Borland DELPHI for Windows

Osmar de Oliveira Braz Junior - 1997.

Tubarão / Santa Catarina / BRAZIL
E-Mail : osmarjr@unisul.rct-sc.br
http://www.geocities.com/wallstreet/5693

Índice

1. O que é Delphi?

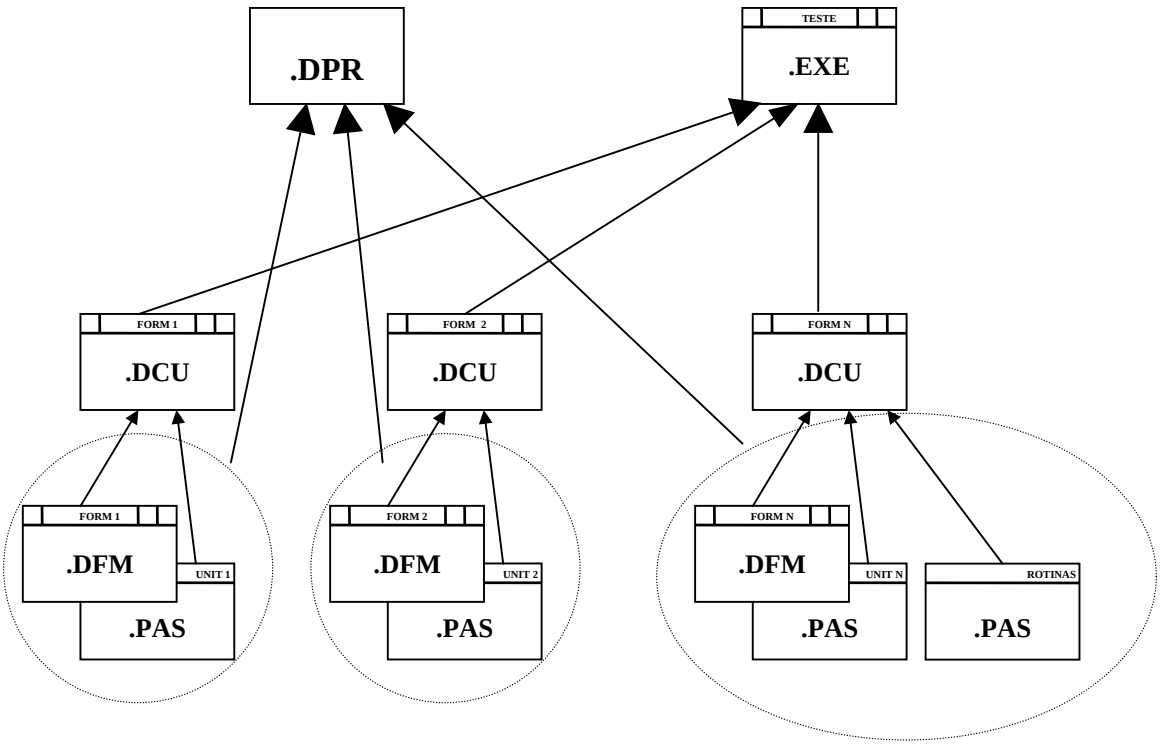
O Delphi é um ambiente de desenvolvimento de aplicações, orientado a objeto, que permite o desenvolvimento de aplicações para os Sistemas operacionais Windows 3.11, Windows 95 e Windows NT, com pouca codificação.

O Delphi tem ferramentas de desenvolvimento, como templates e experts de aplicações e formulários, que aumentam muito a produtividade, facilitando a programação da aplicação.

2. Como é formado uma Aplicação em Delphi

Quando você abre um projeto no Delphi, ele já mostra uma UNIT com várias linhas de código. Este texto tem como objetivo explicar um pouco desta estrutura que o ele usa. Um projeto Delphi tem, inicialmente, duas divisórias: uma UNIT, que é associada a um Form, e outra Project, que engloba todos os FORM e UNITs da aplicação.

Em Delphi temos: o Project, os Forms e as Units. Para todo Form temos pelo menos uma Unit (Código do Form), mas temos Units sem form (códigos de procedures, funções, etc).



2.1 Arquivos que Compõem um Aplicação

2.1.1 Arquivos Gerados no desenvolvimento

Extensão Arquivo	Definição	Função
.DPR	Arquivo do Projeto	Código fonte em Pascal do arquivo principal do projeto. Lista todos os formulários e units no projeto, e contém código de inicialização da aplicação. Criado quando o projeto é salvo.
.PAS	Código fonte da Unit(Object Pascal)	Um arquivo .PAS é gerado por cada formulário que o projeto contém. Seu projeto pode conter um ou mais arquivos .PAS associados com algum formulário. Contem todas as declarações e procedimentos incluindo eventos de um formulário.
.DFM	Arquivo gráfico do formulário	Arquivo binário que contém as propriedades do desenho de um formulário contido em um projeto. Um .DFM é gerado em companhia de um arquivo .PAS para cada formulário do projeto.
.OPT	Arquivo de opções do projeto	Arquivo texto que contém a situação corrente das opções do projeto. Gerado com o primeiro salvamento e atualizado em subsequentes alterações feitas para as opções do projeto.
.RES	Arquivo de Recursos do Compilador	Arquivo binário que contém o ícone, mensagens da aplicação e outros recursos usados pelo projeto.
.~DP	Arquivo de Backup do Projeto	Gerado quando o projeto é salvo pela segunda vez.
.~PA	Arquivo de Backup da Unit	Se um .PAS é alterado, este arquivo é gerado.
.~DF	Backup do Arquivo gráfico do formulário	Se você abrir um .DFM no editor de código e fizer alguma alteração, este arquivo é gerando quando você salva o arquivo.
.DSK	Situação da Área de Trabalho	Este arquivo armazena informações sobre a situação da área de trabalho especifica para o projeto em opções de ambiente(Options Environment).

Obs.: .~DF, .~PA , .~DP são arquivos de backup(Menu Options, Enviroment, Guia Editor Display, Caixa de Grupo Display and file options, opção Create Backup Files, desativa o seu salvamento).

Devido a grande quantidade de arquivos de uma aplicação, cada projeto deve ser montado em um diretório específico.

2.1.2 Arquivos Gerados pela Compilação

Extensão Arquivo	Definição	Função
.EXE	Arquivo compilado executável	Este é um arquivo executável distribuivel de sua aplicação. Este arquivo incorpora todos os arquivos .DCU gerados quando sua aplicação é compilada. O Arquivo .DCU não é necessário distribuir em sua aplicação.
.DCU	Código objeto da Unit	A compilação cria um arquivo .DCU para cada .PAS no projeto.

Obs.: Estes arquivos podem ser apagados para economizar espaço em disco.

2.2 Código fonte do arquivo Project(.DPR)

Nesta arquivo está escrito o código de criação da aplicação e seus formulários. O arquivo Project tem apenas uma seção.

Esta seção é formada pelo seguinte código:

PROGRAM - Define o Projeto;
USES - Cláusula que inicia uma lista de outras unidades.
Forms = É a unidade do Delphi que define a forma e os componentes do aplicativo
In = A clausula indica ao compilador onde encontrar o arquivo Unit.
Unit1 = A unidade que você criou
{$R *.RES} - Diretiva compiladora que inclui o arquivo de recursos.

Abaixo veja como fica o Project quando você abre um projeto novo:

```
program Project1;

uses
  Forms,
  Unit1 in 'UNIT1.PAS' {Form1};

{$R *.RES}

begin
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

2.3 Código fonte do arquivo Unit (.PAS)

Nesta divisória serão escritos os códigos dos seus respectivos forms (Unit1 = Form1). Aqui serão definidos os códigos de cada procedimento dos componentes que você colocar no form.

2.3.1 Seção Unit

Declara o nome da unit.

2.3.2 Seção Uses

Contém as units acessadas por este arquivo.

2.3.3 Seção Interface

Nesta seção estão as declarações de constantes, tipos de variáveis, funções e procedures gerais da Unit/Form. As declarações desta seção são visíveis por qualquer Unit. Esta seção é formada pelo seguinte código:

INTERFACE - Palavra que inicia a seção;
USES - Cláusula que inicia uma lista de outras unidades compiladas (units) em que se basea:
SysUtils = utilitários do sistema (strings, data/hora, gerar arquivos)
WinProcs = acesso a GDI, USER e KERNEL do Windows
Wintypes= tipos de dados e valores constantes
Messages=constantes com os números das mensagens do Windows e tipos de dados das Mensagens
Classes=elementos de baixo nível do sistema de componentes
Graphics=elementos gráficos
Controls=elementos de nível médio do sistema de componentes
Forms=componentes de forma e componentes invisíveis de aplicativos
Dialogs=componentes de diálogo comuns

2.3.4 Seção Type

Declara os tipos definidos pelo usuário. Subseções: Private, declarações privativas da Unit. Public declarações publicas da Unit.

2.3.5 Seção Var

Declara as variáveis privadas utilizadas.

2.3.6 Seção Implementation

Contém os corpos das funções e procedures declaradas nas seções Interface e Type. Nesta seção também estão definidos todos os procedimentos dos componentes que estão incluídos no Form. As declarações desta seção são visíveis apenas por ela mesma. Esta seção é formada pelo seguinte código:

{$R*.DFM} - Diretiva compiladora que inclui toda a interface, propriedades da forma e componentes do arquivo *.DFM

{$S+} - Diretiva compiladora que ativa verificação de pilha.

2.3.7 Seção uses adicional

Serve para declarar Units que ativam esta.

2.3.8 Inicialization

Nesta seção, que é opcional, pode ser definido um código para proceder as tarefas de inicialização da Unit quando o programa começa. Ela consiste na palavra reservada inicialization seguida por uma ou mais declarações para serem executadas em ordem.

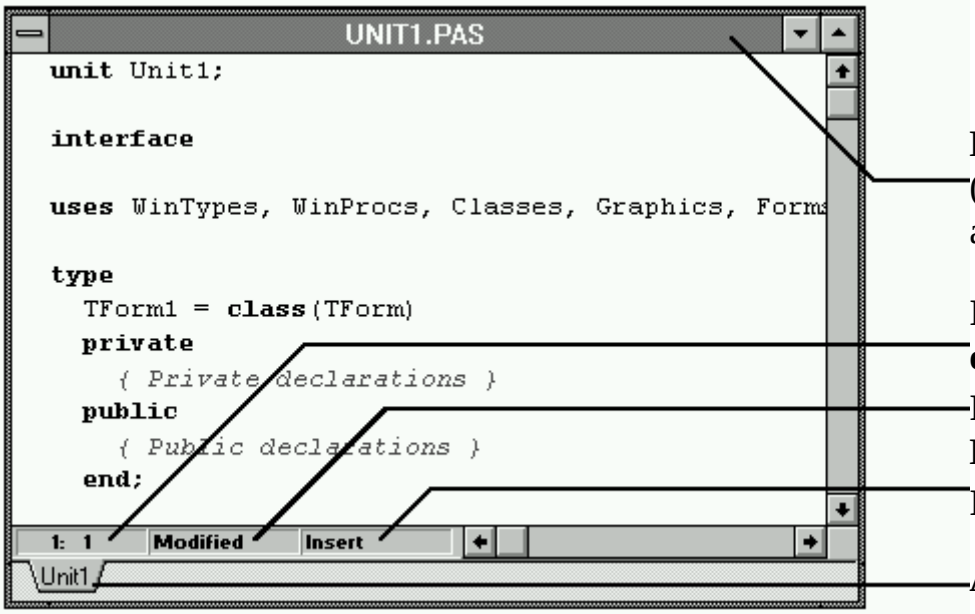
2.3.9 Exemplo

Abaixo veja como fica a unit quando você abre um projeto novo:

```
unit Unit1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
Forms, Dialogs;

type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

{Uses Adicional}

{Initialization}

end.
```

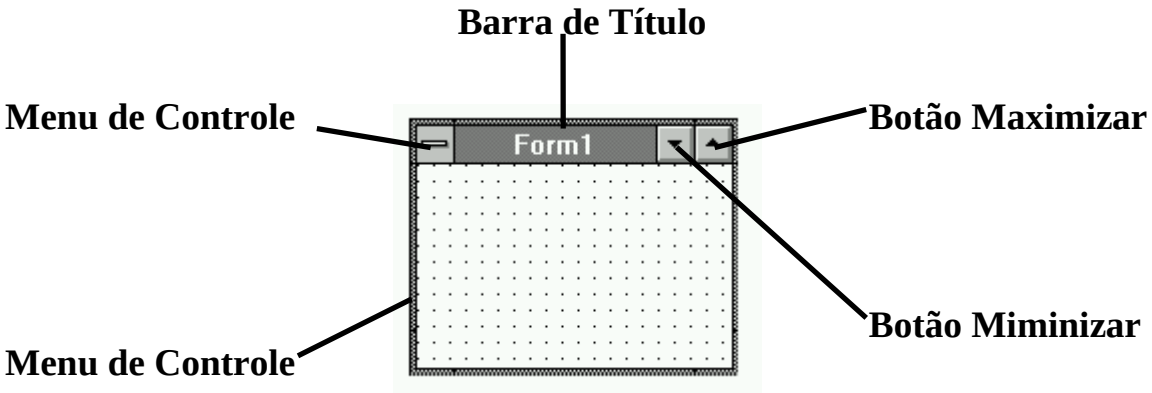
3. Janelas

3.1 Code Editor (Editor de Código)



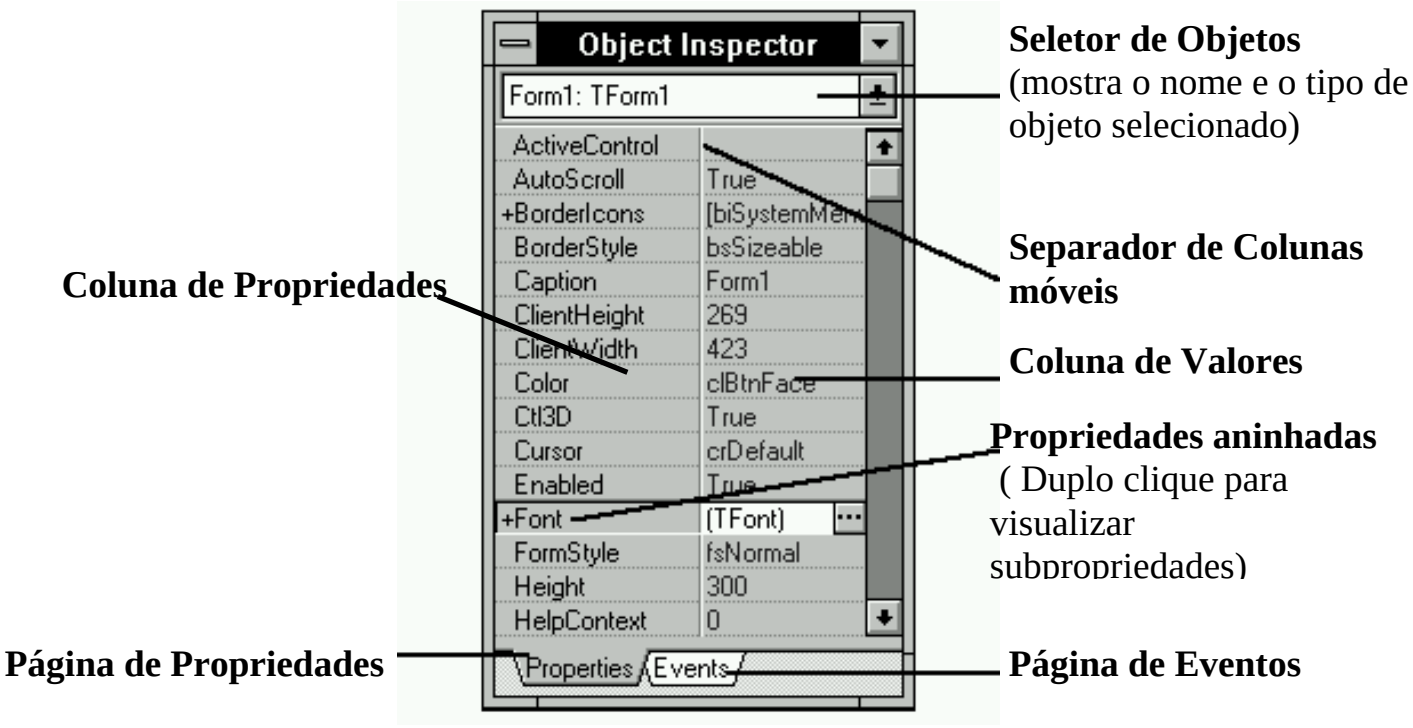
3.2 Form (Formulário)

Você usa formulários para fazer interface com o usuário, nele são inseridos os componentes. O formulário é uma janela, e portanto, possui os atributos de uma janela (menu de controle, botões de maximizar e minimizar), barra de título, bordas redimensionáveis).



3.3 Object Inspector (Inspetor de Objetos)

É uma ferramenta composta de duas páginas: Properties (Propriedades) e Events (Eventos).

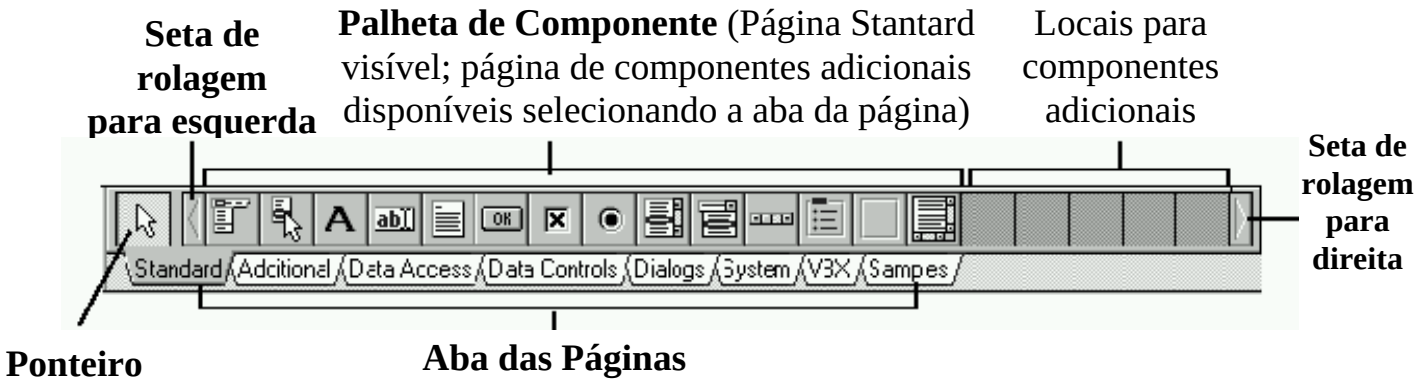


A página Properties([Propriedades) permite que você estabeleça parâmetros de formulários e componentes. Estes parâmetros especificam os valores iniciais de características como nome do componente e sua posição no formulário.

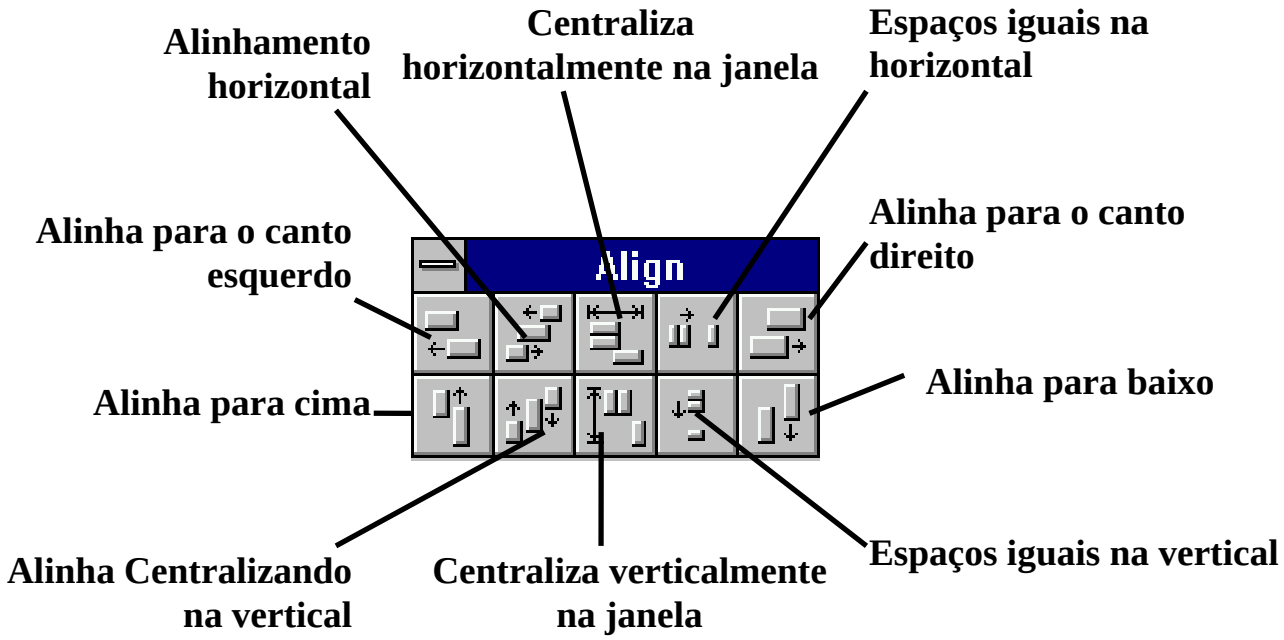
A páginas Events (Eventos) permite associar os componentes com ações do usuário.

3.4 Component Palette(Palheta de Componentes)

É composta de várias páginas contendo cada uma grupos de componentes. Para mudar de página na palheta de componentes, clique sobre a aba da página que você quiser acessar.



3.5 Alignment Palette(Palheta de Alinhamento)



Outras: Browser, BreakPoints, CallStack, Watches, Component List e Windows List.

Obs.: Devido a grande quantidade de janelas é interessante que se tenha no mínimo a resolução de vídeo 800X600, para se ter espaço para o desenvolvimento.

4. Orientação a Objetos

Por ser baseado no Pascal Object, o Delphi permite que se construa aplicações orientadas a objetos. Em linhas gerais, aplicações orientadas a objetos se baseiam no conceito de classe. A classe é um tipo de dados, contendo atributos e serviços. O objeto é uma variável de determinada classe. Por exemplo, um formulário nada mais é do que um objeto da classe Formulário (Tform). Contém atributos e serviços.

4.1 Eventos

Os programas feitos em Delphi são orientados a eventos. Eventos são ações normalmente geradas pelo usuário. Ex.: Clicar o mouse pressionar uma tecla, mover o mouse etc. Os eventos podem ser também gerados pelo windows.

Existem eventos associados ao formulário e cada componente inserido neste.

Exemplos:

- Ao formulário está ligado *on create*, que ocorre quando mostramos o formulário na tela.
- Ao componente botão está ligado o evento *on click*, que ocorre quando damos um click com o mouse sobre o botão.

4.1.1 Eventos comuns ao formulário e aos componentes.

Alguns eventos ligados tanto ao formulário quanto aos componentes estão listados a seguir.

- ***OnClick:*** ocorre quando o usuário clica o objeto.
- ***OndblClick:*** ocorre quando o usuário da um duplo clique.
- ***OnKeyDown:*** ocorre quando o usuário pressiona uma tecla enquanto o objeto tem foco.
- ***OnKeyUp:*** ocorre quando o usuário solta uma tecla enquanto o objeto tem o foco.
- OnKeyPress: ocorre quando usuário da um clique numa tecla ANSI.
- OnMouseDown: ocorre quando o usuário pressiona o botão do mouse.
- OnMouseUp: ocorre quando o usuário solta o botão do mouse.
- OnMouseMove: ocorre quando o usuário move o ponteiro do mouse.

4.1.2 Rotinas que Respondem a Eventos

Cada evento gera uma procedure, aonde você deve inserir as linhas de código que envolvem este evento. Por exemplo, o evento OnClick, que é gerado ao clicarmos em um botão chamado BTNSair, cria a procedure:

```
Procedure TForm1.BTNSairClick(Sender: Tobject);
```

onde Tform1 é o objeto Tform que contém o botão BTNSair, e Sender é um objeto Tobject que representa o componente que deu origem ao evento.

Se você quiser inserir uma rotina que trate um determinado evento de um componente, faça o seguinte:

- clique sobre o componente;
- no Object Inspector, seleciona a página Events;
- dê um duplo clique sobre o evento para o qual quer inserir o código;
- entre no editor de código e escreva as linhas de código.

Exemplo:

```
Procedure TForm1.BTNSairClick(Sender: Tobject);
begin
  Form1.Close;
end;
```

Obs.: Escreva seu código entre o begin e o end, se por acaso você quiser retirar o evento e o componente, retire primeiro os eventos do componente removendo somente o código que você colocou e depois o componente; os resto dos procedimentos o DELPHI tira para você.

4.2 Propriedades

Como vimos, eventos podem estar associados a modificações em propriedade de componente e formulário, ou seja, você pode modificar propriedades de formulários e componentes durante a execução do sistema. Para isto você deverá usar a sintaxe:

<componente>.<propriedade>;

Por exemplo, para modificar a propriedade *text* de uma caixa de edição Edit1 para “Bom Dia” faça:

Edit1.Text := ‘Bom Dia’;

Se a propriedade do componente tiver subpropriedades, para acessa-lá, utilize a seguinte sintaxe:

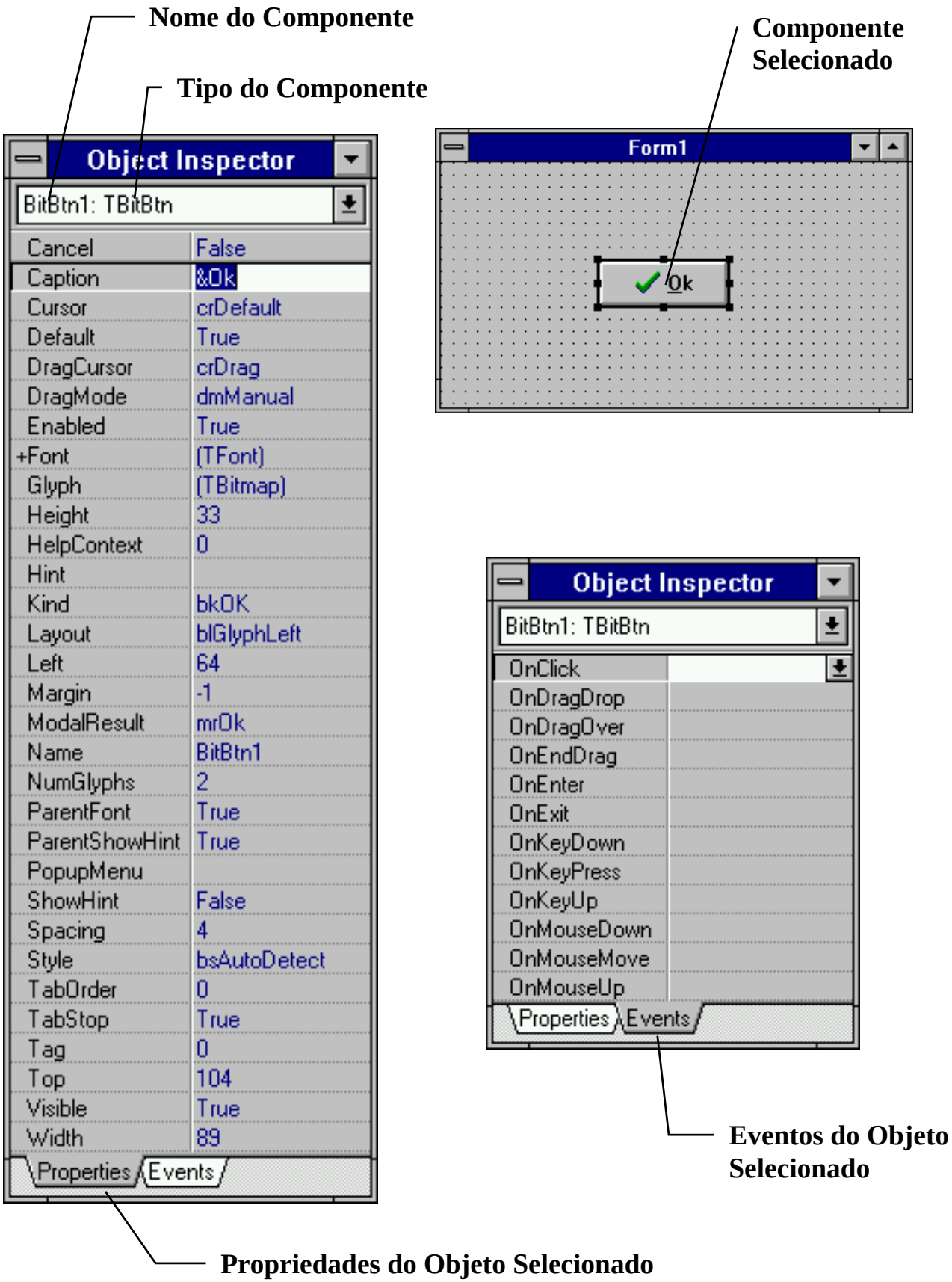
<componente>.<propriedade>.<subpropriedade>

Por exemplo, para modificar a subpropriedade Name referente a propriedade fonte, de uma caixa de edição Edit1, para ‘Script’, faça:

Edit1.Font.name := ‘Script’;

Obs.: Verifique o tipo da propriedade para antes de mandar o valor, consultando no Objetc Inspector.

4.2.1 Consultando Propriedades e Eventos de um componente.



Obs.: Cada componente tem sua próprias propriedades e eventos, e podem existir propriedades iguais de um componente para o outro, portanto lembre-se das propriedades e eventos comuns entre eles.

4.3 Métodos

São procedures ou funções embutidas nos componentes e formulários, previamente definidas pelo Delphi.

Alguns métodos são descritos a seguir:

- Show : Mostra um formulário;
- Hide : Esconde um formulário mais não o descarrega;
- Print : Imprime um formulário na impressora;
- SetFocus : Estabelece o foco para um formulário ou componente;
- BringtoFront: Envia para frente.

4.3.1 Chamado de métodos como resposta a eventos:

Um evento pode gerar a chamada para um método, ou seja, uma subrotina previamente definida para um componente.

No código, para utilizar um método, use a seguinte sintaxe:

<nome do objeto>.<método>

Por exemplo, clicar em um botão pode dar origem ao evento Show de um outro formulário, mostrando este novo formulário na tela:

Form2.show;

5. Application

Sempre que você processa um projeto DELPHI, o Delphi cria automaticamente o objeto Application. Você tem de trabalhar com propriedades e métodos desse objeto para poder fazer coisas como especificar o ícone do aplicativo, especificar o nome de arquivo de Help do projeto ou o titulo da aplicação.

5.1 Métodos:

MessageBox - Exibe uma caixa mensagem padrão Windows.
Método Minimize - Minimiza o aplicativo a um ícone;
Método Restore - Restaura o aplicativo ao estado anterior
ProcessMessages - Permite ao seu aplicativo deixar o Windows processar eventos para outros aplicativos.
Terminate - Finaliza a aplicação.

5.2 Propriedades:

ExeName - Traz o nome do executável que é o nome projeto
HelpFile - Nome de arquivo de Help.
Icon - Ícone da aplicação por completo, sendo que cada janela tem seu próprio ícone.
Title - Título da aplicação.

Obs.: Algumas Propriedades podem ser acessadas pelo Menu Options, Opção Project, Guia Application.

6. MDI Application

Uma aplicação MDI (Multiple Document Interface) é aquela em que vários formulários ou janelas podem ser abertas em um pai.
Exemplos: Word, Excel.

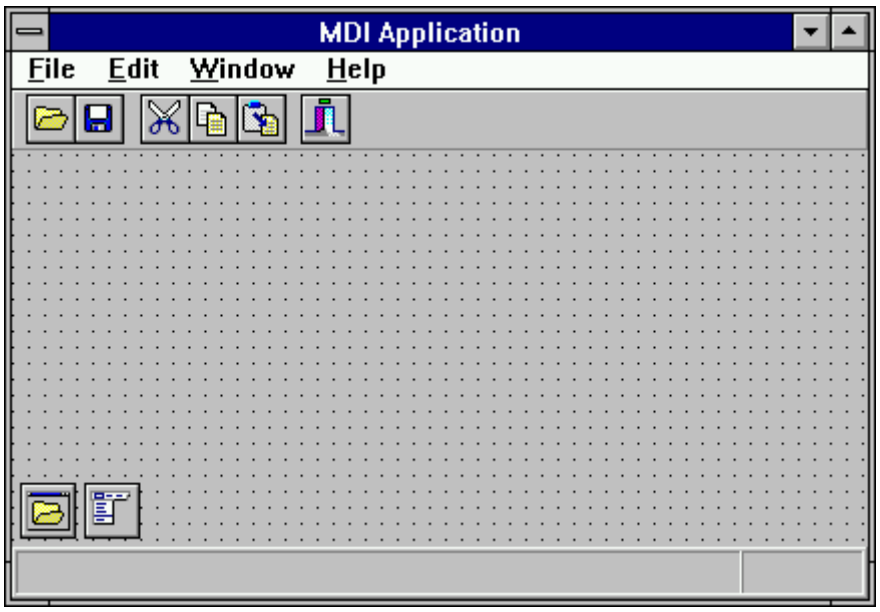
6.1 Usando o Modelo de Aplicação MDI

Para criar uma aplicação MDI automaticamente:

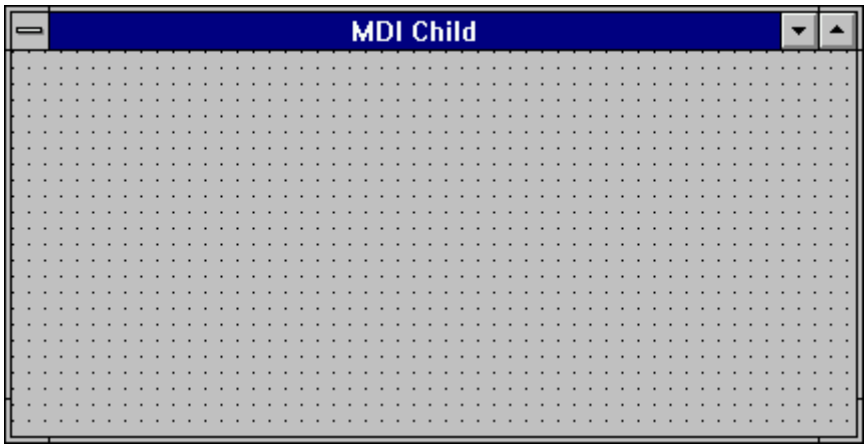
Ative a galeria de projetos, através do Menu Options, opção Environment, guia preferences, marcando a opção *Use on New Project*, do quadro Gallery;

No Menu do Delphi, escolha File opção New Project.
Clique sobre o ícone da aplicação MDI,
Clique no botão OK.
O DELPHI pede então o diretório onde você irá armazenar a aplicação.
Escolha o diretório, e clique no botão OK.

O DELPHI gera automaticamente um projeto MDI, contendo dois formulários: um formulário pai e um formulário Filho. O Formulário pai, de nome *MainForm*, ligado a unit *Main*, possui o seguinte aspecto:



O Formulário Filho modelo, de nome MDIChild, ligado a unit Childwin, apresenta o seguinte aspecto:



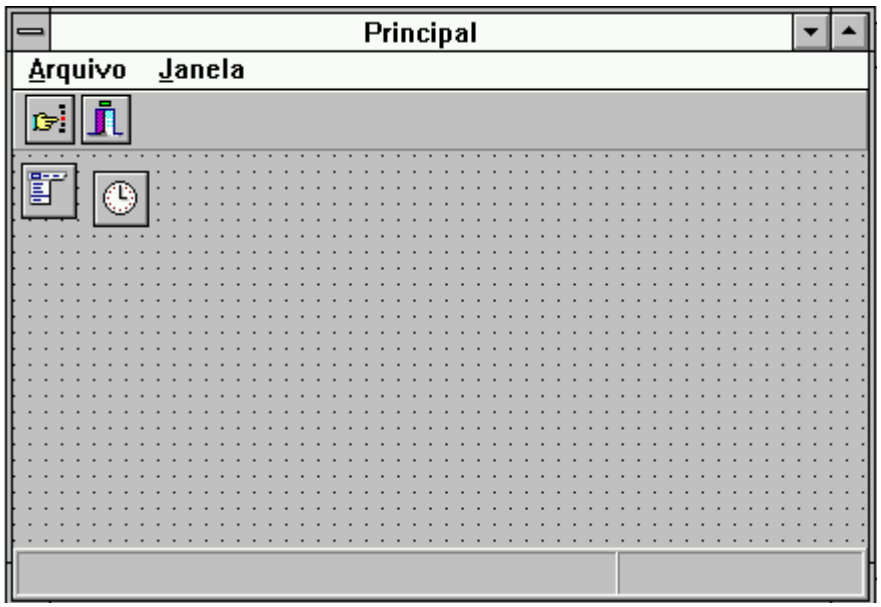
Obs.: A diferença entre os dois formulários esta na propriedade **FormStyle**, onde a MainForm está como fsMDIForm(Pai) e ChildWin como fsMDIChild (Filho).

6.2 Exemplo MDI

6.2.1 Projeto.DPR

```
program Projeto;
uses
  Forms,
  Principa in 'PRINCIPA.PAS' {FrmPrincipal},
  Cadastro in 'CADASTRO.PAS' {FrmCadastro};
{$R *.RES}
begin
  Application.CreateForm(TFrmPrincipal, FrmPrincipal);
   Application.Run;
end.
```

6.2.2 Principa.DFM



6.2.3 Principa.PAS

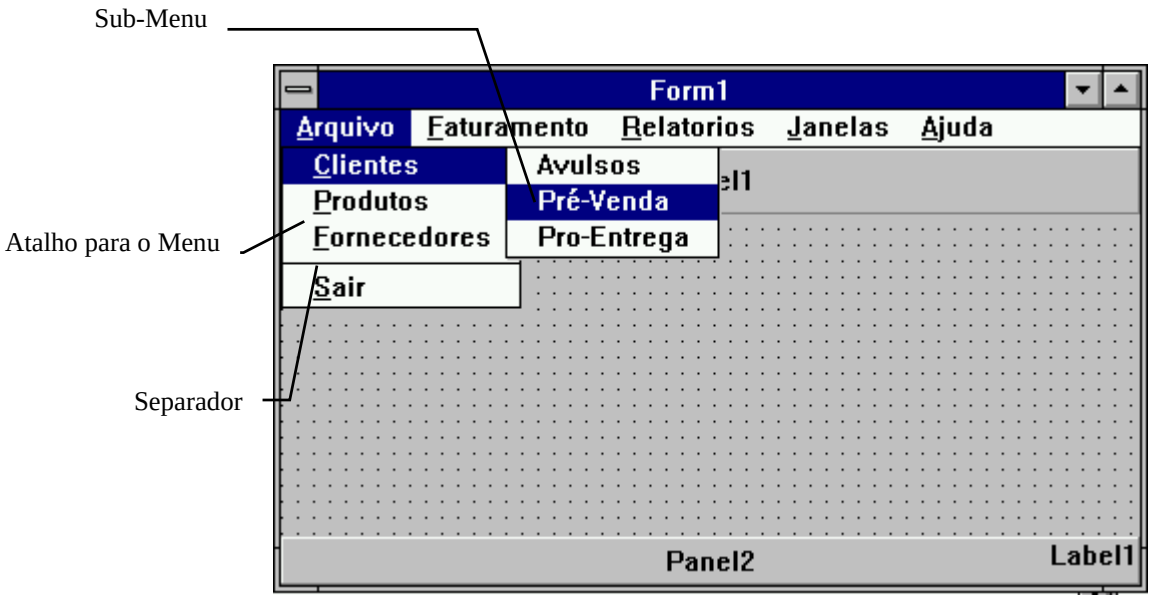
```
unit Principa;

interface

uses WinTypes, WinProcs, SysUtils, Classes, Graphics, Forms, Controls,
Menus,StdCtrls, Dialogs, Buttons,Messages, ExtCtrls, Cadastro;
                        (* Aqui eu declaro as janelas filha *)

type
  TFrmPrincipal = class(TForm)
    MenPrincipal: TMainMenu;
    Panel1: TPanel;
    PanLinhaStatus: TPanel;
    MenArquivo: TMenuItem;
    PanTempo: TPanel;
    Window1: TMenuItem;
    MenArquivoSair: TMenuItem;
    WindowCascadeItem: TMenuItem;
    WindowTileItem: TMenuItem;
    WindowArrangeItem: TMenuItem;
    WindowMinimizeItem: TMenuItem;
    PanBotoes: TPanel;
    SpbSair: TSpeedButton;
    TimTempo: TTimer;
    MenArquivoCadastro: TMenuItem;
    procedure FormCreate(Sender: TObject);
    procedure WindowCascadeItemClick(Sender: TObject);
    procedure WindowTileItemClick(Sender: TObject);
    procedure WindowArrangeItemClick(Sender: TObject);
    procedure MenArquivoSairClick(Sender: TObject);
    procedure WindowMinimizeItemClick(Sender: TObject);
    procedure TimTempoTimer(Sender: TObject);
    procedure MenArquivoCadastroClick(Sender: TObject);

  private
    { Private declarations }
    procedure ShowHint(Sender: Tobject);
    Function Formexiste(nomejanela:Tform):boolean;

  public
    { Public declarations }
    FrmCadastro : TFrmCadastro;  (* Declara a variável que vai receber
                                    a janela caso você mude seu nome*)
  end;

var
```

```
  FrmPrincipal: TFrmPrincipal;

implementation

{$R *.DFM}

Function TForm1.Formexiste(Nomejanela:TForm):boolean;
var
  I : integer;
begin
  Formexiste := false;
  for i := 0 to componentcount - 1 do
     if Components[i] is TForm then
        if TForm(Components[i])= nomejanela then
           Formexiste := true
end;

procedure TFrmPrincipal.FormCreate(Sender: TObject);
begin
  Application.OnHint := ShowHint;
  ContaCadastro := 0;  (* Zero a variavel contadora da janela  quando
                          eu crio o formulario principal *)
end;

procedure TFrmPrincipal.ShowHint(Sender: TObject);
begin
  PanlinhaStatus.Caption := Application.Hint;
end;

procedure TFrmPrincipal.MenArquivoSairClick(Sender: TObject);
begin
  Close;
end;

procedure TFrmPrincipal.WindowCascadeItemClick(Sender: TObject);
begin
  Cascade;
end;

procedure TFrmPrincipal.WindowTileItemClick(Sender: TObject);
begin
  Tile;
end;

procedure TFrmPrincipal.WindowArrangeItemClick(Sender: TObject);
begin
  ArrangeIcons;
end;

procedure TFrmPrincipal.WindowMinimizeItemClick(Sender: TObject);
var
  I: Integer;
begin
  for I := MDIChildCount - 1 downto 0 do
    MDIChildren[I].WindowState := wsMinimized;
end;

procedure TFrmPrincipal.TimTempoTimer(Sender: TObject);
begin
  Pantempo.caption:= timetostr(now)+ ' - ' + Datetostr(date)+ ' ';
end;

procedure TFrmPrincipal.MenArquivoCadastroClick(Sender: TObject);
begin
If Formexiste(FrmCadastro) = false then (* Se for falso então cria a
                                     janela *)
   Begin
      Screen.Cursor := CrHourGlass; (* Coloca o cursor na forma de uma
                                       ampulheta *)
      FrmCadastro := TFrmCadastro.Create(Self); (*Cria a janela filha
                               e atribui a variável declarada *)
   End
Else
  If Formexiste(Frmcadastro) = true  then (* Se for tentar abrir a
                        janela e ela ja estiver sido criada *)
   Begin
```

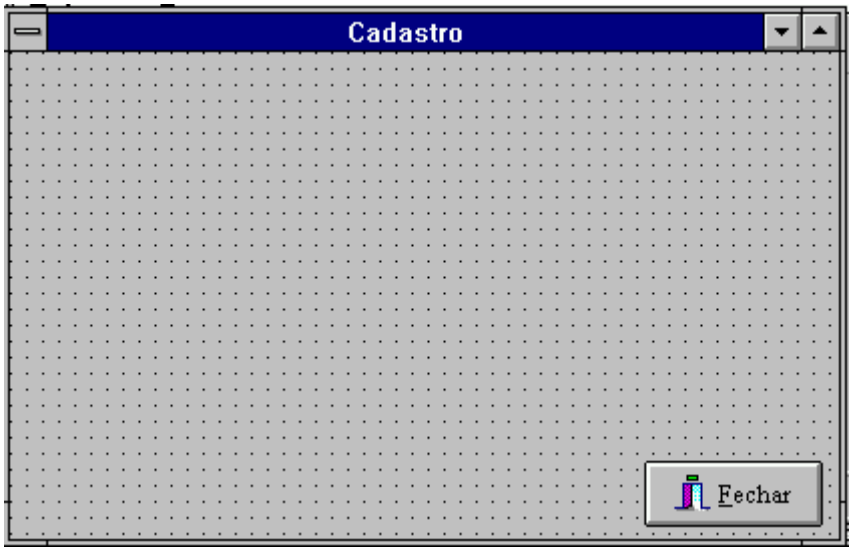
```
      FrmCadastro.WindowState := WsNormal; (* Se estiver minimizada
                                          restaura *)
      FrmCadastro.BringToFront; (*Se estiver atrás, coloca na frente*)
      FrmCadastro.Setfocus; (* Coloca o foco na janela *)
   End;
end;
end.
```

Obs.:

- No menu para colocar separadores coloca "- " no caption, de um item do menu.
- Para sublinhar a letra do item de menu que vai ser o atalho coloque o "&" antes da letra escolhida, isto também é aplicado a outros componentes como os buttons, radiogrupou e outros.
- Para abrir um Sub-Menu pressione a tecla Control e a seta para direita no item que for necessário o Sub-Menu.



6.2.4 Cadastro.DFM



6.2.5 Cadastro.Pas

```
unit Cadastro;
```

```
interface
uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
Forms, Dialogs, StdCtrls, Buttons, ExtCtrls, DBCtrls, Mask, DB,
DBTables;

type
  TFrmCadastro = class(TForm)
    BitBtn1: TBitBtn;
    procedure BitBtn1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
   procedure FormClose(Sender: TObject; var Action:
TCloseAction);

private
    { Private declarations }
public
    { Public declarations }
  end;
var
  FrmCadastro: TFrmCadastro;

implementation

{$R *.DFM}

procedure TFrmCadastro.FormCreate(Sender: TObject);
begin
  Screen.Cursor := CrDefault; (* Coloca o cursor em modo normal *)
end;

procedure TFrmCadastro.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
   Action := Cafree; (* Libera a variavel da janela da memória *)
end;

procedure TFrmCadastro.BitBtn1Click(Sender: TObject);
begin
  Close; (* Fecha a janela *)
end;
end.
```

7. Trabalhando com Banco de Dados

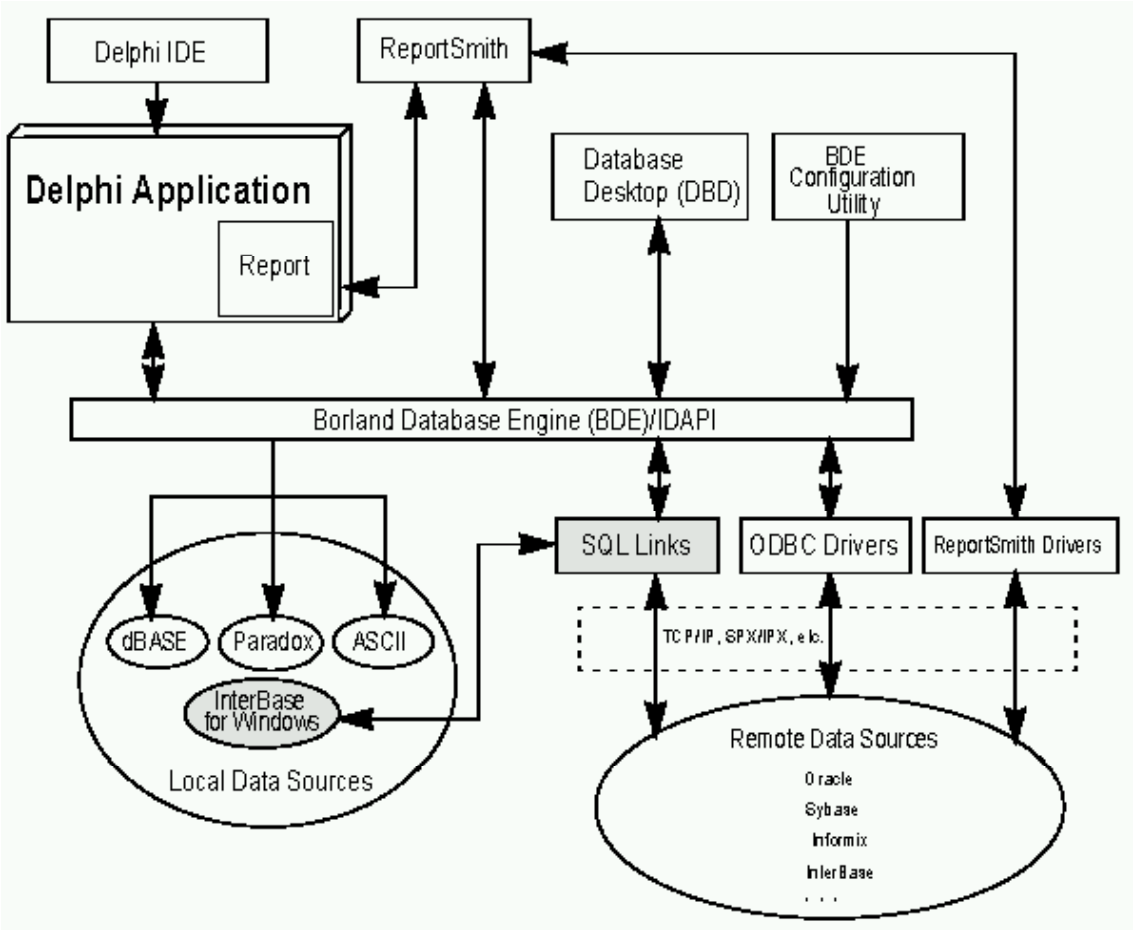
7.1 BDE

O Borland Database Engine é o coração do Delphi e suas aplicações com banco de dados usando o mesmo database engine usado pelo Paradox e dBase. Paradox e dBase é claro trazem capacidades adicionais além do database engine (como o Delphi também o faz), mas isso é longe de dizer que o valor agregado destes recursos adicionais é maior de 2 % das capacidades - os outros 98% são providos pelo database engine e estão disponíveis para todos os usuários deste engine.

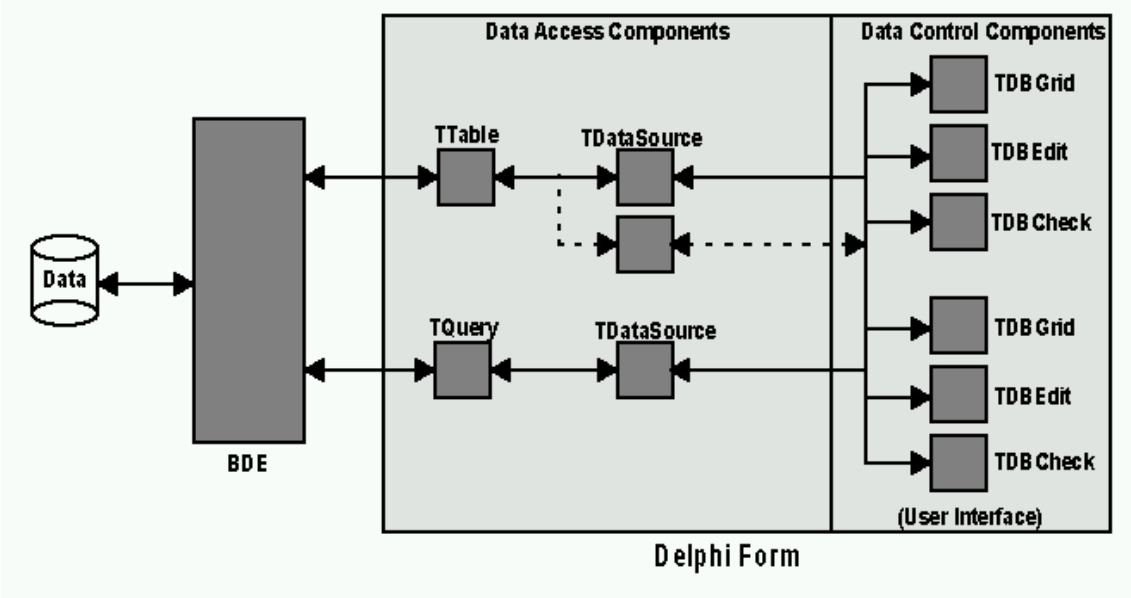
O Borland Database Engine (BDE) é uma coleção de DLLs que as aplicações de banco de dados irão fazer chamadas. Cada estação de trabalho que tiver a aplicação de banco de dados instalada deverá ter, também, o BDE instalado (o Delphi vem com a instalação do BDE para você adicionar a sua aplicação).

O BDE permite a você usar tabelas dBase, Paradox ou ODBC em modo multi-usuário. A versão Cliente/Servidor do Delphi também vem com links para servidores de banco de dados como Oracle, Sybase, MS SQL Server, Informix, e InterBase.

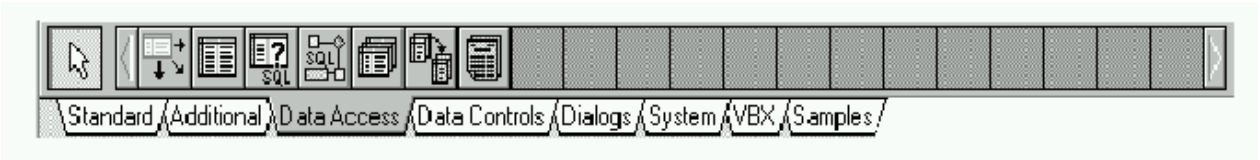
7.2 Arquitetura do Banco de Dados do Delphi



7.3 Arquitetura dos Componentes de Banco de Dados



7.4 Paleta de Componentes da página de Acesso de Dados



Componente	Utilidade
TDataSource	Atua como um conduto entre componentes TTable, TQuery, TStoredProc e componentes de controle, como um TDBGrid
TTable	Recupera dados de um tabela de banco de dados via o BDE e abastece ele para um ou mais componentes de controle de dados através do componente TDataSource. Envia dados recebidos de um componente para uma banco de dados via o BDE.
TQuery	Usa estruturas em SQL para recuperar dados de uma tabela de banco de dados via o BDE, e abastece ele para um ou mais componentes de controle de dados através de um TDataSouce, ou usa estruturas em SQL para enviar dados de um componente para um banco de dados via o BDE.
TStoreProc	Habilita uma aplicação para acessar procedimentos armazenados em um servidor. Envia dados recebidos de um componente para um banco de dados via o BDE.
TDataBase	Instala uma conexão permanente com um banco de dados, especialmente um banco de dados remoto requerendo um login do usuário e uma senha.
TBatchMove	Copia a estrutura de uma tabela ou seus dados. Pode ser usado para mover tabelas inteiras de uma formato de banco de dados para outro.
TReport	Habilita imprimir e visualizar relatórios de um banco de dados através do ReportSmith.

7.5 Paleta de Componentes da página de Controle de Dados

Componente	Utilidade
TDBNavigator	Botões de navegação para controle de dados que move o apontador do registro corrente de uma tabela para o posterior ou anterior, inicia inserção ou modo de edição; confirma novos ou modificações de registros; cancela o modo de edição; e refresca(refaz) a tela para recuperar dados atualizados.
TDBText	Rótulo de controle de dados que pode mostrar um campo corrente do registro ativo.
TDBEdit	Caixa de edição de controle de dados que pode mostrar ou editar um campo corrente do registro ativo.
TDBCheckBox	Caixa de verificação de controle de dados que pode mostrar ou editar dados lógico de uma campo corrente do registro ativo.
TDBListBox	Caixa de Lista de controle de dados que pode mostrar valores da coluna de uma tabela.
TDBComboBox	Caixa de Lista móvel de controle de dados que pode mostrar ou editar valores da coluna de uma tabela.
TDBRadioGroup	Grupos de radio de controle de dados com botões de rádio que pode mostrar ou setar valores de colunas.
TDBGrid	Grade padrão de controle de dados que possibilita visualizar e editar dados de forma tabular, semelhante a uma planilha; faz uso extensivo das propriedades do TField (estabelecidos no Editor de Campos(Fields Editor)) para determinar a visibilidade de uma coluna, formato da visualização, ordem, etc.
TDBMemo	Caixa memo de controle de dados que pode mostrar ou editar dados textos BLOB do registro corrente ativo.
TDBImage	Caixa de Imagem de controle de dados que pode mostrar, cortar ou colar imagens BLOB bitmap's do registro corrente ativo.
TDBLookupList	Caixa de Lista de controle de dados que mostrar valores mapeados através de outra tabela em tempo de compilação.
TDBLookupCombo	Caixa de Combo de controle de dados que mostrar valores mapeados através de outra tabela em tempo de compilação.

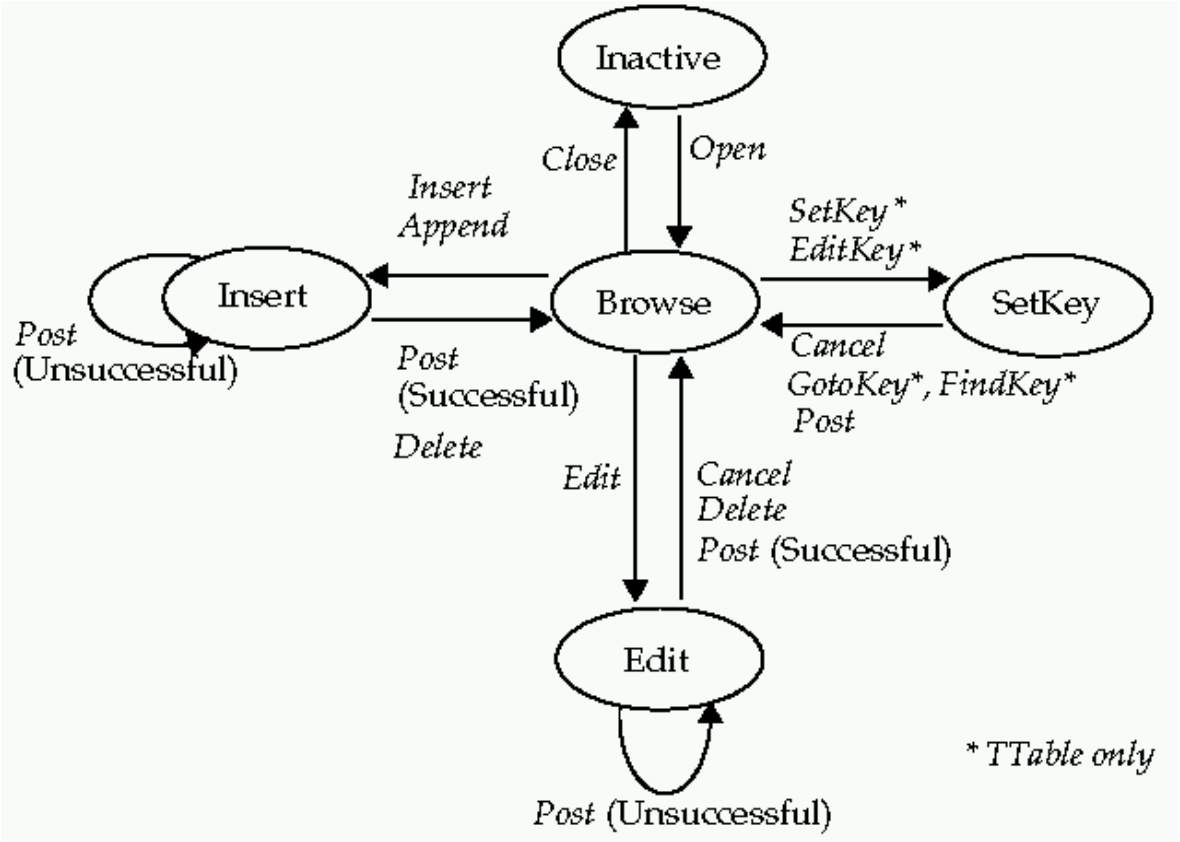
7.6 Usando DataSets

As classes de componentes TTable e TQuery são descendentes de TDataset através do TDBDataSet. Esta classe de componente herda uma parte de propriedades, métodos e eventos.

7.6.1 Estados do DataSet

Estado	Descrição
dsInactive	O Dataset esta fechado.
dsBrowse	Estado de Espera. O estado default quando um dataset é aberto. Registros pode ser visualizados mas não mudados ou inseridos.
dsEdit	Habilita o a linha corrente para ser editada.
dsInsert	Habilita uma nova linha para ser inserida. Uma chamada para o Post inserir a nova linha.
dsSetKey	Habilita FindKey, GotoKey, and GoToNearest para procurar valores nas tabelas do banco de dados. Estes métodos somente pertencem para o componente TTable. Para TQuery, a procura é feita com a syntax do SQL.
dsCalcFields	Modo quando os OnCalcFields é executado; previne qualquer mudança para outros campos ou campos calculados. Raramente é usado explicitamente.

7.6.2 Diagrama de Estados do Dataset



7.6.3 Usando os Eventos do DataSource

7.6.3.1 Evento StateChange

```
procedure TForm1.DataSource1.StateChange(Sender:TObject);
var S:String;
begin
   case Table1.State of
     dsInactive: S := 'Inactive';
      dsBrowse: S := 'Browse';
      dsEdit: S := 'Edit';
      dsInsert: S := 'Insert';
      dsSetKey: S := 'SetKey';
   end;
   Label1.Caption := S;
end;


procedure Form1.DataSource1.StateChange(Sender: Tobject);
begin
   InsertBtn.Enabled := (Table1.State = dsBrowse);
   CancelBtn.Enabled := Table1.State in [dsInsert, dsEdit,
      dsSetKey];
   ...
end;
```

7.6.4 Abrindo e Fechado DataSets

Antes que uma aplicação possa acessar dados através de um dataset, o dataset deve ser aberto. Existem dois meios para abrir o dataset:

- Ativando a propriedade Active para *True*, isto pode ser feito através do Object Inspector ou programavelmente em tempo de execução.

```
Table1.Active := True;
```

- Chamando o método do Dataset *Open* em tempo de execução.

```
Table1.open;
```

Para fechar o processo e semelhante só muda o a propriedade para *False* e o evento para ***Close;***

7.6.5 Navegando no Dataset

Propriedades e Métodos de Navegação

Método ou Propriedade	Descrição
Método First	Move o cursor para a primeira linha em um dataset.
Método Last	Move o cursor para a ultima linha em um dataset.
Método Next	Move o cursor para a próxima linha em um dataset.
Método Prior	Move o cursor linha anterior em um dataset.
Propriedade BOF	True quando o cursor está no inicio do dataset, em outro caso é false.
Propriedade EOF	True quando o cursor está no final do dataset, em outro caso é false.
Método MoveBy(n)	Move o cursor para n linhas a frente em um dataset, quando n é um numero inteiro positivo ou negativo.

Exemplo:

```
Table1.Last;  (* Move o cursor para o último registro da tabela *)
Table1.Next;  (* Move o cursor para o próximo registro da tabela *)
```

7.6.6 Modificando dados no Dataset

Os seguintes métodos habilitam uma aplicação para inserir, atualizar, e deletar dados no dataset.

Método	Descrição
Append	Comfirma qualquer dado pendente e move um registro em branco para o final do dataset, e coloca o dataset em estado de Insert.
Cancel	Cancela a operação corrente e coloca o dataset em estado de Browse.
Delete	Apaga o registro corrente e coloca o dataset em estado de Browse.
DeleteTable	Apaga uma tabela
Edit	Coloca o dataset em estado de edição. Se um dataset já está estado de Edit ou Insert, uma chamada para Edit não tem efeito.
EmptyTable	Esvazia uma tabela.
Insert	Confirma qualquer dado pendente, e coloca o dataset em estado de inserção.
Post	Tentativa para confirmar um registro novo ou alterado. Se sucesso, o dataset e colocado em estado de Browse; se insucesso, o dataset fica no estado corrente.
Refresh	Atualiza a visão do banco de dados.

7.6.7 Lendo Valores do Campo

Existem algumas maneiras de ler dados de um dataset:

```
1. Edit1.text := Table1Nome_Clie.asstring;
2. Edit1.text := Table1.Fields[1].asstring;
3. Edit1.text := Table1.FieldByName('Nome_Clie').asstring;
```

Para associar outros tipos de campos que não texto a uma caixa de edição (que só aceita valores do tipo string), devemos utilizar propriedades de conversão do componente TField: AsBoolean, AsDataTime, AsFloat (Real), AsInteger, AsString.

Para atribuir valores para o dataset o procedimento é o mesmo só que no sentido inverso, desde que que a tabela esteja em modo de edição ou inserção.

7.6.8 Marcando Dados

Os métodos para marcar são

- GetBookmark - Pega a posição do registro e coloca em uma variável do tipo TBookmark;
- GotoBookmark - Leva o apontador para uma variavel do tipo TBookmark;
- FreeBookmark - Desaloca uma variavel do tipo TBookmark;

Exemplo:

```
procedure DoSomething;
var
   Marca: TBookmark;
begin
   Marca  :=  Table1.GetBookmark; {Marca a posição}
   Table1.DisableControls; {Desabilita o controle de dados}
   try
       Table1.First;
       while not Table1.EOF do
            begin
               {Faz alguma coisa}
               Table1.Next;
            end;
    finally
       Table1.GotoBookmark(Marca);
       Table1.EnableControls;
       Table1.FreeBookmark(Marca); {Desaloca a variavel}
    end;
end;
```

7.6.9 Procurando dados em uma Tabela

Os métodos GoToKey e GoToNearest habilita uma aplicação para procurar uma chave em uma tabela. SetKey coloca a tabela em modo de procura. No estado SetKey, atribui-se valores para a procura em um campo indexado. GoToKey então move o cursor para a primeira linha da tabela que encontrou este valor. Como GoToKey é uma função ela retorna True ou False se teve sucesso na procura. O GoToNearest é uma função que leva para o valor mais parecido encontrado.

Exemplo:

```
procedure TSearchDemo.SearchExactClick(Sender: TObject);
begin
  Table1.SetKey; {Primeiro campo é a chave}
  Table1.Fields[0].AsString := Edit1.Text;  (*
Table1.Fieldbyname('Cidade') := Edit1.Text *)
  Table1.GoToKey;
end;
```

Todo procedimento acima pode ser substituído usando a função *Find*
Exemplo:

```
procedure TSearchDemo.SearchExactClick(Sender: TObject);
begin
  Table1.FindKey([Edit1.text]);
end;
```

Exemplo de outra forma de procura em uma tabela.

```
procedure TSearchDemo.SearchExactClick(Sender: TObject);
begin
  Table1.SetKey; {Primeiro campo é a chave}
  Table1.Fieldbyname('Cidade') := Edit1.Text;
  Table1.GoToKey;
end;
```

Procura em um indíce secundário.

```
procedure TSearchDemo.SearchExactClick(Sender: TObject);
begin
  Table1.indexname := 'Nome_Clie';
  Table1.open;
  Table1.setkey;
  Table1Nome_clie.asstring := Edit1.text;
  Table1.gotonearest;
end;
```

Procura em um indíce primário ou secundário.

```
procedure TSearchDemo.SearchExactClick(Sender: TObject);
begin
  Table1.indexfieldnames := 'Codi_Clie';
  Table1.open;
  Table1.setkey;
  Table1Nome_clie.asstring := Edit1.text;
  Table1.gotonearest;
end;
```

7.6.10 Filtrando Registros em uma Tabela

Você pode filtar registros de uma tabela através de métodos específicos:

- SetRangeStart: Estabelece o limite inicial do filtro.
- SetRangeEnd: Estabelece o limite final do filtro.
- ApplyRange: Aplica o filtro à tabela.
- CancelRange: Cancela o filtro aplicado à tabela.
- KeyExclusive: Exclui a chame do indíce setado.

Exemplo:

```
Table1.SetRangeStart;
Table1.FieldByName('Codigo'):=100;(*Table1Codigo.asinteger := 100 *)
Table1.KeyExclusive := False;
Table1.SetRangeEnd;
Table1.FieldByName('Codigo'):=200;(*Table1Codigo.asinteger := 200 *)
Table1. KeyExclusive := True;
Table1.ApplyRange;
```

Obterá um filtro dos registros de clientes com código entre 100 e 199.

7.6.11 Índices de uma Tabela

Método	Descrição
GetIndexNames	Retorna uma lista de nomes disponiveis de índeces para uma tabela do banco de dados.
IndexFieldCount	Número de Campos do Índice.
IndexFields	Vetor de nomes de campos usados no índice
IndexName	Para usar o índice primário de uma tabela Paradox.
IndexFieldNames	Para muda o índice para qualquer campo.

As propriedades IndexName e IndexFieldNames são mutualmente exclusivas. Colocando uma propriedade, limpa o valor da outra.

7.6.12 Sincronizando Tabelas

O método GotoCurrent é um método que sincroniza dois componentes table ligados a uma mesma tabela em um banco de dados e usando o mesmo indíce.

```
Table1.GotoCurrent(Table2);
```

8. A Linguagem SQL

A linguagem SQL (Structured Query Language - Linguagem Estruturada de Pesquisa) foi criada para ser uma linguagem padrão para consulta, atualização e manipulação de banco de dados em um banco de dados relacional.

Comercialmente implementada pela IBM, se tornou um padrão de linguagem de acesso a dados em vários bancos de dados relacionais, como Oracle, DB2, SQL Server, Sybase, Interbase, etc.

Usaresmos as Declarações em SQL para extrair/Atualizar registros de uma ou mais tabelas que atendam as condições especificadas, manipulando, assim, somente os dados que sejam de nosso interesse.

Por exemplo, a declaração a permite que somente os registros cujo o campo Nome Começando pela letra A da tabela de Clientes sejam exibidos na tela:

```
Select *
From Clientes
Where Nome="A*"
```

Podemos dividir os comandos da linguagem SQL em três categorias distintas:

- Comandos de Definição de Dados: permitem definir ou alterar tabelas em um banco de dados.
- Comandos de Controle de Dados: servem para gerenciar o acesso dos usuários a determinadas tabelas.
- Comandos para a Manipulação de Dados: servem para manipular os dados contidos nas tabelas.

8.1 Comandos de Manipulação de Dados

8.1.1 Update

Atualiza registros.

8.1.2 Delete

Deleta Registros

8.1.3 Select

Recupera registros que satisfaçam a uma condição.

8.1.4 Insert

Insere um novo registro.

8.2 Operadores

Podemos usar o seguintes operadores:

- Aritiméticos + , - , * , /
- Comparação: = , >, <, <>, <=, =>
- Intervalo: BETWEEN....AND....
- IN e NOT IN
- LIKE
- IS NULL
- AND, OR e NOT
- Quantidade: ALL, SOME, ANY
- || Concatenação

8.3 Expressões

Podemos construir uma declaração SQL usando as seguintes expressões:

8.3.1 From

Especifica o nome(s) da(s) tabela(s) que originarão os dados.

8.3.2 Where

Especifica uma condição.

8.3.3 Group By

Separa os registros pelo grupo especificado.

8.3.4 Having

Estabelece a condição que cada grupo deve atender.

8.3.5 Order By

Especifica a ordenação dos registros selecionados. Pode-se usar ainda o parâmetros ASC(Ascendente) e DESC (Descendente)

8.3.6 Distinct

Retira as repetições

8.4 Funções

Podemos ainda, especificar as seguintes funções:

- COUNT (Contador);
- AVG(Média);
- FIRST(Primeiro);
- LAST(Último);
- MIN(Mínimo);
- MAX(Máximo);
- SUM(Soma);
- UPPER(Maisculo);
- LOWER(Minusculo);
- TRIM(Remove repetições de um caracter espcificado da esquerda ou direita de uma string);

8.5 Exemplos

1)
```
Select *
From Clientes
```

Seleciona todos(*) os campos de todos registros da tabela clientes.

2)
```
Select Codigo,Nome
From Clientes
Where Codigo > 10 And Codigo < 200
```

Seleciona os campos Código e Nome da tabela Clientes para os registros que tenham os campo Código > (Maior) que 10 e <(Menor) que 200.

3)
```
Select *
From Clientes
Group By Cidade
```

Seleciona todos os campos e registros da tabela Clientes agrupada pelo campo Cidade.

4)
```
Select *
From Clientes
Order By Codigo
```

Seleciona todos os campos e registros da tabela Clientes ordenada pelo campo Codigo.

```
5)    Select *
      From Fornecedores, Produtos
      Where Fornecedores.Codigo = Produtos.Codigo
```

Seleciona todos os campos e registros das tabelas de Fornecedores e Produtos que tenham o campo codigo de Fornecedores igual ao campo Codigo de Produtos.

```
6)    Select *
      From Clientes
      Where Nome Like "S*"
```

Seleciona todos os campos e registros da tabela de Clientes cujo o campo Nome comece pela letra S.

```
7)    Update Funcionarios
      Set Salario = Salario * 1.2
```

Atualiza o campo Salario de todos os registros da tabela de Funcionarios para o conteúdo atual multiplicado por 1.2 (aumento de 20%).

```
8)    Update Funcionarios
      Set Salario = Salario * 1.2
      Where Cargo = "Diretor"
```

Atualiza o campo Salario de todos os registros da tabela de Funcinarios que campo Cargo seja igual a "Diretor" para o conteúdo atual multiplicado por 1.2 (aumento de 20%).

```
9)    Delete  From Produtos
      Where Codigo > 5 And Codigo < 20
```

Apaga todos os registros da tabela Produtos para Codigo >(maior) que 5 e < (menor) que 20.

8.6 Construindo uma Consulta Dinamicamente

Procedimento para criar construir uma consulta em SQL dinamicamente.

```
Query1.Close; {Fecha a query}
Query1.SQL.Clear; {Limpa o conteúdo da propriedade SQL}
Query1.SQL.Add('Select *'); {Adiciona}
Query1.SQL.Add('From Country');
Query1.SQL.Add('Where Name = "Argentina" ');
Query1.Open; {Abre e executa a Query}
```

8.6.1 Passando Parâmetros

Conteudo da Propriedade SQL da Query1:

```
      Select *
      From Country
      Where Name Like :NomeRegiao

Procedure TForm1.ButtonClick(Senderl: Tobject);
Begin
   Query1.close;
   Query1.Params[0].AsString := edit1.text;
   Query1.open;
end;
```

Para NomeRegiao e passado o conteúdo de Edit1.text.

Obs.1:

```
      Query1.open;  =  Query1.ExecSQL;
```

Obs.2:

Para caregar de um arquivo texto:

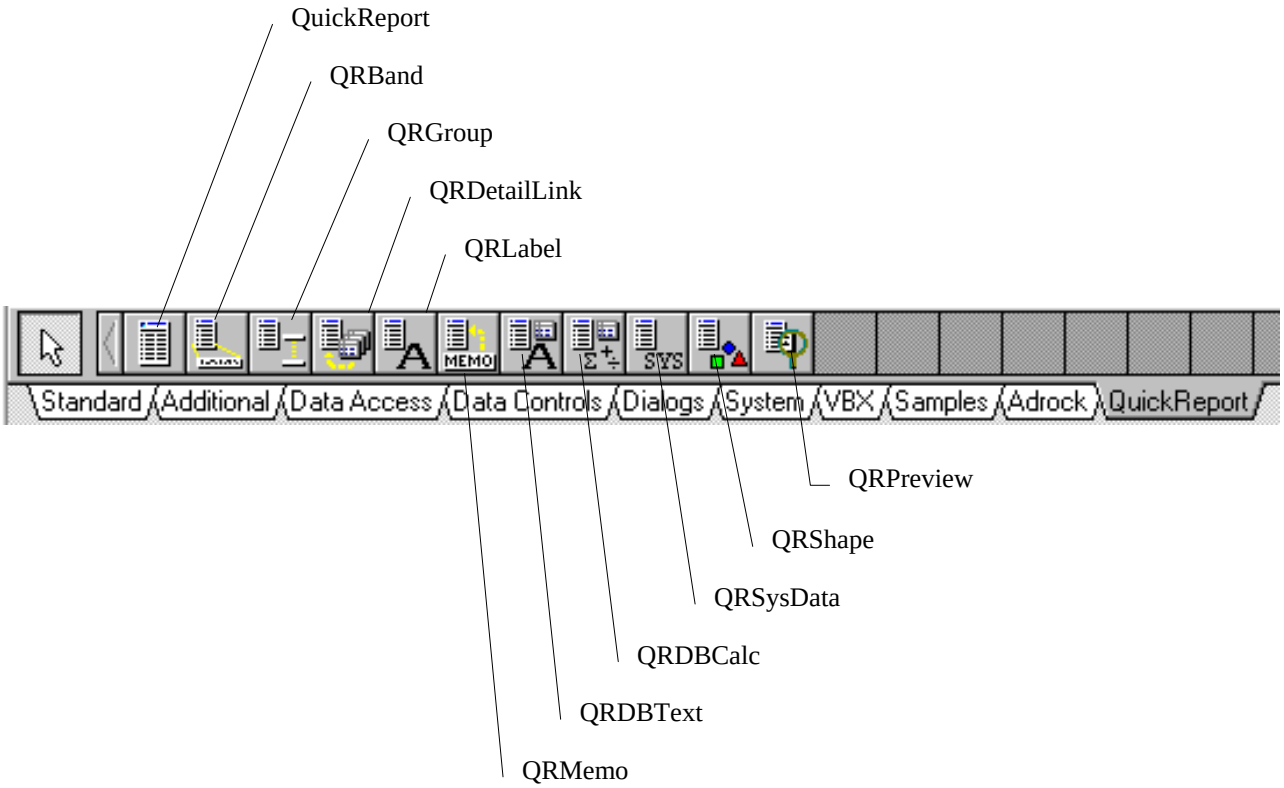
```
      Query1.SQL.LoadFromFile('C:\MYQUERY.TXT');
```

9. Construindo Relatórios com o QuickReport

O Presente trabalho tem como objetivo demostrar os procedimentos para criar relatórios em programas feitos em Delphi utilizando o QuickReport. O Delphi 1.0 já possui um gerador de relatórios o ReportSmith que perde em desempenho e desenvoltura em relação ao QuickReport, além de necessitar de sua instalação prévia quando da sua utilização. Por ser um programa a parte torna-se muito lento em relação a o QuickReport que é uma biblioteca de componentes que passa a fazer parte da aplicação quando da sua utilização após a compilação.

9.1 QuickReport básico

QuickReport é um gerador de relatórios por banda. Seu relatório é construído em cima de vários tipos de banda, onde são colocados componentes visíveis como campos de banco de dados, rótulos, imagens e outros componentes imprimíveis.



Objetos do QuickReport na Barra de Ferramentas do Delphi.

9.1.1 Componentes do QuickReport.

9.1.1.1 QuickReport

Este é o componente principal do QuickReport, pois ele transforma a form padrão em um relatório.

9.1.1.2 QRBand

Você constrói seus relatórios com diferentes QRBands e colocando componentes visíveis de controle do QuickReport sobre ela. Existem diferentes tipos de bandas, como banda de detalhes, banda de cabeçalho, banda de rodapé e outras.

9.1.1.3 QRGroup

Existe duas maneiras de imprimir relatórios mestre/detalhes com o QuickReport. Um é usar um componente TQuery ligado com um Query(Consultas) e QRGroups para grupos de datas, a outra maneira é usar o componente DetailLink combinado com bandas DetailGroup. Cada método tem o

mesmo resultado. Se você tiver necessidade de relatórios mestres/detalhes com mais de duas camadas então use QRGroup e TQuery terá uma probabilidade de ser melhor desempenho.

9.1.1.4 QRDetailLink

QRDetailLink é o segundo caminho para criar relatórios mestre/detalhes no QuickReport. Tipicamente você usaria QRDetailLink com TTables (ou TQueries) quando você tem somente dois níveis de conexão. A força do QRDetailLink contudo, é que você pode ter muitas tabelas detalhes.

9.1.1.5 QRLabel

O Componente QRLabel é usado para mostrar textos em um relatório. A propriedade Caption(Título) pode ser alterado em alguns eventos durante a preparação do relatório. Se você necessitar um texto para saída de um campo de dados use o componente QRDBText.

9.1.1.6 QRMemo

O Componente QRMemo é utilizado montar linhas que precisam ser composta por vários itens de dados ou textos que precisem ser digitados para compor o relatório.

9.1.1.7 QRDBText

O Componente QRDBText é uma controle de texto de dados cliente. Texto é conectado com um campo de banco de dados.

9.1.1.8 QRDBCalc

Este componente automatiza processos de soma e contagem de campos de dados. Em adição para esta propriedade o QRDBCalc tem todas a propriedades e eventos herdados do QRDBText.

9.1.1.9 QRSysData

O Componente QRSysData mostra várias informações do sistema. Em adição para as propriedade do QRSysData tem-se todos os componentes que o QRLabel tem, exceto a propriedade Caption.

9.1.1.10 QRShape

QRShape é usado para mostrar linhas simples e figuras em um relatório.

9.1.1.11 QRPreview

QRPreview é usado para criar uma form de pré visualização. Este componente mostra todas as saídas de uma visualização do preview atual. Para chamar sua própria form de pré visualização ao invés de usar uma form padrão com QuickReport atribuindo o evento QRPrinter.OnPreview. Neste evento você traz sua próprio form a qual deve conte o componente QRPreview. Você não pode selecionar que página para visualizar usando a propriedade PageNumber.

9.2 Criando Relatórios

QuickReport pode criar vários tipos de relatórios, de fato com um breve conhecimento de Delphi e um pouco de criatividade pode ser usado para criar qualquer tipo de relatório.

Lembrar que ainda que se os relatórios são criados como as forms, mas não com a intenção de ser mostrado na tela como forms. Em uma form relatório deve ser incluído um componente TQuickReport, e você usa neste componente os métodos PRINT e PREVIEW para criar seu relatório. Não tente usar em um relatório os métodos de form SHOW ou SHOWMODAL. Uma Form relatório nunca deverá ser em sua aplicação o form principal.

9.2.1 Relatório de lista Simples

Criar um relatório de lista simples do conteúdo de uma tabela ou uma consulta é muito fácil. Siga as instruções passo a passo a seguir para criar um relatório listando todos os clientes de DBDemos. A tabela clientes está incluída no Delphi.

1. Criar um novo projeto com um form e sobre um único form. Esta é sua principal forma de aplicação. Salvar unidade do form como MAINFORM e a aplicação como QRTEST.

2. Adicionar um form em branco para o projeto . Estabelecer o nome para a propriedade MYREPORT. Estabelecer a largura(Width) da form para mais ou menos 800. O atual número não é crítico.

3. Adicionar uma TTable para o componente form. Nomear o componente CustomerTable, estabelecer a propriedade DataBaseName para DBDEMOS e a propriedade TableName para COSTUMER. Estabelecer a propriedade ativa para TRUE.

4. Adicionar um componente TDataSource para o form. Nomear o CustomerDS e estabelecer a propriedade DateSet para CustomerTable.

5. Adicionar um componente TQuickReport para o form. Nomear o componente REP. Estabelecer a propriedade DataSource para CustomerDS.

6. Adicionar um componente QRBand para o form. Estabelecer a propriedade BandType para rbDetail.

7. Colocar um componente TQRDBText sobre o band. Estabelecer a propriedade DataSource para CustomerDS e a propriedade DataField para CompanyName.

8. Salvar o form como REPORT.

9. Voltar à MAINFORM (form principal) e adicionar a cláusula USES MYREPORT no topo do código fonte.

10. Dar um duplo clique no botão da form principal. Isto irá mostrar o evento ONCLICK do botão. Adicionar o seguinte código para o evento : MyReport.Rep.Preview;

11. Rode seu projeto .

Você criou agora um projeto com relatório. Seu programa pode agora: apresentar, imprimir, salvar e carregar relatórios!

9.2.2 Criando um relatório composto

O QuickReport pode imprimir relatórios mestre e detalhes muito facilmente. Estes relatórios podem ser imprimidos juntamente com componentes TQuery (consulta) e de componentes TTable (tabela) conectadas com MasterSource (fonte mestre)/propriedades MasterField(campo mestre). Em acréscimo o QuickReport pode criar mestres complexos/relatórios detalhes com registros de um número ilimitado de tabelas detalhes.

Você poderá ver como criar ambos os tipos de relatórios a seguir. Relatórios mestre/detalhes usando o componente TTable (tabela).

Criando um mestre/detalhes com o componente TTable (tabela) deixa o relatório muito mais rápido e flexível. Em adição você pode incluir dados detalhes de muitas outras tabelas detalhes que você queira. Você pode criar um relatório de uma relação resumida de clientes de todos os seus clientes (master data) e para cada cliente da lista todas as ordens que o cliente tenha feito (detail data), todos os seus contatos pessoais com seus clientes (detail data), todos os produtos de uma categoria que o cliente esteja interessado (detail data) e muitas outras tabelas que você quiser unir com a tabela clientes. Cada uma destas tabelas detalhes podem conter a sua banda cabeçalho, banda detalhe e banda rodapé.

Para criar relatórios mestre usando componentes TTable siga as instruções a seguir. Você vai ter a necessidade de olhar o relatório MDREP do projeto demonstração. Ele e um relatório mestre/detalhe que você pode usar como modelo.

Crie um relatório padrão seguindo o exemplo deste capítulo Criando Relatórios.

1. Alterar a propriedade MyReport.ReportType para qrMasterDetail.

2. Acrescentar um novo componente TTable para o relatório. Nomear ele OrderesTable, colocar a propriedade DataBaseName para DBDEMOS, a propriedade TableName para Orders, o MasterSource para CustomerDS e na caixa de dialogo MasterFields conecte o índice secundário (custno) para Customer. Custno. Coloque a propriedade Active da TTable para TRUE.

3. Adicione um componente TDataSource para o relatório. Nomeie ele OrdersDS e coloque a propriedade DataSet para OrdersTable

4. Adicione um TQRBand para o relatório. Coloque a propriedade BandType para rbSubdetail. Nomeie ele para OrdersDetailBand.

5. Adicione um componente TQRDetailLink para o relatório. Coloque a propriedade DataSource para OrdersDs. Coloque a propriedade DetailBand para OrderDetailBand. Coloque a propriedade Master para REP.

6. Coloque um componente TQRDBText sobre o OrdersDetailBand. Coloque a propriedade DataSource para OrdersDS e a Propriedade DataField para OrderNo.

7. Duplo clique no componente QuickReport para visualizar o relatório.

Se você desejar mais tabelas de detalhes você deverá repetir os passos 3 a 7 para cara tabela de detalhes que você ira incluir. Para mudar a ordem de impressão reorganize a ordem de criação dos componentes TQRDetailink. As Tabelas de detalhes são imprimidas na ordem correspondente com a criação dos componentes TQRDetailink.

10. Arquivos de Inicialização

10.1 O Objeto TInifile

Para manipular arquivos .INI, o Delphi oferece o objeto Tinifile. O Objeto TIniFile, declarado na unit Inifiles, permite que sua aplicação escreva e leia um arquivo .INI.

Para usar o objeto TIniFile, você deve adicionar a unit Inifiles à cláusulas uses de sua aplicação.

10.2 Criando um Arquivo de Configuração

Use o método Create para criar memória para um objeto do tipo TIniFile, passando como parâmetros o nome do arquivo a ser criado.

```
NomeObjeto := TIniFile.Create('NomedoArquivo.INI');
```

Pode ser informado o junto ao nome o caminho de onde se encontra o arquivo, mas o padrão é armazenar o arquivo .INI no subdiretório do Windows.

Para criar as seções, as entradas e os valores das entradas, utilize os seguintes métodos:

- `WriteBool` - Para criar um valor booleano;
- `WriteInteger` - Para criar um valor inteiro;
- `WriteString` - Para criar uma string.

Use o método Free para liberar a memória utilizada pelo objeto TIniFile criado:

```
NomeObjeto.Free;
```

O exemplo abaixo cria, ao clicar no botão Button1, um arquivo .INI chamado NovoJogo, com duas seções: Opcoes (com duas entradas) e Configuracao (com uma entrada):

```
Procedure TForm1.Button1Click(Sender:TObject);
Var
   IniJogo : TIniFile;
Begin
   IniJogo := TIniFile.Create('NOVOJOGO.INI');
   IniJogo.WriteBool('Opcoes','Som',True);
   IniJogo.WriteInteger('Opcoes','Nivel',3);
   IniJogo.WriteString('Configuracao','Nome','Oliveira');
   IniFile.Free;
End;
```

O arquivo de configuração NOVOJOGO.INI é armazenado no diretório \Windows, e possui o seguinte formato:

```
[Opcoes]
Som=1
Nivel=3

[Configuracao]
Nome=Oliveira
```

Quando você for distribuir sua aplicação, não esqueça de oferecer o arquivo .INI. Outra opção é criar o arquivo .INI através de um programa de instalação do seu aplicativo, oferecendo ao usuário a possibilidade de personalizar o arquivo .INI ja na instalação.

10.3 Lendo o Arquivo de Configuração

Quando seu aplicativo for executado, deve ler o arquivo .INI existente. Isto pode ser feito, por exemplo, no evento OnCreate do Formulário, ou na seção Inicialization.

Para ler um arquivo .INI, utilize os métodos:

- `ReadBool` - Para ler uma entrada com valores booleanos;
- `ReadInteger` - Para ler uma entrada com valores inteiros;
- `ReadString` - Para ler uma entrada com valores string.

Atribua o resultado da leitura a variáveis globais (criadas na cláusula var da seção interface da unit do formulário principal), que podem ser utilizadas por toda a aplicação, quando necessário. Não

esqueça de adicionar a unit do formulário principal a cláusula uses da seção implementation quando for utilizar as variáveis.

O exemplo abaixo lê as entradas do arquivo NOVOJOGO.INI, mostrando para o usuário em Labels:

```
Var
   Form1 : Tform1;
   Vsom : Boolean;
   Vnivel : Integer;
   Vnome : String;
   IniJogo : Tinifile;

Implementation

{$R *.DFM}

Procedure TForm1.FormCreate(Sender:TObject);
Begin
   Label1.Caption := 'Nível'+InttoStr(Vnivel);
   If Vsom then
      Label2.Caption := 'Som Ligado');
   else
      Label2.Caption := 'Som Desligado');
   Label3.Caption := 'Nome : '+ Vnome;
End;

Initialization
Begin
   IniJogo := TIniFile.Create('NOVOJOGO.INI');
   Vsom := IniJogo.ReadBool('Opcoes','Som',False);
   Vnivel := IniJogo.ReadInteger('Opcoes','Nivel',-1);
   Vnome := IniJogo.ReadString('Configuracao','Nome','');
   IniFile.Free;
end;
```

10.4 Modificando um Arquivo .INI

Você pode utilizar os métodos WriteString, WriteBool e WriteInteger para modificar as entradas de um arquivo .INI já existente. Por exemplo o formato de mode estabelecido no WIN.INI, seção [Intl], entrada sCurrency, para R$:

```
Procedure TForm1.Button1Click(Sender:TObject);
Var
   IniMoeda : TIniFile;
Begin
   IniMoeda := TIniFile.Create('WIN.INI');
   IniMoeda.WriteString('Intl','sCurrency','R$');
   IniMoeda.Free;
End;
```

10.5 Outras Operações com Arquivos .INI

Através do método ReadSection, você lê todas as entradas de um seção de um arquivo .INI para um objeto da classe Tstring, como ListBox, ComboBox ou Memo:

```
ReadSection(Const Seção:String; ListaEntradas: TStrings);
```

A constante Seção determina a seção que será lida. O parâmetro ListaEntradas corresponde ao objeto do tipo TString que armazenará as entradas (por exemplo, ListBox1.Items).

Através do método ReadSectionValues, você lê todas as entradas e seus valores de uma seção do arquivo .INI para um objeto da classe TString, como ListBox, ComboBox ou Memo:

```
ReadSectionValues(Const Seção:String; ListaEntradas: TStrings);
```

A constante Seção determina a seção que será lida. O parâmetro ListaEntradas corresponde ao objeto do tipo TString que armazenará as entradas (por exemplo, memo1.lines). Você pode usar a propriedade Values de uma string ou lista de strings para acessar uma string específica na lista de string.

Para apagar uma seção inteira de uma arquivo .INI use o método EraseSection.

A propriedade FileName contém o nome do arquivo .INI que está sendo manipulado pelo objeto TIniFile.

10.6 Manipulação de Arquivos

Alguns comandos podem ser uteis para manipular o arquivo .INI :

10.6.1 GetDir

Retorna o diretório corrente do drive especificado.
D pode assumir os seguintes valores:

0 - Default
1 - A
2 - B
3 - C

Sintaxe:
GetDir(D: Byte; var S: String);

10.6.2 FileExists

A função FileExist retorna True se o arquivo especificado existe.

Sintaxe:
FileExists(const FileName: string): Boolean;

10.6.3 DeleteFile

A função DeleteFile apaga o arquivo especificado do disco. Se o arquivo não poder ser apagado ou não existir retorna falso.

Sintaxe:
DeleteFile(const FileName: string): Boolean;

11. Criando HELP

11.1 Visão Geral

Inicialmente você cria os tópicos do help em um arquivo texto, através de um editor de textos convencional (word).

A parte, cria outro arquivo texto (HPJ) contendo informações sobre o arquivo de Help para o compilador.

Por fim, utiliza um programa específico (HC31) para compilar o arquivo de Help, gerando o arquivo de extensão .HLP.

O programa Win Help, gerenciador de Help do Windows, gerencia a visualização do arquivo .HLP.

11.2 Escrendo os tópicos

Para criar o arquivo Help, utilize um editor de textos que permita a gravação em um arquivo RTF, como o Word.

Para escrever os tópicos do Help, abra um novo documento no editor de texto e:

- Escreva o título do tópico, você pode formatar este título para destacar do resto do texto;
- Escreva o texto do tópico;
- Crie uma crebra de página;
- Escreva o novo tópico.

11.3 Inserindo notas de rodapé

Você precisa inserir em seus tópicos notas customizadas para que o Win Help possa identificar e criar os quadros de diáloga padrão, como o quadro procura.

: String de contexto, identifica o tópico. O usuário não a vê, mas será usada para links entre tópicos.

$: Título. Aparece como título do tópico da caixa Procura. Normalmente é igual ao atributo do tópico.

K : Palavras chaves. Aparecem na lista Procura. Você pode escolher mais de uma palavra chave, separando-as como o ponto e vírgula (;).

+ : Sequência de folheamento. Determina a ordem dos tópicos quando o usuário folheia o help. O identificador pode ser um membro (01) ou um nome de grupo seguido de um sinal de dois pontos (:) e um membro (doc:01).

11.4 Adicionando um Hotspot ao tópico

Um hotspot é uma palavra ou gráfico que se clicado leva a um outro tópico.

Para criar um hotspot:

- Digite o texto ou insira o gráfico que você quer que seja o hotspot;
- Selecione o texto e aplique o formato duplo sublinhado a ele;
- Imediatamente após o texto ou gráfico hotspot, digite a string de contexto do tópico do destino;
- Selecione a string de contexto, e aplique o fundo de texto oculto a ele.

Depois de compilado aparecerá sublinhado, em verde.

11.5 Criando um hotspot para uma janela pop-up

Para criar um hotspot para um tópico do mesmo arquivo de help, mostrando este tópico em uma janela pop-up.

- Formate a palavra que será o hotspot com um sublinhado simples;
- Imediatamente após esta palavra digite o nome da string de contexto que representa o tópico, mudando seu formato para oculto.

11.6 Se for usar um tópico de outro arquivo de help:

- Formate a palavra que será o hotspot com sublinhado simples;
- Imediatamente após a palavra, digite o nome da string de contexto que representa o tópico, mudando seu formato para oculto;
- Digite um @;
- Digite o nome do arquivo de help que contém o tópico a ser acessado.

Ex.: Criando um novo documentoCriarDoc@outrohlp.hlp

11.7 Escrevendo Arquivo de Projeto:

O arquivo de projeto é um arquivo texto contendo informações que o compilador de Help usa para construir o arquivo de Help. O arquivo de projetopode incluir muitas instruções que controlam aspectos do arquivo help.

Para escrever um arquivo de projeto para o Help:

Crie um arquivo texto. Dê a ele o mesmo nome que você gostaria que fosse dado ao seu arquivo de help compilado, acrescentando a extensão .HPJ.

Adicione as seguintes linhas ao aruqivo (existem outras seções além das expostas abaixo).

```
[OPTIONS]
CONTENSTS =  String_Contexto
TITLE = Título
COMPRESS = Nível_de_Compressão
ERRORLOG = Nome_do_arquivo_log
[CONFIG]
Browse Buttons()
[FILES]
NomeArquivoRTF_1
NomeArquivoRTF_2
NomeArquivoRTF_3
```

Onde:

String_Contexto: É a string de contexto do tópico de conteúdos. Esta linha não é requerida. Se você não inclui-la, o Winhelp usará o primeiro tópico do primeiro arquivo como tópico de conteúdos.

Título: É o nome que aparece na Barra de Título da janela de Help. Não coloque entre aspas.

Nível_de_compressão: Determina quanto arquivo é comprimido quando ele é compilado. Esta linha não é requerida. Se você a deixar em branco, compilará o arquivo sem compressão. Os valores podem ser:

Valor	Compressão
False	Sem compressão
Medium	Média compressão
High	Alta compressão

Nome_do_arquivo_log: É o nome de um arquivo onde os erros ocorridos durante a compilação sã armazenados. A linha não é requerida. Se você não determinar o arquivo, os erros serão mostrados na tela.

Se você incluir a linha BrowseButtons() os botões >> (próximo) e << (anterior) são adicionados à barra de botões do Help. Esta linha não é requerida.

11.8 Compilando o arquivo de Help:

Há dois compiladores de Help para Windows 3.1:

- HCP.EXE
- HC31.EXE , HC31.ERR

Quando você compilar o arquivo de Help os arquivos abaixo devem de preferência estar no mesmo diretório:

- Todos os RTF
- O compilador de Help (HC31.EXE)
- O arquivo de recursos contendo as mensagens de erro do compilador de Help (HC31.ERR)
- O arquivo de projeto (.HPJ)
- Qualquer Bitmap utilizado.

Para compilar o arquivo de Help no prompt do dos:

nome_do_compilador nome_arquivo_de_projeto

11.9 Inserindo gráficos em um tópico:

Você pode inserir gráficos como si,mples ilustrções, como um Hotspot ou como hipergráfico (gráfico contendo diversos hot spots).

Os bitmaps devem se encontrar no diretório do arquivo de projeto.

No arquivo RTF deve ser inserido a seguinte referência:

```
{bmc bitmap.bmp} ou
{bml  bitmap.bmp}
```

11.10 Criando um Hot Spot para uma janela secundária

Para criar um hot spot para um tópico mostrado em uma janela secundária:

- Formate a palavra que será o hot spot com um sublinhado duplo.
- Imediatamente após esta palavra, digite o nome da string do contexto que representa o tópico, mudando seu formato para oculto.
- Digite um >
- Digite o nome da janela secundária

O nome, tamanho e posição da janela secundária precisam ser definidos na seção [WINDOWS] do arquivo projeto do Help.

```
[WINDOWS]
nome_da_janela  =  "título",  (x,  y,  largura,  altura),  tamanho,
(clientRGB), (nonscrollRGB), (ftop)
```

Por exemplo:

```
[WINDOWS]
Jan1 = "Exemplo", (123,123,256, 256), 0, (0,255,255), (255,0,0)
```

11.11 Tornando o arquivo de Help sensível ao contexto

Após criar o arquivo de Help, você pode utilizá-lo na sua aplicação em Delphi , ligando partes de sua aplicação (como itens de menu, botões, etc...) a tópicos do arquivo de Help.

Para ligar o arquivo de Help a Aplicação:

- Copie o HLP para o diretório do Projeto
- Entre na aplicação em Options/Project.
- Na caixa HelpFile, entre com o nome do arquivo de Help (.HLP) que será ligado a aplicação

O Delphi liga um componente a um tópico do Help através de um número de contexto. Este número de contexto deve ser atribuído 'a string de contexto do tópico, na seção MAP do arquivo de Porjeto. A seção [MAP] mapeia todos os números de contexto, associando-os as strings de contexto respectivas.

Por exemplo:

```
[MAP]
Abrir_doc 001
Criar_doc 002
```

11.12 Chamadas ao arquivo de Help

Chamar o tópico do conteúdo:

```
Application.helpcommand(help_contents,0)
```

Chamar um tópico específico:

```
Application.helpcommand(help_contents,xx)
```

xx = Índice da string de contexto definido na seção [MAP] do arquivo de projeto do Help.

Fechar o arquivo de Help:

```
Application.HelpCommand(help_quit,0);
```

12. Exceções

12.1 A Estrutura Try…Finally…End

Seu código precisa garantir que, mesmo ocorrendo um erro, os recursos alocados sejam desalocados. Entre estes recursos estão: arquivos, memória, recursos do Windows, objetos.

Para garantir a desalocação dos recursos, usamos a estrutura abaixo:

```
{Aloca os recursos}
Try
      {Comandos que usam os recursos}
Finally
      {Libera os recursos}
end;
```

A aplicação sempre executará os comandos inseridos na parte Finally do bloco, mesmo que uma execeção ocorra. Quando um erro ocorre no bloco protegido, o programa pula para a parte finally, chamada de código limpo, que é executado. Mesmo que não ocorra um erro, estes comandos são executados.

No código a seguir, alocamos memória e geramos um erro, ao tentarmos uma divisão por 0. Apesar do erro o programa libera a memória alocada:

```
Procedure TForm1.Button1Click(Sender:TObject);
Var
   Apointer : Pointer;
   AnInteger, Adividend : Integer;
Begin
   Adividend := 0;
   GetMem(Apointer, 1024);
   Try
       Aninteger := 10 Div Adividend;
   Finally
      FreeMem(Apointer, 1024);
   End;
End;
```

12.2 A Estrutura Try…Except…End

Um tratamento de exceção é um código que trata erros que ocorrem dentro de blocos protegidos.

Para definir um tratamento de exceção, utilize a seguinte construção:

```
Try
      {Comandos que você deseja proteger}
Except
      {Comandos de tratamento de erros}
end;
```

A aplicação irá executar os comandos na parte except somente se ocorrer um erro. Se na parte try você chamar uma rotina que não trata erros, e um erro ocorrer, ao voltar para este bloco a parte except será executada. Uma vez que a aplicação localiza um tratamento para exceção ocorrida, os comandos são executados, e o objeto exceção é destruído. A execução continua até o fim do bloco.

Dentro da parte except definimos um código a ser executado para manipular tipos específicos de exceção. Por exemplo, o código abaixo trata o erro de divisão por zero, através da exceção EDivByZero:

```
Function Divisão (Soma, Numero : Integer) : Integer;
Begin
   Try
       Result := Soma div Numero;
   Except
      on EDivByZero do Result 0;
   End;
End;
```

A palavra reservada *on* define respostas para uma exceção. *On* está sempre junto de *do*, para manipular a exceção.

Para ler informações específicas sobre o erro ocorrido, você usa uma variação da estrutura *on…do*, que provê uma variável temporária que engloba a exceção. Nesse caso, você poderá criar seu próprio quadro de mensagem contendo a string da mensagem da exceção:

```
Try
   {Comandos}
Except
   on E:EInvalidOperation do
       MessageDlg('Ígnorando a exceção : '+E.Message,
        mtinformation,[mbOk],0);
End;
```

Onde a variável temporária *E* é do tipo *EInvalidOperation.*

Você pode prover um tratamento padrão de erro para tratar exceções que não tem tratamentos especificados. Para isto, adicione uma parte *else* na parte except do bloco:

```
Try
   {Comandos}
Except
   on EPrimeiroTipo do
      {Código especifico para o primeiro tipo de erro]
   on ESegundoTipo do
      {Código específico para o segundo tipo de erro}
   Else
     {Código padrão de tratamento de erros}
End;
```

Além disso, você pode utilizar as exceções genéricas para tratar um erro, em vez de uma exceção específica. Por exemplo, se você quer tratar um erro relacionado a uma operação com inteiros, mas não sabe exatamente o erro, poderá utilizar a exceção *EIntError*, que é a exceção genérica da qual derivam outras exceções relacionadas a inteiros:

```
Try
   {Comandos}
Except
   on EIntError do
   {Código de tratamento de erros}
End;
```

12.3 Exceções Silenciosas

Você pode definir exceções que não mostram um quadro de mensagem para o usuários quando aparecem. São chamadas exceções sileciosas.

O caminho mais curto para criar esta exceção é através da procedure *Abort.* Esta procedure automaticamente gera uma exceção do tipo *EAbort*, que abortará a operação sem mostrar uma mensagem.

O exemplo abaixo aborta a operação de inclusão de itens em um ListBox quando tentamos inserir o terceiro elemento:

```
Begin
   For i := 1 to 10 do
      begin
          ListBox1.items.Add(IntToStr(i));
          If i = 3 then
            Abort;
       end;
End;
```

13. Dicas

13.1 Criação de MDIChild

```
procedure TForm1.Teste1Click(Sender: TObject);
begin
  if formexiste(Form2)= false then
     begin
       Screen.Cursor := CrHourGlass; {Cursor ampulheta}
        Form2 := TForm2.Create(Self);
     end;
  else
    begin {Traz a janela para frente}
      Form2.WindowState := WsNormal;
      Form2.BringToFront;
      Form2.Setfocus;
    end;
end;
```

Obs1.:

```
interface

uses {Declaração da unit da form a ser criada}
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
  Controls, Forms, Dialogs, Menus, unit2, StdCtrls;
```

Obs2.:

```
public
  { Public declarations }
  Form2 : Tform2; {Declaração da janela a ser criada}
end;
```

Obs3.:

```
{Na form Filha criada}
procedure TForm2.FormCreate(Sender: TObject);
begin
  Screen.Cursor := CrDefault; {Cursor na forma normal}
end;

procedure TForm2.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Action := cafree; {Libera da memória a janela}
end;
```

13.2 Form Existe

```
Function TForm1.Formexiste(Nomejanela:TForm):boolean;
var
  I : integer;
begin
  Formexiste := false;
  for i := 0 to componentcount - 1 do
     if Components[i] is TForm then
        if TForm(Components[i])= nomejanela then
           Formexiste := true
end;
```

Obs.:

```
public
  { Public declarations }
  Function Formexiste(nomejanela:Tform):boolean;
end;
```

13.3 Criação ShowModal

```
procedure TForm1.Teste2Click(Sender: TObject);
begin
  try
    Application.CreateForm(TForm3, Form3);
    Form3.showmodal;
  Finally
    Form3.free;
  end;
end;
```

Obs.:

```
interface

uses {Declaração da unit da form a ser criada}
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
  Controls, Forms, Dialogs, Menus, unit3, StdCtrls;
```

13.4 Fechar Todas

```
procedure TForm1.Teste31Click(Sender: TObject);
Var
  I: Integer;
begin
  for I := MDIChildCount - 1 downto 0 do
    MDIChildren[I].close;
end;
```

Obs.: Pode ser utilizada para minimizar todas e maximizar todas.

13.5 Relógio e Teclado

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  {Atualiza a hora do Relógio }
  label1.caption:=timetostr(now)+' - '+Datetostr(now)+ ' ';

  {Atualiza o Painel de Teclas}
  if (GetKeyState(vk_NumLock) and $01) <> 0 then
     Label2.Caption := '  Num  ' {Tecla NumLock}
  else
     Label2.Caption := '';
  if (GetKeyState(vk_Capital) and $01) <> 0 then
     Label3.Caption := '  Cap  ' {Tecla CapsLock}
  else
     Label3.Caption := '';
  if (GetKeyState(vk_Insert) and $01) <> 0 then
     Label4.Caption := '  Ins  ' {Tecla Insert}
  else
     Label4.Caption := '';
end;
```

13.6 Informações Sistema

```
procedure TForm2.FormCreate(Sender: TObject);
type
  FlagSet = set of 120..135;
var
  Flgs : FlagSet;
  L : Longint;
  s : String;
begin
image1.picture.graphic := application.icon;
titulo.caption := application.title;
L := GetVersion;

windows.caption := Format ('Windows %u.%.2u', [Lobyte(loword(L)),
Hibyte(loword(L))]);

dos.caption := Format ('DOS %u.%.2u',[Hibyte(Hiword(L)),
Lobyte(HIword(L))]);

L := GetWinFlags;
If L and WF_Enhanced = 0 then
   modo.caption := 'Modo Standard';
If L and WF_80x87 = 0 then
   coprocessador.caption := 'Ausente';

memoria.caption:=FormatFloat('#######',MemAvail DIV 1024) + ' KB
Livres';

{---------- Processador --------------}

l := GetWinFlags;
Flgs := FlagSet(Word(l));
s := 'unknown';
if 121 in Flgs then
   s:= '286';
if 122 in Flgs then
   s:= '386';
if 123 in Flgs then
   s:= '486 ou Pentium';
Processador.Caption:= s;

{--------- GDI Recursos -------------}

l := GetFreeSystemResources(gfsr_GDIResources);
gauge1.progress := l;

{--------- User Recursos -------------}

l := GetFreeSystemResources(gfsr_UserResources);
gauge2.progress := l;

{--------- Espaço Disco -------------}

l:= (DiskFree(0) div 10240) div (DiskSize(0) div 1024000);
  Label18.Caption:= IntToStr(DiskFree(0) div 1024000) +
                    +' MB ';
gauge3.progress := l;
end;
```

14. Bibliografia

Delphi por Adelize Generini de Oliveira
BookStre Editora
Fone: 048-222-1125
E-mail: Bookstore.livraria@omnibbs.ax.apc.org

Aplicações em Delphi por Adelize Generini de Oliveira
BookStore Editora
Fone: 048-222-1125
E-mail: Bookstore.livraria@omnibbs.ax.apc.org

Manipulando Banco de Dados com Delphi por Adelize Generini de Oliveira
BookStore Editora
Fone: 048-222-1125
E-mail: Bookstore.livraria@omnibbs.ax.apc.org

Delphi - Técnicas Avançadas por Adelize Generini de Oliveira
BookStore Editora
Fone: 048-222-1125
E-mail: Bookstore.livraria@omnibbs.ax.apc.org

Programação em Delphi - para Leigos por Neil J. Rubenking
Editora Berkeley
Fone: 011-832-8039

Aprendendo Delphi Avançado por Américo Damasceno Jr.
Editora Érica
Fone: 011-295-3066

Delphi Segredos e Soluções por Gary Cornell/Troy Strain
Editora Makron Books
Fone: 011-829-8604

Borland Delphi - Guia de Referência Inteligente por Marcelo Leão
Axcel Books do Brasil Ltda

Borland Delphi - Curso Básico e Rápido por Marcelo Leão
Axcel Books do Brasil Ltda

Manuais Borland Delphi 1.0 for Windows
Manuais Borland Delphi Professional 3.0 for Windows

Sites
Delphi Club
http://www.geocities.com/Colosseum/1870/

Delphi Brasil
http://www.geocities.com/SiliconValley/8314/delphi.htm

Delphi Super Page
http://sunsite.icm.edu.pl/delphi/index.html