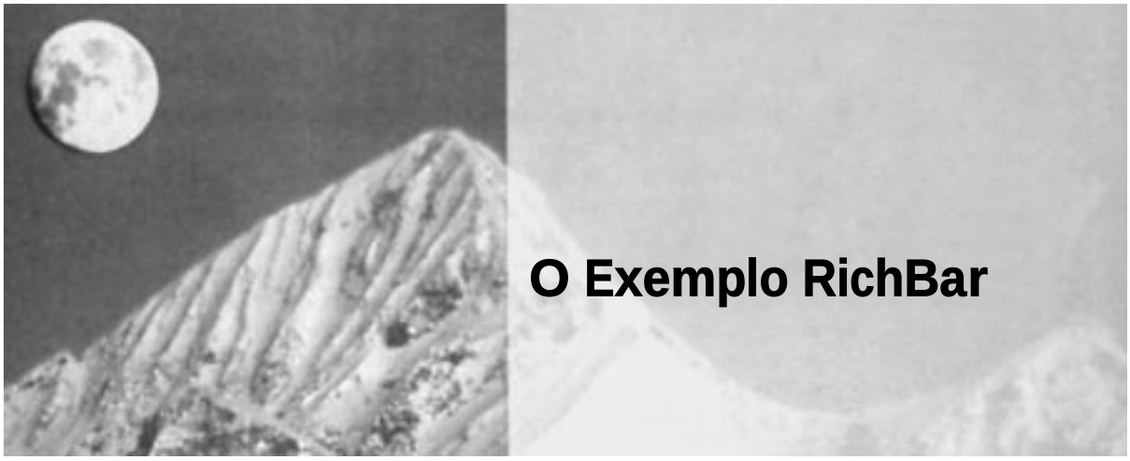

O Exemplo RichBar

Este capítulo de bônus é fornecido com o livro *Mastering Delphi 6*. Trata-se de uma introdução aos recursos básicos do exemplo RichBar, discutido no Capítulo 7 do livro e do exemplo MdEdit acompanhante, discutido no Capítulo 8.

Este documento explica como você cria um editor simples baseado no controle RichEdit, usando o Delphi 6. O programa tem uma barra de ferramentas e implementa vários recursos, incluindo um esquema completo para abrir e salvar os arquivos de texto, discutido neste documento. Na verdade, queremos solicitar ao usuário para que salve qualquer arquivo modificado, antes de abrir um novo, para evitar a perda de quaisquer alterações. Parece um aplicativo profissional, não?

Operações de Arquivo

A parte mais complexa desse programa é a implementação dos comandos do menu suspenso File: New, Open, Save e Save As (Arquivo: Novo, Abrir e Salvar e Salvar como). Em cada caso, precisamos verificar se o arquivo corrente mudou, salvando o arquivo apenas se tiver mudado. Devemos solicitar ao usuário para que salve o arquivo sempre que o programa criar um novo, carregar um já existente ou terminar.

Para fazer isso, adicoinamos um campo, uma propriedade e três métodos públicos na classe que descreve o formulário do aplicativo:

```
private
  FModified: Boolean;
  FileName: string;
  procedure SetModified(const Value: Boolean);
  property Modified: Boolean read FModified write SetModified;
public
  function SaveChanges: Boolean;
  function Save: Boolean;
  function SaveAs: Boolean;
```

A string `FileName` e a propriedade `Modified` são configuradas quando o formulário é criado, executando o código usado para definir um novo arquivo. Esses valores são alterados posteriormente, quando um novo arquivo é carregado ou quando o usuário renomeia um arquivo com o comando Save As. Aqui está o código de inicialização:

```
procedure TFormRichNote.FormCreate(Sender: TObject);
begin
```

```
  Application.Title := Caption;
  NewExecute (Self);
end;
```

O valor do flag muda assim que você digita novos caracteres no controle RichEdit (em seu manipulador de evento OnChange):

```
procedure TFormRichNote.RichEdit1Change(Sender: TObject);
begin
  // permite operações de salvamento
  Modified := True;
end;
```

Quando um novo arquivo é criado, o programa verifica se o texto foi modificado. Se assim for, ele chama a função SaveChanges, que pede para que o usuário salve as alterações, descarte-as ou pule a operação corrente:

```
procedure TFormRichNote.New1Click(Sender: TObject);
begin
  if not Modified or SaveChanges then
  begin
    RichEdit1.Text := '';
    Modified := False;
    FileName := '';
    Caption := 'RichNote - [Untitled]';
  end;
end;
```

Se a criação de um novo arquivo for confirmada, algumas operações simples ocorrerão, incluindo o uso de '*Untitled*' em vez do nome do arquivo no título do formulário.

Avaliação de Curto-Circuito

A expressão `if not Modified or SaveChanges then` exige alguma explicação. Por padrão, o Pascal realiza o que é chamada de "avaliação de curto-circuito" de expressões condicionais complexas. A idéia é simples: se a expressão `not Modified` for verdadeira, teremos certeza de que a expressão inteira vai ser verdadeira e não precisaresmo avaliar a segunda expressão. Nesse caso em particular, a segunda expressão é uma chamada de função e a função é chamada apenas se `Modified` for verdadeira. Esse comportamento de expressões `or` e `and` pode ser mudado configurando-se uma opção de compilação do Delphi chamada Complete boolean eval. Você pode encontrá-la na página Compiler da caixa de diálogo Project Options.

A caixa de mensagem apresentada pela função `SaveChanges` tem três opções. Se o usuário selecionar o botão Cancel, a função retornará um valor falso. Se o usuário selecionar No, nada acontecerá (o arquivo não será salvo) e a função retornará um valor verdadeiro para indicar que, embora não tenhamos salvo o arquivo, a operação solicitada (como a criação de um novo arquivo) pode ser executada. Se o usuário selecionar Yes, o arquivo será salvo e a função retornará um valor verdadeiro.

No código dessa função, observe em particular a chamada da função MessageDlg, usada como o valor de uma instrução case:

```
function TFormRichNote.SaveChanges: Boolean;
begin
  case MessageDlg ('The document ' + filename + ' has changed.' + #13#13 +
```

```
    'Do you want to save the changes?', mtConfirmation, mbYesNoCancel, 0) of
  idYes:
    // chama Save e retorna seu resultado
    Result := Save;
  idNo:
    // não salva e continua
    Result := True;
  else // idCancel:
    // não salva e aborta a operação
    Result := False;
  end;
end;
```

NOTA Na chamada de `MessageDlg` acima, adicionamos caracteres de nova linha explícitos (`#13`) para melhorar a legibilidade da saída. Como uma alternativa ao uso de uma constante caractere numérica, você pode chamar `Chr(13)`.

Para realmente salvar o arquivo, outra função é chamada: `Save`. Esse método salva o arquivo, se ele já tiver um nome de arquivo correto, ou pede para que o usuário introduza um nome, chamando a função `SaveAs`. Estas são duas funções mais internas, não conectadas diretamente com itens de menu:

```
function TFormRichNote.Save: Boolean;
begin
  if Filename = '' then
    Result := SaveAs // solicita um nome de arquivo
  else
  begin
    RichEdit1.Lines.SaveToFile (FileName);
    Modified := False;
    Result := True;
  end;
end;
function TFormRichNote.SaveAs: Boolean;
begin
  SaveDialog1.FileName := Filename;
  if SaveDialog1.Execute then
  begin
    Filename := SaveDialog1.FileName;
    Save;
    Caption := 'RichNote - ' + Filename;
    Result := True;
  end
  else
    Result := False;
end;
```

Usamos duas funções para executar as operações Save e Save As por completeza, mesmo que o programa RichBar tenha apenas um botão Save e não um botão Save As. A versão MdEdit, no Capítulo 8, oferece esse recurso extra. Além disso, o botão Save é ativado somente se o arquivo não tiver sido modificado, como indicado no método `SetModified`:

```
procedure TFormRichNote.SetModified(const Value: Boolean);
begin
  FModified := Value;
```

```
  tbtnSave.Enabled := Modified;
end;
```

Abrir um arquivo é muito mais simples. Antes de carregar um novo arquivo, o programa verifica se o arquivo corrente foi alterado, pedindo ao usuário para salvá-lo com a função SaveChanges, como antes. O método OpenExecute é baseado no componente OpenDialog, outra caixa de diálogo padrão fornecida pelo Windows e suportada pelo Delphi:

```
procedure TFormRichNote.OpenExecute(Sender: TObject);
begin
  if not Modified or SaveChanges then
    if OpenDialog1.Execute then
    begin
      Filename := OpenDialog1.FileName;
      RichEdit1.Lines.LoadFromFile (FileName);
      Modified := False;
      Caption := 'RichNote - ' + FileName;
    end;
end;
```

O único outro detalhe relacionado às operações de arquivo é que os componentes OpenDialog e SaveDialog do formulário têm um valor específico para suas propriedades `Filter` e `DefaultExt`, como você pode ver no trecho a seguir da descrição textual do formulário:

```
object OpenDialog1: TOpenDialog
  DefaultExt = 'rtf'
  FileEditStyle = fsEdit
  Filter = 'Rich Text File (*.rtf)|*.rtf|Any file (*.*)|*.*'
  Options = [ofHideReadOnly, ofPathMustExist,ofFileMustExist]
end
```

A string usada para a propriedade `Filter` contém quatro pares de substrings, separados pelo símbolo |. Cada par tem uma descrição do tipo de arquivo que aparecerá na caixa de diálogo Abrir arquivo ou Salvar arquivo e o filtro a ser aplicado nos arquivos do diretório, como `*.RTF`. Para configurar os filtros no Delphi, você pode simplesmente chamar o editor dessa propriedade, o qual apresenta uma lista com duas colunas.

Os métodos relacionados a arquivo anteriores também são chamados a partir do método `FormCloseQuery` (o manipulador do evento `OnCloseQuery`), que é chamado sempre que o usuário tenta fechar o formulário, terminando o programa. Podemos fazer isso acontecer de várias maneiras: dando um clique duplo no ícone do menu de sistema, selecionando o comando `Close` do menu de sistema, pressionando Alt+F4 ou chamando o método Close no código, como no comando de menu File | Exit.

Em `FormCloseQuery`, você pode decidir se vai realmente fechar o aplicativo, configurando o parâmetro `CanClose`, que é passado por referência. Novamente, se o arquivo corrente tiver sido modificado, podemos chamar a função `SaveChanges` e usar seu valor de retorno. Novamente, podemos usar a técnica de avaliação de curto-circuito:

```
procedure TFormRichNote.FormCloseQuery(Sender: TObject;
  var CanClose: Boolean);
begin
  CanClose := not Modified or SaveChanges;
end;
```

O último comando relacionado a arquivo é Print. O componente RichEdit inclui recursos de impressão e eles são muito simples de usar. Aqui está o código, que produz uma saída impressa realmente interessante:

```
procedure TFormRichNote.PrintExecute (Sender: TObject);
begin
  RichEdit1.Print (FileName);
end;
```

Conclusão

Conforme mencionado no início, o suporte a arquivo fornecido por esse exemplo é bastante complexo. Isso é algo que você provavelmente precisará manipular em qualquer aplicativo relacionado a arquivo. Como isso era longo demais para incluirmos no livro impresso, decidimos colocá-lo no CD, em vez de omiti-lo completamente. Agora, você pode voltar ao livro (principalmente aos Capítulos 7 e 8) para ver como o exemplo pode ser estendido de várias maneiras diferentes.