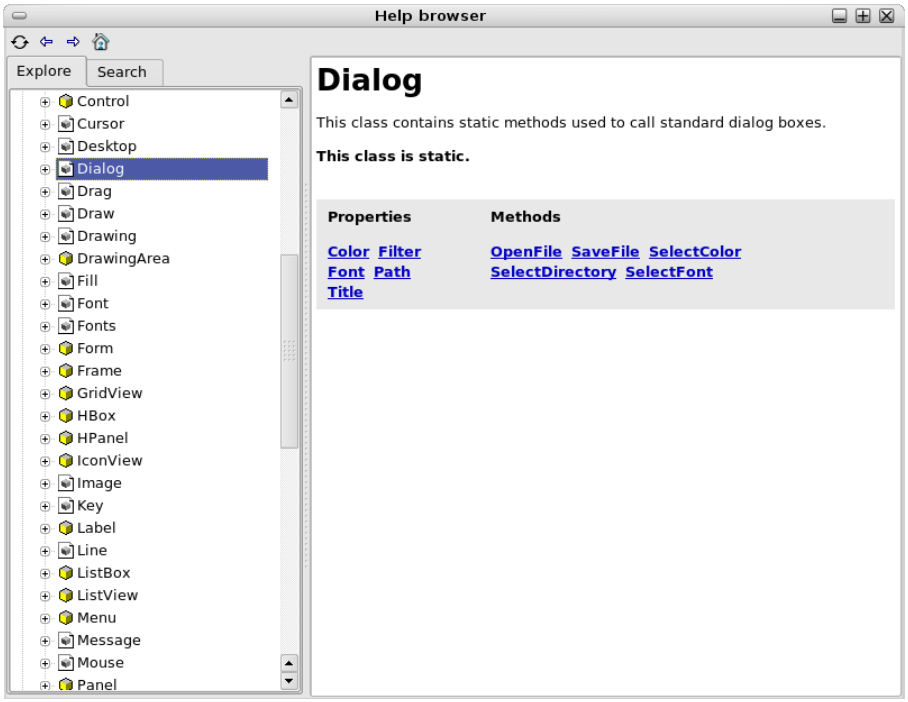
Действие "Open"

Что это действие должно сделать? Выяснить у пользователя путь к файлу, прочитать этот файл и загрузить данные в ListBox. Вот соответствующий код:

```
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
```

Эта часть кода представляет интересную особенность языка Gambas, "non instanceable - не запрашиваемые" или статические классы (2). Мы не можем создавать объекты этих классов, но мы можем прямо использовать их. В этом коде мы можем видеть два из этих классов в действии: класс "File" и класс "Dialog".

Например, класс Dialog предоставляет доступ к типичному стандартному диалогу для выбора файлов, системных цветов и т.д. Это документировано в теме "help" gb.qt

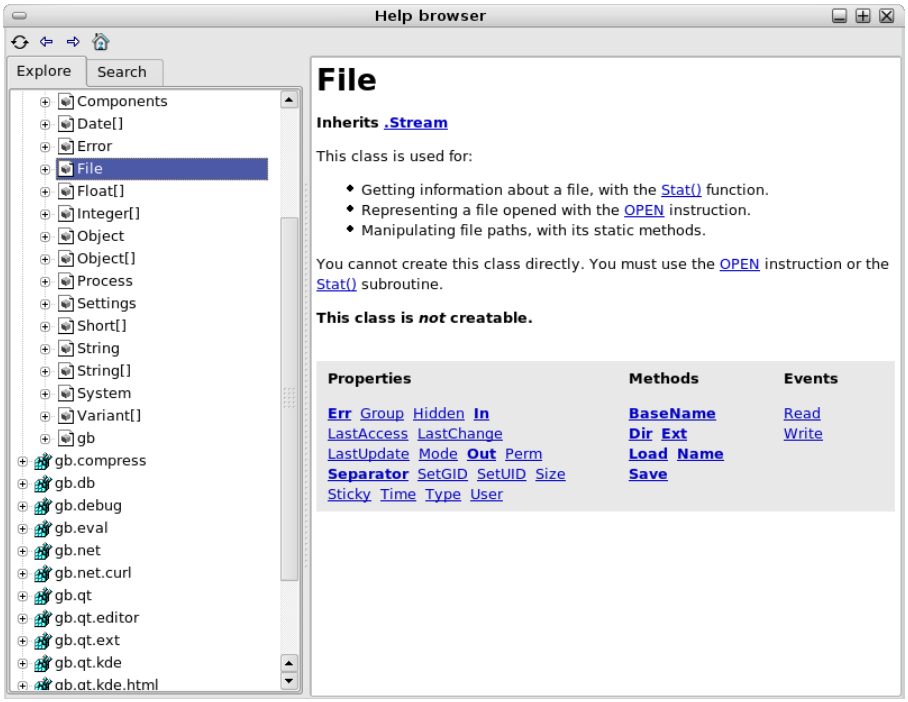


В нашей программе мы хотим выбрать файл и загрузить его содержимое в ListBox. Мы будем использовать класс `Dialog` следующим образом:

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

Мы задали заголовок диалога, предоставили фильтр для выбора типа файла по расширению (*.data) и, наконец, мы вызвали метод класса `OpenFile()`. Есть особенность в класее `Dialog`: если пользователь НЕ выбрал файл (то есть, пользователь нажал клавишу отказа (cancel) или клавишу ESC) возвращаемое значение в методе `OpenFile()` становится `True`. Когда же пользователь выбирает файл, мы можем получить доступ к полному пути через свойство `Dialog.Path`

Класс `File` (его документация под пунктом `gb` в системе помощи) предоставляет несколько методов, позволяющих нам работать с файлами:



В документации к Gambas в разделе, озаглавленном "How do I ..." есть несколько примеров кода для чтения и записи файлов. Мы будем использовать метод "Load()", чьими аргументами являются путь к файлу и возвращаемая String,которая содержит все данные в этом файле. Мы можем разделить строки возвращаемых данных, используя функцию Split(). Ее аргументы - это строки, которые следует разделить и разделитель (символ новой строки в данном случае \n) и возвращаемый Array of Strings. Из этих соображений мы должны объявить переменную arr_strings as String[]:

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

Когда мы получим все строки данных, содержащиеся в файле, мы должны очистить ListBox и добавить каждую строку, используя метод Add() ListBox.

Действие "Save"

Когда пользователь нажимает клавишу "Save - сохранить", программа должна вывести содержимое ListBox в текстовый файл. Мы покажем диалог "Save File", запрашивая у пользователя имя файла для сохранения данных. Вот код для этого действия:

```
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
```

Мы хотим сохранить данные в файле с расширением .data, так что, если имя файла предоставленное пользователем не заканчивается с ".data", мы будем дописывать расширение. Мы используем метод Save(), который предоставляется классом File.

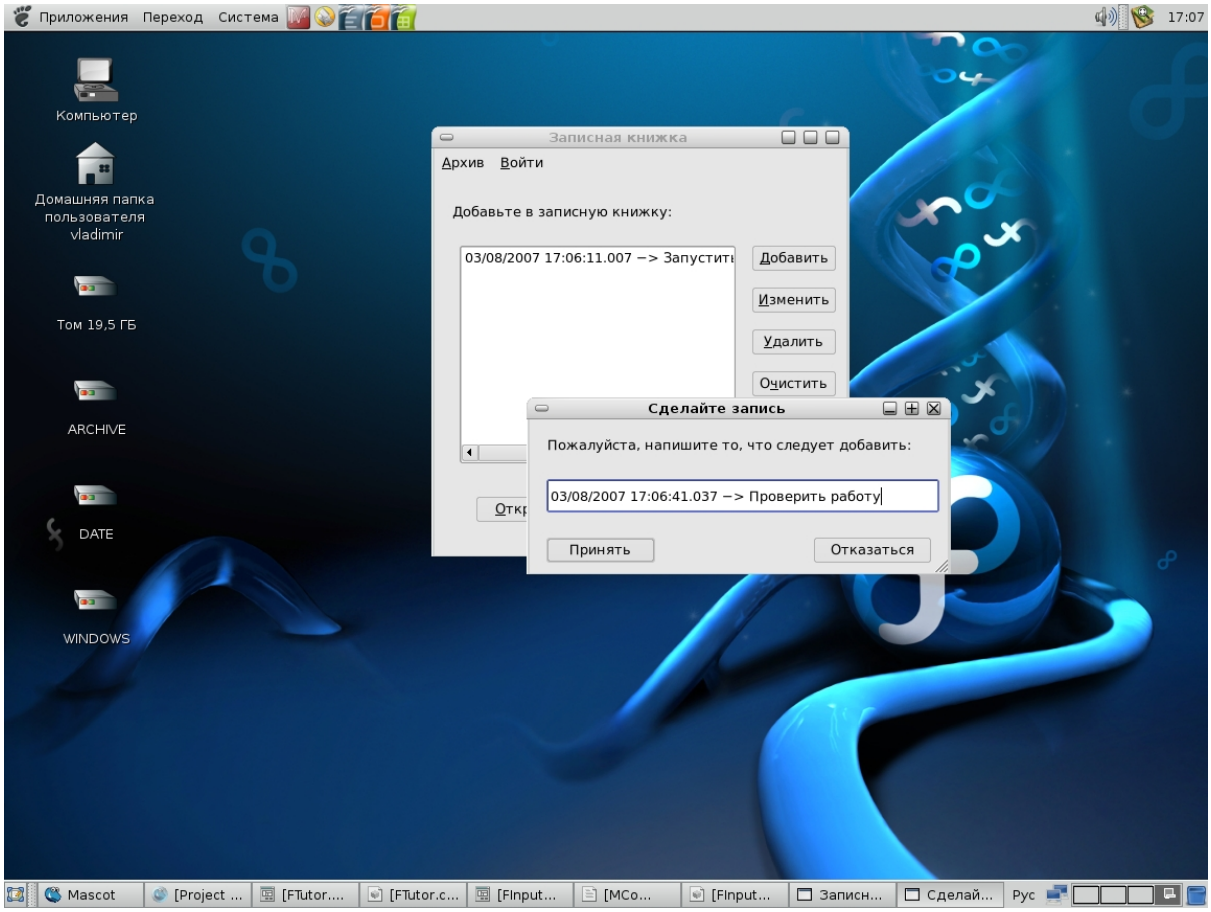
Аргументы этого метода - путь к файлу и текст, который мы хотим сохранить. Мы получаем доступ к содержимому ListBox, используя свойство Contents, которое возвращает String, с "new line" (\n) разделителем каждого ввода в ListBox.

Окончательная подгонка

Пользователю интересно увидеть весь текст записи, а listbox может оказаться не достаточно широким, чтобы отобразить его, если запись очень длинная. Мы реализуем эту возможность следующим образом: когда пользователь дважды щелкает клавишей мышки по вводу, мы покажем весь текст в окне диалога:

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

Наша программа запущена



Развертывание завершения программы

OK. Наша программа готова. Мы можем проверить ее в любое время внутри IDE, нажав на клавишу F5.

Теперь мы хотим использовать программу, как обычное приложение, то есть, без IDE. Для выполнения этого есть опция основного меню: "Project > Create executable". Этим создается "монолитный" исполняемый файл, включающий все ресурсы проекта. Этот файл не будет машинным кодом, но "байткодом", выполняемым интерпретатором Gambas - gbx. Это предполагает, что мы нуждаемся, чтобы этот интерпретатор был установлен в системе для запуска программ, написанных в Gambas. (Это похоже на другие языки программирования. Например, нам нужно установить Java для выполнения программ, написанных на Java).

Все дистрибутивы, в которые включен Gambas, компонеты Gambas разделены и есть пакет "Gambas runtime", который включает интерпретатор, но не полный IDE.

Мы можем создать RPM или DEB архив нашей программы. Эти пакеты будут иметь интерпретатор Gambas (`gambas runtime` программа) по зависимости. Есть мастер для помощи в генерации такого пакета. Он очень прост и интуитивно понятен. Вы можете найти его в меню "Project > Create installation package ...".

Окончание

Мы убедились, что это очень легко создавать простые, но функциональные приложения в Gambas. Этот инструмент предлагает некоторые средства управления и предопределенные классы. Есть также множество расширений или компонентов, доступных в порядке создания клиент/серверных приложений, доступа к базам данных, приложений мультимедиа и т.д.

IMHO, Я думаю, что Gambas очень многообещающий инструмент. К счастью, Gambas' mailing−list разработчиков очень активен, а ошибки устраняются очень быстро.

Спасибо, Benoit (et col.)! Хорошая работа!

Об этом документе и его авторе

Как говорилось выше, мы работали с версией 1.0−1 Gambas в этом руководстве (предварительно скомпилированный пакет для Debian Sid). Во время написания этого документа была выпущена версия 1.0.3, и весьма вероятно, что в момент прочтения этого, появится новая версия. Весьма благоразумно почитать журнал изменений, чтобы избежать возможных расхождений от версии к версии.

Все комментарии, предложения, коррективы и улучшения этого документа приветствуются.

Мой mail - forodejazz (arroba) gmail (punto) com

Примечания

1. События нуждаются в поддержке процедуры или подпрограммы: функция, которая не возвращает какого-либо значения.
2. Я не эксперт в технологии объектно-ориентированного программирования. Мои извинения, если я использовал некорректные термины ;−)