Создание визуальных компонент для Lazarus

30.05.2005
Сергей Смирнов

Компоненты или элементы управления являются именно теми "кирпичиками", из которых и позволяют создавать программы интегрированные средства быстрой разработки. Lazarus также придерживается этой концепции. Однако, в отличие от VB и подобно Delphi, Lazarus абсолютно самодостаточен в вопросе создания программистами своих собственных компонент.

Постановка задачи

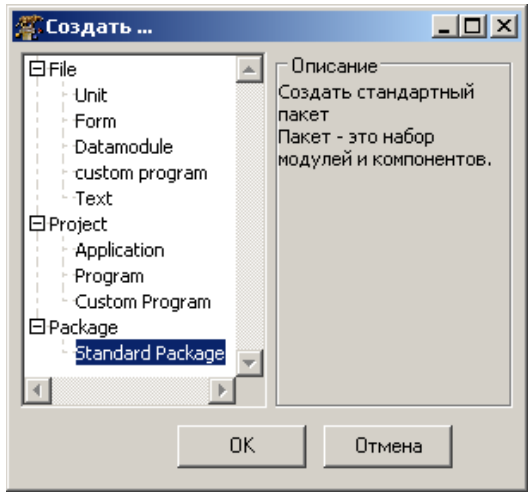
В действительности, создать свой собственный компонент очень легко. Настолько легко, что не имеет смысла описывать этот процесс на простом примере. Поэтому, поставим себе некоторую нетривиальную задачу. Вот, к примеру, имеется стандартный элемент управления TStatusBar, который представляет собой относительно несложную панель состояния, которая располагается вдоль нижней границы окна. Его можно разбить на несколько панелей, в каждой из которых выводить какие-либо сообщения. К сожалению, ничего кроме текста эти панели отображать не умеют. Что там можно ещё отображать? Например, неплохо было бы разместить ProgressBar, чтобы не просто сообщить пользователю о начале какого либо продолжительного процесса, а постоянно держать его в курсе текущего состояния. Вот этим и займёмся.

Слишком уж мудрить не будем. В качестве предка возьмём TStatusBar и предусмотрим всего пару дополнительных свойств: одно для непосредственного доступа к встроенному ProgressBar, чтобы можно было им управлять, а другое - для указания, в какой из панелей строки состояния будет отображаться ProgressBar.

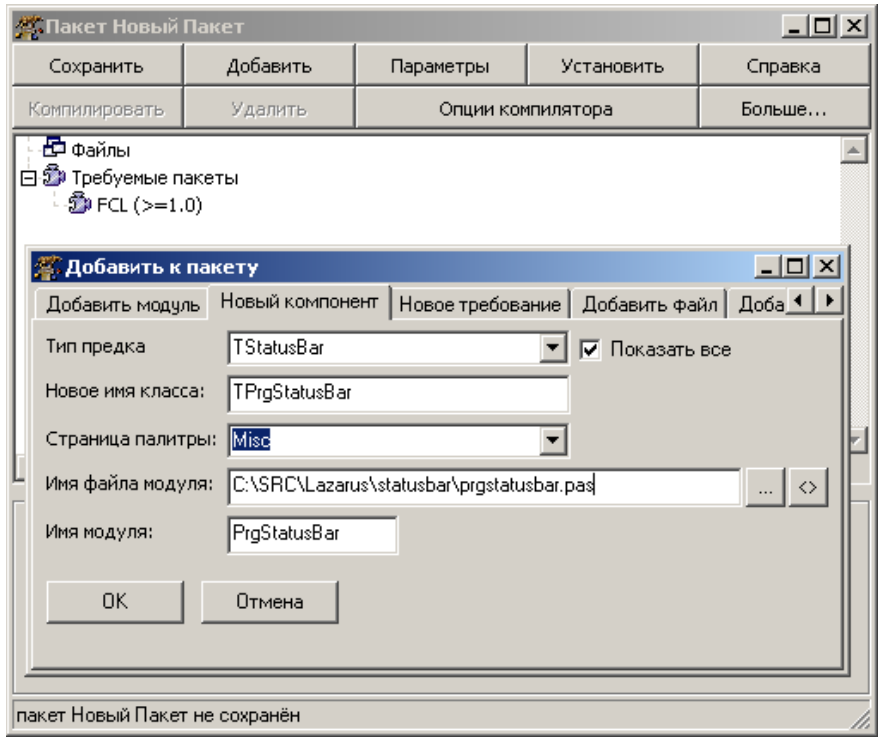
Скачать архив с уже готовым комплектом файлов компонента можно **здесь**.

Создание пакета

Так как для добавления компонентов в палитру компонент Lazarus используются пакеты, сначала создадим новый пакет. Для этого в меню "Файл" выберем пункт "Создать". В левой части появившегося окна выберем "Standard Package" и нажмём кнопку "OK", как показано на рисунке:



На экране возникнет окно управления пакетом. Сохраним наш новый и пока пустой пакет в отдельном каталоге под именем, например, prgstatusbarlaz.lpk. Теперь добавим новый компонент. Вообще говоря, один пакет может содержать сколько угодно компонентов, но нам пока достаточно одного. Нажмите кнопку "Добавить" и в появившейся форме перейдите на вкладку "Новый компонент". Заполните все поля примерно так, как показано на рисунке:



Нажмите кнопку "OK" и новый файл с заготовкой компонента будет добавлен в пакет. Также, поскольку мы создаём визуальный компонент, будет добавлена зависимость от пакета LCL. Чтобы приступить непосредственно к разработке компонента, двойным щелчком по имени модуля откройте его в редакторе исходного кода. То, что Вы увидите будет являться результатом автоматической генерации кода на основе сделанных настроек и уже может быть установлено как пользовательский элемент управления, который пока просто дублирует своего родителя.

Разработка компонента

Для начала, добавим необходимые свойства. Для этого в секции published напечатаем следующее:

```
    property ProgressBar: TProgressBar read;
    property ProgressBarPanel: Integer;
```

Разместим курсор на имени одного из свойств и нажмём Ctrl-Shift-C. Сработает автогенерация кода, в результате которой будут

добавлены необходимые внутренние переменные и методы (включая и объявление и реализацию). Теперь в раздел Public нашего класса добавим объявления конструктора и деструктора:

```
    constructor Create(TheOwner: TComponent); override;
    destructor Destroy; override;
```

Снова нажмём Ctrl-Shift-C и получим заготовки реализации этих методов. Наконец, можно начать программировать жизнедеятельность нашего компонента. Сначала определим действия, необходимые при создании компонента. В процедуре Create, ниже строки inherited Create(TheOwner), которая создаёт родительский компонент будем добавлять строки:

```
  FProgressBarPanel := -1;
```

Это означает вот что. Для указания, в какой из панелей будет отображаться наш ProgressBar мы будем просто задавать номер панели (нумерация начинается с 0), а чтобы вовсе его не показывать будем использовать -1. Именно с этого значения и начнём, так как при первоначальном размещении компонента панели ещё не созданы. Далее создадим сам ProgressBar и укажем, что он является встроенным компонентом. Это позволит сохранять его свойства вместе со свойствами основного компонента в файле определения формы.

```
  FProgressBar := TProgressBar.Create(Self);
  Include(FProgressBar.ComponentStyle, csSubComponent);
```

В следующей строке мы назначаем наш компонент родительским по отношению к ProgressBar, чтобы последний отрисовывался внутри него. Затем обнуляем все размеры и координаты, чтобы непосредственно при создании компонента ProgressBar не отображался.

```
  FProgressBar.Parent := self;
  FProgressBar.Top := 0;
  FProgressBar.Height := 0;
  FProgressBar.Width := 0;
  FProgressBar.Left := 0;
```

И в заключение, добавим такую строку:

```
  FIsLoaded := False;
```

Это очень важный момент, к которому мы вернёмся чуть позже, а пока просто добавим объявление закрытого поля FIsLoaded типа Boolean к автоматически сгенерированным полям нашего класса.

Теперь перейдём к процедуре SetProgressBarPanel установки значения свойства ProgressBarPanel. Сгенерированного автоматически кода явно недостаточно, потому, что простое на первый взгляд присвоение значения свойству на самом деле связано со значительным количеством проверок и других действий. Итак, по-порядку.

```
  if FProgressBarPanel=AValue then Exit;
  if not FIsLoaded then // Это происходит загрузка свойств из потока
  begin
    FProgressBarPanelTemp := AValue; // Сохраним свойство, чтобы потом присвоить его в Loaded.
    Exit; // Следующая проверка не имеет смысла, пока панели не загружены из потока.
  end;
  if (AValue >= self.Panels.Count) then Exit;
```

Сразу после автоматически добавленной проверки на присваивание одного и того же значения идёт проверка того самого поля, к которому я обещал вернуться. Дело тут вот в чём. Обратите внимание, что в последней строке проверяется, не пытаемся ли мы отобразить наш ProgressBar в несуществующей панели. На первый взгляд, это вполне логично, но в реальности происходит следующее: сначала при создании компонента срабатывает метод Create, а потом начинается загрузка свойств компонента из потока. И нет никакой гарантии, что панели в нашей строке состояния будут созданы до присвоения сохранённого значения свойству ProgressBarPanel. При этом Panels.Count будет равен 0 и мы никогда не сможем установить значение таким, каким оно сохранено в файле формы.

Единственный способ выйти из этой неприятной ситуации - временно сохранить значение, которое мы собираемся установить. Только когда все свойства компонента будут загружены, а панели созданы, мы сможем присвоить значение свойству ProgressBarPanel. Для этого нам надо переопределить метод Loaded. Наберём в секции Protected декларацию

```
  procedure Loaded; override;
```

и выполним завершение кода. После этого добавим в автоматически созданную процедуру пару строк так, что получится следующее:

```
procedure TPrgStatusBar.Loaded;
begin
  inherited Loaded;
  FIsLoaded := True;
  SetProgressBarPanel(FProgressBarPanelTemp);
end;
```

Как видите, здесь мы просто устанавливаем поле-флаг FIsLoaded и повторно вызываем метод для установки значения свойства, используя предварительно запомненное в поле FProgressBarPanelTemp значение. Кстати, Вы не забыли добавить его декларацию в раздел Private?

Теперь, когда учтены все нюансы поведения программы, закончим работу с методом SetProgressBarPanel, добавив довольно бесхитростный код, который будет размещать ProgressBar внутри выбранной панели. В итоге должно получиться следующее:

```
procedure TPrgStatusBar.SetProgressBarPanel(const AValue: Integer);
var
  i: Integer;
  L: Longint;
```

```
begin
  if FProgressBarPanel=AValue then Exit;
  if not FIsLoaded then // Это происходит загрузка свойств из потока
  begin
    FProgressBarPanelTemp := AValue; // Сохраним свойство, чтобы потом присвоить его в Loaded.
    Exit; // Следующая проверка не имеет смысла, пока панели не загружены из потока.
  end;
  if (AValue >= self.Panels.Count) then Exit;
  if AValue = -1 then
  begin
    FProgressBar.Top := 0;
    FProgressBar.Height := 0;
    FProgressBar.Width := 0;
    FProgressBar.Left := 0;
    FProgressBarPanel := -1;
    Exit;
  end;
  FProgressBar.Top := 2;
  FProgressBar.Height := self.Height-2;
  L := 0;
  for i := 1 to AValue do
    L := L + self.Panels[i-1].Width;
  if AValue = 0 then
  begin
    FProgressBar.Left := 0;
        FProgressBar.Width := self.Panels[AValue].Width;
  end
  else
  begin
    FProgressBar.Left := L + 2;
        FProgressBar.Width := self.Panels[AValue].Width - 2;
  end;
  FProgressBarPanel:=AValue;
end;
```

Завершим разработку компонента, определив код деструктора. Единственное, что может понадобиться сделать - это уничтожить созданный нами ProgressBar. Вообще говоря, поскольку при его создании мы назначили владельцем наш компонент, то при разрушении компонента ProgressBar должен уничтожиться автоматически. Но всё-же подстрахуемся:

```
destructor TPrgStatusBar.Destroy;
begin
  FProgressBar.Free;
```

```
  FProgressBar := nil;
  inherited Destroy;
end;
```

И в заключение - два момента. Во-первых, определим значение по умолчанию для свойства ProgressBarPanel. Это совсем не тоже самое, что присвоить значение по умолчанию в конструкторе. Значение по умолчанию в объявлении свойства позволяет Lazarus не сохранять в файле значения свойств, если они равны умолчательным, что в итоге уменьшает размер программы. Обратите внимание, что значение по умолчанию не соответствует той -1, которую мы присвоили полю свойства в конструкторе. Значение 1 кажется более логичным.

```
    property ProgressBarPanel: Integer read FProgressBarPanel write SetProgressBarPanel default 1;
```

Во-вторых, давайте ненадолго остановимся на процедуре регистрации компонента в палитре Lazarus. Как видите, в нашем простом случае она была сгенерирована автоматически и состоит всего из одного вызова процедуры RegisterComponents. В качестве первого параметра указывается имя вкладки, в которой будет размещён компонент. Если вкладки с таким именем не существует, она будет создана для нас. Второй параметр - массив компонентов, которые мы хотим разместить в этой вкладке. В нашем случае - всего один компонент.

```
procedure Register;
begin
  RegisterComponents('Misc',[TPrgStatusBar]);
end;
```

Скачать архив со всеми необходимыми файлами можно **здесь**.

Пиктограммы

Чтобы отличать компоненты друг от друга в палитре компонент Lazarus, разработчики стараются сделать их значки как можно более уникальными. Кроме того, с их помощью стараются показать назначение компонента. Давайте и мы сделаем иконку для нашего нового компонента. Правила очень простые:

1. Надо создать изображение размером максимум 23x23 пиксела и сохранить его в формате XPM. Имя файла должно совпадать с именем класса компонента (регистр не имеет значения).
2. Все изготовленные файлы с рисунками (в нашем случае всего один) нужно скомпилировать в ресурсный файл с помощью утилиты lazres, которая находится в каталоге tools (её саму сначала надо собрать, запустив make в этом каталоге).

   ```
   C:\lazarus\tools\lazres.exe prgstatusbarlaz.lrs tprgstatusbar.xpm
   ```

3. Теперь надо подключить файл ресурсов в секции инициализации модуля компонента:

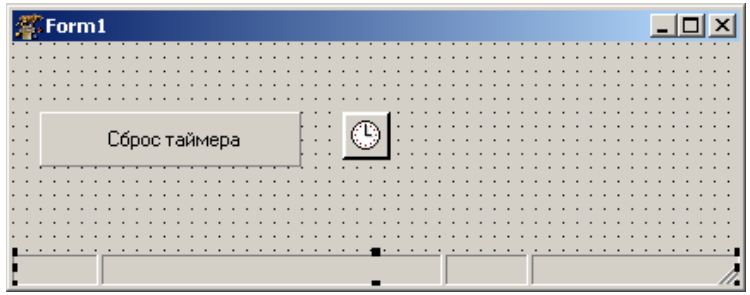
   ```
   initialization
   ```

```
{$I prgstatusbarlaz.lrs}
```

Вот и всё. Включать какие либо из этих файлов в состав пакета не нужно. Теперь при установке пакета соответствующие значки появятся в палитре компонент Lazarus.

Тестовая программа

Для проверки работы нашего нового компонента установим пакет и создадим простейшую программу тестирования. В главной форме разместим нашу модернизированную строку состояния, в которой определим 4 панели, кнопку и таймер. Примерно так, как показано на рисунке:




```
unit Unit1;

{$mode objfpc}
{$H+}

interface

uses
  Classes, SysUtils, LResources, Forms, Controls, Graphics, Dialogs, ComCtrls,
  Buttons, ExtCtrls, PrgStatusBar;

type

  { TForm1 }

  TForm1 = class(TForm)
    PrgStatusBar1: TPrgStatusBar;
    SpeedButton1: TSpeedButton;
    Timer1: TTimer;
    procedure SpeedButton1Click(Sender: TObject);
```

```
    procedure Timer1Timer(Sender: TObject);
  private
    { private declarations }
  public
    { public declarations }
  end;

var
  Form1: TForm1;

implementation

{ TForm1 }

procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
  PrgStatusBar1.ProgressBar.Position := 0;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  PrgStatusBar1.ProgressBar.StepIt;
end;

initialization
  {$I unit1.lrs}

end.
```

В работе это выглядит вот так: