DESENVOLVENDO APLICAÇÕES PARA BANCOS DE DADOS DESKTOP COM BORLAND DELPHI 6

Introdução

No início, programar em Windows era algo extremamente complicado e acessível apenas a programadores dispostos a investir muito tempo e fosfatos na leitura de pilhas de livros, intermináveis testes e análise de programas exemplos que mais confundem do que explicam. Mas porque era tão difícil fazer programas para Windows? Para começar, o Windows usa o conceito de GUI (Graphic User Interface – Interface Gráfica com o Usuário), que embora fosse muito familiar para usuários do Unix e do Mac OS, era novidade para usuários do DOS. O uso de um sistema GUI implicava em aprender vários conceitos que eram estranhos ao usuário de um sistema baseado em texto como o DOS. Para complicar um pouco mais, o Windows é um sistema multi-tarefa, e as aplicações são orientadas a eventos, o que implica em aprender um novo estilo de programação. Finalmente, o programador tinha que ter alguma familiaridade com as centenas de funções oferecidas pela API do Windows. Por tudo isso, programação em Windows era um assunto que costuma provocar arrepios nos programadores.

Felizmente as linguagens visuais chegaram para mudar esta situação. Foi só com elas que o Windows consegui cumprir sua promessa de ser um sistema amigável e fácil de usar também para os programadores, que sempre tiveram que pagar a conta da facilidade de uso para o usuário.

Entre as linguagens visuais que surgiram, nenhuma veio tão completa e bem acabada quanto o Delphi. Desde o início ele possuía um compilador capaz de gerar código diretamente executável pelo Windows, proporcionando uma velocidade de execução de 5 a 20 vezes maior que as linguagens interpretadas como o Visual Basic e Visual FoxPro que geravam executáveis Pcode que precisam de arquivos auxiliares de run-time. Além disso, o Delphi também possuía uma engine para acesso a diversos bancos de dados e um gerador de relatórios. O tempo de desenvolvimento de qualquer sistema foi reduzido a uma fração do tempo que seria necessário usando outras linguagens e o resultado é sempre muito melhor. É por isso que o Delphi fez e faz tanto sucesso no mundo inteiro, sempre ganhando prêmios como melhor ferramenta de desenvolvimento para Windows.

O objetivo principal de qualquer ferramenta de desenvolvimento ou linguagem de programação é a criação de aplicações. Determinadas linguagens ou ferramentas devido aos recursos que possuem são mais indicadas para a criação de aplicações comerciais, outras se destinam mais a aplicações científicas ou ainda para a criação de sistemas operacionais.

O Delphi é uma ferramenta RAD (Rapid Application Development – Desenvolvimento Rápido de Aplicações) criada pela Borland. É uma ferramenta de propósito geral, permitindo o desenvolvimento de aplicações tanto científicas como comerciais com a mesma facilidade e alto desempenho.
Integra-se facilmente com a API (Application Program Interface) do Windows, permitindo a criação de programas que explorem ao máximo os seus recursos, assim como os programas escritos em linguagem C/C++.

Possui um compilador extremamente rápido, que gera executáveis nativos (em código de máquina, não interpretado), obtendo assim melhor performance e total proteção do código fonte.

O Delphi é extensível, sua IDE (Integrated Development Environment – Ambiente de Desenvolvimento Integrado) pode ser ampliada e personalizada com a adição de componentes e ferramentas criadas utilizando-se o Object Pascal, a linguagem de programação do Delphi. Neste ambiente constroem-se as janelas das aplicações de maneira visual, ou seja, arrastando e soltando componentes que irão compor a interface com o usuário.

O Object Pascal é uma poderosa linguagem Orientada a Objeto, que além de possuir as características tradicionais das mesmas como classes e objetos, também possui interfaces (semelhantes às encontradas em COM e Java), tratamento de exceção, programação multithreaded e algumas características não encontradas nem mesmo em C++, como RTTI (Runtime Type Information). Assim como o C++, o Object Pascal é uma linguagem híbrida, pois além da orientação a objeto possui também uma parte da antiga linguagem estruturada (Pascal)

Devido ao projeto inicial da arquitetura interna do Delphi e da orientação a objeto, suas características básicas mantêm-se as mesmas desde o seu lançamento em 1995 (ainda para o Windows 3.1, pois o Windows 95 ainda não havia sido lançado), o que demonstra um profundo respeito com o desenvolvedor. Isto permite que uma aplicação seja facilmente portada de uma versão anterior para uma nova, simplesmente recompilando-se o código fonte.

Obs: Embora as características, teorias e exemplos abordadas aqui sejam sobre o Delphi 6 (última versão disponível), tudo pode ser aplicado em versões anteriores do Delphi, excetuando-se o caso da utilização de componentes e ferramentas introduzidos apenas nesta versão.

O Delphi é distribuído em três versões diferentes: Personal, Professional e Enterprise.

- A versão Personal consiste no Ambiente de Desenvolvimento, com compilador e debugador e os componentes básicos. É indicado para o aprendizado do Object Pascal e da utilização da IDE e para o desenvolvimento de pequenas aplicações. Esta versão não contém os componentes de acesso a bancos de dados. Atualmente foi liberada para download gratuito no site do fabricante (http://www.borland.com);
- A versão Professional tem todos os componentes da versão Personal mais componentes de acesso a banco de dados (BDE) Desktop e através de ODBC, para impressão de relatórios (Quick Report), elaboração de gráficos (TeeChart), uma versão especial do InstallShield para criar programas de instalação para distribuir seus aplicativos, componentes para criação de aplicativos para internet, componentes

CLX para criação de aplicativos multiplataforma (podem ser compilados no Kylix, o Delphi para Linux) e o código fonte dos componentes;

- A versão Enterprise contém os mesmos componentes da versão Professional mais ferramentas e drivers para acesso a bancos de dados cliente-servidor (Oracle, DB/2, Sybase, MS-SQL Server ou Interbase); o BizSnap, componentes para criação de aplicativos B2B (Business-To-Business) ; DataSnap para criação de aplicativos multicamadas; o Team Source, ferramenta para gerenciamento do desenvolvimento em equipe; componentes ADO para acesso direto (OLE/DB) ao MS-SQL Server.

CAPÍTULO 1

Princípios da Programação para Windows

Antes de começar a trabalhar com o Delphi, é importante ter algumas noções do que está envolvido na programação Windows e no Delphi em particular. Algumas coisas tornam a tarefa de programação no Windows (e ambientes baseados em eventos e interface gráfica) bem diferente de outros ambientes e das técnicas de programação estruturada normalmente ensinadas nos cursos de lógica de programação:

Independência do Hardware: No Windows, o acesso aos dispositivos de hardware é feito com intermédio de *drivers* fornecidos pelo fabricante do hardware, o que evita que o programador tenha que se preocupar com detalhes específicos do hardware. Como acontecia com a programação em DOS.

Configuração Padrão*:* O Windows armazena centralmente as configurações de formato de números, moeda, datas e horas, além da configuração de cores, livrando o programador de se preocupar com esses detalhes específicos.

Multitarefa: Antigamente, no DOS (não estamos falando do Prompt do MS-DOS), um programa geralmente tomava o controle da máquina só para si, e outros programas não rodavam até que o mesmo fosse fechado. Já no Windows vários programas são executados de maneira simultânea e não há como evitar isso.

Controle da Tela: No DOS geralmente um programa ocupa todo o espaço da tela, e o usuário via e interagia apenas com aquele programa. Já no Windows, todas informações mostradas e todas entradas recebidas do usuário são feitas por meio de uma *janela*, uma área separada da tela que pode ser sobreposta por outras janelas do mesmo ou de outros programas.

Padrões de Interface: No Windows, todos os elementos de interface aparecem para o usuário e interagem da mesma forma. Além disso, existem *padrões* definidos pela Microsoft que são recomendados para conseguir a consistência entre aplicativos. Falaremos de alguns deles no curso, mas a melhor forma de aprendê-los é analisar os aplicativos Windows mais usados do mercado.

Eventos e a Cooperação com o Sistema: Num programa criado para DOS (como os programas escritos em Clipper) ele é responsável pelo fluxo de processamento, temos que definir claramente não só que instruções, mas também em que ordem devem ser executadas, ou seja a execução segue uma ordem preestabelecida pelo programador, e o programa só chama o sistema operacional quando precisa de alguma coisa dele. Em Windows não é bem

assim. Nosso programa não controla o fluxo de processamento, ele responde e trata *eventos* que ocorrem no sistema. Existem muitos *eventos* que podem ocorrer, sendo que os principais são aqueles gerados pelo usuário através do mouse e do teclado. A coisa acontece mais ou menos assim: O usuário clica o mouse e o Windows verifica que aplicação estava debaixo do mouse no momento em que foi clicado. Em seguida ele manda uma mensagem para a aplicação informando que ocorreu um clique e as coordenadas do cursor do mouse na tela no momento do clique. A aplicação então responde à mensagem executando uma função de acordo com a posição do mouse na tela. É claro que o Delphi toma conta do serviço mais pesado e facilita muito as coisas para o programador. Detalhes como as coordenadas da tela em que ocorreu o clique, embora estejam disponíveis, dificilmente são necessários nos programas. Isso, como veremos, afeta radicalmente o estilo de programação e a forma de pensar no programa. A seqüência de execução do programa depende da seqüência de eventos.

Conhecendo o CD de Instalação do Delphi 6

Dentre as opções presentes no CD de Instalação, para este curso será necessário além da instalação do próprio Delphi (primeira opção) a instalação do InstallShield (última opção), que será responsável pela criação dos discos de instalação dos seus aplicativos.

Segue abaixo uma pequena explicação sobre as demais opções:

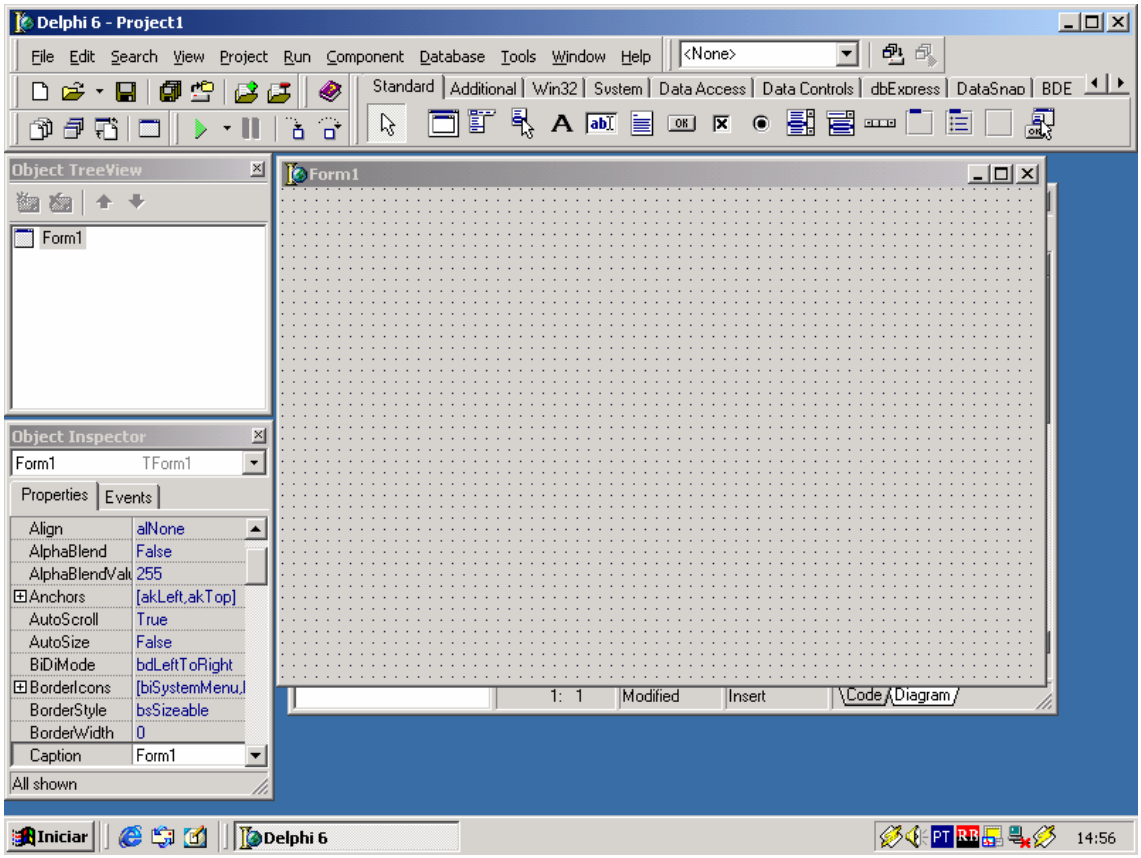
- TeamSource: utilizado em grandes projetos, para permitir o desenvolvimento em equipe, controlando versões e o acesso ao código fonte;

- InterBase 6.0 Server: Servidor de Banco de Dados SGDB da Borland, pode ser instalado tanto em Windows NT/2000/XP quanto no Windows 95/98/Me. Existem também versões para Novell, Solaris e Linux;
- InterBase 6.0 Desktop Edition: Cliente do InterBase Server, incluindo drivers e ferramentas de manipulação do banco;
- Remote Debugger Server: Permite depurar um programa que está sendo executado em outra máquina.

Conhecendo o Delphi 6

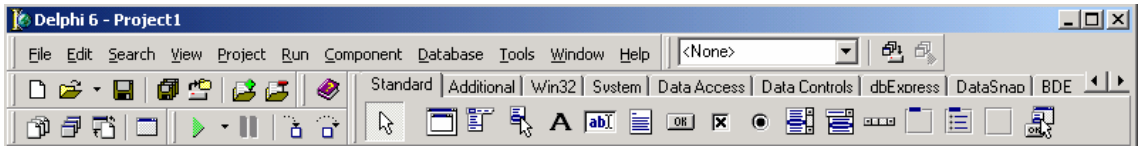
O Delphi oferece dois níveis de programação distintos. Existe o nível que o manual chama de *designer*, que utiliza os recursos de programação visual e aproveita os componentes prontos, e o nível do *component writer*, onde escrevemos os componentes para o *designer* utilizar nas aplicações. Podemos dizer que o *component writer* programa em um nível mais baixo e o *designer* em um nível mais alto. Para este curso, consideraremos apenas a programação no nível do *designer*.

Inicie o Delphi clicando no ícone Delphi 6 que se encontra no menu *Iniciar* / *Programas* / *Borland Delphi 6*.



Quando ativamos o Delphi, a tela inicial é parecida com a acima. Os itens que você está vendo formam o que chamamos de IDE, com um projeto novo aberto. Na janela superior, temos a barra do menu principal do Delphi, à esquerda a *SpeedBar*, com as opções mais comuns e à direita a paleta de componentes. Estes componentes formam a base da programação visual e é onde o *designer* vai buscar recursos para criar sua aplicação.
A seguir, vamos analisar as ferramentas que compõe o ambiente de desenvolvimento e os arquivos que constituem um projeto.

Janela Principal



A janela principal é o próprio Delphi, se a fecharmos estaremos fechando todo o Delphi. Esta janela é composta basicamente pelo menu e mais duas áreas distintas, o SpeedBar e a Component Palette - Paleta de Componentes.

SpeedBar

SpeedBar (“Barra de Velocidade”) foi o nome dado pela Borland à barra de ferramentas com atalhos para os procedimentos comumente executados durante a fase de desenvolvimento de um projeto. São eles:

- **New**. Abre uma caixa de diálogo que permite selecionar o tipo de objeto a ser criado (novo aplicativo, formulário, DLL, relatórios, entre outros). Equivalente ao menu File | New | Other.
- **Open**. Abre uma Unit, Form, Projeto ou Package. Equivalene ao menu File | Open.
- **Save**. Salva a Unit/Form atual. Equivalente ao menu File | Save.
- **Save All**. Salva todas as Units/Forms abertos que sofreram alteração. Equivalente ao menu File | Save All ou as teclas Shift+Ctrl+S.
- **Open Project**. Abre um arquivo de projeto (.dpr – Delphi Project). Equivalente ao menu File | Open Project ou as teclas Ctrl+F11.
- **Add file to Project**. Acrescenta um arquivo já existente ao projeto atual. Equivalente ao menu Project | Add to Project ou as teclas Shift+F11.

- **Remove file to Project**. Remove um arquivo do projeto. O arquivo não será removido do disco, apenas deixará de fazer parte projeto. Equivalente ao menu Project | Remove from Project.
- **Help Contents**. Aciona o Help do Delphi. Equivalente ao menu Help | Delphi Help.
- **View Unit**. Permite escolher uma Unit do projeto para ser exibida. Equivalente ao menu View | Units ou as teclas Ctrl+F12.
- **View Form**. Permite escolher um Form do projeto para ser exibido. Equivalente ao menu View | Forms ou as teclas Shift+F12.
- **Toggle Form/Unit**. Permite alternar entre um formulário e seu respectivo código fonte. Equivalene ao menu View | Toggle Form/Unit ou a tecla de função F11.
- **New Form**. Adiciona um novo formulário ao projeto. Equivalente ao menu File | New Form.
- **Run**. Executa a aplicação, compilando-a se necessário. Equivalente ao menu Run | Run ou a tecla de função F9.
- **Pause**. Suspende a execução do programa. Equivalente ao menu Run | Pause Program.
- **Trace Into**. Executa o programa passo a passo, linha a linha, dentro da rotina que for invocado e dentro de todas as rotinas que forem acessadas posteriormente. Equivalente ao menu Run | Trace Into ou a tecla de função F7.
- **Step Over**. Semelhante ao Trace Into, porém a execução passo a passo ocorrerá somente dentro da rotina em que for invocado. Equivalente ao menu Run | Step Over ou a tecla de função F8.

Component Palette (Paleta de Componentes)

Cada ícone na paleta refere-se a um componente que, quando colocado em um Form, executa determinada tarefa: por exemplo, um TLabel mostra um texto estático, e um TEdit é uma caixa de edição que permite mostrar e alterar dados, o TComboBox é uma caixa que permite selecionar um dentre os itens de uma lista, etc.

A paleta de componentes tem diversas guias, nas quais os componentes são agrupados por funcionalidade. Outras guias podem ser criadas com a instalação de componentes de terceiros.

Segue abaixo a relação completa de todas as guias que compõe o Delphi 6 Enterprise:

Standard: componentes padrão da interface do Windows, usados para barras de menu, exibição de texto, edição de texto, seleção de opções, iniciar ações de programa, exibir listas de itens etc. Geralmente são os mais usados.

Additional: componentes especializados que complementam os da página Standard. Contém botões com capacidades adicionais, componentes para exibição e edição de tabelas, exibição de imagens, gráficos etc.

Win32: componentes comuns de interface que são fornecidos pelo Windows 95/NT para os programas. Contém componentes para dividir um formulário em páginas, edição de texto formatado, barras de progresso, exibição de animações, exibição de dados em árvore ou em forma de ícones, barras de status e de ferramentas etc.

System: componentes que utilizam funções avançadas do sistema operacional, como temporização (timers), multimídia e conexões OLE e DDE.

Data Access: componentes de acesso a bancos de dados e conexão com controles de exibição de dados.

Data Controls: componentes semelhantes aos encontrados nas guias Standard e Additional, porém ligados a banco de dados.

dbExpress: componentes de conexão com bancos de dados SQL introduzida no Delphi 6 e no Kylix (Delphi para Linux). Entre os principais recursos desta nova arquitetura estão dispensar o uso do BDE (Borland Database Engine)

para acesso a dados e fazer um acesso muito mais rápido e leve. A configuração e instalação também são mais simples que a do BDE. Atualmente existem drivers para Oracle, DB/2, Interbase e MySQL entre outros. Não existe suporte para bancos de dados Desktop, assim, não permite acesso a dBase, Paradox ou Access para estes você deverá continuar a utilizar o BDE.

DataSnap: componentes que permitem a criação de middleware de alto desempenho capazes de trabalhar com Web services, possibilitando fácil conexão de qualquer serviço ou aplicação de cliente com os principais bancos de dados, como Oracle, MS-SQL Server, Informix, IBM, DB2, Sybase e InterBase, através de Serviços Web padrão da indústria e XML, DCOM ou CORBA. (No Delphi 5 esta guia era chamada de Midas).

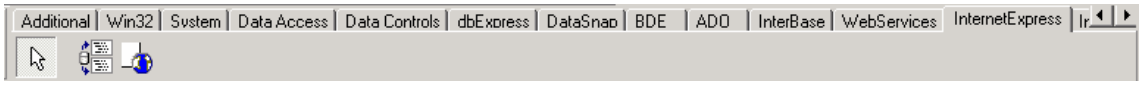
BDE: componentes de acesso a dados utilizando a BDE (até o Delphi 5 faziam parte da guia Data Access). A BDE é a engine que acompanha o Delphi desde sua primeira versão. Muito completa, permite acessar desde bases de dados desktop, como Paradox, dBase ou Access, até bases de dados SGDB, como Interbase, DB/2, Oracle, Informix, SyBase ou MS-SQL Server, todos em modo nativo. Permite ainda o acesso a outros bancos de dados através de ODBC.

ADO: componentes de acesso a dados da interface dbGo (introduzida no Delphi 5 como o nome de ADO Express) através da tecnologia ADO (ActiveX Data Objects), da Microsoft. Tanto a ADO como o OLE DB (drivers de acesso) estão incluídos no MDAC (Microsoft Data Access Components) e permitem acesso a uma série de bancos de dados e à ODBC. Sua principal vantagem é estar incorporada as versões mais recentes do Windows (2000/XP e ME) não sendo necessário nenhuma instalação de engine de acesso. Também é escalável, permitindo acesso desde bases de dados desktop até aplicações multicamadas. A desvantagem é que não é portável para outras plataformas, caso queira portar seu sistema para Linux, terá que trocar todos os componentes de acesso a dados.

Interbase: componentes para acesso nativo ao Interbase, através de sua API, constituindo o método de acesso mais rápido e eficiente para este banco de dados. Por não ser uma interface genérica permite utilizar todos os recursos

que o Interbase disponibiliza. A desvantagem, no entanto é que ao utiliza-los perde-se a possibilidade de alterar o banco de dados sem mudar o programa, visto que os mesmos se destinam apenas ao Interbase.

WebServices: componentes que formam a BizSnap, uma plataforma RAD para desenvolvimento de Web services, que simplifica a integração B2B criando conexões e Web services baseados em XML/SOAP

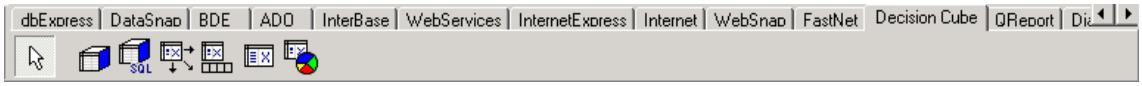


InternetExpress: componentes para manipulação de XML e produção de páginas para internet.

Internet: componentes para manipulação de Sockets, páginas e web browser.

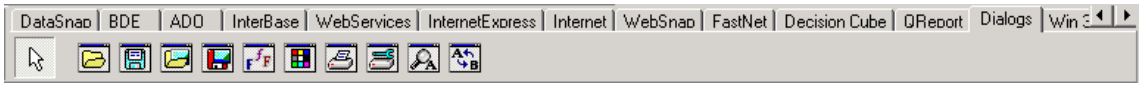
WebSnap: componentes para o desenvolvimento de aplicações Web que suporta os principais Servidores de Aplicações Web, inclusive Apache, Netscape e Microsoft Internet Information Services (IIS);.

FastNet: componentes para manipulação de protocolos e serviços da internet como http, nntp, ftp, pop3 e smtp entre outros. (São mantidos por compatibilidade com o Delphi 5, nesta versão foram inseridos os componentes Indy com maior funcionalidade).

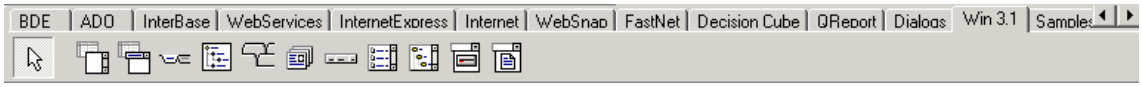


Decision Cube: componentes para tomada de decisão através da análise multidimensional de dados, com capacidades de tabulação cruzada, criação de tabelas e gráficos. Estes componentes permitem a obtenção de resultados como os obtidos por ferramentas OLAP (On-Line Analytical Processing – Processamento Analítico On-Line) utilizados para análise de Data Warehouses.

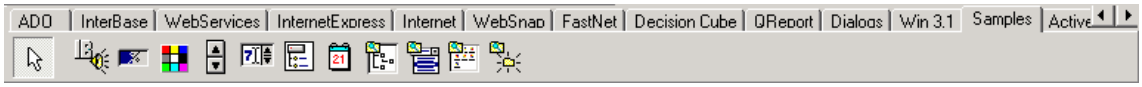
Qreport: QuickReport é um gerador de relatórios que acompanha o Delphi e se integra totalmente ao mesmo, sem a necessidade de run-time ou ferramentas externas como o Cristal Report, etc. Estes componentes permitem a criação de diversos tipos de relatórios de forma visual e também a criação de preview personalizado.



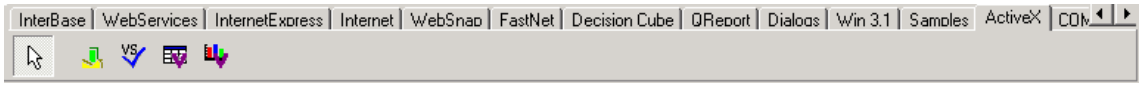
Dialogs: O Windows tem caixas de diálogo comuns, como veremos, que facilitam mostrar uma interface padrão dentro do seu programa para as tarefas comuns, como abrir e salvar arquivos, impressão, configuração de cores e fontes etc. Esta guia tem componentes que permitem utilizar essas caixas de diálogo comuns.



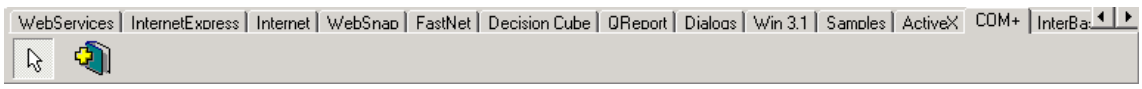
Win 3.1: esta guia contém controles considerados obsoletos, que estão disponíveis apenas para compatibilidade com programas antigos. Não crie programas novos que utilizem esses controles.



Samples: contém exemplos de componentes para que você possa estudá-los e aprender a criar seus próprios componentes. O código fonte desses exemplos está no subdiretório SOURCE\SAMPLES do diretório de instalação do Delphi.

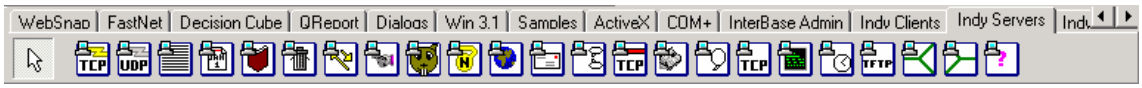


ActiveX: um componente ActiveX é um tipo de componente que pode ser criado em outra linguagem (como C++) e utilizado no Delphi. Esta página contém alguns exemplos de componentes ActiveX prontos para utilizar, que têm funções de gráficos, planilha, etc. O Delphi também pode criar componentes ActiveX, que podem ser utilizado em ambientes como Visual Basic e Visual FoxPro.



COM+: catálogo de objetos COM (Component Object Model), tecnologia desenvolvida pela Microsoft que possibilita a comunicação entre clientes e aplicações servidores. Uma interface COM é a maneira como um objeto expõe sua funcionalidade ao meio externo.

Indy Clients: componentes para criação de aplicativos clientes para protocolos HTTP, TCP, FTP, DNS Resolver, POP3, SMTP, TELNET, entre outros.

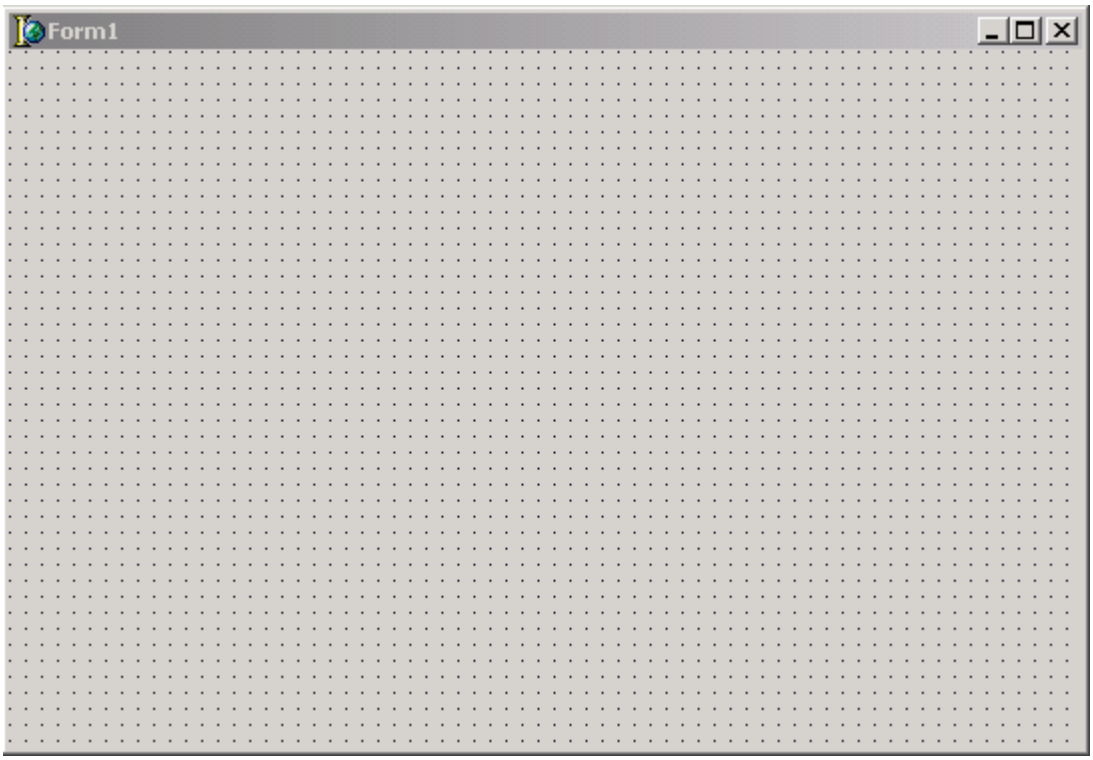


Indy Servers: componentes para criação de aplicativos servidores de HTTP, TCP, FTP, TELNET, GOPHER, IRC, entre outros.

Indy Misc: componentes complementares aos das guias Indy Clients e Indy Servers, para criação de aplicativos clientes e servidores com acesso a internet, como clientes de ftp, irc e browsers.

Servers: componentes para automatização do Microsoft Office, permitindo o controle, impressão e criação de documentos destes aplicativos dentro do seu programa.

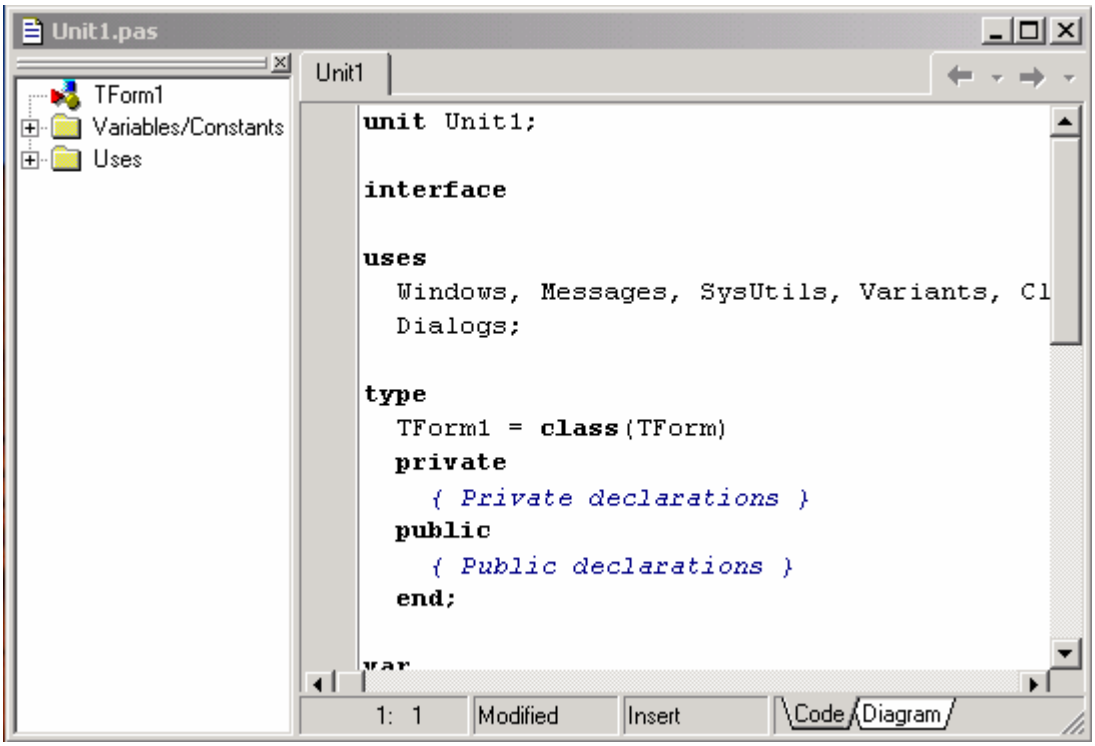
Formulário para criação do Aplicativo



Cada aplicativo criado em Delphi é chamado de projeto e pode ser formado por um ou mais formulários (Janelas), ou ainda em casos especiais não possuírem janela alguma (Console Application).

É sobre estes formulários que serão colocados os componentes para a criação da interface do aplicativo.

Todo formulário possui um arquivo de programa-fonte correspondente, chamado **Unit**, que pode ser visualizado no editor de código (Code Editor). A janela do editor de código pode ser acessada clicando-se no botão Toggle Form/Unit da SpeedBar caso o Formulário esteja selecionado; você também pode clicar na borda que aparece logo abaixo do formulário, ou ainda pressionando-se a tecla F12 que permite alternar entre o editor de código e o formulário.

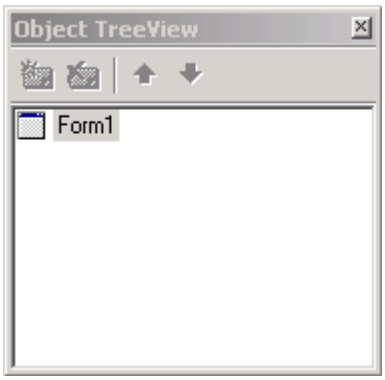


A Unit está intimamente ligada ao formulário, chamado **Form**: quando se adiciona um componente ao Form, o Delphi inclui na Unit deste Form o código referente à inclusão do mesmo, ou seja, uma mudança no lado visual resulta em uma alteração automática no código. A Borland, empresa que fez o Delphi, denominou isso de Two-Way-Tool (ferramenta de duas vias).

À esquerda do editor de código esta o Code Explorer, uma ferramenta que permite visualizar a acessar no código fonte as units, classes, variáveis, constantes e objetos (componentes) que fazem parte do Form.

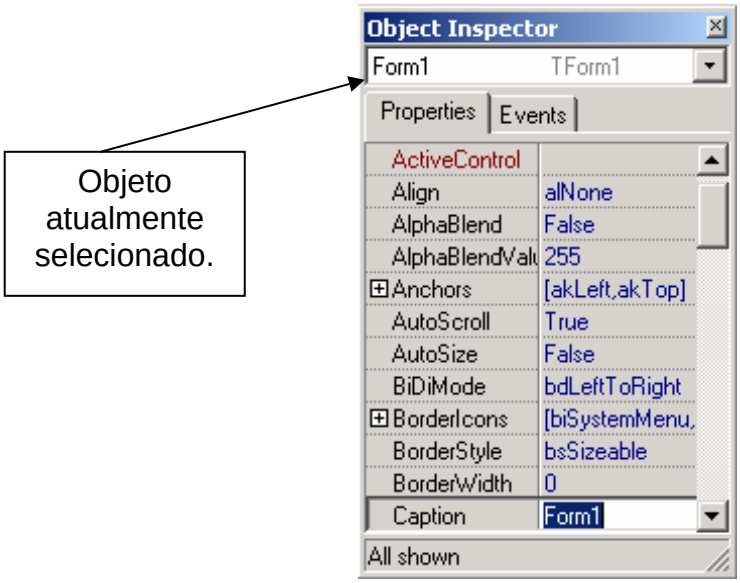
Na mesma janela do editor de código pode ser acessada a guia Diagram que permite documentar o relacionamento entre os componentes, além de modificar algumas características destas relaçãoes. Inicialmente ela está vazia, para incluir os componentes, você deve arrastá-los do Object TreeView e solta-los sobre o diagrama.

Object TreeView



O Delphi 6 introduziu uma nova janela, o Object TreeView, que mostra o relacionamento entre os componentes que são colocados no Form. Como veremos adiante, existem componentes que funcionam como “recipientes”, ou seja, podem conter outros componentes dentro de si. O Object TreeView permite a visualização destes relacionamentos e o acesso rápido a estes objetos.

Object Inspector (Propriedades e Eventos)

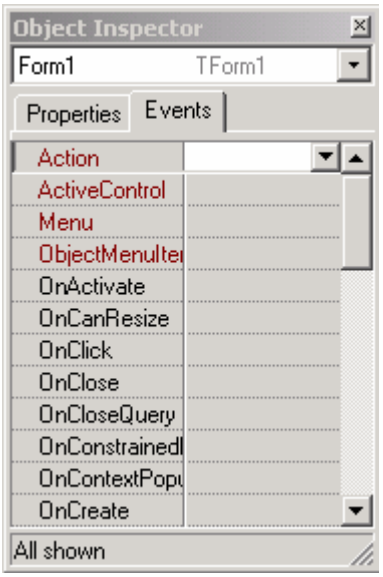


O Object Inspector é a ferramenta responsável por permitir a modificação das propriedades dos componentes/objetos de forma visual, durante o projeto (Design Time). Por enquanto, pense em propriedades como sendo as características dos componentes, tanto visuais quanto funcionais. O combobox na parte superior desta janela dá acesso a todos os objetos do Form atual e sempre exibe o nome do objeto/componente selecionado. Uma vez selecionado você pode inspecionar suas propriedades e altera-las se desejar. A alteração das propriedades pode modificar a aparência de objetos visuais e também seu funcionamento padrão.

A alteração o valor de uma propriedade depende de seu tipo, para textos e números basta apenas digita-los no quadro correspondente (em frente ao nome da propriedade). Caso o mesmo seja formado por uma lista, será fornecido um combobox com os valores possíveis neste mesmo local.
Se existir um sinal de adição à esquerda do nome da propriedade, isto indica que a mesma possui subpropriedades, para acessa-las clique no sinal de adição que as mesmas serão exibidas. Algumas propriedades possuem um editor de propriedade, nestes casos é fornecido um botão com reticências. Clicando-se neste botão a tela de edição da propriedade deve ser exibida.

A ordem e as propriedades que devem ser exibidas no Object Inspector podem ser configuradas clicando-se com o botão direito do mouse sobre o mesmo para acionar seu menu popup. A opção *View* permite selecionar as categorias a serem exibidas, desativando uma categoria, todas as propriedades que pertencem a elas não serão mostradas. A opção *Arrange* do menu permite ordenar por categoria ou por nome.

Além da modificação das propriedades, os componentes sofrem a ação de eventos. Um evento ocorre quando o programa está sendo executado e o usuário pressiona o botão do mouse ou uma tecla, por exemplo. Você pode querer que seu programa execute uma determinada tarefa se uma dessas ações ocorrem.



Cada um dos eventos exibidos no Object Inspector para um determinado objeto é acionado quando é executada uma ação específica. Por exemplo, fechar a janela ativaria o evento OnClose da mesma. Se houver algum código associado a esse evento, ele será executado naquele momento. Assim, você pode observar que o programa no Windows é essencialmente passivo, isto é, ele espera até que algo aconteça e ele seja chamado. Se um evento é

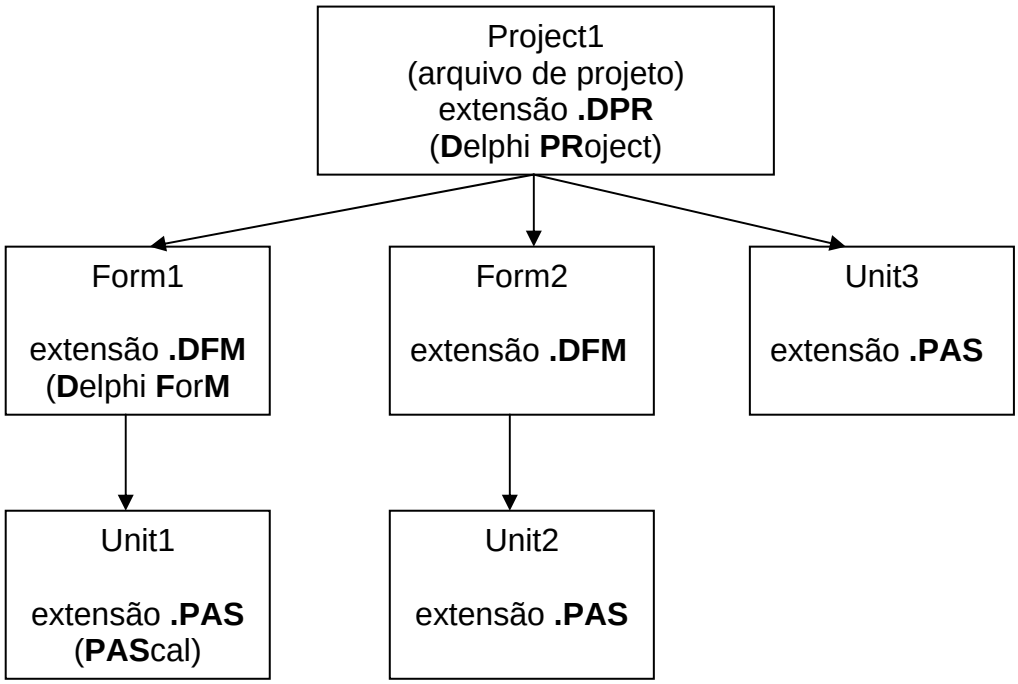
codificado, mas o mesmo não ocorre durante a execução do aplicativo seu código não é executado.

Conhecendo a Estrutura de uma Aplicação

Uma aplicação feita em Delphi é composta de um arquivo de projeto, que gerencia quais Forms e Units compõem a aplicação. O nome dado ao arquivo do projeto, normalmente será o nome dado ao executável da aplicação quando a mesma for compilada.

Em alguns casos podemos ter uma Unit sem Form, um exemplo seria uma Unit com funções para serem utilizadas por toda a aplicação (em vários Forms), mas todo Form obrigatoriamente deve possuir sua Unit correspondente.

Vejamos um esquema gráfico de como é estruturado um projeto em Delphi:



Estrutura da Unit de um Form

As Units do Delphi possuem uma estrutura que deve ser obedecida.
Quando um Form é criado também é criada uma Unit associada ao mesmo.

A estrutura básica de uma Unit pode ser visualizada observando-se o código fonte da mesma no editor de código. Será semelhante ao exibido a seguir:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs;

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

var
  Form1: TForm1;

implementation

{$R *.dfm}

end.
```

Vamos analisar o código acima:

- Na primeira linha, o nome em frente à palavra **unit**, no caso Unit1, indica o nome dado ao arquivo com a programação do formulário. Se o formulário fosse salvo com este nome ele geraria um arquivo externo com o nome de Unit1.pas e outro com o nome de Unit1.dfm. (Quando for salvar seus formulários você deve dar nomes mais significativos).
- Na linha seguinte, a palavra **interface** delimita a seção de interface na qual serão colocadas as definições de funções, procedimentos, tipos e variáveis que poderão ser vistos por outras units da aplicação.
- A cláusula **Uses** dentro da seção interface indica quais units deverão ser ligadas para poder complementar a nossa . Ao criar um Form as Units definidas no código acima são inseridas automaticamente, pois fornecem o suporte para criação do mesmo. Ao inserir componentes num Form, outras Units podem ser adicionadas a esta lista.
- A seguir temos a definição dos tipos do programa, identificada pela palavra **type**. Neste ponto temos a definição de uma classe TForm1 que é derivada da classe base TForm. Ao se acrescentar componentes no Form também será gerado no código definição correspondente aos mesmos. (O conceito de classes e objetos será explicado no Capítulo 2)

- O próximo item é a definição de variáveis e constantes globais, através da palavra reservada **var**. Neste ponto é criada uma variável com visibilidade global (pode ser vista em outras units nas quais a mesma seja incluída na cláusula uses)
- A palavra chave **implementation** delimita a segunda seção da unit, onde serão colocadas as funções e variáveis que serão acessadas apenas por ela mesma (não são visíveis em outras units).
- O símbolo ***{$R *.dfm}*** *faz a associação da unit com seu respectivo form e não deve ser modificado. Uma unit de funções não possui esta diretiva.*
- Ao Final da Unit, temos uma linha com **end.** Ele é o marcador de final de arquivo. Qualquer coisa colocada após esta linha será ignorada.
- Opcionalmente, uma unit pode ter ainda duas seções: **initialization** e **finalization**, com comandos que são executados quando a aplicação é iniciada ou finalizada.

Criando seu Primeiro Programa em Delphi

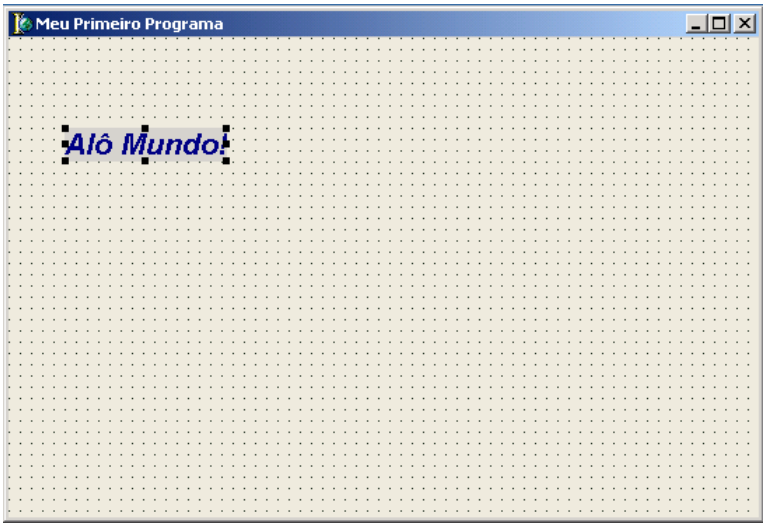
É comum que todo programador ao iniciar numa linguagem de programação crie um programa chamado “Alô Mundo!” (Hello World!). Não quebraremos esta tradição aqui.

Com um projeto novo aberto, vá ao Object Inspector e procure a propriedade *Caption* do Form, que representa o Título da Janela. No lugar do texto atual (Form1) digite o texto “Meu Primeiro Programa” (sem as aspas). A seguir, localize a propriedade *Name* e altere seu conteúdo de Form1 para FormPrincipal. (Obs: Até agora alteramos o Título da Janela e seu nome interno).

Agora iremos escrever o texto “Alô Mundo!”, para isso iremos precisar de um componente para este fim. Vá até a paleta de componentes, na guia Standard e clique no componente Label, quarto item da guia, representado por uma letra **A**. Em seguida clique no Form onde deseja que o texto apareça.

Pressione F11 para selecionar a janela do Object Inspector e procure a propriedade *Caption*. No Label ela é responsável pela exibição de um Texto. Digite “Alô Mundo!” (novamente sem as aspas). Localize a propriedade *Name* e altere-a para LabelMensagem. Procure a propriedade *Font* e clique no editor de propriedade (o botão com reticências), na caixa de diálogo selecione a fonte Arial estilo Negrito Itálico, tamanho 16 cor Azul-marinho.

Se você acompanhou tudo até aqui, seu Form deve estar semelhante a figura a seguir:



Agora vamos aprender a salvar nosso projeto. Para isso vá ao menu *File* e escolha a opção *Save All*, ou ainda clique no botão *Save All* do SpeedBar. Você também pode utilizar o atalho de teclado *Shift + Ctrl +S*.

Será apresentada uma caixa de diálogo com o título “Save Unit1 As”, o “Unit” do título indica que estamos salvando o Form (a Unit do mesmo). Observe no item *Salvar em* qual o local onde será salvo o projeto e modifique se necessário ou então crie uma pasta para o mesmo. Em *Nome do arquivo* digite **Principal.pas** e pressione Enter (ou clique no botão *Salvar*).

Uma nova caixa de diálogo será apresentada, desta vez com o título “Save Project1 As”. No item *Nome do arquivo* digite **AloMundo.dpr**.

Com estes procedimentos acabamos de salvar nosso projeto e o formulário do mesmo. Se você observar o diretório em que os salvou, notará a existência dos seguintes arquivos:

- AloMundo.dpr: o arquivo de projeto
- Principal.pas: arquivo com a Unit do Form
- Principal.dfm: arquivo do Form, que recebe automaticamente o mesmo nome dado a sua Unit.
- AloMundo.res: arquivo de recursos do projeto, normalmente contém apenas o ícone que será usado no executável, mas pode receber outros recursos.
- AloMundo.dof e AloMundo.cfg: arquivos de configuração de compilação e ambiente.

Os arquivos principais são os .**dpr**, **.pas** e **.dfm**, os demais caso não sejam encontrados serão recriados com as opções padrão.

Vá ao editor de código e observe que na primeira linha onde estava escrito **unit Unit1;** agora se encontra **unit Principal;** o nome que atribuímos ao formulário quando o salvamos.

Agora que já salvamos nosso programa, vamos executa-lo. Para isso vá até o menu *Run* e escolha a opção *Run*, ou ainda clique no botão de mesmo nome no SpeedBar. Você também pode usar o teclado, pressionando a tecla F9.

O projeto será compilado e executado. Parabéns, você criou seu primeiro programa em Delphi!

Para fecha-lo pressione *Alt+F4* ou clique no botão fechar na barra de título.

Se você observar novamente a pasta onde os arquivos foram salvos verá que dois novos arquivos foram criados:

- Principal.dcu: a Unit e o Form compilados, dcu significa Delphi Compiled Unit. A união de todos os .dcu de um projeto juntamente com outras Units internas do Delphi irão formar o arquivo executável .exe.
- AloMundo.exe: o aplicativo independente (executável que roda sem o Delphi), é criado com o mesmo nome dado ao projeto.

Caso se deseje fazer uma cópia de segurança de um projeto, os arquivos .dcu e .exe podem ser apagados, pois podem ser recriados através de uma nova compilação do projeto. Após a compilação, apenas o executável é necessário. É ele que será enviado ao usuário final.

Este foi um capítulo extenso, no qual procuramos exibir algumas das principais características e recursos Delphi, para lhe dar uma visão geral do que esta fantástica ferramenta de desenvolvimento é capaz de fazer.

No próximo capítulo introduziremos conceitos sobre OOP (Object Oriented Programming – Programação Orientada a Objeto) e analisaremos em maior profundidade as propriedades e os eventos do Form e de alguns componentes.

Exercícios

1) O que é Delphi?
2) O que é IDE? Quais os itens que a compõe?
3) Para que serve o Object Inspector?
4) Quais são os principais arquivos de um projeto?
5) O que é um evento?

Capítulo 2

Programação Orientada a Objeto (POO)

Embora não seja objetivo deste curso ensinar POO, uma breve introdução é necessária, já que o Delphi é essencialmente orientado a objeto.

De maneira prática, podemos pensar no objeto sendo uma estrutura que agrupa dados e funções para manipular estes dados. Como as funções são sempre “intimamente ligadas” aos dados, o sistema todo funciona de maneira mais segura e confiável. Além disso, a POO utiliza conceitos como *encapsulamento* e *herança* que facilitam muito a programação e manutenção dos programas.

Os dados de um objeto costumam ser chamados de *variáveis de instância* e as funções de *métodos*. As *variáveis de instância* definem as *propriedades* (também chamadas de *atributos*) do objeto e os *métodos* definem o seu *comportamento*.

Encapsulamento - Como as variáveis e métodos estão na mesma estrutura, pode-se pensar em variáveis e métodos privados, ou seja, dados e funções que só podem ser manipulados pelas funções que estão dentro da estrutura. Desta maneira é possível formar uma camada protetora nos dados e evitar atribuições desastradas que comprometeriam o funcionamento do programa. Os defensores mais ortodoxos da POO dizem que todos os dados de um objeto deveriam ser privados e o número de funções públicas deve ser o menor possível, mas isso nem sempre é viável ou prático. O Delphi implementa este conceito e oferece dados/funções publicas (*public*) e privadas (*private*). Outra conseqüência do encapsulamento é que os objetos podem ser “caixas pretas”. Não é necessário (teoricamente) conhecer detalhes de funcionamento de um objeto para usá-lo, basta enviar as mensagens apropriadas que ele responde com a ação desejada.

Classes - A classe representa um tipo ou categoria de objetos, o modelo a partir do qual um objeto pode ser construído. É a estrutura propriamente dita, que define os dados e métodos daquela classe de objetos. O objeto em si é uma instância da classe. Podemos fazer uma analogia com a programação estruturada entre os tipos e variáveis, onde a classe equivale ao tipo e o objeto à variável desse tipo.

Herança - É a capacidade que uma classe de objetos tem de herdar variáveis e métodos de outra classe. Esta capacidade permite que o código já escrito seja reutilizado de maneira muito mais eficiente e simples do que na programação estruturada. Um programa orientado a objeto costuma implementar verdadeiras árvores genealógicas de classes, com vários níveis de herança.

Para programar no nível do *designer* não é necessário um conhecimento profundo de POO. Mas é preciso conhecer pelo menos a sintaxe. Todas as aplicações para Windows precisam de pelo menos uma janela, que no Delphi é chamada de *Form*. Cada *form* (assim como todos os objetos visuais) tem um objeto associado a ele e sua representação visual, que vemos na tela. Todos os componentes que incluímos no form passam a fazer parte do objeto que define o form. Exemplo: Se colocarmos um botão no form, a classe deste form será modificada para incluir este botão. Os eventos e métodos deste form, também estão na classe. Assim, supondo que o form se chame Form1 (nome default), por exemplo, para desativar o botão que incluímos (de nome Buttom1) escreveríamos:

```
Form1.Buttom1.Enabled := false;
```

Note como o Buttom1 faz parte da estrutura que define o form. Mas se quisermos ativar método RePaint (Repintar) do form faríamos:

```
Form1.Repaint;
```

Veja que Repaint, não é uma variável, tecnicamente é uma procedure (método), mas fazemos referência a ela como parte da estrutura Form1. Isso pode parecer confuso no início, mas facilita muito a programação.

Conhecendo propriedades, métodos e eventos básicos

No Anexo **A**, temos uma referência sobre propriedades, métodos e eventos dos componentes mais utilizados numa aplicação. Para que possamos prosseguir veremos agora algumas propriedades, métodos e eventos comuns a maior parte dos componentes do Delphi:

Propriedades Básicas

***Name*:** É comum a todos os componentes da paleta. O Delphi nomeia automaticamente todos os componentes que são incluídos no form (inclusive o proprio form). Usa o nome da classe do componente mais um número sequencial. O nome atribuído pelo Delphi pode ser mantido, mas é aconselhável renomear os componentes que serão referidos no programa. No nosso primeiro exemplo, o programa AloMundo, o Form e o Label que inserimos tinham os nomes de Form1 e Label1, mas nós os alteramos para FormPrincipal e LabelMensagem. Como não fizemos nenhuma programação isso seria até desnecessário, mas você deve criar o hábito de renomear todos os componentes que utiliza, caso seja feita referência a eles no código fonte, é muito mais claro imaginar que um botão chamado ButtonFechar seja para fechar seu form do que se o nome fosse Button1 ou Button13. Quando você renomeia um componente, o Delphi atualiza automaticamente todo o código gerado por ele, o que inclui o cabeçalho da Unit, os eventos do componente e as propriedades de outros componentes que fazem referência ao componente renomeado, **mas não atualiza o código gerado por você.** Exemplo: se

renomearmos um botão de Button1 para ButtonIncluir, o Delphi atualizará o cabeçalho da unit, o nome dos eventos de Button1, mas você terá que atualizar todas as referências que você fez ao Button1 em seu código com o novo nome. Aliás, esta é uma regra geral no Delphi: ele nunca modifica automaticamente o código gerado pelo programador, mesmo que esteja em comentário. E por segurança, você deve fazer o mesmo, não modifique o código gerado por ele, a não ser que saiba exatamente o que está fazendo, poderá inutilizar seu form, em alguns casos tendo problemas até para salva-lo.

Caption: Todos os componentes que podem apresentar um rótulo tem esta propriedade. Armazena a string que será mostrada quando o componente for desenhado. No Form ele corresponde ao texto exibido na barra de título.

Left e Top: Esquerda e Topo. Armazenam a posição do componente em relação ao form ou painel que o contém. Movendo o componente, estas propriedades se atualizam automaticamente, e alterando estas propriedades, o componente é movido.

Height e Width: Altura e comprimento do componente, alterando estas propriedades, o componente terá seu tamanho alterado.

Font: Permite selecionar a fonte, o tamanho, o estilo e a cor da fonte que será usada para escrever o texto no componente.

Color: Cor do componente. Existe uma lista de cores padrão usadas pelo Windows e pelo Delphi, mas é possível definir qualquer cor através de seus componentes RGB.

TabOrder: Ordem do componente no Form ou painel. Quando há vários componentes selecionáveis no Form ou painel, a tecla Tab permite navegar entre os componentes. Esta propriedade define a ordem em que os componentes são selecionados quando o usuário tecla Tab.

Hint: (Dica) Este é um recurso muito útil e fácil de usar. Permite que apareça um texto de ajuda quando o usuário posiciona o cursor do mouse sobre um componente. Todos os componentes podem ter Hint. Na propriedade Hint, voce deve digitar a frase que deve aparecer. Veja a propriedade abaixo.

ShowHint: Ativa o hint para o componente. Se estiver desligado, o hint não é mostrado.

Eventos básicos

OnClick: É gerado cada vez que o botão esquerdo do mouse é pressionado e solto. O evento só ocorre quando o usuário libera o botão. O Delphi já direciona o evento para o componente que está debaixo do cursor do mouse.

OnDblClick: Gerado quando é feito um duplo clique no botão esquerdo.

OnKeyPress: Gerado quando o usuário pressiona (e libera) uma tecla no teclado.

OnEnter: Gerado quando o foco de atenção do Windows "cai" sobre o componente. Exemplo: Suponha uma tela de entrada de dados, onde vários campos (caixas de texto) devem ser digitados. Quando a tela é apresentada, o foco, ou o cursor de texto, está sobre o primeiro campo. Depois de digitar o campo, o usuário pressiona Tab para *passar* para o campo seguinte. Veja que o que *passa* para o campo seguinte é a atenção da aplicação e do Windows. Essa atenção é chamada de foco, ou ***focus*** em inglês. Este evento é gerado assim que o componente recebe o foco, antes de executar qualquer código do componente.
OnExit: Gerado imediatamente antes de o foco deixar o componente.

OnResize: Gerado quando o tamanho do componente é alterado. É usado principalmente em forms e painéis.

OnChange: Gerado quando o valor do componente muda. Só é aplicado em componentes que permitem edição de seu conteúdo.

Métodos Básicos

Show: Desenha o componente. Se for uma janela (form) ela é desenhada e ativada.

Close: Fechar. Aplicado geralmente em forms e componentes de acesso a dados. Quando utilizado no form principal, encerra a aplicação.

Repaint: Repintar. Redesenha o componente ou form.

Refresh: Tem o mesmo efeito que o ***Repaint***, mas antes de desenhar, apaga o componente. Quando aplicado em componentes de acesso a dados, faz com que o buffer do arquivo seja recarregado.

Create: Aloca memória para criar um componente ou form, dinâmicamente.

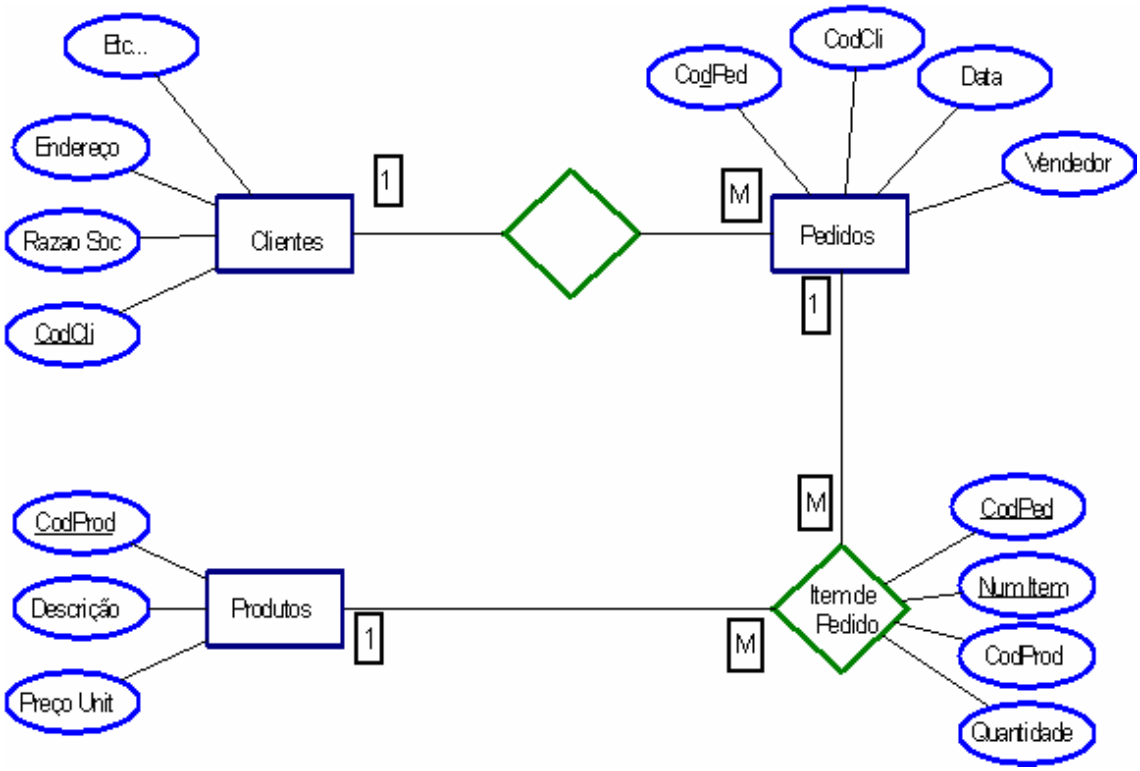
Free: Libera a memória alocada com o ***Create***.

Criando um Projeto Piloto

Para continuar nosso aprendizado, desenvolveremos um projeto piloto, um sistema de entrada de pedidos. Nele introduziremos os conceitos de acesso a

dados, os comandos de Object Pascal e toda teoria necessária à criação de aplicativos em Delphi.

Para começar vamos dar uma boa olhada no *DER* (Diagrama Entidade-Relacionamento) da figura abaixo. Temos o cadastro de clientes que armazena dados sobre os clientes. Da mesma maneira, o cadastro de produtos. Para o cadastro de pedidos, existe um problema: Se o Pedido relacionar diretamente o Cliente com o Produto, teremos uma relação de M para M (Muitos para Muitos), por que um cliente pode pedir muitos produtos e um produto pode fazer parte de muitos pedidos. Procuramos sempre evitar esse tipo de relação, então incluímos mais uma relação: o Item de Pedido. Relacionando Pedidos com Produtos através de Itens de Pedido, ficamos apenas com relações de 1 para M. Um pedido pode ter muitos itens, mas um item só pode fazer parte de um pedido. Da mesma maneira, um produto pode fazer parte de muitos itens, mas um item só pode ter um produto associado.



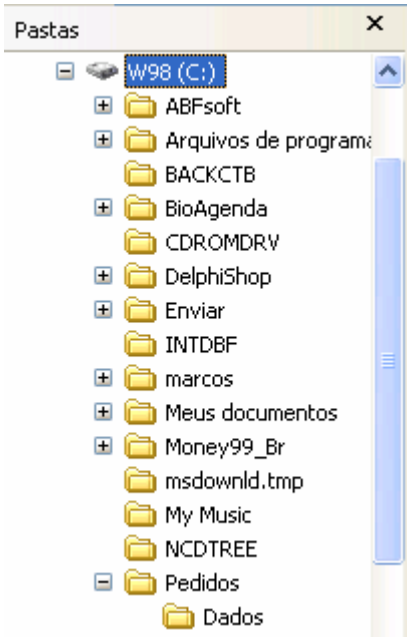
Agora, precisamos decidir qual será nosso banco de dados e qual a engine que utilizaremos: BDE, dbGO (ADO), IBX (Interbase Express) ou dbExpress. Optamos por utilizar o BDE pois o mesmo se encontra disponível em todas as versões do Delphi. Utilizaremos uma base de dados Desktop, o Paradox, que para nossos propósitos será mais que suficiente.

Antes de criarmos nossa base de dados, é interessante observar só mais um detalhe. O Paradox é uma base de dados onde cada tabela é armazenada em um arquivo externo separado. Cada um destes arquivos de dados pode ter associado a ele vários outros arquivos (necessários para a criação de índices, verificação de referências, validação e etc). Portanto o número total de arquivos

na base de dados aumenta rapidamente conforme vamos implementando novos cadastros. Por esse motivo, não é aconselhável deixar a base de dados no mesmo diretório que os programas fontes. Recomendo que debaixo do diretório que você criou para o projeto, crie um subdiretório chamado **Dados** ou **DB.** É nesse diretório que você deverá criar sua base de dados.

Criando o diretório para o seu Projeto

Abra o Windows Explorer e crie no drive C: (ou onde achar mais adequado) uma pasta para seu projeto. Por exemplo, criamos uma pasta chamada **Pedidos**, e dentro desta uma outra chamada **Dados**. A estrutura de diretórios ficou assim:



Conhecendo o BDE (Borland Database Engine)

Quando o Delphi precisa acessar um banco de dados, ele o faz através dos serviços do Borland Database Engine (BDE). O BDE funciona da mesma maneira para ler dados armazenados localmente em arquivos DB (Paradox) e DBF (dBase/FoxPro), bem como para os mais sofisticados sistemas client/server.

Muitos anos atrás, a Borland percebeu um problema com o acesso a banco de dados em seus softwares. Eles produziram vários produtos que eram usados para acesso a bancos de dados, mas cada um tinha um método diferente para conectar-se e utilizar os dados. A Borland percebeu que uma abordagem de conectividade de banco de dados unificada forneceria muitas vantagens. Então decidiram criar um novo software que deveria abstrair toda a funcionalidade de um banco de dados dentro de um engine.

Há vários anos atrás, a Borland vendia um Paradox engine para programadores. Era composto pela parte de acesso do DBMS Paradox e foi desenvolvido para ser usado em programas que precisavam ler e gravar informações em tabelas Paradox (.DB). O BDE substituiu este engine, mas adicionou funcionalidade para conexão com outros tipos de bancos de dados. O BDE também está disponível separadamente do Delphi, caso você precise de uma performance sólida em um ambiente de desenvolvimento que não seja da Borland.

Ter todas as ações mais comuns de banco de dados dentro de um pedaço de software produz muitos efeitos positivos. Um único conjunto de drivers permite a melhoria de um driver em particular sem ter que reivindicar a roda cada vez que um novo pacote tem acesso a um banco de dados. Isto significada que o acesso à banco de dados pode ser atualizado sem ter que atualizar um pacote de software inteiro. Se você instalar uma nova versão do BDE em um sistema com uma versão antiga do Paradox pode imediatamente beneficiar-se dos novos drivers.

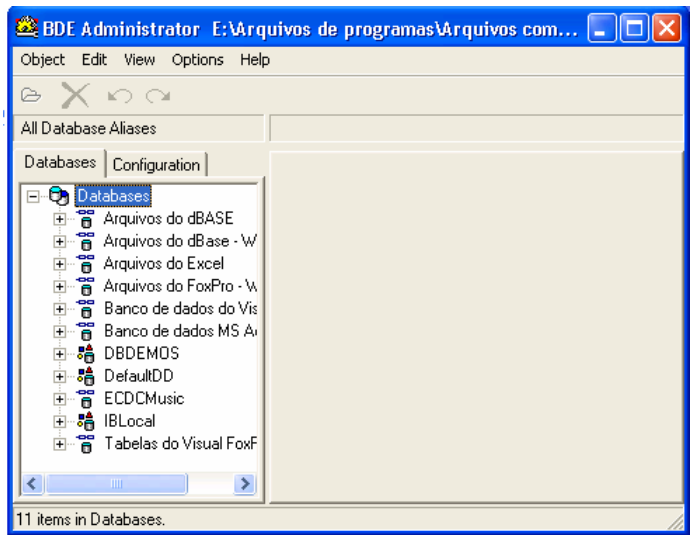
O conceito de driver unificado também salva você do armazenamento permanente que desaparece quando você tem muito código redundante. Um sistema com Paradox, Delphi e BDE requer muito menos espaço em disco porque o Paradox e o Delphi não necessitam ter seus próprios engines de acesso à banco de dados. Hoje em dia onde cada nova geração de programas requerem mais espaço em disco rígido, cada bit economizado ajuda.

Por último e, contudo mais importante, usar um método de acesso à banco de dados comum garante que o código para acessar um tipo particular de banco de dados seja escrito apenas uma vez.

Este capítulo se concentrará em lhe dar uma introdução aos recursos e capacidades do BDE.

O Utilitário BDE Administrador

Quando o Delphi é instalado, ele cria um ícone ou atalho para o utilitário BDE Administrador. (Abaixo temos o BDE Administrator rodando no Windows XP).



Ele pode ser acessado pelo menu do Windows em **Programas | Borland Delphi 6 | BDE Administrator**.
O Delphi 6 coloca todos os executáveis e arquivos de configuração no diretório **Arquivos de programas\Arquivos comuns\Borland Shared\BDE**. No Delphi 1 estes arquivos eram colocados em um diretório chamado IDAPI, nomeado depois de utilitário de configuração antigo. IDAPI significa Independent Database Application Programmer Interface, e dá este nome para muitos dos arquivos e diretórios no sistema. O BDEAdmin atua como sua interface para o BDE. O utilitário permite que você mude muitos aspectos de como o BDE trabalha. O BDE armazena suas informações de configurações no Registry do Windows. Depois que você mudar suas definições no utilitário BDE Administrator, você deverá gravar suas definições selecionando **Object | Apply** ou dando um clique sobre o botão Apply.

Página Database

Na página Database do BDE Administrator estão os aliases para os bancos de dados disponíveis. Os bancos do dados são exibidos em uma árvore hierárquica parecida com o Windows Explorer. Para visualizar seus aliases de bancos de dados, simplesmente dê um clique sobre o símbolo de mais próximo ao banco de dados. Uma vez exibido, a definição de cada alias pode ser facilmente visualizada ou modificada selecionando o alias para exibi-lo no painel a direita do BDE Administrator.

Aliases de Bancos de Dados

Os aliases de banco da dados dão a você a habilidade para mudar a maneira que trabalha com seus dados em uma maneira muito poderosa. No mundo das aplicações de negócios, os bancos de dados são realocados freqüentemente quando as organizações geram novos arquivos e movem suas operações de bancos de dados para servidores de bancos de dados dedicados. Alguns ambientes de desenvolvimento necessitam que você recompile sua aplicação toda vez que um banco de dados mude de local, ou para construir uma camada adicional de abstração dentro do seu código. O BDE automaticamente dá a você esta habilidade através do uso dos aliases. Antes de vermos em detalhes como os aliases do BDE trabalham, veremos um pouco além a natureza de dados e tabelas de bancos de dados.

Banco de Dados versus Tabelas

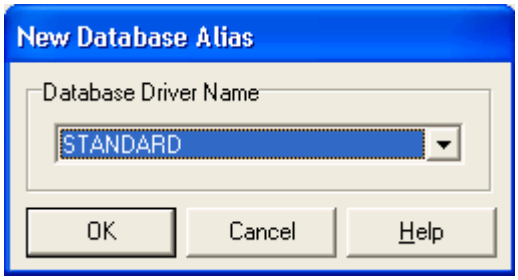
Quando Wayne Radcliff projetou a estrutura de arquivo do dBASE nos tempos dos PC's-Compatíveis, ele usou a extensão DBF (Database File) para descrever isso. Este é um termo errado porque banco de dados significa uma coisa específica que não se aplica a arquivos DBF.

Cada arquivo DBF possui uma *tabela*, uma coleção ordenada de dados. Uma tabela divide informações entre colunas conhecidas como *campos* e linhas (Fields and Rows), que relacionados formam um *registro*. Esse termo "Banco de Dados" geralmente denomina a coleção de tabelas que se relacionam a outras no mesmo caminho. Exceto para aplicações simples, o banco de dados consiste em múltiplas tabelas relacionadas e cada tabela contém vários registros e campos.

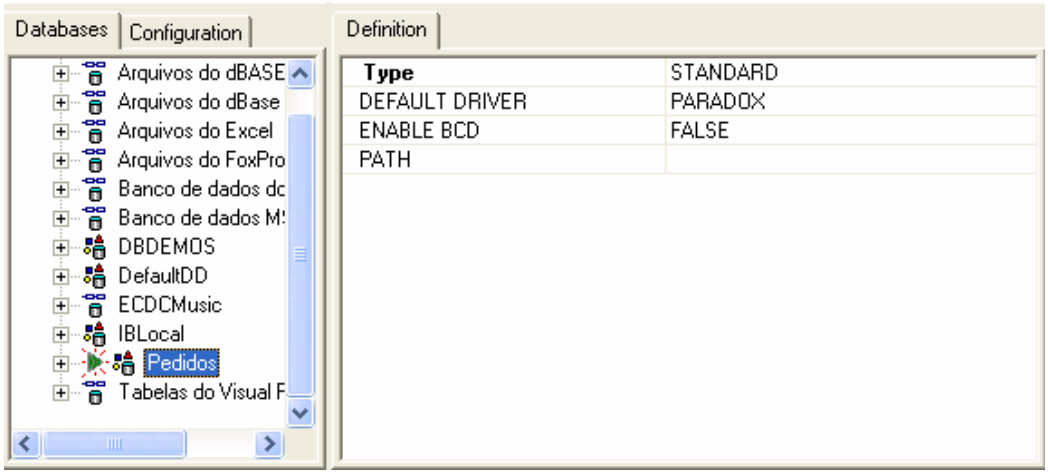
Agora que você entendeu que um banco de dados é composto por um conjunto de tabelas relacionadas, nós precisamos observar como implementar esse conjunto. Servidores de banco de dados como Interbase, SyBase e Oracle têm uma estrutura de banco de dados que pode ser composta por múltiplas tabelas. Desenvolvedores de banco de dados muitas vezes precisam utilizar esse tipo de estrutura, mas em ambientes sem servidores geralmente falta um mecanismo para linkar múltiplas tabelas. Você pode criar seu próprio formato de banco de dados para conter todas as suas tabelas, mas depois você perderá a compatibilidade com outros programas que utilizam banco de dados mais antigos.

O BDE possui uma solução avançada para esse problema. Colocado um grupo de tabelas relacionadas em uma mesma pasta, você pode associar essas tabelas juntas usando o BDE. Basta dar um alias, ou nome ao banco de dados, para que o grupo de arquivos em uma pasta particular se torne um conjunto de tabelas logicamente ligadas. O BDE chama um conjunto como este de Pseudo Database. Um Pseudo-Database não possui características avançadas como encontramos em um Database Server, como preservar integridade referencial. De qualquer modo ele permite a você utilizar o nome do banco de dados como ponteiro para o local de seus dados.

Vamos criar um novo alias, para nosso projeto. Comece selecionando **Object | New** no menu principal.
Você será questionado a selecionar um tipo de driver na caixa de diálogo New Database Alias.



O driver Standard fornece acesso às tabelas DB (Paradox), DBF (dBase/FoxPro) e arquivos Txt. Dê um clique sobre o OK para retornar a pagina Database. Agora, você precisará entrar com um nome para o alias no painel esquerdo. Vamos chamá-lo de **Pedidos**.



Você precisa também definir a localização das tabelas no campo PATH. Defina o caminho utilizado na criação da pasta do projeto: **C:\Pedidos\Dados**. Certifique-se de gravar a configuração, clicando no botão Apply ou indo ao menu **Object | Apply**. Também é possível utilizar o atalho de teclado **Ctrl+A**. Agora quando você precisar pegar dados em uma tabela, você pode usar o alias ao invés do nome diretório.

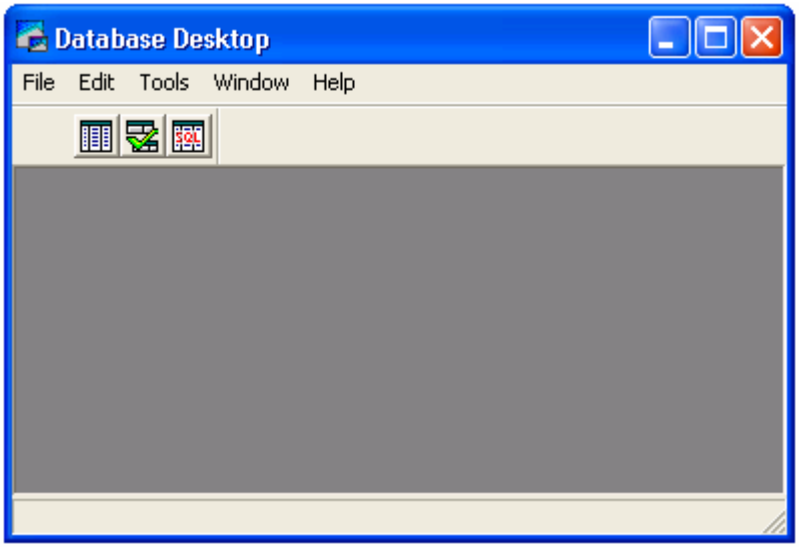
Até agora, parece que nós simplesmente mudamos um nome - o diretório, para outro - o alias. Isto nos salva de problemas quando chegar a hora de mudarmos a localização do banco de dados. Nós podemos colocá-lo em um servidor de arquivos de rede, movendo-o para um disco rígido maior, ou até mesmo fazer a mudança para um sistema de banco de dados cliente-servidor. Você não terá que mudar ou recompilar seus programas após as tabelas terem sido removidas. Você precisará criar um novo ponteiro para o alias no BDE.

A meta dos aliases do BDE é criar um ponteiro para um ponteiro. Qualquer rotina de acesso à banco de dados do Delphi fará referência ao alias de banco de dados, o qual faz referência às tabelas. Agora, mudando o alias, você tem uma maneira rápida de achar seus dados quando eles forem movidos de um lugar para outro.

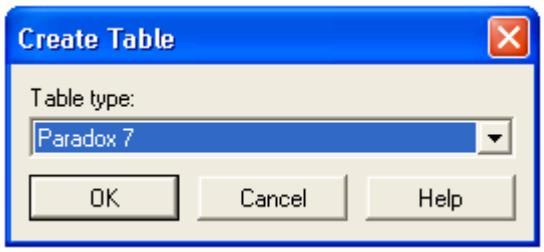
Você deverá, sempre que criar uma nova tabela de banco de dados ou acessar uma antiga, tentar referenciar-se a ela pelo alias ao invés do path.

Criando as Tabelas de Dados no Database Desktop

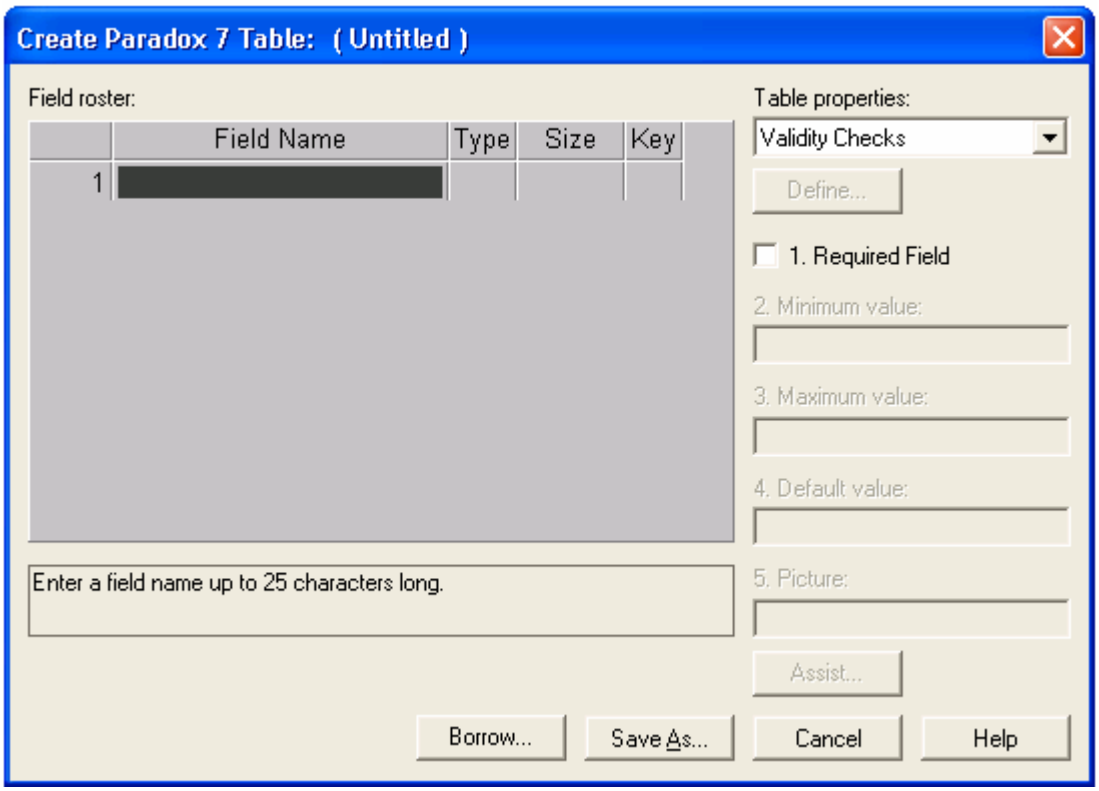
O Database Desktop é a ferramenta utilizada para a criação/modificação de tabelas de “bancos de dados” desktop, como Paradox. Você pode acessa-lo no menu do Windows em **Programas | Borland Delphi 6 | Database Desktop**, ou ainda no menu do Delphi em **Tools | Database Desktop**. Será apresentada uma janela como a figura a seguir.



Selecione o menu **File| New | Table**, será exibida a caixa de diálogo Create Table. Na caixa de combinação Table type está uma lista com os tipos de arquivos disponíveis. O tipo padrão, Paradox 7 vem selecionado.



Como utilizaremos este tipo mesmo, simplesmente clique no botão OK. Será exibida a caixa de diálogo Create Paradox 7 Table.



Na coluna **Field Name**, digitamos o nome para o campo; em **Type** escolhemos o tipo da informação que desejamos armazenar; em **Size** digitamos o tamanho para esta informação, se aplicável, pois alguns tipos possuem tamanho predefinido; em **Key** selecionamos se este campo pertence à chave primária[1]. Segue abaixo as regras para definição de nome de campos bem como de tipo e tamanho:
O tamanho máximo para o nome de um campo é de 25 caracteres.
Cada nome de campo deve ser único, você não pode ter dois campos com o mesmo nome na mesma tabela.

[1] Chave primária é o nome do índice que será utilizado para identificar exclusivamente um registro de uma tabela em nosso banco de dados, bem como classificá-la inicialmente.

O nome não pode iniciar com espaços em branco, mas pode conter espaços (exceto no final). No entanto, não é uma prática muito interessante, pois isto limitaria a maneira de se acessar a informação mais diretamente, pois o nome do campo não seria um identificador válido em Object Pascal, assim, é uma boa prática não utilizar espaços no nome dos campos.
Caracteres especiais não devem ser utilizados como parte do nome, apenas letras, números e o caractere de sublinhado. Portanto não acentue as palavras. (Use “Codigo” e não “Código”).

Evite utilizar palavras reservadas ou palavras chave de comandos SQL como SELECT ou COUNT para dar nomes aos campos, isto poderia trazer problemas caso você deseje no futuro acessar uma base de dados cliente-servidor. Procure atribuir nomes que descrevam claramente o conteúdo do campo e sua finalidade.

Obs: Estas regras se aplicam à utilização da estrutura das tabelas do Paradox, outros bancos de dados (desktop ou não) podem possuir regras próprias diferentes das utilizadas pelo Paradox. Um bom princípio seria nunca definir nomes de campos que não poderiam ser identificadores válidos no Delphi, isto pode lhe poupar horas de esforços tentando encontrar erros de codificação e resolvendo mensagens de exceção. Para obter maiores informações sobre regras de outras tabelas/bancos de dados, consulte o Help do Database Desktop ou do próprio banco de dados sendo utilizado.

Segue abaixo, uma tabela com a relação dos tipos e tamanhos dos campos em Paradox:

Símbolo	Tamanho	Tipo
A	1-255	Alpha - Pode conter strings (texto) formadas por letras, números e símbolos especiais ($, %, #, *, =) É o campo que você utilizará para armazenar Nome, Endereço, Telefone, CEP, etc.
N	Automático	Number - Pode conter apenas números, valores positivos ou negativos. Os valores aceitos estão na faixa de -10^{307} a 10^{308} com uma precisão de até 15 dígitos.
$	Automático	Money - Campo numérico, pode conter apenas números, valores positivos ou negativos. É formatado para exibir símbolo monetário e separar as casas decimais.
S	Automático	Short - Pode conter apenas números no intervalo de -32.767 a 32.767
I	Automático	Long Integer - Campo de 32 bits, pode conter apenas números no intervalo de -2147483648 to 2147483647. Requer espaço para armazenamento maior do que o campo Short.
#	0 – 32	BCD - Armazena números no formato BCD (Binary Coded Decimal). São utilizados para executar cálculos de alta precisão, porém a velocidade dos cálculos é lenta, se

Símbolo	Tamanho	Tipo
		comparada com os outros tipos numéricos. Usado para compatibilidade com outros aplicativos que utilizem este tipo de campo. (O tamanho representa o número de dígitos após o ponto decimal)
D	Automático	Date - Armazena qualquer data no intervalo de 1 de janeiro de 9999 AC até 1 de janeiro de 9999 DC.
T	Automático	Time - Armazena horários no padrão de 24 horas. (A exibição pode ser modificada dentro dos programas).
@	Automático	Timestamp - Armazena uma combinação de data e hora com as mesmas características dos campos separados (Date e Time)
M	1 – 240	Memo - Campos memo são similares aos campos Alpha, porém não possuem a limitação de 255 caracteres de armazenamento. O valor definido em tamanho representa a quantidade de caracteres que será armazenada como um campo na tabela, o restante dos caracteres digitados serão armazenados em um arquivo separado com o mesmo nome da tabela e extensão .MB. Se a maior parte de seus memos são de 50 caracteres, você poderia definir este campo com tamanho 50, assim, os dados seriam armazenados na tabela principal, sem necessidade de acessar o arquivo .MB, este seria acessado somente quando o tamanho de um campo ultrapassasse este limite. A quantidade de informação armazenada é limitada apenas pelo espaço em disco disponível em seu Hard Disk.
F	0 – 240	Formatted Memo - Este campo é similar ao campo Memo, porém consegue reter informação de formatação como definição de Tipo e tamanho de Fonte (letra), Estilo (Negrito, Itálico, etc.) e cor. Definindo o tamanho em 0, todo o texto será armazenado em um arquivo externo .MB, como acontece com o Memo..
G	0 - 240	Graphic - Armazena gráficos criados por aplicativos de desenho como Paint do Windows, ou imagens scaneadas. Dentro do Database Desktop podemos selecionar imagens de formatos .BMP, .PCX, .TIF, .GIF e .EPS, porém eles serão convertidos para o formato BMP quando forem armazenados. A melhor prática é definir seu tamanho como 0 pois armazenará toda a imagem num arquivo separado.
L	Automático	Logical - Armazena valores lógicos (boolean) verdadeiro (true) ou falso (false).
+	Automático	Autoincrement - Campo numérico que contém valores do tipo long integer, apenas para leitura (não pode ser alterado). Inicia com o número 1 e este valor é incrementado em um a cada registro adicionado a tabela.

Símbolo	Tamanho	Tipo
		Normalmente é usado para criar um campo de numeração automática como um código de cliente, por exemplo. A exclusão de registros não faz com que esta numeração recomece ou retroceda. E um número excluído jamais será inserido novamente, o valor será sempre crescente.
B	0 - 240	Binary - Utilizado para armazenar informações que não devem ser interpretadas, devem ser armazenadas em modo binário. A utilização mais comum é o armazenamento de som. Assim como campos Graphic, seu tamanho não precisa ser obrigatoriamente ser definido, pois a informação será armazenada em um arquivo .MB.
Y	1 - 255	Bytes - Armazena informações que não devem ser interpretadas, devem ser armazenadas em modo binário. A utilização mais comum é o armazenamento de códigos de barra e magnetic strips de cartões. Diferente dos campos Binary, a informação é armazenada na tabela Paradox para um acesso mais rápido. (Tamanho máximo de 255 bytes)

Para darmos prosseguimento ao desenvolvimento de nosso sistema de Controle de Pedidos, apresentaremos a baixo as definições de estrutura das tabelas de nosso bancos de dados que será formado pelas quatro tabelas que descrevemos anteriormente em nosso *DER*: Clientes, Produtos, Pedidos e Itens do Pedido.

Tabela de Clientes: Clientes.DB

Nº	Nome do Campo	Tipo	Tamanho	Chave
1	Código	+		Sim
2	RazaoSocial	A	35	
3	Fantasia	A	20	
4	Tipo	A	1	
5	CGC_CPF	A	14	
6	Insc_RG	A	15	
7	Endereco	A	35	
8	Bairro	A	20	
9	Cidade	A	30	
10	UF	A	2	
11	CEP	A	8	
12	Telefone	A	15	
13	Fax	A	15	
14	EMail	A	30	
15	HomePage	A	50	
16	Obs	F		
17	Cadastro	D		

Nº	Nome do Campo	Tipo	Tamanho	Chave
18	Ativo	L		
	Índices Secundários:			
Nº	**Nome**	**Campos / Expressão de classificação**		
1	IRazaoSocial	RazaoSocial		
2	ICidadeUF	Cidade e UF		

Tabela Produtos: Produtos.DB

Nº	Nome do Campo	Tipo	Tamanho	Chave
1	Codigo	I		Sim
2	Descricao	A	35	
3	Preco	$		
4	Unidade	A	5	
	Índices Secundários:			
Nº	**Nome**	**Campos / Expressão de classificação**		
1	IDescricao	Descrição		

Tabela de Pedidos: Pedidos.DB

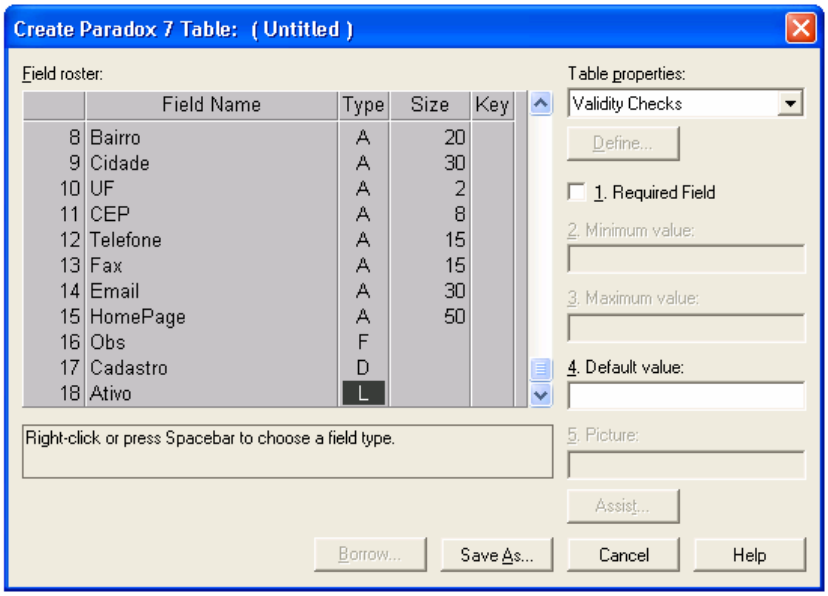
Nº	Nome do Campo	Tipo	Tamanho	Chave
1	NumeroPedido	+		Sim
2	Cliente	I		
3	DataPedido	D		
4	FormaPgto	A	30	
	Índices Secundários:			
Nº	**Nome**	**Expressão de classificação**		
1	ICliente	Cliente e NumeroPedido		
2	IDataPedido	DataPedido		

Tabela de Itens do Pedido: Itens.DB

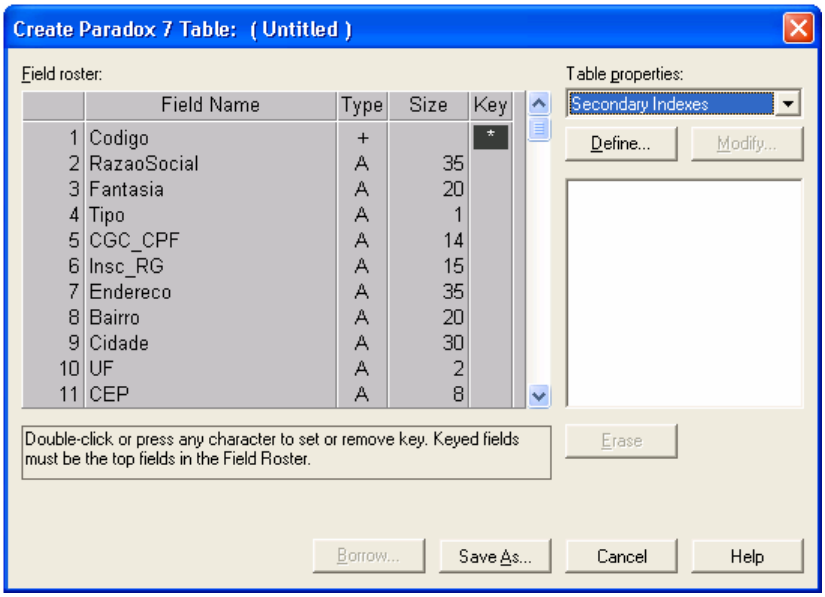
Nº	Nome do Campo	Tipo	Tamanho	Chave
1	NumeroPedido	I		Sim
2	CodigoProduto	I		Sim
3	Quantidade	N		
4	ValorUnitario	$		

Obs: As estruturas definidas para as tabelas, visam apenas a aplicação didática do desenvolvimento de software para banco de dados. O sistema que iremos desenvolver será funcional e com ele aprenderemos todas as técnicas necessárias para criação de sistemas versáteis, rápidos e precisos. No entanto, para o desenvolvimento profissional de software, devemos trabalhar sobre uma análise mais profunda do sistema e da empresa onde este será implantado. As tabelas certamente teriam muitos outros campos e provavelmente outras tabelas também seriam necessárias. Porém, como o objetivo deste sistema é apenas o aprendizado, manteremos estas tabelas desta forma mais simplificada, para facilitar o processo de compreensão.

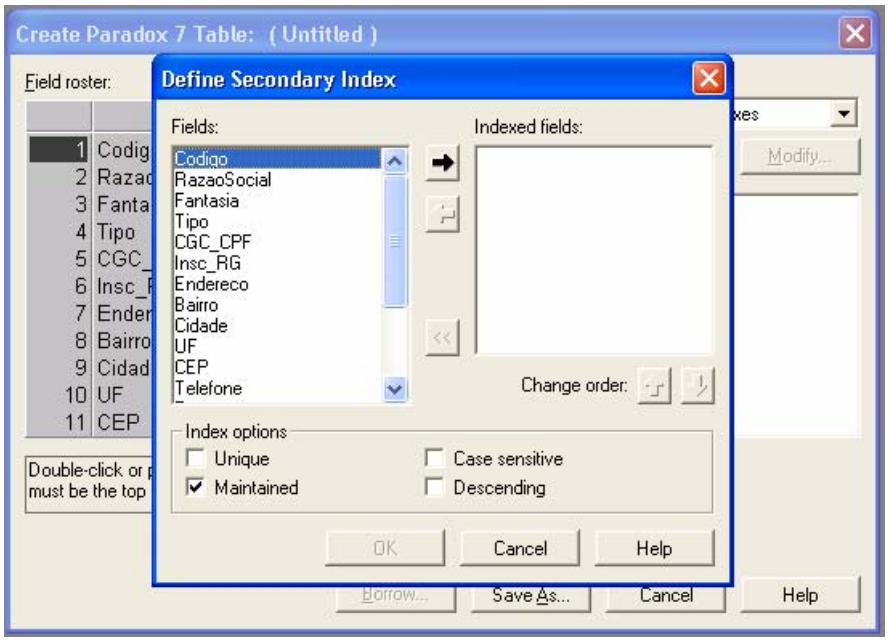
Vamos iniciar criando passo a passo a primeira tabela, Clientes. Digite em Field Name o nome do primeiro campo, *CODIGO*, na coluna Type digite +, para criar um campo autoincrement e na coluna Key dê um duplo clique ou pressione uma tecla para marcar este campo como sendo campo chave. Proceda da mesma forma para os campos seguintes (observe que na definição da tabela apenas o primeiro campo é campo chave, então não marque os demais).
Após digitar os campos sua tela deve ficar semelhante a esta:



Agora vamos criar o índice secundário. Para isso, na caixa de combinação (combo) **Table properties**, selecione a opção **Secundary Indexes**.
O lado direito da caixa de diálogo será alterado e ficará semelhante à figura a seguir:

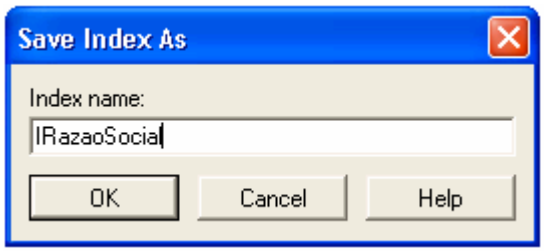


Clique no botão Define e uma nova caixa de diálogo será aberta:

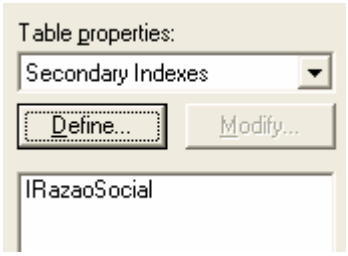


Clique no campo RazaoSocial na coluna **Fields**, selecionando-o. Em seguida, clique no botão com a seta para direita, o que irá transferir o campo para a coluna **Indexed fields**. Sua janela deverá ser algo como a figura abaixo.

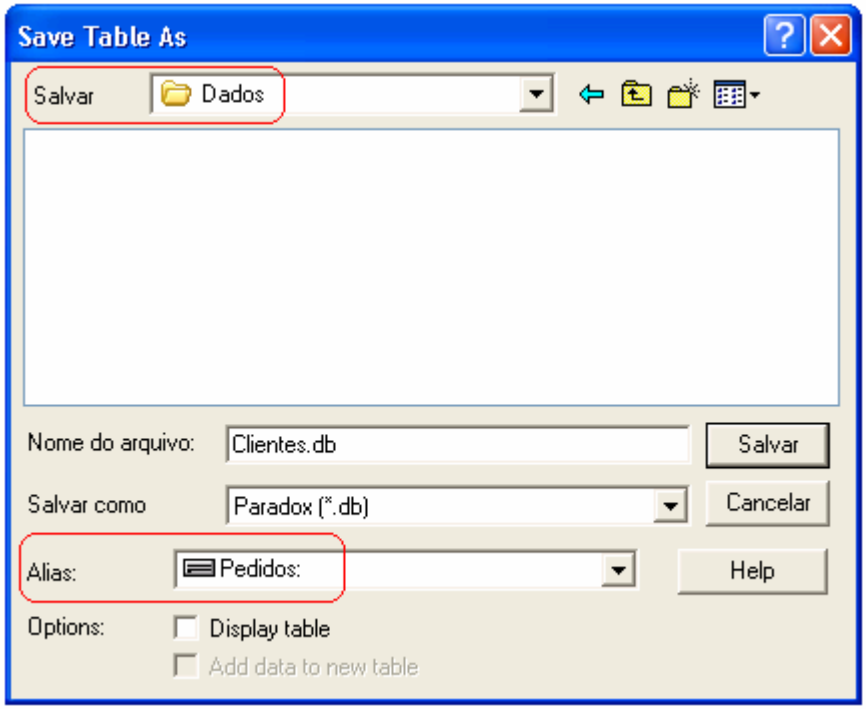
Clique no botão OK e na caixa de diálogo **Save Index As** digite o nome do índice secundário (**IRazaoSocial**), clicando em OK novamente.



Você retornará a caixa de diálogo anterior, no entanto repare que na caixa de listagem aparece o nome do índice recém criado.



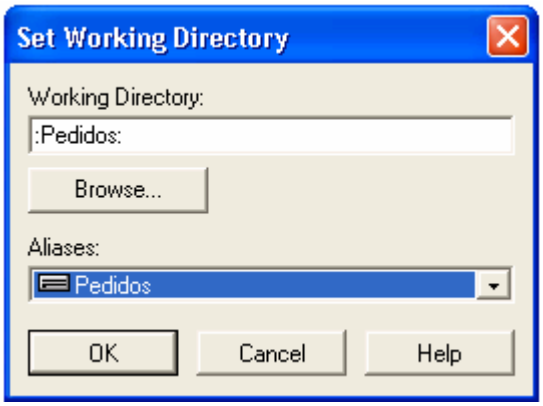
Clique no botão **Save As** para salvar nossa tabela. Na caixa de diálogo **Save Table As** selecione no combo Alias o alias que criamos para nosso banco de dados (**Pedidos**). Note que ao selecionar o alias, a combo Salvar indica a Pasta **Dados**, para onde nós definimos o path do mesmo.



Em nome do arquivo, digite o nome que será dado a tabela, neste caso Clientes.db (ou apenas Clientes, pois a extensão será adicionada automaticamente de acordo com o tipo de arquivo sendo criado).

Bem, com isso criamos nossa primeira tabela. Utilize o mesmo procedimento para a criação das outras três, observando para selecionar os campos marcados como chave.

Antes de você criar as tabelas restantes, vamos a uma dica: Para que você não precise selecionar o alias toda vez que for gravar uma nova tabela que criar ou ao abri-la para modificação, podemos informar ao Database Desktop que estaremos trabalhando com um alias em particular, tornando-o o alias padrão. Para isso vá ao menu **File | Working directory** e selecione o alias **Pedidos** clicando em seguida no botão **OK**.



Está feito, dá próxima vez que for salvar a tabela, o nosso alias já estará selecionado.

Crie as tabelas restantes, pois necessitaremos delas nas próximas aulas, onde estaremos iniciando a criação do nosso aplicativo.

Exercícios

1) O que são Classe e Objeto?
2) O que são método e propriedade?
3) O que é e para que serve o BDE?
4) Qual a vantagem de se utilizar Alias?
5) Qual a função do Database Desktop?

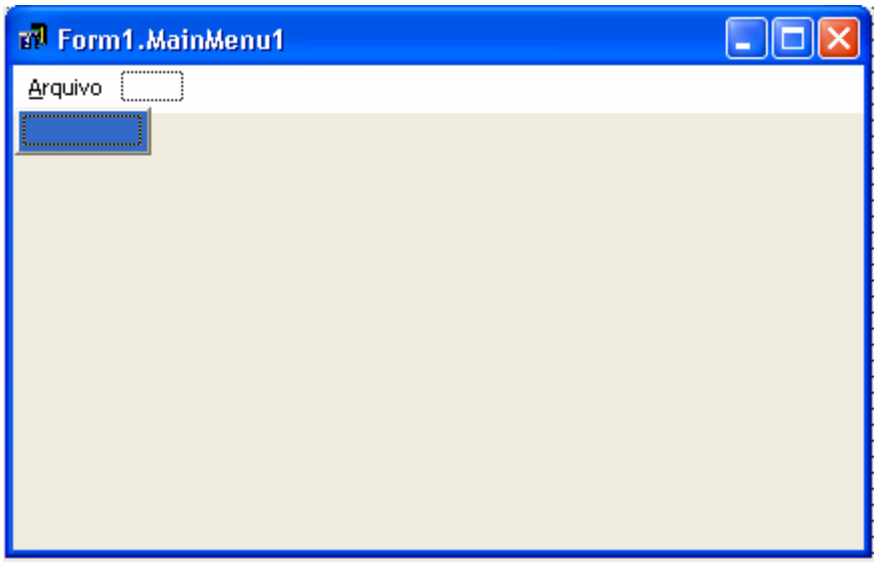
Capítulo 3

Criando nosso Formulário Principal

A partir de agora, passaremos a desenvolver uma aplicação que vai usar os recursos básicos para a maioria das aplicações. Conforme vamos incrementando a aplicação, veremos os principais conceitos da programação em Delphi. Como definimos anteriormente, nossa aplicação será um mini sistema de pedidos. O sistema manterá cadastros de clientes e produtos e será capaz de gerar e armazenar os pedidos dos clientes.

Criando o Menu

Como primeiro passo, cuidaremos do sistema de menus. O componente do menu é o segundo na página Standard da paleta de componentes (o primeiro, Frames, na verdade não é um componente, apenas permite inserir um frame dentro do form). Coloque o menu no form. O ícone aparece mas ainda não há nenhuma barra de menu. Para entrar no Menu Editor (editor de menu) é necessário um duplo clique no ícone do componente que foi colocado form. Note que quando o editor é ativado o *Object Inspector* passa a mostrar as propriedades e eventos dos itens de menu.
Selecione a propriedade *Caption* e digite *&Arquivo.* O menu passa a mostrar agora uma opção para Arquivo e um retângulo pontilhado ao lado. Note que o **&** utilizado antes da palavra **Arquivo** serviu para identificar que a letra **A** será um atalho para esta opção de menu, que poderá ser acessada no sistema através das teclas Alt+A.



Clique no retângulo, selecione a propriedade *Caption* e digite *C&adastros*. Repita a operação e coloque mais um ítem para *A&juda*.
Neste ponto já temos o que num sistema DOS chamaríamos de menu principal. As opções que aparecem horizontalmente na barra de menu, normalmente ativam sub-menus que oferecem mais opções, e estas eventualmente podem abrir mais sub-menus, num leque que pode crescer bastante, se necessário. (Embora seja prudente não estender em excesso esta estrutura, pois tornaria o sistema complexo para o usuário final).
Para criar um sub-menu do menu principal, basta clicar na opção desejada e utilizar o retângulo que aparece abaixo da opção. Crie agora os seguintes sub-menus:

Para Arquivo: &Sair
Para Cadastro: &Clientes, &Produtos e P&edidos
Para Ajuda: &Sobre o sistema...

Observe que quando escolhemos as letras a que funcionarão como atalho para as opções do menu através de ALT + <letra escolhida>. As letras não precisam ser necessariamente a primeira letra da palavra. Também é preciso tomar cuidado para escolher letras diferentes para cada opção no mesmo nível. Um exemplo é o sub-menu Pedido dentro do menu Cadastro, se não tivéssemos escolhido a primeira letra teríamos dois itens com o mesmo atalho.

Para criar sub-menus em sub-menus, selecione a opção que será a raiz e pressione CTRL + Seta Direita. Crie as opções *Cadastro* e *Listagem* no cadastro de Clientes e no de Produtos. No cadastro de Pedidos crie três opçoes: Cadastro, Impressão e Listagem. O menu completo fica assim:

Arquivo	Cadastros	Ajuda
Sair	Clientes	Sobre o sistema...
	Cadastro	
	Listagem	
	Produtos	
	Cadastro	
	Listagem	
	Pedidos	
	Cadastro	
	Impressão	
	Listagem	

Antes de fechar definitivamente o editor de menus, temos ainda outra tarefa. Note que a opção Cadastro aparece em 3 sub-menus. Em cada um deles, o item de menu para Cadastro deve receber um nome diferente, e o Delphi resolve o problema chamando estes itens de *Cadastro1*, *Cadastro2* e *Cadastro3*. Apenas para deixar o código fonte mais claro e legível, poderíamos renomear estes itens para algo como *mnuCadCli*, *mnuCadProd* e *mnuCadPed*. Estamos usando o prefixo *mnu* para indicar que é um item de menu. É um bom hábito utilizar prefixos que indiquem a classe do componente, isso facilita muito

a leitura do código fonte. Para fixar melhor este hábito, renomeie também as opções de Listagem.

Agora já podemos fechar o editor de menus, clicando no botão fechar na barra de título da janela.

Falta agora criar o código para cada opção do menu. A maioria das opções deixaremos para codificar mais adiante, de imediato vamos apenas codificar a opção de saída do sistema. Já com o editor de menus fechado, selecione a opção *Arquivo/Sair* no próprio form da aplicação. O editor de código será ativado. Na área onde o cursor aparece piscando escreva **Close;** Este método será responsável por fechar este Form, como estamos no formulário principal a aplicação toda será fechada.

O código do programa ficará assim:

```
procedure TForm1.Sair1Click(Sender: TObject);
begin
   Close;
end;
```

Clique no Form para selecioná-lo e altere a propriedade **Name** do mesmo para FormPrincipal. Na propriedade **Caption** digite **Controle de Pedidos**.

Observe que ao trocar o Name do form o Delphi fez as modificações no código fonte também.

```
procedure TFormPrincipal.Sair1Click(Sender: TObject);
begin
   Close;
end;
```

Note também que o componente de menu que incluímos foi adicionado à definição da classe do Form, juntamente com os itens de menu que incluimos através do Menu Editor. Isso pode ser visto no editor de código se você for para o início da unit. A nova definição será semelhante ao exemplo abaixo:

```
type
  TFormPrincipal = class(TForm)
    MainMenu1: TMainMenu;
    Arquivo1: TMenuItem;
    Cadastros1: TMenuItem;
    Ajuda1: TMenuItem;
    Sair1: TMenuItem;
    Clientes1: TMenuItem;
    Produtos1: TMenuItem;
    Pedidos1: TMenuItem;
    mnuCadCli: TMenuItem;
    mnuListCli: TMenuItem;
    mnuCadProd: TMenuItem;
    mnuListProd: TMenuItem;
    mnuCadPed: TMenuItem;
```

```
    mnuImpPed: TMenuItem;
    mnuListPed: TMenuItem;
    procedure Sair1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormPrincipal: TFormPrincipal;
```

Salvando o Projeto

Salve o projeto na pasta que criamos (**C:\Pedidos**), como aprendemos na nossa primeira aula. Caso não se recorde, vamos fazer juntos para fixar:

Clique no botão *Save All* do SpeedBar. Na caixa de diálogo *Save Unit1 As* no item *Nome do arquivo* digite **Principal.pas** e verifique a pasta que está selecionada, se não for a pasta Pedidos localize e selecione-a antes de clicar no botão *Salvar* ou teclar Enter.

Na caixa de diálogo *Save Project1 As* no item *Nome do arquivo* digite o nome que daremos ao projeto, **Pedidos.dpr**.

Definindo o tipo de aplicação

Se você executar o projeto, notará que o menu já está funcionando, incluindo a opção Sair do menu Arquivo, que encerra a execução do mesmo. Também deve ter notado que o formulário aparece na mesma posição e do mesmo tamanho que estava quando o desenhamos. Se quisermos, por exemplo, que ele apareça maximizado, poderemos fazer isso definindo a propriedade **WindowState** em nosso Form. Para isso, vá até o Object Inspector e defina WindowState para **wsMaximized** (Não esqueça de fechar a janela se o estiver executando).

Também devemos decidir que tipo de aplicação nós teremos. Basicamente existem dois tipos de aplicativos, SDI (Simple Document Interface – Interface Simples de Documentos) e MDI (Multiple Document Interface - Interface de Múltiplos Documentos). O SDI é tipo padrão, onde cada form é independente do outro. Se tivermos um formulário principal com um menu e acionarmos um segundo formulário, poderemos minimixar o formulário principal e este outro continuaria ativo e ainda em alguns casos poderiam aparecer dois botões na barra de tarefas do Windows.

Já em um aplicativo MDI temos um formulário principal (Pai), que contém todos os demais formulários (filhos). Assim, se você minimizar o principal, todos os filhos são minimizados com ele. Apenas um botão é exibido na barra de tarefas e os formulários filhos também não podem ser arrastados para fora da área do formulário Pai.

Poderíamos trabalhar com um formulário principal SDI maximizado ou reduzindo-o ao menu e uma barra de ferramentas, como é o caso do próprio Delphi. Existe até um exemplo deste estilo de sistema dentro do diretório de exemplos que acompanha o Delphi. Se você fez a instalação mantendo diretórios sugeridos poderá encontrar este projeto em:

C:\Arquivos de programas\Borland\Delphi6\Demos\Db\MastApp.

Optamos por trabalhar com um formulário MDI para criar uma interface próxima da qual a maior parte dos usuários do Windows já estão acostumados ao utilizarem editores de texto como o Microsoft Word que trabalham desta forma. (Obs: A partir da versão 2000 do Office o Word deixou de ser um aplicativo puramente MDI, pois se abrirmos dois documentos o mesmo exibe dois botões na barra de tarefas do Windows).

Para criarmos um Formulário MDI devemos definir a propriedade **FormStyle** como sendo **fsMDIForm**.

Obs: Só pode existir um formulário com *FormStyle* definido com *fsMDIForm* por projeto. Os formulários filhos (Children Forms) deverão ter seu *FormStyle* definido como *fsMDIChild*. (Não há limite para o número de formulários filhos).

Preparando o Acesso aos Dados

Para fazer o acesso aos dados, poderíamos criar um DataModule, uma espécie de formulário especial onde podemos colocar componentes de acesso a dados e outros componentes da categoria *Invisible components* (componentes que não criam interface com o usuário, e por isso não são visíveis em tempo de execução). No entanto o uso do DataModule aumentaria um pouco mais o nível de dificuldade pois teríamos que fazer muitas outras considerações. Assim, optamos por manter os componentes de acesso a dados dentro dos próprios formulários, o que de certa forma também deixará nossa aplicação mais leve no final.

Vá até a guia *BDE* da paleta de componentes, selecione o componente *DataBase* e adicione-o ao Form. Para acesso a tabelas Paradox isso é desnecessário, visto que o mesmo só precisa da informação de path do alias, mas facilitaria para uma futura mudança para um banco de dados cliente-servidor, assim não vemos razão para não utiliza-lo sempre. Defina no componente DataBase as seguintes propriedades:

Propriedade	Conteúdo
AliasName	Pedidos
DatabaseName	DBPedidos
Name	DatabasePedidos

A propriedade AliasName indica qual alias do BDE iremos utilizar e DatabaseName define qual será o nome interno desse alias. Assim, caso instale seu sistema em um micro que já possua um sistema em Delphi e que por infelicidade utilize o mesmo alias, você pode simplesmente criar um outro o nome para o seu alias e selecionar este nome novo no AliasName do seu componente DataBase.

Salve as alterações que fizemos até agora clicando no botão *Save* do SpeedBar ou utilizando o menu do Delphi se preferir. Habitue-se em ir salvado seu projeto na medida que o vai modificando ou adicionando novos códigos e componentes. Pois no caso de uma queda de energia ou travamento do Windows o que não estiver salvo será perdido, como em qualquer outro programa.

Criando o Cadastro de Clientes

Para criarmos nosso cadastro de Clientes devemos incluir um novo formulário no projeto. Fazemos isso clicando no botão *New Form* do SpeedBar ou então no menu **File | New | Form**.
Inserido o novo Form, teremos novamente um Form1. Defina as propriedades deste Form como na tabela abaixo:

Propriedade	Conteúdo
Name	FormClientes
Caption	Cadastro de Clientes
FormStyle	fsMDIChild
Position	poMainFormCenter

A propriedade Position define em qual a posição o Form irá aparecer::

- poDesigned: o Form aparece na posição que estava no projeto (design time).
- poDefault: Tanto a posição do Form (Top e Left) como seu tamanho (Height e Width) é definida pelo sistema operacional. A cada execução o Form é exibido à direita e abaixo da posição anterior, formando o que chamamos de alinhamento de janelas em cascata.
- poDefaultPosOnly: O tamanho definido no projeto será mantido, mas a posição é definida pelo sistema operacional.
- poDefaultSizeOnly: A posição definida na projeto é mantida, mas o tamanho pode ser alterado pelo sistema operacional.

- poScreenCenter: O Form será exibido no centro da tela virtual, caso se utilize dois monitores o mesmo aparecerá centralizado na área definida pelos dois, ficando assim metade do form em cada monitor.
- poDesktopCenter: O Form será centralizado considerando-se apenas a área de trabalho do monitor, caso se utilize dois monitores ele não considerará o segundo.
- poMainFormCenter: aparece no centro de um Form MDI (pai), é utilizada para formulários com FormStyle igual a fsMDIChild.
- poOwnerFormCenter: Mantém o tamanho definido no projeto, mas centraliza o Form no centro do formulário definido na propriedade Owner. Caso a mesma não seja definida o resultado será o mesmo de poMainFormCenter.

Agora vamos começar a criar a interface. Na guia Stardard, selecione o componente Panel e coloque-o no Form. Defina as propriedades assim:

Propriedade	Conteúdo
Align	alBottom
Caption	*Apague o conteúdo deixando em branco*
Name	PanelBotoes

Observe que Align define o alinhamento deste componente, assim definindo alBottom o alinhamos na parte inferior do Form ou objeto que o contém.
Adicione um ScrollBox (ele criará uma caixa com barra de rolagem), que se encontra na guia Additional, tendo o cuidado de coloca-lo na área do form que não contém o primeiro Panel. Para este componente defina as propriedades:

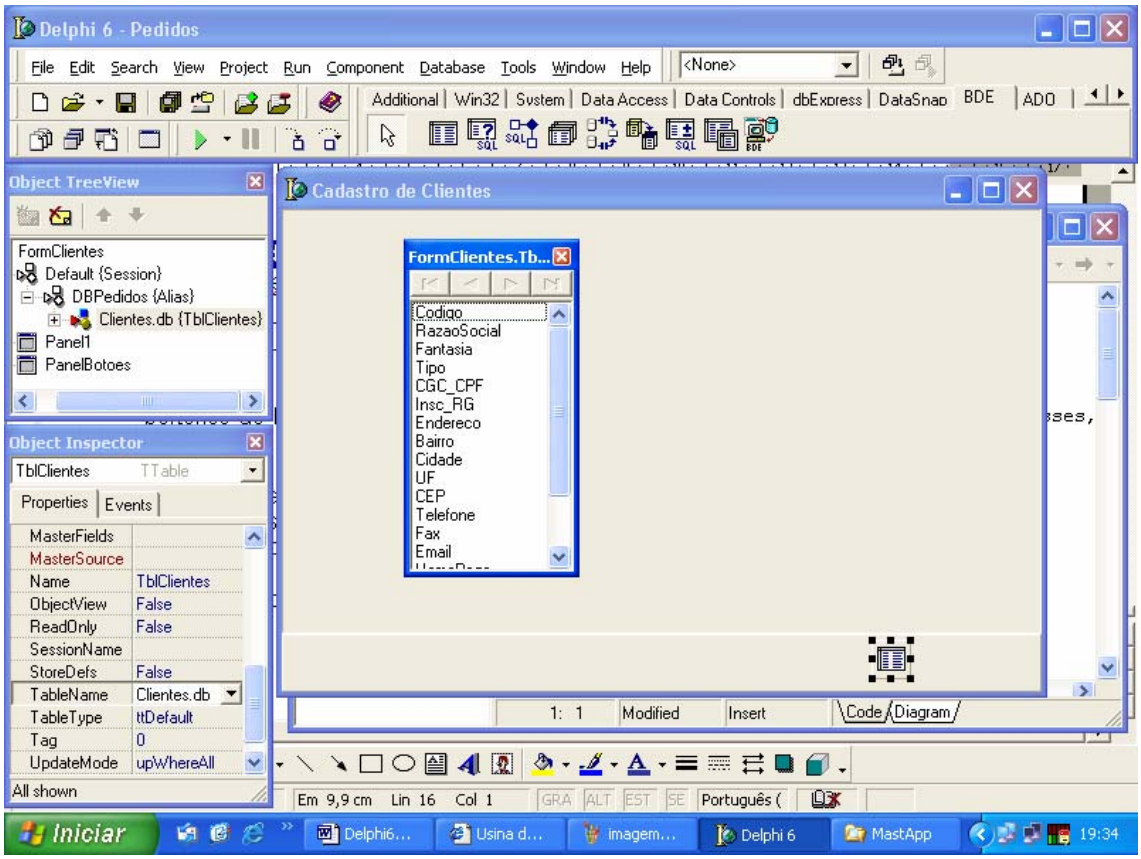
Propriedade	Conteúdo
Align	AlClient
Name	ScrollBoxDados

Agora vamos inserir o componente que dará acesso a nossa tabela. Vá até a guia BDE, selecione o componente Table e coloque-o no Form. Como este é um componente invisível o local onde ficará não tem muita importância, então coloque no fim do Form, para não atrapalhar sua visão do local onde colocaremos os objetos para digitação de dados.
Precisamos agora conectar nossa Table ao alias que desejamos acessar ou ao componente Database que aponta para esse alias. Como nosso componente Database está no nosso FormPrincipal, devemos permitir que este Form possa visualizar o outro. Para isso vá até o menu **File | Use Unit** (ou pressione Alt+F11) e selecione a unit que você quer permitir visualizar dentro desta. Selecione **Principal** (que é a única que temos além da atual) e clique em OK.
Agora configure o componente Table como segue:

Propriedade	Conteúdo
DatabaseName	DBPedidos
Name	TableClientes
Table	Clientes.db

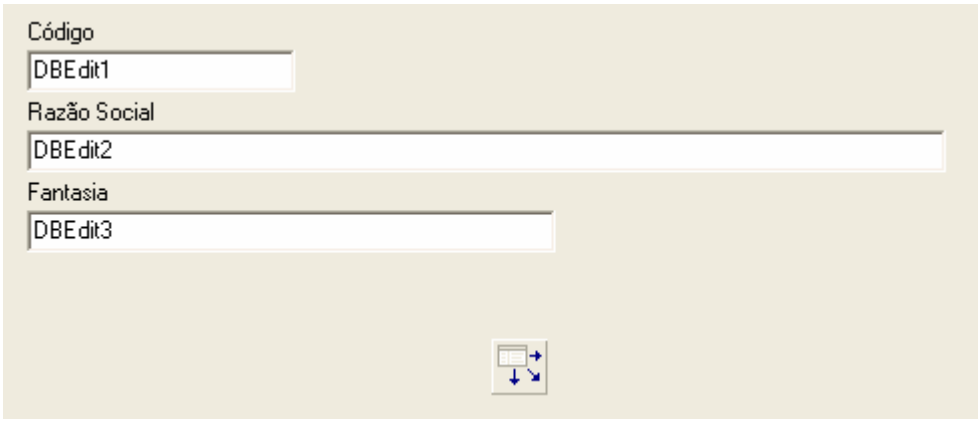
Assim, definimos que este componente irá acessar a tabela Clientes.db que pertence ao Database DBPedidos, que por sua vez sabemos que aponta para o alias Pedidos que criamos no início.

Dê um clique duplo no componente Table de seu Form. Será exibido o Fields Editor. Pressione Ctrl+F e serão adicionadas definições de campo para cada um dos campos da tabela. Você também pode fazer isso clicando com o botão da direita do mouse sobre o Fields Editor e selecionando a opção *Add all fields* no menu de contexto do mesmo. Seu ambiente de trabalho deve estar apresentando algo assim:



Clique no primeiro campo da lista, selecionando-o. Vá ao Object Inspector e defina a propriedade DisplayName para Código. (O DisplayName servirá para preencher a propriedade Caption dos respectivos Label's que serão inseridos). Faça o mesmo com os campos seguintes, alterando por exemplo, RazaoSocial para Razão Social, CGC_CPF para CGC/CPF, Insc_RG para Insc.Estadual/RG e assim por diante.

Agora volte ao primeiro campo da lista, segure a tecla Shift e utilizando a seta para baixo selecione até Fantasia. Agora arraste os campos selecionados para o Form, na área definida pelo ScrollDados. Você deve obter algo assim:



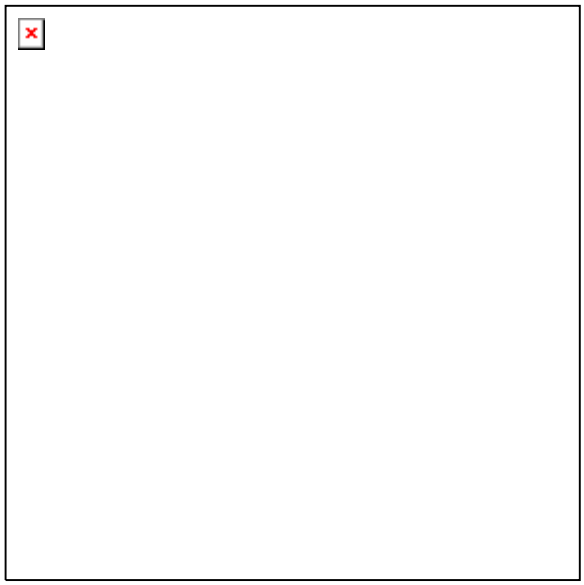
Observe que os campos selecionados ao serem arrastados inseriram um Label com o Texto definido na propriedade DisplayLabel do mesmo e um objeto DBEdit, responsável por permitir a edição de dados, já configurado para acessar este campo. Também foi inserido um componente DataSource. Este é responsável por interligar os dados acessados pelo componente Table com os componentes de exibição de dados (da guia Data Controls). Vamos continuar com a criação do nosso Form e depois explicaremos mais detalhadamente como estes componentes se relacionam.

Como o componente DataSource também é invisível mova-o para uma área livre, como ao lado do Table e altere sua propriedade Name para DataSourceClientes (ou DSClientes se preferir um nome mais curto).

Para o campo tipo, vamos usar um outro tipo de controle, então iremos inseri-lo manualmente. Vá até a guia Data Controls, selecione o componente DBRadioGroup e coloque-o no Form. Defina as seguintes propriedades:

Propriedade	Conteúdo
Caption	Tipo
Columns	2
DataField	Tipo
DataSource	DataSourceClientes
Height	38
Items	Pessoa Física Pessoa Jurídica
Name	DBRadioGroupTipo
Values	F J
Width	208

As propriedades Items e Values definem respectivamente os textos que serão exibidos no objeto DBRadioGroup e quais os valores serão gravados na tabela. Estas propriedades possuem um editor (botão com reticências), para defini-las clique no botão e informe os textos um em cada linha, como no exemplo abaixo:



Clique em OK para voltar ao Object Inspector novamente. E defina a propriedade Values da mesma forma
.
Tenha o cuidado de ir arrumando os campos no Form, de maneira a abrir espaço para os demais campos e procurando deixar o form com uma aparência mais agradável. Também seria prudente você salvar seu Form a esta altura, para evitar o risco de perder tudo o que já fez no caso de uma queda de energia ou um esbarrão no estabilizador (o que não é muito incomum). Salve esta Unit como **Clientes.pas**.

Abra o Fields Editor novamente, com um clique duplo no componente Table. Selecione agora os campos CGC_CPF e Insc_RG e arraste-os para o Form.
Faça o mesmo com os outros até checar ao campo Obs, pois neste campo também utilizaremos outro componente.

Talvez seja necessário redimensionar seu Form dependendo da aparência que você deseja dar ao mesmo. Para fazer isso você pode alterar os valores nas propriedades Height e Width do Form ou então modifica-lo com o mouse, como faria com qualquer outra janela ou ainda utiliza-lo maximizado caso prefira.

Insira um Label (guia Standard) e defina seu Caption como Observação. Abaixo do Label insira um componente DBRichEdit definindo suas propriedades para:

Propriedade	Conteúdo

DataField	Obs
DataSource	DataSourceClientes
Name	DBRichEditObs

Volte ao Fields Editor e insira o campo Cadastro, se ainda não o Fez.
O último campo, Ativo nós também definiremos manualmente, será um componente DBCheckBox (guia Data Controls). Insira-o no Form e defina suas propriedades para:

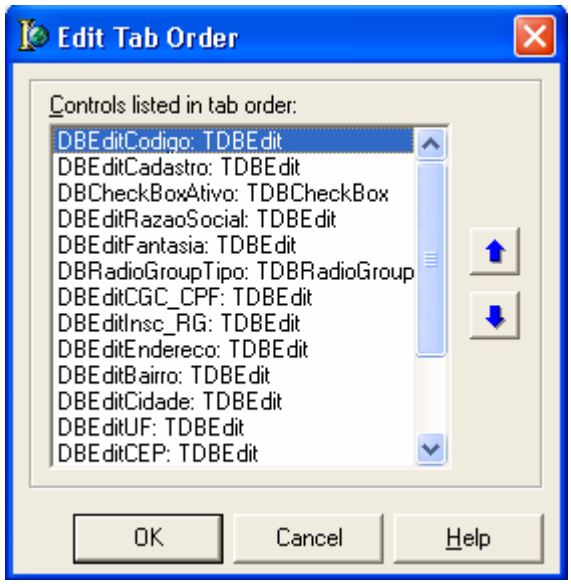
Propriedade	Conteúdo
Caption	Cliente Ativo
DataField	Ativo
DataSource	DataSourceClientes
Name	DBCheckBoxAtivo

Agora vamos voltar aos componentes que foram inseridos automaticamente (arrastados do Fields Editor para o Form) e vamos definir a propriedade Name dos mesmos para algo mais significativo. Sugerimos que utilize DBEdit (nome da classe) como prefixo e adicione no lugar dos números o nome do campo, como fizemos abaixo:

Campo	Name
Código	DBEditCodigo
RazaoSocial	DBEditRazaoSocial
Fantasia	DBEditFantasia
CGC_CPF	DBEditCGC_CPF
Insc_RG	DBEditInsc_RG
Endereço	DBEditEndereco
Bairro	DBEditBairro
Cidade	DBEditCidade
UF	DBEditUF
CEP	DBEditCEP
Telefone	DBEditTelefone
Fax	DBEditFax
Email	DBEditemail
HomePage	DBEditHomePage
Cadastro	DBEditCadastro

Dependendo da maneira como você arrumou os componentes DataControl em seu Form eles podem ter ficado com a tabulação (ordem em que são selecionados ao pressionarmos a tecla Tab) incorreta, para redefinir clique com o botão da direita no ScrollDados em uma área vazia (sem DataControl's) e selecione a opção *Tab Order* no menu de contexto. Será exibida a caixa de

diálogo Edit Tab Order, que lhe permitirá ordenar os objetos na seqüência correta.



Agora, para podermos navegar e modificar os dados vamos inserir o componente DBNavigator. Selecione-o na guia Data Controls e insira-o no PanelBotoes (que nós alinhamos na parte inferior do formulário) e defina suas propriedades como segue:

Propriedade	Conteúdo
DataSource	DataSourceClientes
Flat	True
Hints	Primeiro Registro Registro Anterior Próximo Registro Último Registro Incluir Registro Excluir Registro Alterar Registro Gravar Inclusão/Alteração Cancelar Inclusão/Alteração Reexibir os Dados
Name	DBNavigatorClientes

A propriedade Flat define que os botões terão aparência plana, assumindo a forma de botão apenas quando o mouse passa sobre eles. Esse é o comportamento das barras de ferramenta das versões mais recentes do Windows e do Internet Explorer, bem como do próprio Delphi.
A propriedade Hints define as dicas que serão exibidas para cada um dos botões que fazem parte do DBNavigator. Para que estas dicas seja exibidas você deve definir a propriedade **ShowHint** no Form para **True**.

Por último, incluiremos um botão do tipo BitBtn, que se encontra na guia Additional. Insira-o também no PanelBotoes, no canto Direito e defina as propriedade assim:

Propriedade	Conteúdo
Kind	bkClose
Caption	&Fechar
Name	BitBtnFechar

Salve novamente seu Formulário e vamos adicionar a codificação mínima e coloca-lo para funcionar.

No Object Inspector selecione o Formulário (FormClientes) utilize o combo no topo do mesmo ou então clique em um objeto qualquer do Form e vá teclando Esc continuamente, até que o Object Inspector exiba no seu combo **FormClientes: TFormClientes**. (Essa dica é uma maneira rápida de acessar um objeto que está sobreposto por outro, como é o caso do Form).
Vá para a guia Events o Object Inspector e localize o evento OnCreate. Dê um duplo clique na caixa em frente ao evento. Isso abrirá o editor de código para você. Nele digite o seguinte:

TableClientes.Open; // abre a tabela de dados

O seu editor de código deve estar assim:

```
procedure TFormClientes.FormCreate(Sender: TObject);
begin
   TableClientes.Open; // abre a tabela de dados
end;
```

Localize agora o evento OnDestroy, dê um duplo clique novamente para abrir o editor de código e insira a linha abaixo:

TableClientes.Close; // fecha a tabela de dados

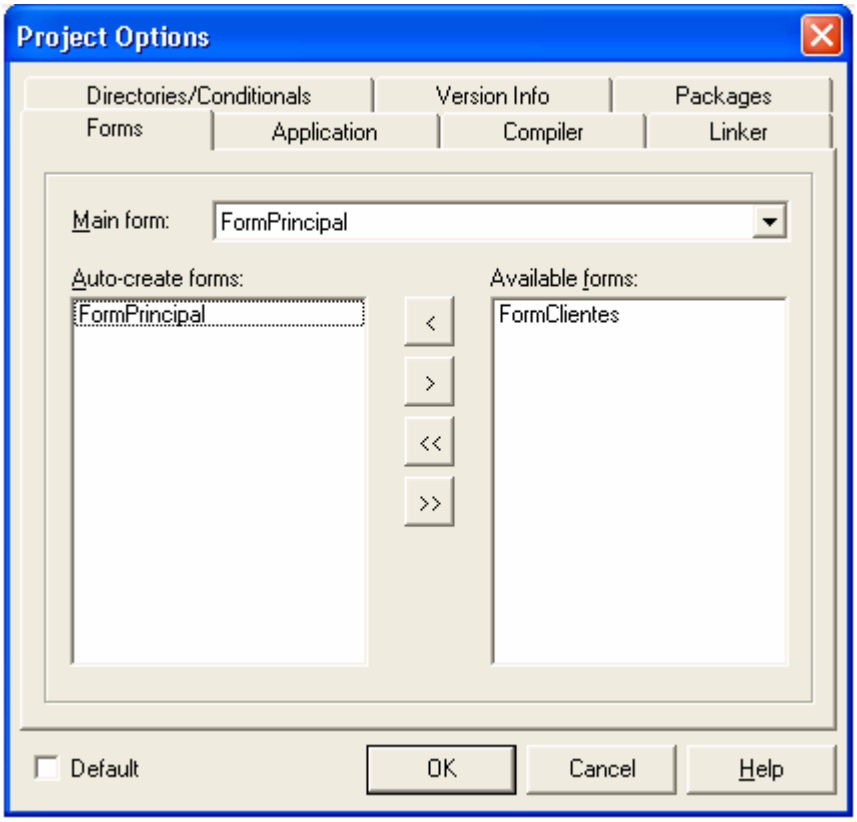
Isso já é o suficiente para que possamos executar nosso formulário. Devemos agora fazer a ligação deste com o menu do FormPrincipal. Isso é feito da seguinte maneira.
Volte ao FormPrincipal, isso pode ser feito clicando-se na guia com o nome Principal no editor de código ou ainda através do botão View Forms do SpeedBar (neste caso você deve selecionar FormPrincipal na lista e clicar em OK).
Selecione no seu FormPrincipal o menu **Cadastros | Clientes | Cadastro**. Isso abrirá o editor de código na opção de menu selecionada. Para chamarmos o FormClientes devemos digitar o seguinte código:

FormClientes.Show; //exibe o formulário

Também precisamos informar que vamos utilizar o FormClientes dentro do FormPrincipal, para isso vá ao menu **File | Use Unit**. E selecione **Clientes** na lista de Unit's. (Caso você não faça este procedimento, o Delphi irá verificar que você está tentando utilizar este Form e irá questiona-lo se ele deve incluir a informação de que você vai utiliza-lo. Na verdade este procedimento insere o nome da Unit na cláusula **uses** da seção **implementation** da unit do form sobre o você chamou a opção **Use Unit**).

Salve o Projeto e execute-o, pressionando F9.
Você deve notar que ao executar seu aplicativo, o formulário do Cadastro de Clientes já aparece aberto. Isso se deve ao fato de que todo formulário filho é exibido no momento em que é criado e o Delphi, por padrão, cria todos os formulários no inicio (quando é executado). Assim se tivessemos 50 formulários filhos todos eles seriam exibidos ao mesmo tempo. Note também que se você tentar fechar o formulário ele será minimizado. Este também é um comportamento padrão para formulários filhos. Como estes comportamentos não são os que desejamos, iremos modifica-los.

Encerre a execução do programa e volte ao Delphi. No menu do Delphi escolha **Project | Options** ou tecle Ctrl + Shift + F11.



Mova o FormClientes da Lista de Auto-create forms para a lista de Available forms, deixando-o como na figura anterior. Clique em OK e o problema do form aparecer sem clicar no menu estará terminado. Devemos adicionar na opção de menu que chama o FormClientes mais uma linha de código, pois agora é nossa a responsabilidade de criar o formulário. O seu código deve ficar assim:

Application.CreateForm(TFormClientes,FormClientes); // cria o form
FormClientes.Show; //exibe o formulário

No FormClientes, devemos codificar um novo evento para eliminar o problema das janelas minimizarem quando tentamos fecha-las. Localize o evento OnClose (do FormClienes), abra sua codificação como foi feito anteriormente e acrescente o seguinte código:

Action := caFree; // remove o form da memória

Este comando fará com que o formulário seja removido da memória quando o fecharmos, resolvendo assim o problema. A esta altura você também deve ter notado que utilizamos duas barras de divisão (**//**) para indicar um comentário. Você pode ver maiores detalhes sobre isso no apêndice **A** que contém a sintaxe básica dos comandos e estruturas de controle do Object Pascal.

Se você executar o aplicativo agora, verá que os problemas anteriores não existem mais. No entanto, caso você acione o menu para abrir o Cadastro de Clientes mais de uma vez, sem fechar a janela na primeira vez, verá que uma nova cópia da janela será criada. Isso permite que se possa estar cadastrando um Cliente e antes de terminar abrir uma nova janela para consultar um outro. Pode ser um recurso interessante, mas também pode ser confuso para usuários menos experientes. Assim, iremos limitar nosso aplicativo a executar apenas uma cópia da mesma janela de cada vez. Para isso, vamos adicionar mais uma linha no código da opção de menu do FormPrincipal que chama o FormClientes. Seu código agora deve ficar assim:

```
if FormClientes = NIL then  //cria o form se ele não foi criado ainda
        Application.CreateForm(TFormClientes,FormClientes);
FormClientes.Show; //exibe o formulário
```

E outra linha de código no evento OnClose do FormClientes:

```
Action := caFree; // remove o form da memória
FormClientes := NIL; // informa que foi destruído (não criado)
```

Com isso, nosso aplicativo já funciona corretamente. Podemos incluir, alterar e excluir informações, bem como nos movimentarmos entre os registros. No entanto, algumas melhorias podem ser feitas:

- Ao clicar na inclusão poderíamos jogar o cursor no primeiro campo que queremos editar, como Razão Social por exemplo;
- Poderíamos trazer a data de Cadastro preenchida com a data do dia;
- Os campos de CGC/CPF, CEP e Cadastro já poderiam vir com máscaras de digitação;
- A mensagem que aparece quando tentamos excluir poderia ser traduzida;
- Seria interessante poder localizar um determinado cliente através de seu Nome.

Certamente, você deve ter pensado em algumas outras também, isso depende muito de como você deseja que seu Form se comporte e que funcionalidade deseja implementar.

Realizaremos estas melhorias no próximo capítulo, neste momento, vamos entender como os dados são acessados e exibidos.

Entendendo como acessamos os dados no Delphi

Para acessar e manipular as tabelas de bancos de dados, devemos utilizar os componentes de acesso a bancos de dados, que estão armazenados nas guias da Paleta de Componentes, de acordo com o tipo de engine utilizado: BDE, dbGO (guia ADO), IBX (guia Interbase) ou dbExpress. No nosso caso, utilizamos o BDE, mas estes conceitos são válidos também para as outras engines de acesso. Os componentes destas guias são responsáveis apenas pelo acesso (leitura/gravação e movimentação).

Para que os mesmos possam ser modificados e visualizados pelo usuário, devemos utilizar os componentes da guia Data Controls. Nesta guia encontramos componentes semelhantes aos da guia Standard, porém criados especificamente para dar acesso aos dados.

Você pode estar se perguntando, qual seria a vantagem de se dividir tanto os métodos de acesso e manutenção e usar toda essa quantidade de componentes. E a resposta é simples, flexibilidade. Cada uma das guias de acesso possui mais de um tipo de componente para acesso a dados, alguns que exploram melhor a capacidade de acesso a tabelas desktop (como é o caso do componente Table do BDE) e outros voltados especificamente para o acesso cliente-servidor (como o componente Query da mesma guia). Separando-se os controles de edição (Data Controls) dos componentes de acesso a dados (BDE, dbGO, IBX ou dbExpress) temos um meio mais fácil de substituir a metodologia de acesso sem contudo afetar a interface visual já construída anteriormente.

Como os componentes de acesso são diferentes (embora possuam propriedades e métodos comuns) para que não fossem necessários Data Controls específicos para cada um, a Borland criou um outro componente, que é responsável por fazer a ligação entre os componentes de exibição e os componentes de acesso. Estamos falando do componente **DataSource**, que se encontra na guia Data Access. Este componente se liga aos componentes de acesso a dados através da propriedade **DataSet**. E os Data Controls se conectam a ele através de uma propriedade também chamada **DataSource**. Assim, se desejarmos trocar os componentes de acesso, simplesmente os inserimos no Form (ou DataModule) e apontamos os DataSource's para os novos componentes de acesso, sem precisar modificar nada nos Data Controls.

Se você já programou em outras linguagens visuais pode ter uma idéia melhor de como esta arquitetura criada pela Borland é realmente superior às outras formas de desenvolvimento utilizadas por outras ferramentas. (A Borland também possui o Borland C++ Builder, uma ferramenta semelhante ao Delphi, mas que é baseada em linguagem C/C++. Nela a forma de desenvolvimento é a mesma utilizada aqui).

Exercícios

1) Qual a função do Menu Editor?
2) Qual a diferença entre aplicações SDI e MDI? Qual tipo podemos criar com o Delphi?
3) Para que servem os componentes da Guia BDE?
4) Qual a função dos componentes da Guia Data Controls?
5) Qual a função do componente DataSource?

Capítulo 4

Aprimorando nosso formulário de Cadastro de Clientes

Caso seu projeto esteja fechado, você pode abri-lo clicando no botão *Open Project* do SpeedBar, teclando Ctrl+F11 ou ainda acessando o menu **File|Open Project**. Se trabalhou recentemente com o projeto, pode tentar um caminho mais rápido, através do menu **File|Reopen**, localizando-o na lista dos últimos projetos abertos.

Retorne ao nosso FormClientes (selecionando-o por exemplo, através do botão View Form no SpeedBar) para que possamos complementa-lo.

Definindo máscaras de edição

Inicialmente, vamos definir máscaras de formatação que irão facilitar a digitação dos dados. Abra o Fields Editor (com um duplo clique no componente Table) e localize o campo CEP. No Object Inspector procure a propriedade **EditMask** e digite: **99999\-999;0;_**
A primeira parte antes da vírgula define a máscara para o CEP, o zero após a vírgula indica que o hífen não será salvo, e o sublinhado no final define o caractere que será exibido quando a posição estiver em branco. (Se achar melhor você pode substituir o sublinhado por um espaço em branco).

Agora localize o campo Cadastro no Fields Editor e voltando ao Object Inspector defina a propriedade EditMask para: **!99/99/9999;1;_**
Esta é a máscara para dadas, a exclamação indica que espaços em branco não serão exibidos, o número 1 indica que as barras (**/**) serão salvas juntamente com os números digitados.

Por último, procure o campo UF no Fields Editor e no Object Inspector defina seu EditMask para: **!ll** (exclamação e duas letras L minúsculas)
A letra **l** indica que será aceito apenas caracteres alfabéticos.

Você encontrará maiores detalhes sobre as máscaras no apêndice A.

Codificando eventos das tabelas de dados

Modificaremos agora o comportamento do nosso aplicativo através da codificação de eventos específicos.

Definindo um valor inicial para um campo

Para que um determinado campo tenha um valor inicial quando acionamos a inclusão, devemos codificar o evento **OnNewRecord**. Para isso, selecione o objeto Table e vá para a guia Events no Object Inspector. Localize este evento e abra o Code Editor com um duplo clique na caixa á direita da propriedade (como fizemos até agora). Na área criada para a codificação do evento digite:

```
TableClientesCadastro.AsDateTime := Date; // atribui a data atual
TableClientesAtivo.AsBoolean := True; // cliente ativo
```

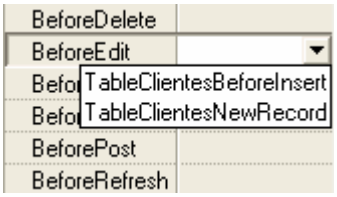
Estamos fazendo assim uma atribuição da data atual (através da função Date) para o campo Cadastro que esta no TableClientes (TableClientesCadastro) sendo do tipo de dados date (AsDateTime). Na segunda linha, definimos que o conteúdo do campo ativo será True (verdadeiro), informando assim que o cliente é ativo (dificilmente cadastraríamos um cliente inicialmente inativo, ele se torna inativo com o tempo).

Posicionando o foco no primeiro item a ser editado.

Quando iniciamos uma inclusão ou alteração pode ser mais prático que o foco esteja num determinado controle, como o DBEdit do campo RazaoSocial. Para isso vamos codificar o evento **BeforeInsert** (que ocorre antes da inclusão):

```
DBEditRazaoSocial.SetFocus; // posiciona o cursor neste controle
```

Como desejamos que a mesma ação seja executada no momento que acionarmos a alteração, devemos codificar a mesma coisa no evento **BeforeEdit**. No entanto, o Delphi fornece um mecanismo maravilhoso (não encontrado em outras linguagens) que permite a você simplesmente indicar que se um determinado evento ocorrer ele deve executar o código contido em outro evento. Para definir isso, simplesmente vá no Object Inspector e localize o evento BeforeEdit. Desta vez, ao invés de dar um clique duplo selecione no combo como é mostrado na figura abaixo:



Traduzindo a mensagem de Exclusão

A mensagem “Delete Record?” que aparece quando você tenta excluir um registro pertence ao componente DBNavigator. Para podermos traduzi-la devemos desativar esta mensagem. Para isso selecione este componente no seu Form e defina a propriedade **ConfirmDelete** para **False**. O próximo passo é codificar no Table o evento BeforeDelete (que ocorre antes da exclusão).

Localize-o e abra o editor de código (com um clique duplo), digitando os comandos a seguir:

```
if Application.MessageBox('Deseja excluir o registro atual?','Confirme',
   MB_YESNO + MB_ICONQUESTION + MB_DEFBUTTON2) = IDNO then
   Abort; //interrompe e não exclui o registro
```

Isso criará uma caixa de diálogo com o título “Confirme” e a mensagem “Deseja excluir o registro atual?”, exibindo os botões “Sim” e “Não” e o ícone padrão para perguntas. O segundo botão (“Não”) foi definido como botão padrão. Observe também que as mensagens foram digitadas entre aspas simples, que é o formato para strings no Delphi.

Validando as informações antes de gravar

Outro item importante a ser implementado é não permitir a gravação de informações incorretas ou em branco. Assim podemos fazer algumas verificações no evento BeforePost (que quando clicamos no botão “Post” antes da gravação dos dados):

Duas aspas simples sem espaço

```
If DBEditRazaoSocial.Text = ‘’ then //nada foi digitado
Begin
        ShowMessage(‘Digite a Razão Social / Nome’);
        DBEditRazaoSocial.SetFocus; // posiciona o cursor
        Abort; // não grava e continua editando
End;
```

Definindo máscaras de formatação dinamicamente

Localize no Form o Label correspondente ao campo RazaoSocial e altere o Name para LabelRazaoSocial. Faça o mesmo com o Label do campo Fantasia, modificando o Name para LabelFantasia.
Localize o controle DBRadioGroupTipo, iremos codificar o evento OnClick do mesmo, definindo o código a seguir:

```
if DBRadioGroupTipo.ItemIndex = 0 then // pessoa física
begin
   LabelRazaoSocial.Caption := 'Nome';
   LabelFantasia.Caption := 'Apelido';
   TableClientesCGC_CPF.EditMask := '999\.999\.999\-99;0;_';
end
else // pessoa jurídica
begin
   LabelRazaoSocial.Caption := 'Razão Social';
   LabelFantasia.Caption := 'Fantasia';
   TableClientesCGC_CPF.EditMask := '99\.999\.999\/9999\-99;0;_';
end;
```

Você também deve codificar de forma semelhante o evento **AfterScroll**, que ocorre depois que nos movemos de um registro para outro, pois assim estaremos exibindo as máscaras corretas para o registro corrente. No evento AfterScroll a codificação será ligeiramente diferente na primeira linha, ao invés de acessarmos o controle, faremos a leitura da informação diretamente do campo:

```
if TableClientesTipo.AsString = 'F' then // pessoa física
begin
   LabelRazaoSocial.Caption := 'Nome';
   LabelFantasia.Caption := 'Apelido';
   TableClientesCGC_CPF.EditMask := '999\.999\.999\-99;0;_';
end
else // pessoa jurídica
begin
   LabelRazaoSocial.Caption := 'Razão Social';
   LabelFantasia.Caption := 'Fantasia';
   TableClientesCGC_CPF.EditMask := '99\.999\.999\/9999\-99;0;_';
end;
```

Digitação em maiúsculo ou minúsculo independente do Caps Lock

Uma última alteração que podemos fazer é forçar a digitação para maiúsculo no campo UF. Para isso vá até o controle DBEditUF e altere a propriedade CharCase para ecUpperCase.

Você pode fazer o mesmo com os outros campos de texto (Razão Social, Fantasia, Endereço, etc) caso deseje que todas as informações sejam digitadas e gravadas em maiúsculo.

O campo de email sempre deve ser digitado em minúsculo, podemos então definir isto automaticamente para que o usuário não tenha que se preocupar com isso. Para tal, selecione o DBEditemail e defina a propriedade ChaCase para ecLowerCase.

Localizando informações

Insira um BitBtn no PanelBotoes e defina suas propriedades como segue:

Propriedade	Conteúdo
Caption	&Procurar
Name	BitBtnProcurar
Glyph	Imagem find.bmp

A imagem find.bmp faz parte do arquivo de imagens do Delphi que está em: C:\Arquivos de programas\Arquivos comuns\Borland Shared\Images\Buttons.

Dê um duplo clique no botão para abrir o editor de código que virá assim:

```
procedure TFormClientes.BitBtnProcurarClick(Sender: TObject);
begin

end;
```

Antes da instrução begin você deve inserir o seguinte:

```
var
    Nome: String;
```

O editor deverá estar apresentando o seguinte código:

```
procedure TFormClientes.BitBtnProcurarClick(Sender: TObject);
var
  Nome: String;
begin

end;
```

Digite o seguinte código entre as instruções begin e end; (onde sempre fizemos até agora):

```
  Nome := ''; // inicia definindo uma string vazia
  if InputQuery('Procurar', 'Razão Social/Nome', Nome) then
    if not TableClientes.Locate('RazaoSocial', Nome,
      [loCaseInsensitive, loPartialKey]) then
      ShowMessage('Razão Social/Nome não encontrado!');
```

Seu editor de código deve estar apresentando algo assim:

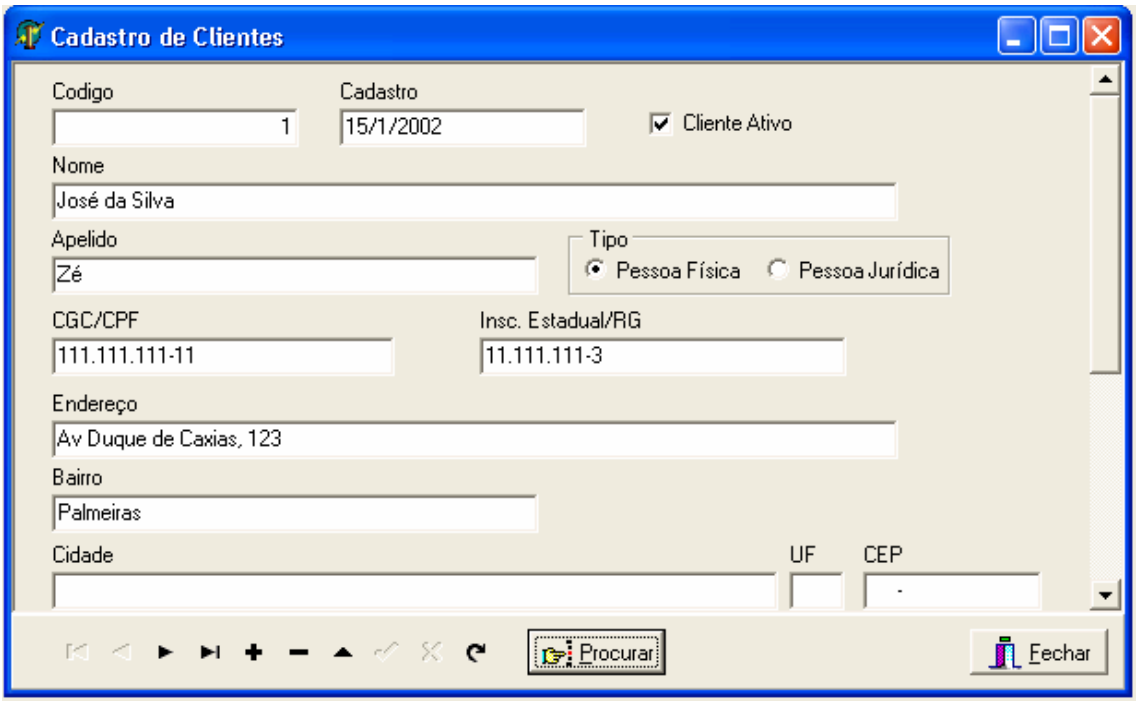
```
procedure TFormClientes.BitBtnProcurarClick(Sender: TObject);
var
  Nome: String;
begin
  Nome := ''; // inicia definindo uma string vazia
  if InputQuery('Procurar', 'Razão Social/Nome', Nome) then
    if not TableClientes.Locate('RazaoSocial', Nome,
      [loCaseInsensitive, loPartialKey]) then
      ShowMessage('Razão Social/Nome não encontrado!');

end;
```

A função InputQuery é responsável por criar uma caixa de diálogo para digitação de informação do tipo texto. O método locate é um dos métodos responsáveis pela localização de informações.

Deve ser necessário acertar a ordem de tabulação dentro do PanelBotoes, uma outra maneira de ajustar é definir a propriedade TabOrder. Defina o TabOrder do BitBtnProcurar para 1. (O DBNavigator deve ter seu TabOrder definido como 0 e conseqüentemente o BitBtnFechar será 2).

O nosso Cadastro de Clientes ficou assim:



Certamente o seu deve ter algumas diferenças, dependendo da maneira que escolheu para dispor os controles, etc.

Finalizaremos este capítulo por aqui. Agora você deve criar o formulário de Produtos, utilizando os conhecimentos que obteve até o momento.

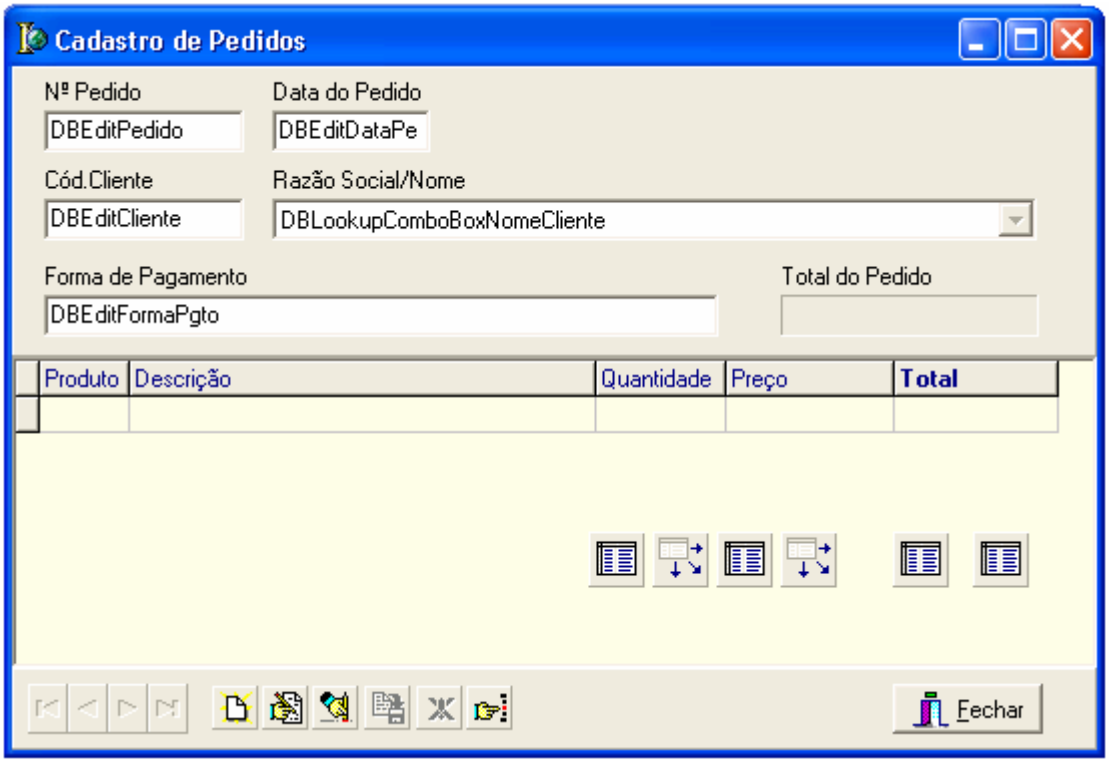
Exercícios:

1) Qual é a função propriedade EditMask?
2) Qual é o evento que devemos utilizar para inicializar valores num campo?
3) Para que serve a propriedade CharCase?
4) Qual a utilizade do método Locate?
5) Para que serve a função InputQuery?

Capítulo 5

Criando o Cadastro de Pedidos

Vamos começar esta aula de forma diferente. Iremos apresentar a aparência final do formulário de cadastro de pedidos, e depois iremos destrincha-lo, fazendo passo a passo cada um dos procedimentos para sua construção.



Esta será a interface que criaremos para o pedido. Passemos então a sua criação:

Crie um novo formulário no projeto e configure as seguintes propriedades:

Propriedade	Conteúdo
Caption	Cadastro de Pedidos
FormStyle	fsMDIChild
Height	375
Name	FormCadPedidos
Position	poMainFormCenter
ShowHint	True
Width	550

Salve-o com o nome **CadPedidos.pas**.

Insira um componente Panel (guia Standard), configurando estas propriedades:

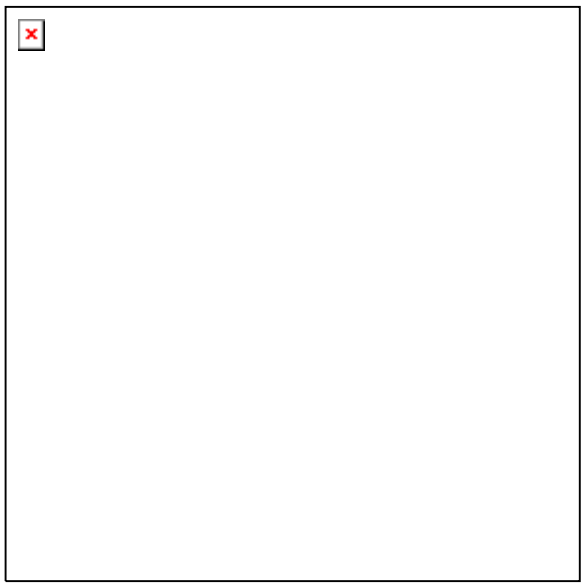
Propriedade	Conteúdo
Align	alTop
Caption	*Apagar o conteúdo*
Enabled	False
Height	145
Name	PanelDados

Insira um segundo Panel (na área livre) configurando-o assim:

Propriedade	Conteúdo
Align	alBottom
Caption	*Apagar o conteúdo*
Name	PanelBotoes

O próximo passo é inserir um componente DBGrid (guia Data Controls) na área entre os dois Panel´s. Com ele poderemos visualizar todos os itens de um pedido. O DBGrid deve ter sua propriedade Align definida para alClient, para ocupar a área restante do form. Defina a propriedade Name para DBGridItens.

O passo seguinte é inclusão do DBNavigator e dos botões no PanelBotoes. Insira um DBNavigador no PanelBotoes e na propriedade VisibleButtons, defina para False todas as subpropriedades a partir de nbInsert, como pode ser observado na figura abaixo:



Isso fará com que o DBNavigator exiba apenas os quatro primeiros botões, responsáveis pela movimentação na tabela. Para as outras opções nós utilizaremos nossos próprios botões, aprendendo como o DBNavigator manipula os métodos para dar manutenção no sistema.

Como os outros botões foram ocultados, vamos reduzir a largura do mesmo para melhorar sua aparência. Assim, defina a propriedade **Width** para **80**. Defina a propriedade **Name** do DBNavigator para **DBNavigatorPedidos**.

Insira agora seis componentes SpeedButton (guia Additional). Para inserir múltiplos componentes segure a tecla Shift antes de clicar no componente na paleta de componentes. Solte a tecla Shift e clique nos locais onde deseja que o componente seja inserido, quando terminar, vá até a paleta de componentes e clique no ícone semelhante à seta do mouse.

Defina para estes botões as propriedades que seguem:

(Obs: As imagens utilizadas na propriedade glyph devem ser encontradas em: C:\Arquivos de programas\Arquivos comuns\Borland Shared\Images\Buttons).

Primeiro botão (SpeedButton1):

Propriedade	Conteúdo
Glyph	Imagem filenew.bmp
Name	SpdBtnIncluir

Segundo botão (SpeedButton2):

Propriedade	Conteúdo
Glyph	Imagem edit.bmp
Name	SpdBtnAlterar

Terceiro botão (SpeedButton3):

Propriedade	Conteúdo
Glyph	Imagem erase.bmp
Name	SpdBtnExcluir

Quarto botão (SpeedButton4):

Propriedade	Conteúdo
Glyph	Imagem filesave.bmp
Enabled	False
Name	SpdBtnGravar

Quinto botão (SpeedButton5):

Propriedade	Conteúdo
Glyph	Imagem ignore.bmp
Enabled	False
Name	SpdBtnCancelar

Sexto botão (SpeedButton6):

Propriedade	Conteúdo
Glyph	Imagem find.bmp
Enabled	False

Name	SpdBtnProcurar

Para criar o botão para fechar o form, utilizamos o nosso já conhecido BitBtn (também da guia Additional). Insira um BitBtn no PanelBotoes e defina suas propriedades como segue:

Propriedade	Conteúdo
Caption	&Procurar
Name	BitBtnProcurar
Glyph	Imagem find.bmp

A etapa seguinte será a inclusão dos componentes de acesso a dados. Insira quatro componentes Table (guia BDE) configurando-os assim:

Primeiro Table (Table1):

Propriedade	Conteúdo
DatabaseName	DBPedidos
Name	TablePedidos
TableName	Pedidos.db

Segundo Table (Table2):

Propriedade	Conteúdo
DatabaseName	DBPedidos
Name	TableItens
TableName	Itens.db

Terceiro Table (Table3):

Propriedade	Conteúdo
DatabaseName	DBPedidos
Name	TableClientes
TableName	Clientes.db

Quarto Table (Table4):

Propriedade	Conteúdo
DatabaseName	DBPedidos
Name	TableProdutos
TableName	Produtos.db

Precisamos inserir agora dois componentes DataSource (guia Data Access):

Primeiro DataSource (DataSource1):

Propriedade	Conteúdo
DataSet	TablePedidos
Name	DSPedidos

Segundo DataSource (DataSource2):

Propriedade	Conteúdo
DataSet	TableItens
Name	DSItens

Selecione o componente DBNavigatorPedidos e defina sua propriedade DataSource para DSPedidos.
Selecione o DBGridItens e defina sua propriedade DataSource para DSItens.

Agora devemos adicionar os campos da tabela Pedidos dentro do seu respectivo componente Table. Como fizemos anteriormente, dê um duplo clique no objeto Table para abrir o Fields Editor. Pressione Ctrl+F para adicionar todos os campos. Pressione Ctrl+N para adicionar um novo campo. Criaremos agora um campo de **LookUp**, responsável por localizar uma determinada informação em outra tabela e retornar o conteúdo de um determinado campo. Observe na figura abaixo como definimos um campo Lookup:

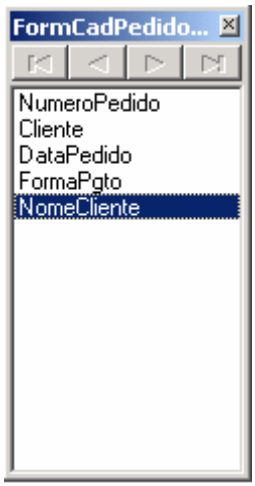
O item Name representa o nome que daremos ao campo (digite NomeCliente). Component será o nome interno do campo, que é criado automaticamente, baseando-se no nome do Table e o nome que você informou em Name. No item Type informamos o tipo do campo (selecione String que é igual ao Alpha do Paradox). Em Size definimos o tamanho do campo quando aplicável (digite 35, o mesmo tamanho do campo Nome na tabela de cliente).

Em Field type devemos indicar Lookup, para termos acesso ao grupo Lookup definition.

O item Key Fields representa qual campo utilizaremos para procurar (selecione Cliente). Em Dataset temos o nome da tabela na qual a busca será realizada

(selecione TableClientes). O item Lookup Keys é o campo da tabela definida em Dataset que deve corresponder ao campo definido em Key Fields (selecione Código). O Item Result Field tem o nome do campo que deve ter seu conteúdo retornado se a informação for encontrada (selecione RazaoSocial).

Ao clicar em OK, seu Fields Editor deve ter ficado como a primeira figura abaixo (do lado esquerdo):



Mova este campo arrastando-o com o mouse, deixando-o abaixo do campo Cliente.
Altere as propriedades DisplayLabel dos campos, para algo mais significativo (como o exibido na figura do nosso formulário no início do capítulo). Defina também o EditMask do campo DataPedido como fizemos no cadastro de clientes.

Arraste os campos para o PanelDados (na parte superior do Form), alinhando-os como fizemos (ou da forma como preferir).

Obs: Para alinhar dois objetos na mesma linha, ajuste a propriedade Top de ambos para o mesmo valor. Para alinhar o lado esquerdo, defina a propriedade Left. Uma maneira é selecionar o primeiro componente, segurar a tecla Shift e selecionar os outros. Vá até a propriedade que quer definir e tecle Enter. Isso fará com que os outros objetos assumam o mesmo valor que estava definido na propriedade do primeiro objeto.

Selecione o TableItens e adicione os seus campos, da mesma forma como fizemos com o TablePedidos. Adicione também um campo de Lookup, configurando-o assim:

- Name: Descriçao
- Type: String
- Size: 35
- Field type: Lookup
- Key Fields: CodigoProduto

- Dataset: TableProdutos
- Lookup Keys: Código
- Result Field: Descrição

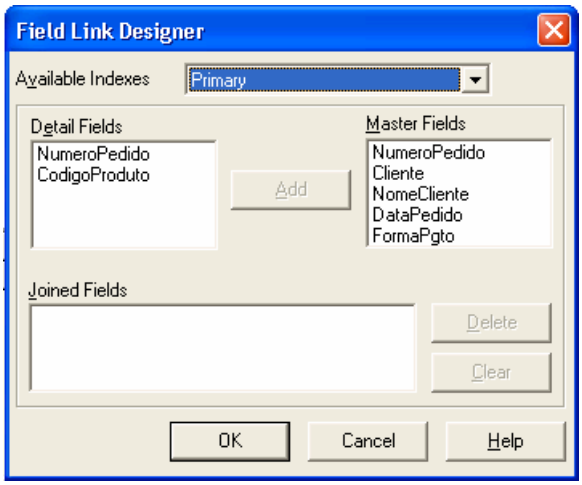
Posicione este campo Lookup abaixo do campo CodigoProduto, da mesma maneira como fizemos na tabela anterior.

Adicione um novo campo, agora será um campo Calculated, responsável por efetuar o cálculo do total do item (quantidade x preço) sem que seja necessário armazena-lo na tabela. Configure-o assim:

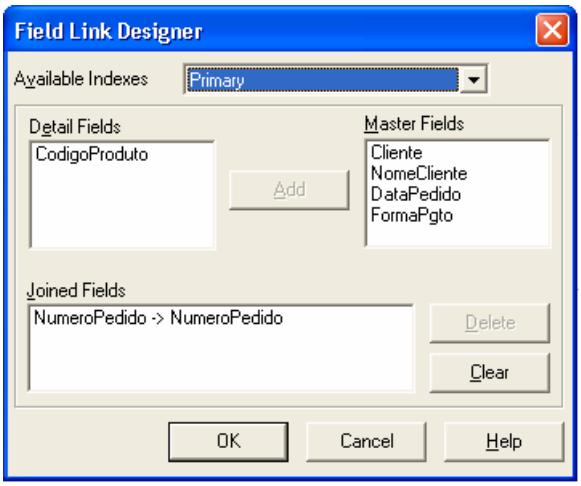
- Name: Total
- Type: Currency
- Field type: Calculated

(Obs: Os outros campos são definidos apenas para campos Lookup).

Faremos agora uma ligação meste-detalhe entre os componentes TablePedidos e TableItens. Vá ao componente TableItens e defina a propriedade **MasterSource** para **DSPedidos**. A seguir clique no editor (botão com reticências) da propriedade MasterFields.

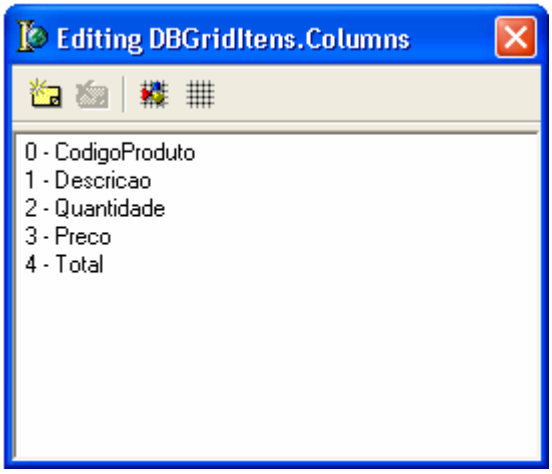


Será exibido o Field Link Designer, como na figura acima. Selecione o campo NumeroPedido na lista Detail Fields e na lista Master Fields. Clique no botão Add. A caixa Joined Fields deve exibir o seguinte:

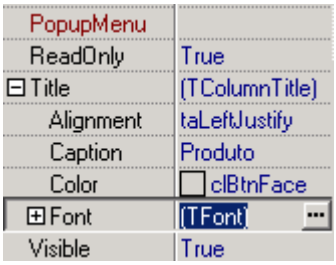


Clique no botão OK para prosseguir. Com isso indicamos que o TableItens está ligado como filho do TablePedidos. Cada vez que movimentarmos a TablePedidos na TableItens serão exibidos apenas os registros que pertençam ao pedido exibido.

Para finalizar, vamos terminar de configurar o DBGridItens. Dê um duplo clique no DBGrid para abrir o editor de colunas. Clique no terceiro botão para inserir todos os campos como exibido abaixo:



Selecione o campo NumeroPedido e o remova teclando Delete. (O número do pedido não precisa aparecer no DBGrid).
Vamos mudar um pouco a aparência do nosso DBGrid, alterando a cor dos títulos das colunas e a cor de fundo para exibição dos dados
Selecione o campo CodigoProduto e localize a propriedade **Title** no Object Inspector. Abra as subpropriedades clicando no sinal de + à esquerda da propriedade.



Selecione a subpropriedade Font e defina a cor para Azul-marinho (se desejar altere também a fonte). Faça o mesmo para os outros campos. No campo Total defina também o estilo da fonte para negrito.
Selecione agora o DBGridItens e defina sua propriedade **Color** para **clInfoBk**.

Nos falta agora inserir dois últimos componentes. Insira um Label no PanelDados e configure sua propriedade **Caption** para **Total do Pedido**. Insira um componente StaticText (guia additional) abaixo do Label, definindo suas propriedades da seguinte forma:

Propriedade	Conteúdo
Alignment	taRightJustify
AutoSize	True
BorderStyle	sbsSunken
Caption	*Apagar o conteúdo*
Height	21
Font	Times New Roman, Negrito, Tamanho 9, cor Azul-marinho
Name	StaticTextTotal

Com isso terminamos toda a parte visual (design). Devemos agora iniciar a codificação, não esquecendo de salvar novamente antes de prosseguir.

Codificando o formulário Cadastro de Pedidos

Vamos começar definindo os eventos de criação, destruição e fechamento do nosso form. Selecione o FormCadPedidos através do combo do Object Inspector ou utilizando o Object TreeView. Localize o evento **OnCreate** do form e abra o editor de código com um duplo clique na caixa de combinação do respectivo evento. Adicione as linhas de código abaixo:

```
// abre as tabelas
TableProdutos.Open;
TableClientes.Open;
TablePedidos.Open;
TableItens.Open;
```

O editor de código deve estar semelhante ao código a seguir:

```
procedure TFormCadPedidos.FormCreate(Sender: TObject);
begin
   // abre as tabelas
   TableProdutos.Open;
   TableClientes.Open;
   TablePedidos.Open;
   TableItens.Open;
end;
```

Continue da mesma forma para os demais eventos. Agora codifique o evento **OnDestroy**:

```
   // fecha as tabelas
   TableProdutos.Close;
   TableClientes.Close;
   TablePedidos.Close;
   TableItens.Close;
```

E o evento **OnClose**:

```
   Action := caFree; // libera form da memória
   FormCadPedidos := NIL; // indica que foi liberado
```

Localize o BitBtnFechar e dê um clique duplo para abrir o código do evento OnClick do mesmo. Como fizemos anteriormente no cadastro de clientes, simplesmente chame o método para fechar o form:

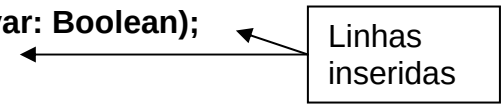
```
Close; // fecha o form
```

No TablePedidos vamos codificar o evento **OnNewRecord**:

```
TablePedidosDataPedido.AsDateTime := Date; //inicia com a data atual
```

Criaremos agora duas procedures (métodos) dentro do nosso formulário, para executar rotinas que podem ser chamadas várias vezes.
Vá até o inicio da unit do formulário na cláusula **private** e acrescente as linhas em destaque à sua definição, ficando como exibido abaixo:

```
 private
  { Private declarations }
  procedure AtivarControles(Ativar: Boolean);
  procedure RecalculaPedido;
 public
```



```
    { Public declarations }
  end;
```

Agora vá ao final da unit e digite todo o código abaixo antes do **end.**

```
procedure TFormCadPedidos.AtivarControles(Ativar: Boolean);
begin
   PanelDados.Enabled := Ativar;
   DBNavigatorPedidos.Enabled := (not Ativar);
   DBGridItens.ReadOnly := (not Ativar);
   SpdBtnIncluir.Enabled := (not Ativar);
   SpdBtnAlterar.Enabled := (not Ativar);
   SpdBtnExcluir.Enabled := (not Ativar);
   SpdBtnGravar.Enabled := Ativar;
   SpdBtnCancelar.Enabled := Ativar;
   SpdBtnProcurar.Enabled := (not Ativar);
end;


procedure TFormCadPedidos.RecalculaPedido;
var
   TmpTable: TTable;
   Total: Currency; // armazena valores do tipo moeda
begin
   // cria um objeto Table via codificação
   TmpTable := TTable.Create(Application);
   try
      // define DatabaseName e TableName via codificação
      TmpTable.DatabaseName := TableItens.DatabaseName;
      TmpTable.TableName := TableItens.TableName;
      TmpTable.Open;
      TmpTable.FindKey([TablePedidosNumeroPedido.AsInteger]);
      Total := 0; // inicializa a variavel totalizadora
      while (not TmpTable.Eof) and
           (TmpTable.FieldByName('NumeroPedido').AsInteger =
            TablePedidosNumeroPedido.AsInteger) do
      begin
         // Acumula o Total da linha
         Total := Total + (TmpTable.FieldByName('Preco').AsCurrency *
                     TmpTable.FieldByName('Quantidade').AsFloat);
         TmpTable.Next; // próximo registro
      end;
   finally
      TmpTable.Close; // fecha a tabela
      TmpTable.Free; // libera objeto da memória
   end;
   StaticTextTotal.Caption := FormatCurr('###,###,##0.00', Total);
end;
```

Selecione novamente o TableItens e localize o evento **OnCalcFields**. O código a ser inserido é o seguinte:

```
if (TableItensQuantidade.AsFloat > 0) and
   (TableItensPreco.AsCurrency > 0) then
   TableItensTotal.AsCurrency := TableItensPreco.AsCurrency *
                        TableItensQuantidade.AsFloat;
RecalculaPedido; // recalcula e exibe novamente
```

No evento **OnNewRecord**, definiremos estas duas linhas:

```
TableItensQuantidade.AsFloat := 1;
DBGridItens.SelectedIndex := 0; //código
```

O evento **BeforeInsert** deve ser codificado como segue:

```
if TablePedidosNumeroPedido.AsString = '' then
begin
   if TablePedidos.State = dsInsert then
   begin
      // grava para salvar o número do pedido na tabela pai
      TablePedidos.Post;
      // ativa a alteração novamente
      TablePedidos.Edit;
   end;
end;
```

O código abaixo você deve inserir no evento **BeforePost**:

```
if TableItensNumeroPedido.AsString = '' then
   TableItensNumeroPedido.AsInteger :=
               TablePedidosNumeroPedido.AsInteger;

if TableItensCodigoProduto.AsString = '' then
begin
   DBGridItens.SelectedIndex := 0; // seleciona a coluna código
   ShowMessage('Código do produto deve ser informado!');
   Abort; // interrompe a gravação
end;

if TableItensQuantidade.AsFloat <= 0 then
begin
   DBGridItens.SelectedIndex := 2; // seleciona coluna quantidade
   ShowMessage('Código do produto deve ser informado!');
```

```
      Abort; // interrompe a gravação
    end;
```

Selecione agora o objeto DBGridItens e localize o evento OnKeyDown, codificando-o como segue:

```
// muda para a próxima coluna se pressionar Enter
if Key = VK_RETURN then // pressionou ENTER
begin
  case DBGridItens.SelectedIndex of
    0: DBGridItens.SelectedIndex := 2; //quantidade
    1: DBGridItens.SelectedIndex := 2; //quantidade
    2: DBGridItens.SelectedIndex := 3; //Preco
  else
    DBGridItens.SelectedIndex := 0; //código
    TableItens.Next;
    if TableItens.Eof then
      TableItens.Append;
  end;
end;
```

Definiremos agora uma validação para o campo CodigoProduto na TableItens. Abra o Fields Editor, selecione o campo CodigoProduto e localize o evento **OnValidate**. Neste evento iremos definir o código abaixo:

```
if TableItensCodigoProduto.AsString <> '' then
begin
  if TableProdutos.FindKey([TableItensCodigoProduto.AsInteger]) then
  TableItensPreco.AsCurrency :=
        TableProdutos.FieldByName('Preco').AsCurrency
  else
  begin
    ShowMessage('Código inválido');
    Abort;
  end;
end;
```

Existe mais um ajuste que devemos fazer na parte de eventos ligados aos campos. Ao digitarmos uma data, num campo onde definimos o EditMask, se apagarmos o conteúdo do mesmo será gerado um erro por definição de data em branco (‘ / / ‘). Este código a seguir é responsável por identificar esta situação e efetuar a correção necessária. Deve ser digitado no evento **OnSetText**:

```
    if Text = '  /  /    ' then
      Sender.Clear // apaga o campo data
```

```
else
   // atribui a data digitada ao campo
   try
      Sender.AsString := Text;
   except
      ShowMessage('Data inválida!');
   end;
```

Agora faltam os códigos do evento **OnClick** dos SpeedButton's. Codifique para o SpdBtnIncluir o seguinte:

```
AtivarControles(True); // ativa os controles para digitação
TablePedidos.Append; // inclui um novo registro na tabela
DBEditDataPedido.SetFocus;
```

O próximo botão a receber código será o SpdBtnAlterar:

```
if TablePedidos.IsEmpty then
begin
   // a tabela está vazia, então devemos incluir
   SpdBtnIncluir.Click; // executa o click no botão
   Exit; // retorna
end;
AtivarControles(True); // ativa os controles para digitação
TablePedidos.Edit; // permite alterar os dados
DBEditDataPedido.SetFocus;
```

O botão seguinte é o SpdBtnExcluir:

```
if Application.MessageBox('Deseja excluir este pedido?','Confirme',
   MB_YESNO + MB_ICONQUESTION + MB_DEFBUTTON2) = IDNO then
   Exit; // retorna (sem fazer nada)
// devemos excluir os itens primeiro, para não termos
// registros órfãos
try
   TableItens.First; // posiciona no primeiro item
   while not TableItens.Eof do // executa até o fim do arquivo
   begin
      TableItens.Delete; // exclui o item
      TableItens.First;
   end;
   TablePedidos.Delete; // exclui o pedido
except
   ShowMessage('Ocorreu um erro durante a exclusão do pedido');
end;
```

Para o botão SpdBtnGravar teremos esta codificação:

```
if TablePedidosNumeroPedido.AsInteger <= 0 then
begin
   ShowMessage('Digite o número do pedido!');
   DBEditPedido.SetFocus;
   Exit;
end;
if TablePedidosDataPedido.IsNull then
begin
   ShowMessage('Digite a data do pedido!');
   DBEditDataPedido.SetFocus;
   Exit;
end;

if TablePedidosCliente.AsInteger > 0 then
begin
   // procura pelo cliente usando o código
   if not TableClientes.FindKey([TablePedidosCliente.AsInteger]) then
   begin
      ShowMessage('Código de cliente inválido!');
      DBEditCliente.SetFocus;
      Exit;
   end;
end
else
begin
   ShowMessage('Digite o código do cliente!');
   DBEditCliente.SetFocus;
   Exit;
end;
TablePedidos.Post;
if TableItens.State in [dsInsert,dsEdit] then
   TableItens.Post;
AtivarControles(False); // desativa os controles
```

E para o botão SpdBtnCancelar o código será este:

```
TablePedidos.Cancel; // cancela inclusão/alteração
AtivarControles(False); // desativa os controles
```

O SpdBtnProcurar nós codificaremos na próxima aula, pois criaremos um formulário especial de pesquisa. Vamos ligar nosso formulário ao menu no FormPrincipal para que possamos vê-lo em ação.

Vá até o FormPrincipal e clique na opção de menu correspondente a este cadastro. Na área que se abriu no editor de código insira as linhas abaixo:

```
if FormCadPedidos = NIL then // cria o form se ele não foi criado ainda
   Application.CreateForm(TFormCadPedidos,FormCadPedidos);
FormCadPedidos.Show; //exibe o formulário
```

Também devemos remover a criação automática deste form, como fizemos com o cadastro de clientes. Para isso vá até a caixa de diálogo Project Options (Shift+Ctrl+F11) e mova o FormCadPedidos para a lista de Available forms.

Salve o projeto e execute-o. Observe como se comporta e quais são as melhorias que você poderia implementar no mesmo. No capítulo seguinte criaremos o form de localização e veremos como associa-lo ao cadastro de pedidos. Faremos também uma análise profunda da lógica de funcionamento deste tipo de interface que criamos e das codificações que foram utilizadas.

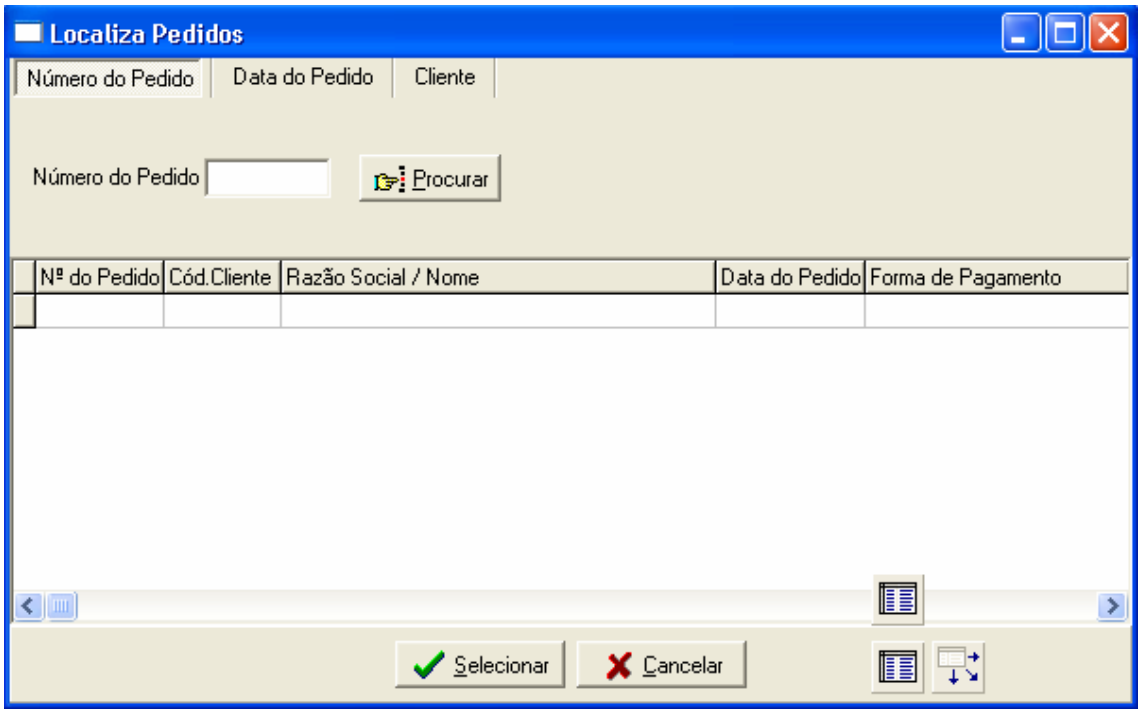
Exercícios

1) Para que serve a propriedade Glyph presente em alguns tipos de botão?
2) Como podemos ocultar os botões do DBNavigator?
3) Qual é a melhor maneira para se inserir um determinado componente em um Form várias vezes?
4) Qual a finalizade de campos LookUp?
5) Qual a finalizade de campos Calculated?

Capítulo 6

Criando um formulário de Pesquisa

Da mesma forma como fizemos no capítulo anterior, vou lhe mostrar o resultado final e veremos como chegar até ele.
Esta será a aparência do Form que vamos criar:



Crie um novo formulário e defina seu Name para FormLocalizaPedidos e o Caption para Localiza Pedidos. Altere as propriedades Height para 375 e Width para 600. Modifique a propriedade BorderStyle para bsDialog. Salve-o com o nome LocalizaPedidos.pas.

Insira um componente PageControl (guia Win32) e definas suas propriedades da seguinte maneira:

Propriedade	Conteúdo
Align	AlTop
Height	105
Name	PgCtrlLocalizar
Style	TsFlatButtons

Clique com o botão da direita sobre o PageControl e selecione **New Page** no menu de contexto. Será inserido um TabSheet (uma espécie de “página”).

Clique neste objeto para seleciona-lo e defina o Name para TabSheetNumPedido e o Caption para *Número do Pedido*.
Insira neste TabSheet um Label, um Edit (ambos da guia Standard) e um BitBtn (guia Additional). No Caption do Label escreva *Número do Pedido*. Apague o texto contido na propriedade Text do componente Edit e altere seu Name para EditPedido. O Name do BitBtn deve ser BitBtnPedido e no seu Caption você deve digitar &Procurar. Na propriedade glyph utilize a imagem find.bmp ou outra mais conveniente.

Insira agora um DBGrid (guia Data Controls) abaixo do PageControl, configurando suas propriedades assim:

Propriedade	Conteúdo
Height	195
Left	0
Name	DBGridPedidos
ReadOnly	True
Top	105
Width	594

Modifique ainda no DBGrid, na propriedade Options a sub-propriedade RowSelecte para True.

Insira abaixo do DBGrid um Panel, configurando-o assim:

Propriedade	Conteúdo
Caption	*Apagar o conteúdo*
Height	41
Left	0
Name	PanelBotoes
Top	300
Width	594

Insira dentro no Panel dois BitBtn, configurando-os nesta seqüência:

Primeiro Botão (BitBtn1):

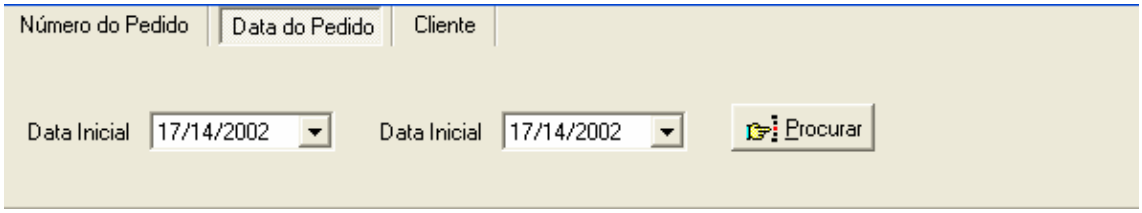
Propriedade	Conteúdo
Kind	bkOK
Caption	&Selecionar
Default	False
Name	BitBtnSelecionar
Width	90

Segundo Botão (BitBtn2):

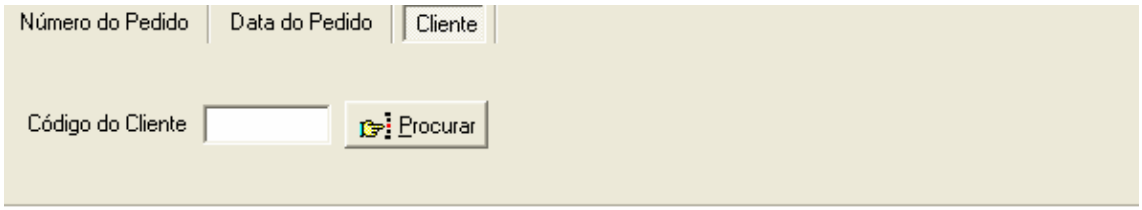
Propriedade	Conteúdo
Kind	bkCancel
Caption	&Cancelar
Name	BitBtnCancelar
Width	90

Retorne ao PageControl e clicando novamente com o botão da direita, escolha New Page para inserir outro TabSheet. Desta vez, escreva no Caption do TabSheet *Data do Pedido* e Altere o Name Para TabSheetData.

Insira um Label neste TabSheet definindo seu Caption como *Data Inicial*. Adicione um componente DateTimePicker (guia Win32) em frente ao Label. Defina o Name para DTPickerInicial e preencha a propriedade Format com dd/mm/yyyy. Repita o processo, adicionando outro Label, agora com o Caption *Data Final* e outro DateTimePicker com o Name DTPickerFinal e o mesmo conteúdo em Format. Adicione outro BitBtn com o Name BitBtnData e as mesmas configurações do BitBtn adicionado à página anterior. Neste ponto este TabSheet deve ser semelhante a figura abaixo:



Selecione o PageControl e insira a última página. Nela você deve colocar um Label com o Caption *Código do Cliente*, um Edit com o nome de EditCliente e com o conteúdo da propriedade Text apagado. Insira também outro BitBtn, com o Nome de BitBtnCliente e as mesmas configurações dos outros dois. Você deve ter um TabSheet semelhante a este:



Adicione agora dois componentes Table (guia BDE) e um DataSource (guia Data Access). Configure o primeiro Table para:

Propriedade	Conteúdo
DatabaseName	DBPedidos
Name	TableClientes
Table	Clientes.db

E o segundo para:

Propriedade	Conteúdo
DatabaseName	DBPedidos
Name	TableClientes
Table	Clientes.db

Altere o nome do DataSource para DSPedidos e aponte sua propriedade DataSet para o TablePedidos.
Adicione os campos no TableClientes e ajuste as propriedades DisplayLabel e crie um campo de Lookup chamado NomeCliente, da mesma forma como fizemos no Cadastro de Pedidos.

Agora vamos iniciar a codificação. No evento OnCreate do Form, escreva o seguinte:

```
TablePedidos.Open;
TableClientes.Open;
Height := 140; // define a "altura" do form
```

No evento OnDestroy, a codificação será esta:

```
TablePedidos.Close;
TableClientes.Close;
```

No evento OnClick do BitBtnPedido codifique como as linhas destacadas abaixo:

```
procedure TFormLocalizaPedidos.BitBtnNumPedidoClick(Sender: TObject);
var
   NumPedido: Integer;
begin
   if EditPedido.Text = '' then
   begin
      ShowMessage('Digite o número do Pedido!');
      EditPedido.SetFocus; // posiciona na caixa de texto novamente
      Exit; // retorna
   end;
   // agora devemos converter o texto digitado
   // para um número inteiro
   try
      // converte string para inteiro
      NumPedido := StrToInt(EditPedido.Text);
   except
      on EConvertError do // ocorreu um erro na conversão
      begin
```

```
        ShowMessage(EditPedido.Text + ' não é um número inteiro
válido!');
        EditPedido.Clear; // limpa o conteudo da caixa de texto
        EditPedido.SetFocus;
        Exit;
      end;
   end;
   // Cancela qualquer definição de faixa de dados (filtragem)
   TablePedidos.CancelRange;

   // define que o indice usado será a chave primária
   TablePedidos.IndexName := '';
   // Procura pelo número do pedido
   if TablePedidos.FindKey([NumPedido]) then
   begin
      // aumenta o formulário para exibir o DBGrid
      // e os botões para selecionar ou cancelar
      Height := 375;
      // posiciona na "linha" 110 para garantir
      // que os botões não fiquem abaixo da barra de tarefas
      Top := 110;
      // como só vai haver um pedido posiciona no botao para selecionar
      BitBtnSelecionar.SetFocus;
   end
   else
   begin
      ShowMessage('Pedido não encontrado!');
      EditPedido.SetFocus;
   end;
end;
```

No evento OnClick do botão BitBtnData, devemos definir o seguinte código:

```
   // Cancela qualquer definição de faixa de dados (filtragem)
   TablePedidos.CancelRange;
   // define que o indice usado será a data do pedido
   TablePedidos.IndexName := 'IDataPedido';
   // Exibe apenas os registros nesta faixa de datas

TablePedidos.SetRange([DateToStr(DTPickerInicial.Date)],[DateToStr(DTP
ickerFinal.Date)]);
   if TablePedidos.Eof then // fim de arquivo
   begin
      TablePedidos.CancelRange;
      ShowMessage('Nenhum pedido encontrado no período informado!');
      DTPickerInicial.SetFocus;
   end
   else
```

```
  begin
    Height := 375;
    Top := 110;
    DBGridPedidos.SetFocus;
  end;
```

O evento OnClick do BitBtnCliente deve ficar assim:

```
procedure TFormLocalizaPedidos.BitBtnClienteClick(Sender: TObject);
var
  NumCliente: Integer;
begin
  if EditCliente.Text = '' then
  begin
    ShowMessage('Digite o Código do Cliente!');
    EditCliente.SetFocus; // posiciona na caixa de texto novamente
    Exit; // retorna
  end;
  // agora devemos converter o texto digitado
  // para um número inteiro
  try
    // converte string para inteiro
    NumCliente := StrToInt(EditCliente.Text);
  except
    on EConvertError do // ocorreu um erro na conversão
    begin
      ShowMessage(EditCliente.Text + ' não é um número inteiro
válido!');
      EditCliente.Clear; // limpa o conteudo da caixa de texto
      EditCliente.SetFocus;
      Exit;
    end;
  end;
  if not TableClientes.FindKey([NumCliente]) then
  begin
    ShowMessage('Nenhum cliente cadastrado com o código ' +
EditCliente.Text);
    EditCliente.Clear; // limpa o conteudo da caixa de texto
    EditCliente.SetFocus;
    Exit;
  end;

  // Cancela qualquer definição de faixa de dados (filtragem)
  TablePedidos.CancelRange;
  // define que o indice usado será a chave primária
  TablePedidos.IndexName := 'ICliente';
  // Exibe apenas os registros deste cliente
  TablePedidos.SetRange([NumCliente],[NumCliente]);
```

```
   if TablePedidos.Eof then // fim de arquivo
   begin
      TablePedidos.CancelRange;
      ShowMessage('Nenhum pedido deste cliente!');
      EditCliente.SetFocus;
   end
   else
   begin
      Height := 375;
      Top := 110;
      DBGridPedidos.SetFocus;
   end;
end;
```

Selecione novamente o primeiro TabSheet (TabSheetNumPedido) pois o TabSheet que estiver selecionado no momento da compilação é o que estará selecionado por padrão.

Vá ao Form e defina a propriedade ActiveControl como EditPedido.
Selecione o PageControl e no evento OnChange escreva o seguinte:

```
Height := 140; // define a "altura" do form
```

Você deve remover a criação automática deste formulário. O próximo passo a chama-lo dentro do Cadastro de Pedidos. Para isso, vá ao formulário de Cadastro de Pedidos. No menu File|Use unit selecione o form LocalizaPedidos.

No botão correspondente a opção procurar, digite o seguinte:

```
   Application.CreateForm(TFormLocalizaPedidos, FormLocalizaPedidos);
   if FormLocalizaPedidos.ShowModal = mrOK then
      TablePedidos.GotoCurrent(FormLocalizaPedidos.TablePedidos);
   FormLocalizaPedidos.Free;
```

Isto fará com que o pedido localizado no form de procura seja selecionado (e exibido) no formulário de pedidos, através da sincronização das posições dos registros (método GotoCurrent do Table).

Salve seu projeto e execute-o. Cadastre alguns pedidos, localize-os. Veja como o aplicativo se comporta.

Comentando a codificação feita

Este form que criamos, quando chamado, exibirá apenas a parte superior (o PageControl). O usuário poderá escolher através das páginas qual tipo de procura deseja fazer. Caso alguma informação seja encontrada o DBGrid com os dados e o Panel com os botões serão exibidos pois o formulário irá alterar seu tamanho. Caso ao invés de selecionar o item o usuário decidir alterar o tipo de procura o mesmo reduzirá seu tamanho novamente.

Para a localização de informações utilizamos basicamente dois métodos do Table, o Locate (para pesquisas flexíveis, que permitam diferenciar ou não letras maiúsculas e minúsculas) e o FindKey (para procuras exatas, contidas na chave primária ou em índices secundários). Ambos os métodos retornam True caso tenham encontrado a informação e False se não encontrarem.

Em nosso Form de Pedidos, utilizamos um Panel (PanelDados) com a propriedade Enabled definida como False. Isso nos permite travar todos os objetos que estejam contidos nele, impedindo que o usuário os acesse até que alteremos o valor de Enabled para True.
A propriedade ReadOnly do DBGridItens tem o mesmo propósito, impedir digitação quando o usuário estiver apenas visualizando o pedido. Você deve notar que ambos são ativados através de uma chamada ao método que criamos, o AtivarControles, que reproduzimos abaixo para facilitar o entendimento:

```
procedure TFormCadPedidos.AtivarControles(Ativar: Boolean);
begin
   PanelDados.Enabled := Ativar;
   DBNavigatorPedidos.Enabled := (not Ativar);
   DBGridItens.ReadOnly := (not Ativar);
   SpdBtnIncluir.Enabled := (not Ativar);
   SpdBtnAlterar.Enabled := (not Ativar);
   SpdBtnExcluir.Enabled := (not Ativar);
   SpdBtnGravar.Enabled := Ativar;
   SpdBtnCancelar.Enabled := Ativar;
   SpdBtnProcurar.Enabled := (not Ativar);
end;
```

Este método/procedure pode receber um parâmetro do tipo Boolean (lógico), ou seja, True ou False. Ao invés de utilizarmos estruturas if de controle, procuramos utilizar o valor passado diretamente, para ativar ou desativar os controles. Assim, quando os botões de gravar e cancelar estiverem ativos, os outros não estarão, por isso usamos “(not Ativar)”. O PanelDados também estará ativo para permitir que os dados seja digitados.

Vejamos o outro método que criamos, o Recalcula Pedido:

```
procedure TFormCadPedidos.RecalculaPedido;
var
   TmpTable: TTable;
```

```
    Total: Currency; // armazena valores do tipo moeda
begin
    // cria um objeto Table via codificação
    TmpTable := TTable.Create(Application);
    try
        // define DatabaseName e TableName via codificação
        TmpTable.DatabaseName := TableItens.DatabaseName;
        TmpTable.TableName := TableItens.TableName;
        TmpTable.Open;
        TmpTable.FindKey([TablePedidosNumeroPedido.AsInteger]);
        Total := 0; // inicializa a variavel totalizadora
        while (not TmpTable.Eof) and
             (TmpTable.FieldByName('NumeroPedido').AsInteger =
              TablePedidosNumeroPedido.AsInteger) do
        begin
            // Acumula o Total da linha
            Total := Total + (TmpTable.FieldByName('Preco').AsCurrency *
                        TmpTable.FieldByName('Quantidade').AsFloat);
            TmpTable.Next; // próximo registro
        end;
    finally
        TmpTable.Close; // fecha a tabela
        TmpTable.Free; // libera objeto da memória
    end;
    StaticTextTotal.Caption := FormatCurr('###,###,##0.00', Total);
end;
```

Neste método criamos um objeto Table (TmpTable) através de programação e o associa a mesma tabela de Itens do pedido ao qual o TableItens esta conectado. Procura pelo pedido atual e totaliza os registros. Note que utilizamos uma estrutura de controle de exceções (try..finally..end) para garantir que a memória alocada com a criação do objeto table seja devolvida ao Windows, quer a operação seja bem sucedida ou não. Isso garante que seu programa seja mais robusto.

Um último tópico a ser comentado é o código do botão de exclusão:

```
    if Application.MessageBox('Deseja excluir este pedido?','Confirme',
        MB_YESNO + MB_ICONQUESTION + MB_DEFBUTTON2) = IDNO then
        Exit; // retorna (sem fazer nada)
    // devemos excluir os itens primeiro, para não termos
    // registros órfãos
    try
        TableItens.First; // posiciona no primeiro item
        while not TableItens.Eof do // executa até o fim do arquivo
        begin
            TableItens.Delete; // exclui o item
            TableItens.First;
```

```
    end;
    TablePedidos.Delete; // exclui o pedido
  except
    ShowMessage('Ocorreu um erro durante a exclusão do pedido');
  end;
```

A idéia é que você deve excluir primeiro os itens (filhos) antes de excluir o pedido (pai), pois caso algum dos filhos não possa ser excluído, por alguém estar utilizando ou algo assim (no caso do sistema estar em rede) você teria registros órfãos na sua tabela. A estrutura de controle de exceção try..except..end foi utilizada para informar interromper a execução (não excluir mais) e avasar caso ocorra algum erro. Se nenhum problema ocorrer, a mensagem não é exibida.

Exercícios

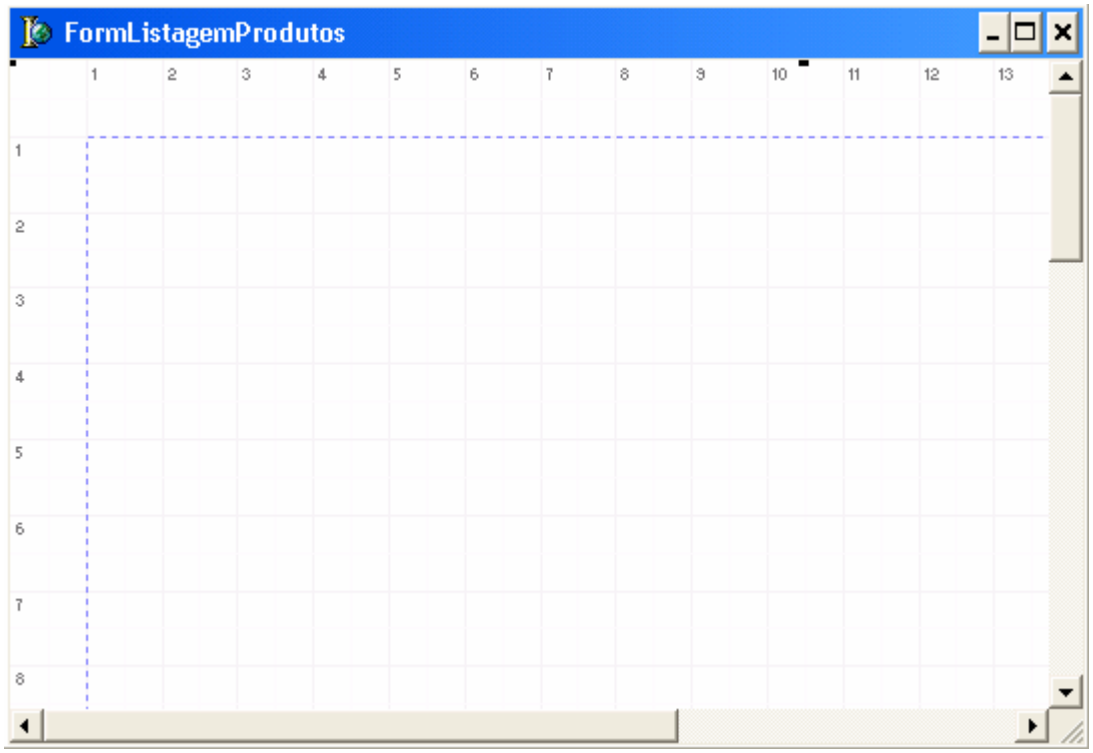
1) Quais são os principais métodos de pesquisa do componente Table?
2) Como podemos alterar dinamicamente o tamanho de um formulário?
3) Qual é a funcionalidade do componente PageControl?
4) Para que serve as estruturas de controle de exceção try..finally..end?
5) Qual o método utilizado para sincronizar a posição de registro de dois componentes Table configurados para acessar a mesma tabela?

Capítulo 7

Criando Relatórios e Listagens

Para criar um relatório, insira um novo Form no seu projeto. Defina o Name para FormListagemProdutos.

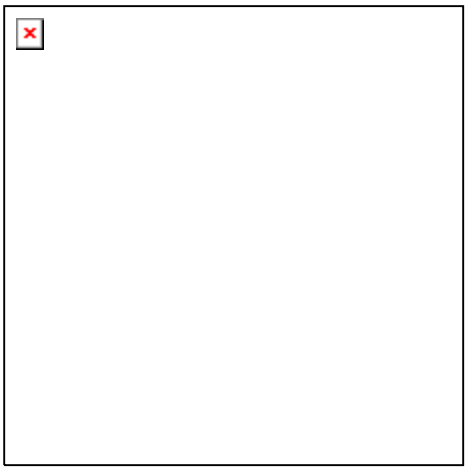
Vá até a guia QReport, selecione o primeiro componente e coloque em seu form. Este componente criará uma espécie de folha de desenho, deixando-o como na figura abaixo:



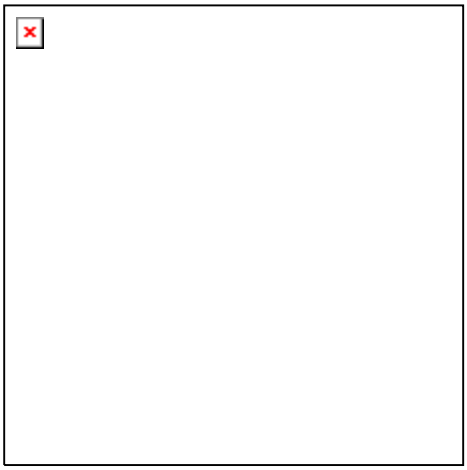
Defina o Name o objeto QuickReport para QuickRepProdutos. Na propriedade ReportTitle escreva *Listagem de Produtos*.

Insira um Table, altere o Name para TableProdutos e configure-o para acessar nossa tabela de produtos. Retorne ao QuickRepProdutos e na propriedade DataSet selecione o TableProdutos.

Clicando com o botão da direita sobre este componente, selecione no menu de contexto a opção Report Settings. Será exibida uma caixa de diálogo que permitirá definir o tamanho do papel, seu lay-out (retrado ou paisagem), margens, etc.



No grupo Bands selecione Page header, Detail band e Page footer, clicando em seguida no OK. Isso introduzirá três band's (áreas de relatório) já configurados para exibir cabeçalho, área de dados e rodapé. (Também é possível inserir e configurar as band's manualmente, através do componente QRBand que se encontra na mesma guia).



Insira no topo do primeiro band (baixa) um componente QRSysData, configure a propriedade Data para qrsReportTitle, Aligment para taCenter e AlignToBand como True. Altere a fonte (propriedade Font) para Arial, Negrito tamanho 16.
Aumente a altura deste primeiro band (utilizando o mouse) para abrir espaço para os outros itens.
Insira quatro componentes QRLabel na primeira band (faixa), definindo seus captions respectivamente para *Código*, *Descrição*, *Preço* e *Unidade*.

No band do meio, insira quatro componentes QRDBText e configure a propriedade DataSet de todos eles para TableProdutos. Depois selecione na propriedade DataField o campo correspondente (Codigo, Descricao, Preco e Unidade).

Na última band, adicione um componente QRSysData configurando a propriedade Data para qrsPageNumber, e a propriedade Text para *Página:* Altere também Alignment para taRightJustify e AlignToBand para True.

Retorne ao FormListagemPedidos e no evento OnCreate insira a linha:

TableProdutos.Open;

Insira também no evento OnClose a instrução complementar para fechar a tabela:

TableProdutos.Close;

Salve-o com o nome de ListagemPedidos. Remova a criação automática.

Vá até o formulário principal e acione no menu da aplicação a opção para Produtos|Listagem e insira as linhas abaixo:

```
Application.CreateForm(TFormListagemProdutos,FormListagemProduto);
FormListagemProdutos.QuickRepProdutos.PreviewModal;
FormListagemProdutos.Free;
```

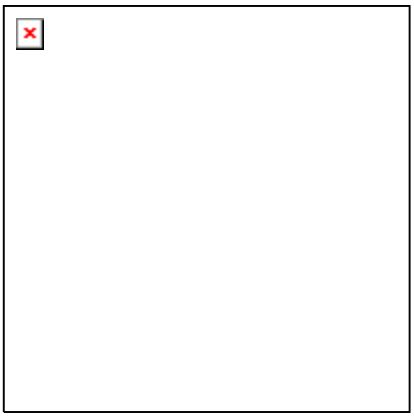
Observe que não chamamos Formulário.Show nem Formulário.ShowModal, estamos acessando o componente QuickReport. O método PreviewModal é responsável por exibir uma janela de visualização de impressão (Idem ao método Preview, mas PreviewModal bloqueia a execução de outras rotinas até que seja fechado). Se quiséssemos fazer a impressão diretamente, utilizaríamos o método Print no lugar de PreviewModal ou Preview.

Uma melhoria que você poderia fazer seria criar um formulário com dois botões e chamar seu relatório a partir deste outro formulário. Um botão para Imprimir e outro para Visualizar.

Agora você deve ter condições de criar as outras listagens também. Tente fazer a listagem de Clientes e de Pedidos. Para a listagem de pedidos você irá precisar configurar dois componentes Table, como fizemos no FormProcuraPedido, para que o nome do cliente possa ser exibido.

Explicarei apenas a Impressão do Pedido, por possuir detalhes que não aprendemos ainda.

Criando a impressão do Pedido



Para a criação da Impressão do Pedido, o procedimento é semelhante ao anterior, no entanto, no Report Settings ao invés de selecionarmos o Page footer selecionamos a band Summary, que será responsável por fazer a totalização de todos os itens. Serão necessários quatro componentes Table, um para cada tabela que possuímos. A configuração dos Tables é a mesma utilizada no cadastro de pedidos, a única diferença é que não criamos o campo Calculated no TableItens. Se você deseja, pode selecionar os componentes Table e o DataSource DSPedidos, copiá-los (Ctrl + C), vir até o form que você está criando para a impressão do pedido e cola-lo (Ctrl+V). Assim, só será necessário excluir o campo Calculated (Total) do TableItens. Para praticar você deveria inserir os Tables e o DataSource tentar configura-los segundo as instruções do cadastro de pedidos, não se esquecendo de interligar as tabelas de Pedidos e itens através das propriedades MasterSource e MasterFields, como fizemos lá.

O segredo deste relatório está nos componentes que aparecem circulados na figura. São componentes QRExpr, que permite a digitação de fórmulas e expressões. Insira um na band detail (do meio) e na propriedade Expression digite: **TableItens.Quantidade * TableItens.Preco**.
Isso fará com que o total seja calculado. Você também pode definir uma máscara na propriedade Mask, usando o seguinte: **###,###,##0.00**
Assim, teremos a separação de decimais e milhar. Por se tratar de valor, você deveria alinha-lo pela direita, isso é feito na propriedade Alignment, definindo-a como taRightJustify. (Pode fazer o mesmo para os campos quantidade e preço.

Na band summary (terceira) insira o mesmo componente, com as mesmas configurações, alterando apenas o conteúdo em Expression para:

SUM(TableItens.Quantidade * TableItens.Preco)

Esta expressão irá fazer uma somatória de todas as linhas com esta fórmula (quantidade x preço).

Defina o Name do Form para FormImprimePedido. Insira no evento OnCreate o seguinte código:

```
// abre as tabelas
TableProdutos.Open;
TableClientes.Open;
```

```
TablePedidos.Open;
TableItens.Open;
```

E no evento OnDestroy este outro:

```
// fecha as tabelas
TableProdutos.Close;
TableClientes.Close;
TablePedidos.Close;
TableItens.Close;
```

Salve este formulário como ImprimePedido.pas. Remova sua criação automática e na opção correspondente no menu do FormPrincipal digite o código em destaque:

```
procedure TFormPrincipal.mnuImpPedClick(Sender: TObject);
var
  NumPedido: Integer;
  Texto: String;
begin
  Application.CreateForm(TFormImprimePedido,FormImprimePedido);
  if (FormCadPedidos <> NIL) and
    (FormCadPedidos.TablePedidosNumeroPedido.AsInteger > 0) then
    NumPedido := FormCadPedidos.TablePedidosNumeroPedido.AsInteger
  else
  begin
    Texto := '';
    if InputQuery('Digite o número do Pedido',
                  'Impressão de Pedido',Texto) then
    begin
      try
        NumPedido := StrToInt(Texto);
      except
        ShowMessage('Número inválido!');
        // libera o formulario de relatorio
        FormImprimePedido.Close;
        Exit;
      end;
    end
    else
    begin
      FormImprimePedido.Close;
      Exit;
    end;
  end;
  if not FormImprimePedido.TablePedidos.FindKey([NumPedido]) then
  begin
    ShowMessage('Pedido não encontrado!');
```

```
      FormImprimePedido.Close;
      Exit;
   end;

   FormImprimePedido.QuickRepPedidos.PreviewModal;
   FormImprimePedido.Free;
end;
```

Salve o projeto e execute. Você notará que se chamar este relatório com o cadastro de pedidos aberto, ele apresentará a visualização do pedido atualmente selecionado. Se o mesmo estiver fechado, será solicitado o número do pedido a ser impresso/visualizado.

Criando uma caixa de diálogo sobre o sistema

Clique no botão New do SpeedBar e na caixa de diálogo New Itens, selecione a página Forms. Escolha o objeto AboutBox e clique em OK.

Será criado um Form semelhante ao dá figura anterior. Altere o Name para FormSobre, o Caption para *Sobre* e no evento OnClose do mesmo digite o seguinte:

```
Action := caFree;
FormSobre := NIL;
```

Utilize os Label's para digitar seus dados. E modifique as fontes como achar mais adequado. Veja exemplo abaixo:

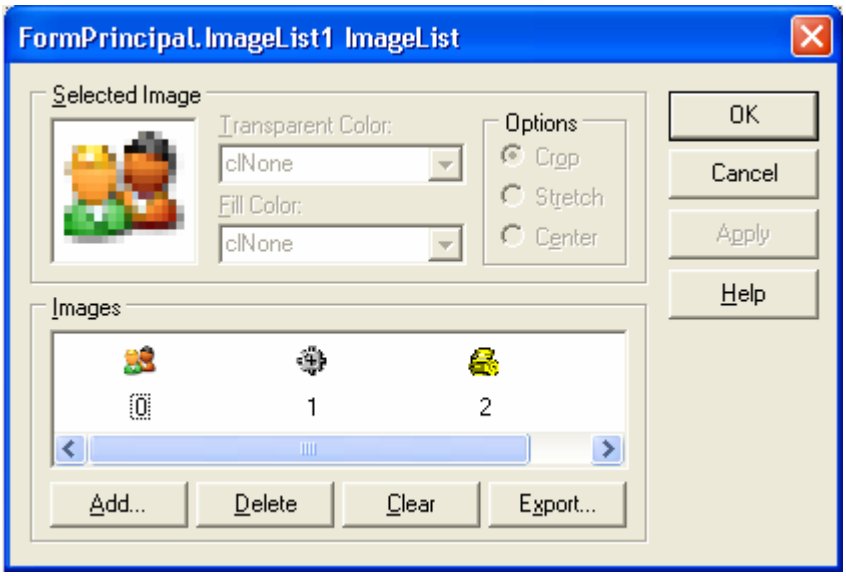
Criando uma barra de Ferramentas

Vá até a guia Win32 e insira no FormPrincipal um componente ToolBar e um componente ImageList.

Clique com o botão da direita do mouse sobre o componente ToolBar (que se alinhou automaticamente na parte superior do Form, logo abaixo do menu) e escolha a opção New Button no menu de contexto.
Repita este procedimento mais duas vezes, inserindo mais dois botões. Clique novamente com o botão da direita do mouse e escolha a opção New Separator. Repita a operação inserindo mais um botão.

O componente ImageList é responsável por manter uma lista de figuras que pode ser associada a menus e barras de ferramenta.

Dê um clique duplo no componente ImageList para abrir a caixa de diálogo para inclusão de figuras (ícones e bmp).



Para inserir um ícone/bmp, clique no botão Add. Selecione quatro ícones ou bmp (pequenos, específicos para botão) para o Cadastro de Clientes, Produtos, Pedidos e uma Calculadora (existe uma no diretório do Delphi). As figuras que usamos nos BitBtn também servem, mas por possuírem duas imagens (e consequentemente o dobro da largura) você deve aceitar que o ImageList as desmembre em duas figuras. Neste caso você exclui a última selecionando-a e clicando em Delete. Quando inserir todas as imagens clique em OK.

Altere a propriedade Images do ToolBar, apontando para o componente ImageList.

Vá ao Object Inspector e no evento OnClick dos três primeiros botões aponte-os para o evento OnClick das opções de menu correspondentes (CadCli, CadProd e CadPed. Para o último botão, o da calculadora, dê um duplo clique sobre o botão ou acione o evento OnClick no Object Inspector e insira o seguinte código:

```
WinExec('CALC.EXE',0);
```

WinExec é uma função da API do Windows, responsável por executar outros programas. Este comando fará com que a calculadora do Windows seja chamada. Você pode fazer o mesmo com o Bloco de Notas (NotePad.exe), o Word (WinWord.exe) ou qualquer outro aplicativo de seu micro. Dependendo do programa o Windows pode não localiza-lo, neste caso seria necessário informar o Path (caminho) completo. Por exemplo, para chamar o Delphi:

```
WinExec('C:\Arquivos de Programas\Borland\Bin\Delphi.exe',0);
```

Exercícios:

1) Para que serve o Componente ImageList?
2) Qual é o Componente responsável pela criação de relatórios?
3) Para que serve o componente QRExpr?
4) Como são construídos os relatórios?
5) Para que serve a função WinExec?

Apêndice A

Object Pascal: Estruturas Básicas

Palavras reservadas

As palavras reservadas independente de qualquer biblioteca, fazem parte do núcleo da linguagem, e por isso são utilizadas por qualquer compilador Delphi. Normalmente elas estão em evidencia no programa (negrito).

Palavra	Descrição
Begin	Início de bloco de programa.
End	Fim de Bloco de Programa.
Type	Definição de tipos.
Var	Definição de variáveis.
Uses	Definição de bibliotecas.
Class	Definição de classes.
Implementation	Seção da unidade, onde estão todas as definições da mesma, variáveis, tipos, constantes, bibliotecas, etc., que são privadas à unidade. Além isso está todo o código dos métodos da unidade.
Interface	Seção da unidade, onde estão todas as definições da mesma, variáveis, tipos, constantes, bibliotecas, etc., que serão tornadas públicas pela unidade.
Do	Comando interno de execução de algum outro comando.
while	Comando de repetição.
for	Comando de repetição.
if	Comando de seleção.
else	Comando de exceção a uma seleção.
Case	Comando de seleção.
Then	Comando de auxiliar de seleção.
Public	Propriedades e Métodos que podem ser acessados por qualquer unidade do programa.
Private	Propriedades e Métodos que só podem ser acessados pela própria unidade.
Protected	Propriedades e Métodos que podem ser acessados apenas pela própria unidade e pelos métodos das unidades derivadas desta.
Unit	Chave para a declaração do nome da unidade.
Array	Palavra para a definição de vetores.
Of	Comando auxiliar para definições de tipos.
Repeat	Comando de Repetição.
Until	Comando auxiliar na repetição.

With	Comando utilizado para referir-se a um registro de um objeto de forma direta.

Tipos de Dados

Tipos Inteiros

Tipo	Domínio	Tamanho
Shortint	-128 à 127	1
Smallint	-32768 à 32767	2
Longint	-2147483648 à 2147483647	4
Byte	0 à 255	1
Word	0 à 65535	2
Integer	-32768 à 32767	2
Cardinal	0 à 65535	2
Integer	-2147483648 à 2147483647	4 (32-bits)
Cardinal	0 à 2147483647	4 (32-bits)

Tipos Booleanos

Tipo	Descrição
Boolean	1 byte, assume apenas valores TRUE ou FALSE
ByteBool	É um byte com características booleanas, 0(FALSE) e diferente de 0 (TRUE)
WordBool	É um Word com características booleanas, 0(FALSE) e diferente de 0 (TRUE)
LongBool	É um Longint com características booleanas, 0(FALSE) e diferente de 0 (TRUE)

Tipos Caracteres

Tipo	Descrição
Char	1 byte, caracter
String	n bytes, caracteres

Tipos Reais

Tipo	Domínio	Alg. Sig.	Tam.
Real	2.9 x 10-39 à 1.7 x 1038	11-12	6
Single	1.5 x 10-45 à 3.4 x 1038	7-8	4
Double	5.0 x 10-324 à 1.7 x 10308	15-16	8
Extended	3.4 x 10-4932 à 1.1 x 104932	19-20	10
Comp	-263+1 à 263 -1	19-20	8
Currency	-922337203685477.5808 à 922337203685477.5807	19-20	8

Definição de Arrays

Um array é um vetor de dados de um determinado tipo. Ele representa uma lista com uma determinada característica, e é definido no Delphi da mesma forma como era definido no Pascal.

Var

Nome_da_variável: **array**[1..n] **of** Tipo_da variável; // ou

Nome_da_variável: **array**[1..n,1..m,1..x,..] **of** Tipo_da variável; // Para matrizes

Ex.:

X: **array**[1..10] **of** Integer; // Vetor de 10 elementos inteiros

S: **array**[1..50] **of** Double; // Vetor de 50 elementos reais

M: **array**[1..10,1..20] **of** Boolean; // Matriz booleana de 10x20

Comentários

Comentário de Uma linha: // comentário

Comentários de várias linhas { comentário }

Formas de Atribuição

O Delphi utiliza a mesma forma de atribuição do Pascal (:=). A única diferença é que a atribuição foi estendida aos objetos e aos novos tipos do Delphi.

Ex.:

X := 10 + Y;

Form1 := Form2; Operadores:

O Delphi Possui uma estrutura de operadores muito parecida com a do pascal, apenas com a inclusão de alguns novos operadores.

Operadores

Operadores Aritméticos

Operador	Operação	Tipos Usados	Tipos Resultantes
+	Adição	Inteiros Reais	Inteiro Real
-	Subtração	Inteiros Reais	Inteiro Real
*	Multiplicação	Inteiros Reais	Inteiro Real
/	Divisão	Inteiros Reais	Real Real
Div	Divisão inteira	Inteiros	Inteiro
Mod	Resto da divisão inteira	Inteiros	Inteiro

Operadores Unários

Operador	Operação	Tipos Usados	Tipos Resultantes
+	Identidade do sinal	Inteiros Reais	Inteiro Real
-	Negação de sinal	Inteiros Reais	Inteiro Real

Operadores Lógicos (Bit a Bit)

Operador	Operação	Tipos Usados	Tipos Resultantes
Not	Negação	Inteiros	Booleano
And	E	Inteiros	Booleano
Or	OU	Inteiros	Booleano
Xor	OU Coincidente (Exclusivo)	Inteiros	Booleano
Shl	Shift para a esquerda	Inteiros	Booleano
Shr	Shift para a direita	Inteiros	Booleano

Operadores Booleanos

Operador	Operação	Tipos Usados	Tipos Resultantes
Not	Negação	Booleanos	Booleano
And	E	Booleanos	Booleano
Or	OU	Booleanos	Booleano
Xor	OU coincidente	Booleanos	Booleano

Operadores Relacionais

Operador	Operação	Tipos Usados	Tipos Resultantes
=	Igual	Tipos simples compatíveis, de classe, referencia de classe, ponteiros, conjunto, string ou string empacotado.	Booleano

Operador	Operação	Tipos Usados	Tipos Resultantes
<>	Diferente de	Tipos simples compatíveis, de classe, referencia de classe, ponteiros, conjunto, string ou string empacotado.	Booleano

<	Menor que	Tipos simples compatíveis, strings ou string empacotado ou Pchar	Booleano
>	Maior que	Tipos simples compatíveis, strings ou string empacotado ou Pchar	Booleano
<=	Menor ou igual	Tipos simples compatíveis, strings ou string empacotado ou Pchar	Booleano
>=	Maior ou igual	Tipos simples compatíveis, strings ou string empacotado ou Pchar	Booleano
<=	Subconjunto de	Tipos de conjuntos compatíveis	Booleano
>=	Superconjunto de	Tipos de conjuntos compatíveis	Booleano
In	Membro de	Operando da esquerda, qualquer tipo ordinal; Operando da direita, conjunto cuja base seja compatível com o operando da esquerda	Booleano
Is	Compatível a	Tipos de conjuntos, mais especificamente classes	Booleano

Operadores Especiais

Operador	Operação	Tipos Usados	Tipos Resultantes
@	Atribuição	Qualquer	Ponteiro
As	Relação	Classes	Classe

Precedência dos Operadores

Precedência	Operadores
Primeiro	@, not, -(unário)
Segundo	*,/, div, mod, and, shl, shr, as
Terceiro	+,-,or, xor
Quarto	=, <, >, <=, >=, in, is

Comandos de seleção

If Then

O if é a estrutura de seleção mais simples que existe, e está presente em todas as linguagem de programação. Nele, uma condição é testada, se for

verdadeira ira executar um conjunto de comandos, se for falsa, poderá ou não executar um outro conjunto de comandos. A sua estrutura básica é:

```
If condição Then
  Begin
        Comandos executados se a condição for verdadeira;
  End
 Else
   Comandos serem executados se a condição for falsa;
```

Ex.:

If x=0 Then
  Write('x é zero');
	If x=0 Then
	  Write('x é zero');
	Else
	  Write('x não é zero');

A utilização dos marcadores de início e fim (Begin e End), e considerada opcional, se for executado apenas um único comando. O If ainda permite o encadeamento de If's.

Ex.:

```
If x=0 Then
  Write('X é zero');
Else
   If x>0 Then
     Write('X é positivo');
   Else
     Write('X é negativo');
```

Case

A instrução case consiste em uma expressão usada para selecionar um valor em uma lista de possíveis valores, ou de faixa de valores. Estes valores são constantes e devem ser únicos e de tipo ordinal. Finalmente pode haver uma instrução else que será executada se nenhum dos rótulos corresponder ao valor do seletor. O seu formato básico é:

```
Case Seletor of
  Const1: Begin
            Comandos referentes a constante 1.
          End;
  Const2: Begin
            Comandos referentes a constante 2.
          End;
  Faixa1..Faixa2: Begin
                    Comandos referentes a Faixa de valores.
                  End;
Else
      Comandos referentes ao else
End;
```

Ex.:

```
Case Numero of
    1: texto := ‘um’;
    2: texto := ‘dois’;
    3: texto := ‘três’;
end;
```

```
Case MeuChar of
    ‘+’: Texto := ‘Sinal de mais’;
    ‘-‘: Texto := ‘Sinal de Menos’;
    ‘0’..’9’: Texto := ‘Números’;
else
     Begin
          Texto := ‘Caracter desconhecido’;
          Meuchar := ‘?’;
     End;
```

A utilização dos marcadores de início e fim (Begin e End), e considerada opcional, se o caso for executar apenas um único comando. O case ainda pode ser encadeado, ou seja um case dentro do outro.

Comandos de repetição

For

O loop for no Pascal baseia-se estritamente em num contador, o qual pode ser aumentado ou diminuído cada vez que o loop for executado. O contador é inicializado, e o loop irá se repetir enquanto o contador não chegar ao fim da contagem. O seu formato básico é:

```
For contador := inicio_da_contagem  to fim_da_contagem do
   Begin
        Comandos;
   End;
```

Ex. aumentando:

```
K := 0;
For i:=1 to 10 do
    K :=  K + i;
```

Ex. diminuindo:

```
K := 0;
For i:=10 downto 1 do
    K :=  K + i;
```

A utilização dos marcadores de início e fim do loop (Begin e End), e considerada opcional, se o loop for executar apenas um único comando. O for ainda pode ser encadeado, ou seja um for dentro do outro:

Ex.:

```
K := 0;
W:= 0;
For i:=1 to 10 do
   For j:=1 to 10 do
       K :=  K + i * j;
```

While

O while é um comando de repetição que não possui controle de um contador e que testa a condição antes de executar o próximo loop. Este loop irá se repetir enquanto a condição for verdadeira. A sua forma básica é:

```
While condição
   Begin
        Comandos;
   End;
```

Ex.:

```
I:=10;
J:=0;
While I>J do
    Begin
        I  := I –1;
         J := J + 1;
     End;
```

No caso do while, as regras do for para os marcadores de início e fim do loop e a do encadeamento, também são válidas.

Repeat Until

O Repeat é um comando de repetição que não possui controle de um contador e que testa a condição depois de executar o loop. Este loop irá se repetir até que a condição seja verdadeira. A sua forma básica é:

```
Repeat
        Comandos;
Until condição
```

Ex.:

```
I:=10;
J:=0;
Repeat
        I  := I –1;
         J := J + 1;
Until J>I;
```

No caso do Repeat, as regras do for para os marcadores de início e fim do loop e a do encadeamento, também são válidas.

Procedimentos e funções

Um conceito importante que integra linguagens como o Pascal, e consequentemente o Object Pascal do Delphi, é o da sub-rotina. No Pascal as subrotinas podem assumir duas formas diferentes: procedimentos ou funções. A única diferença real entre as duas, é que as funções têm um valor de retorno, enquanto os procedimentos não. Abaixo está a sintaxe do Pascal para procedimentos e funções:

Procedure ola;
Begin
ShowMessage(‘Olá !’);
End;
	Function Dois: Integer;
	Begin
	Result := 2;
	End;

Os procedimentos e funções podem ainda ter parâmetros de entrada de qualquer tipo definido.

Procedure ImprimeMensagem (Mens: string);
Begin
ShowMessage(Mens);
End;
	Function Duplo (valor: Integer): Integer;
	Begin
	Result := valor * 2;
	End;

Estes parâmetros de entrada, podem ainda ser de passagem por valor, como o visto nos exemplos acima, ou de passagem por referência, como o visto abaixo:

Procedure ImpMens (Var Mens: string);
Begin
ShowMessage(Mens);
End;
	Function Duplo (Var valor: Integer): Integer;
	Begin
	Duplo := valor * 2;
	Valor := 0;
	End;

Definições básicas da Orientação ao Objetos

Se você olhar o mundo real, nosso idioma tem dois componentes principais: substantivos (objetos) e verbos(operações). Para que nossas aplicações reproduzam a realidade, nossa linguagem de computador precisa fazer o mesmo. A maioria das linguagens “tradicionais” tem uma grande variedade de operações que pode efetuar, mas um pequeno conjunto de substantivos para descrever os objetos.

Como você define um objeto? Um objeto é uma entidade que tem um estado; isto é, tem algum valor. O comportamento de um objeto é definido pelas ações que ele sofre e vice-versa. Todo objeto é, na verdade, uma instância de alguma classe de objetos.

O objetivo da Orientação a Objetos é que cada módulo do sistema represente um objeto ou uma classe de objetos do mundo real. Um programa que implementa um modelo de realidade pode, assim, ser visto como um conjunto de objetos que interage com o outro. Você pode projetar um sistema usando essa mentalidade orientada a objetos, segundo estes passos:

1. Identificar os objetos e seus atributos
2. Identificar as operações que afetam cada objeto, e as operações que cada objeto deve iniciar

3. Estabelecer a visibilidade de cada objeto em relação aos outros objetos.
4. Estabelecer a interface de cada objeto.
5. Implementar cada objeto.

Classes e objetos

Estes são dois termos que freqüentemente são erroneamente utilizados. Esclarecendo isto, as suas definições são as seguintes:

- Uma classe é um tipo de dado definido pelo usuário, o qual tem um estado, uma representação e algumas operações ou comportamentos. Uma classe tem alguns dados internos e alguns métodos, na forma de procedimentos e funções. Uma classe usualmente descreve as características genéricas e o comportamento de uma série de objetos muito semelhantes. As classes são usadas pelo programador para organizar o código fonte e pelo compilador para gerar o aplicativo.
- Um objeto é uma instancia de uma classe, ou, usando outras palavras, é uma variável do tipo de dados definido pela classe. Objetos são entidades reais. Quando o programa é executado, os objetos ocupam parte da memória para a sua representação interna.

Propriedades, Métodos e Eventos

Pela definição da orientação à objetos de uma classe, ela pode conter métodos, propriedades, métodos e eventos. As propriedades são os atributos que definem a classe (são as variáveis do Pascal). Os métodos são as operações que a classe pode realizar, nas quais o programador tem o controle de sua execução (são os procedimentos e funções do Pascal). Os eventos são as operações que a classe pode realizar, nas quais o programador não tem o controle sobre a sua execução. A chamada da execução de um evento é feita pelo próprio sistema operacional (Windows 95 ou NT). Cada classe pode conter suas propriedades, métodos e eventos específicos, mas nas classes definidas pelo Delphi existem algumas propriedades e eventos, que são bem comuns a grande maioria. Entre eles estão:

Propriedades genéricas:

Propriedade	**Descrição**	**Valores**
Caption	String que se refere ao objetivo da classe. Esta string é mostrada no componente relacionado a classe.	String
Color	Cor atribuída a classe. Em alguns casos ela está relacionada com a cor de fundo do componente.	Cor
Cursor	Tipo de cursor a ser mostrado quando o mouse estiver sobre o componente	Cursor
Enable	Se o componente esta habilitado ou não	True, False
Font	Fonte de letra utilizado pela classe	Fonte de letra
Height	Altura do componente, em relação ao Top	Integer

Propriedade	Descrição	Valores
Hint	String que aparece sobre um componente, quando o mouse fica algum tempo parado sobre este.	String
Left	Posição esquerda do componente. Se o componente for um form, este será em relação a tela, se for outro qualquer, será em relação ao form aonde ele estiver.	Inteiro
Name	Nome da propriedade para o código	Nome
ShowHint	Habilita ou desabilita o Hint	True, False
Top	Posição superior do componente. Se o componente for um form, este será em relação a tela, se for outro qualquer, será em relação ao form aonde ele estiver.	Inteiro
Visible	Torna a classe visível ou Invisível	True, False
Width	Largura do componente, em relação ao Left.	Integer

Eventos genéricos:

Evento	Descrição
OnClick	É executado quando o mouse é clicado encima do componente
OnClose	É executado quando um form é fechado
OnCreate	É executado logo após a criação do componente.
OnDblClick	É executado quando o mouse é clicado duas vezes consecutivas sobre o componente
OnExit	É executado no termino do programa.
OnKeyPress	É executado quando o componente esta selecionado e alguma tecla for apertada.
OnMouseMove	É executado quando o mouse se movimenta sobre o componente.
OnShow	É executado antes de exibir o componente

Código básico

```
unit Unit1;    // Nome da unidade

interface      // Inicio da unidade

uses           // Bibliotecas Adicionadas
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;

type           // Tipos definidos
  TForm1 = class(TForm)    // Definição de uma Classe
  private
```

```
   { Definição dos métodos e propriedades privados à unidade }
  public
   { Definição dos métodos e propriedades públicos }
  end;

var           // Variáveis definidas
  Form1: TForm1;

implementation // Início do fonte dos métodos da unidade

{$R *.DFM}     // Código declara a existência de um form relacionado ao código
                 // da unidade. Este form terá o mesmo nome da unidade, mas
com
                 // a extensão DFM.

            // Espaço destinado ao código fonte dos métodos da unidade.

end.          // Última linha da unidade
```

Relações de uma Classe

Forma de uma classe no Object Pascal

Uma classe deve ser definida dentro da seção de tipos (**Type**). O código desta classe será implementado dentro da seção (**Implementation**) da mesma unidade. Nenhuma parte do código desta classe pode estar em outra unidade.

Na definição dos métodos e propriedades de uma classe, podemos considerá-los públicos, privados ou protegidos. A finalidade disto é a de tornar a classe mais fácil de utilizar, pois estaremos escondendo aquele código auxiliar que é desinteressante ao usuário da nossa classe, bem como traz consigo também o aspecto segurança da classe, pois poderemos deixar visível somente as propriedades e métodos que a nossa classe possuí total controle. Abaixo esta a forma com deve ser definida uma classe na linguagem Object Pascal.

```
NomeDaClasse = class
  public
   { Definição dos métodos e propriedades públicos }
  private
   { Definição dos métodos e propriedades privados à unidade }
  protected
   { Definição dos métodos e propriedades protegidos }
  end;
```

O construtor

O construtor é o evento de uma classe que é executado no momento da criação do objeto. Este evento é normalmente utilizado para a inicialização, e pode ter qualquer quantidade de parâmetros de entrada, e nenhum parâmetro de saída.

Ex.:

```
Data = Class
        Dia, Mes, ano: Integer;
        Constructor init(d, m, a: Integer);
        Procedure Ledata(Var d,m,a: Integer);
        Procedure gravadata(d,m,a: Integer);
End;
```

Um construtor pode ter qualquer nome, e cada classe pode ter apenas um construtor.

O destrutor

O destrutor é o evento de uma classe que é executado no momento do fechamento do objeto. Este evento é normalmente utilizado para a liberação de memória alocada dinamicamente pelo objeto. Ele não possui parâmetros de entrada ou saída:

Ex.:

```
Data = Class
        Dia, Mes, ano: Integer;
                                        Constructor init(d, m, a: Integer);
        Procedure Ledata(Var d,m,a: Integer);
        Procedure gravadata(d,m,a: Integer);
        Destructor destroi;
End;
```

Métodos estáticos, dinâmicos e virtuais

Os métodos do Object Pascal usualmente baseia-se na ligação estática. Isto significa que uma chamada de método é resolvida pelo compilador e pelo linkeditor. Eles Substituem a chamada por uma chamada à localização de memória específica em que a função ou procedimento reside, a qual é conhecida como endereço da função.

As linguagens orientadas a objetos, permitem o uso de outra forma de ligação conhecida como ligação dinâmica. Neste caso, o endereço de fato a ser chamado é determinado em tempo de execução. A vantagem desta abordagem é conhecida como poliformismo, como veremos mais adiante.

Existem duas formas de se realizar esta ligação dinâmica, a ligações do tipo **dynamic** e **virtual**. Elas são equivalentes sendo a sua única diferença a implementação da forma de chamada dos métodos em tempo de execução. Esta diferença faz com que o método **virtual** tenham um melhor desempenho em termos da velocidade, que o do método **dynamic**. Em contrapartida o método **dynamic** geram um código executável menor. Em modo geral o método **virtual** é a forma mais eficiente para a implementação do poliformismo. Métodos do tipo **dynamic** são utilizados somente em situações onde a classe

básica declara um número muito grande de métodos não estáticos, e a aplicação declara um número muito grande de classes descendentes com pouca utilização do poliformismo.

Relações entre Classes

Uma classe pode acessar métodos de outra classe, desde que estes sejam do tipo público, colocando-se o nome completo, ou seja, o nome da classe ponto o nome do método (MinhaClasse.Meumetodo).

Herança

A herança é um dos recursos mais poderosos de uma linguagem orientada a objetos. Ela permite que as classes secundárias assumam as propriedades de suas classes principais. Elas herdam propriedades, métodos e eventos de sua classe principal, a classe secundária pode acrescentar novos componentes naqueles que herdam. Isso permite que você peque uma classe que tenha praticamente todas as partes fundamentais de que precisa, e insira novos objetos que a personalizam exatamente de acordo com suas necessidades.

Quando uma classe secundária herda propriedades de uma primária, a classe secundária terá acesso a todas as propriedades, métodos e eventos, que forem públicos ou protegidos.

Inherit

Esta é uma palavra reservado do Object Pascal que é utilizada antes de um método. Ela pode ser utilizada quando uma classe secundária, possui um método com o mesmo nome de um da classe primária. Neste caso se nos quisermos acessar o método da classe primária teremos que utilizar o Inherit antes deste método.

Override

Esta é uma palavra reservada do Object pascal que é utilizada após um método quando em uma classe secundária, queremos redefinir uma função definida em uma classe primária. Baseado nas definições de override e das ligações dinâmicas e que se define o poliformismo.

Poliformismo

O poliformismo é um recurso das linguagens orientadas a objetos, que literalmente indica a capacidade de um objeto assumir várias formas. Em outras palavras, o poliformismo permite que você referencie propriedades e métodos de classes diferentes por meio de um mesmo objeto. Ele também possibilita que você execute operações com esse objeto de diferentes maneiras, de acordo com o tipo de dado e com a classe atualmente associada a esse objeto.

Por exemplo, posso declarar um objeto genérico da classe caixa (digamos Minhacaixa) e então associar a ele objetos da classe Tcaixadeferramentas ou Tcaixadedinheiro. Agora suponhamos que cada classe tenha um procedimento Open, cada um deles com uma implementação diferente. Esse método deve usar a ligação dinâmica, isso significa declará-la na classe-pai como virtual ou dynamic e redefini-la como override na classe-filha.

Quando você aplica o método Open a MinhaCaixa, o que acontece? É chamado o procedimento Open do tipo atual do objeto. Se Minhacaixa for atualmente um objeto Tcaixadedinheiro, a caixa será aberta mediante uma senha secreta, se for Tcaixadeferramentas a caixa será simplesmente aberta.

Descrição dos Principais Procedimentos e Funções Pré definidas

procedure Beep;

Toca um beep

procedure ChDir(S: string);

Troca o diretório corrente para o diretório especificado em S.
Ex.:

```
begin
  {$I-}
  { Muda para o diretório especificado em Edit1 }
  ChDir(Edit1.Text);
  if IOResult <> 0 then
    MessageDlg('Diretório não encontrado', mtWarning, [mbOk], 0);
end;
```

function Chr(X: Byte): Char;

Retorna o caracter com o código ASCII X
Ex.:

```
begin
  Canvas.TextOut(10, 10, Chr(65));  { A letra 'A'}
end;
```

function Concat(s1 [, s2,..., sn]: string): string;

Concatena as strings
Ex.:

```
var
  S: string;
begin
  S := Concat('ABC', 'DEF');     { 'ABCDE' }
end;
```

function Copy(S: string; Index, Count: Integer): string;

Retorna uma substring de S, começando a partir de Index e tendo Count caracteres
Ex.:

```
var S: string;
begin
  S := 'ABCDEF';
  S := Copy(S, 2, 3); { 'BCD' }
end;
```

function CreateDir(const Dir: string): Boolean;
Cria um novo diretório e retorna o sucesso da operação

function Date: TDateTime;

Retorna a Data atual
Ex.:
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := 'Hoje é  ' + DateToStr(Date);
end;
```

function DateToStr(Date: TDateTime): string;

Converte Data para String
Ex.:
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := DateToStr(Date);
end;
```

function DayOfWeek(Date: TDateTime): Integer;

Retorna o dia da semana especificado entre 1 e 7, onde domigo é um e Sábado é 7
Ex.:
```
procedure TForm1.Button1Click(Sender: TObject);
var
  ADate: TDateTime;
begin
  ADate := StrToDate(Edit1.Text);
  Label1.Caption := IntToStr(DayOfWeek(ADate)) + '° dia da semana';
end;
```

procedure DecodeDate(Date: TDateTime; var Year, Month, Day: Word);

Quebra os valores especificados no parâmetro Date em Year, Month e Day.
Ex.:
```
procedure TForm1.Button1Click(Sender: TObject);
var
  Present: TDateTime;
  Year, Month, Day, Hour, Min, Sec, MSec: Word;
 begin
  Present:= Now;
  DecodeDate(Present, Year, Month, Day);
  Label1.Caption := 'Hoje é ' + IntToStr(Day) + ' do Mês '
    + IntToStr(Month) + ' do Ano ' + IntToStr(Year);
  DecodeTime(Present, Hour, Min, Sec, MSec);
  Label2.Caption := 'A hora é ' + IntToStr(Hour) + ' horas e '
    + IntToStr(Min) + ' minutos.';
end;
```

procedure DecodeTime(Time: TDateTime; var Hour, Min, Sec, MSec: Word);

Quebra os valores especificados em Time nos par6ametros Hour, Min, Sec e MSec.
Ex.:

```
Procedure TForm1.Button1Click(Sender: TObject);
var
  Present: TDateTime;
  Year, Month, Day, Hour, Min, Sec, MSec: Word;
 begin
  Present:= Now;
  DecodeDate(Present, Year, Month, Day);
  Label1.Caption := 'Hoje é ' + IntToStr(Day) + ' do Mês '
    + IntToStr(Month) + ' do Ano ' + IntToStr(Year);
  DecodeTime(Present, Hour, Min, Sec, MSec);
  Label2.Caption := 'A hora é ' + IntToStr(Hour) + ' horas e '
    + IntToStr(Min) + ' minutos.';
end;
```

procedure Delete(var S: string; Index, Count:Integer);

Remove a substring de Count caracters da string S partir da posição Index
Ex.:

```
var
   s: string;
 begin
   s := 'Delphi 3 - Client/Server';
   Delete(s,8,4);
   Canvas.TextOut(10, 10, s); { ' Delphi 3 - Client/Server ' }
 end;
```

function DeleteFile(const FileName: string): Boolean;

Apaga o arquivo FileName do disco. Se o arquivo não puder ser apagado a função retorna False.
Ex.:

```
DeleteFile('TEMP0001.TMP');
```

function DirectoryExists(Name: string): Boolean;

Verifica se Name diretorio existe
Ex.:
function DiskFree(Drive: Byte): Integer;

Retorna o número de bytes livre no driver especificado em Drive.
Onde : 0 = Corrente, 1 = A, 2 = B,...
DiskFree retorna –1 se o driver for inválido
Ex.:

```
var
  S: string;
begin
  S := IntToStr(DiskFree(0) div 1024) + ' Kbytes livres.';
  Canvas.TextOut(10, 10, S);
end;
```

function DiskSize(Drive: Byte): Integer;

Retorna o tamanho em bytes do driver especificado.
Onde : 0 = Corrente, 1 = A, 2 = B,...
DiskFree retorna –1 se o driver for inválido
Ex.:

```
var
  S: string;
begin
  S := 'Capacidade de ' + IntToStr(DiskSize(0) div 1024) + ' Kbytes.';
  Canvas.TextOut(10, 10, S);
end;
```

function EncodeDate(Year, Month, Day: Word): TDateTime;

Retorna uma Data formada por Year, Month e Day
Ex.:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyDate: TDateTime;
begin
  MyDate := EncodeDate(83, 12, 31);
  Label1.Caption := DateToStr(MyDate);
end;
```

function EncodeTime(Hour, Min, Sec, MSec: Word): TDateTime;

Retorna a Hora formada por Hour, Min, Sec e MSec
Ex.:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyTime: TDateTime;
begin
  MyTime := EncodeTime(0, 45, 45, 7);
  Label1.Caption := TimeToStr(MyTime);
end;
```

function ExtractFileDir(const FileName: string): string;

Retorna o diretório adequado para ser passado para as funções CreateDir, GetCurrentDir, RemoveDir e SetCurrentDir.
O resultado da função é uma string vazia se FileName não contiver um drive e um caminho.

Function ExtractFileDrive(const FileName: string): string;

Retorna uma string contendo o drive do path de um arquivo.

Function ExtractFileExt(const FileName: string): string;

Retorna a extensão do arquivo FileName

function ExtractFileName(const FileName: string): string;

Retorna o nome do arquivo

```
Form1.Caption := 'Editando '+ ExtractFileName(FileName);
```

function ExtractFilePath(const FileName: string): string;

Retorna o Path de um arquivo
Ex.:

```
ChDir(ExtractFilePath(FileName));
```

function FileAge(const FileName: string): Integer;

Retorna a data e a hora de um arquivo num valor que pode ser convertido para TDateTime através da função FileDateToDateTime. Retorna –1 se o arquivo não existir

function FileExists(const FileName: string): Boolean;

Retorna verdade se o arquivo existir
Ex.:

```
if FileExists(FileName) then
  if MsgBox('Tem certeza que deseja excluir' + ExtractFileName(FileName)
    + '?'), []) = IDYes then DeleteFile(FileName);
```

function FileSize(var F): Integer;

Retorna o tamanho de um arquivo, para usar FileSize o arquivo deve esta aberto. Se o arquivo estiver vazio FileSize(F) retorna 0. F é uma variavel do tipo arquivo. FileSize não pode ser usada com arquivo texto
Ex.:

```
var
   f: file of Byte;
   size : Longint;
   S: string;
   y: integer;
 begin
   if OpenDialog1.Execute then begin
     AssignFile(f, OpenDialog1.FileName);
     Reset(f);
     size := FileSize(f);
     S := 'Tamanho do arquivo em bytes: ' + IntToStr(size);
     y := 10;
     Canvas.TextOut(5, y, S);
     y := y + Canvas.TextHeight(S) + 5;
     S := 'Posicionando no meio do arquivo...';
     Canvas.TextOut(5, y, S);
     y := y + Canvas.TextHeight(S) + 5;

     Seek(f,size div 2);
     S := 'Posição agora é ' + IntToStr(FilePos(f));
     Canvas.TextOut(5, y, S);
     CloseFile(f);
   end;
 end;
```

procedure FillChar(var X; Count: Integer; value: Byte);

Preenche um vetor com determinado caracter. Value pode ser um byte ou char
Ex.:

```
var
  S: array[0..79] of char;
begin
  { preenche tudo com espaços}
  FillChar(S, SizeOf(S), ' ');
end;
```

function FloatToStr(Value: Extended): string;

Converte um valor em ponto flutuante (real) para uma string

procedure ForceDirectories(Dir: string);

Cria multiplos diretórios de uma só vez
Ex.:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Dir: string;
begin
  Dir := 'C:\SISTEMAS\VENDAS\DADOS';
  ForceDirectories(Dir);
  if DirectoryExists(Dir) then
    Label1.Caption := Dir + ' foi criado'
end;
```

function FormatDateTime(const Format: string; DateTime: TDateTime): string;

Formata o valor DateTime usando o formato de Format.
Os especificadores de formato abaixo são válidos

Especificador Exibe
c Exibe a data usando o formato definido da variável global ShortDateFormat, seguido pela hora se o formato definido pela variável global LongTimeFormat. A hora não é exibida se a parte fracional de DateTime for zero.
d Exibe o dia como um número, sem zero à esquerda (1-31).
dd Exibe o dia como um número, com zero à esquerda (01-31).
ddd Exibe o nome do dia abreviado (Sun-Sat) usando as strings definidas pela variaável global ShortDayNames
dddd Exibe o nome do dia (Sunday-Saturday) usando as strings definidas pela variaável global LongDayNames.
ddddd Exibe a data utilizando o formato definido na variável global ShortDateFormat.
dddddd Exibe a data utilizando o formato definido na variável global LongDateFormat.
m Exibe o mês como um número sem zeros a esquerda (1-12). Se o especificador m for seguido de um especificador h ou hh, será exibido os minutos do mês.
mm Exibe o mês como um número com zeros a esquerda (01-12). Se o especificador m for seguido de um especificador h ou hh, será exibido os minutos do mês.
mmm Exibe o mês de forma abreviada (Jan-Dec) usando as strings definidas na variável global ShortMonthNames.
mmmm Exibe o mês por extenso (January-December) usando as strings definidas na variável global LongMonthNames.
yy Exibe o ano com dois dígitos (00-99).
yyyy Exibe o ano com quatro dígitos (0000-9999).
h Exibe a hora sem zero a esquerda (0-23).
hh Exibe a hora com zero a esquerda (00-23).
n Exibe o minuto sem zero a esquerda (0-59).
nn Exibe o minuto com zero a esquerda (00-59).
s Exibe o segundo sem zero a esquerda (0-59).
ss Exibe o segundo com zero a esquerda (00-59).
t Exibe a hora usando o formato definido na variável global ShortTimeFormat.
tt Exibe a hora usando o formato definido na variável global LongTimeFormat.
am/pm Exibe a hora no formato de 12 horas seguido de 'am' ou 'pm'.
a/p Exibe a hora no formato de 12 horas seguido de 'a' ou 'p'.

ampm Exibe a hora no formato de 12 horas de acordo com o conteúdo das variáveis globais TimeAMString (se antes do meio dia) e TimePMString (se depois do meio dia).
/ Exibe o caracter separador da data definido na variável global DateSeparator.
: Exibe o caracter separador de hora definido na variável global TimeSeparator.
Obs: Caracteres dentro de aspas simples ou duplas ('xx'/"xx") são exibidos como estão e não afetam a formatação.

Os especificadores de formato podem ser utilizados tanto em maiúsculo quanto em minúsculo - ambos produzem o mesmo resultado.

Se a string definida pelo parametro Format estiver vazia, os valores de data e hora serão formatados como as if a 'c' format specifier had been given.

```
S := FormatDateTime('"The meeting is on" dddd, mmmm d, yyyy, ' +

  '"at" hh:mm AM/PM', StrToDateTime('2/15/95 10:30am'));
```

function FormatFloat(const Format: string; Value: Extended): string;

Transforma um Float numa string usando a formatação contida em Format
Os especificadores de formato abaixo são válidos

Especificador	Representa
0	Posição do dígito. Se o valor sendo formatado tem um digito na posição que especificador '0' aparece na string de formatação, este dígito será copiado para a string de saída. Senão, um '0' é armazenado nesta posição na string de saída.
#	Se o valor sendo formatado tem um digito na posição que especificador '#' aparece na string formatada, este dígito será copiado para a string de saída. Senão, nada será arrmazenado na string de saída.
.	Ponto decimal. O primeiro caracter '.' na string de formatação determina a localização do ponto decimal no valor sendo formatado; quaisquer caracteres '.' adicionais são ignorados. O caracter que será utilizado como separador decimal é determinado pelo conteúdo da variável global DecimalSeparator. O valor padrão de DecimalSeparator é especificado no Painel de Controle do Windows no ícone Propriedades de Configurações Regionais, guia Número, ítem Símbolo decimal.
,	Separador de milhar. Se a string de formatação contém um ou mais caracteres ',' (vírgula), a saída terá caracteres separadores de milhar inseridos entre os grupos de três dígitos à esquerda do ponto decimal. A posição e o número de caracteres ',' na string de formatação não afetam a saída, exceto para indicar que separadores de milhar são necessários. O caracter que será utilizado como separador de milhar é determinado pelo conteúdo da variável global ThousandSeparator. O valor padrão de ThousandSeparator é especificado no Painel de Controle do Windows no ícone Propriedades de Configurações Regionais, guia Número, ítem Símbolo decimal.
E+	Notação científica. Se qualquer uma das strings 'E+', 'E-', 'e+', ou 'e-' estiverem contidas na string de formatação, o número é formatado usando notação científica. Um grupo de quatro caracteres '0' (zero) pode seguir imediatamente após os especificadores 'E+', 'E-', 'e+', ou 'e-' para determinar o número mínimo de dígitos no expoente. Os especificadores 'E+' e 'e+' fazem com que o sinal do expoente (positivo ou negativo) seja exibido. Os especificadores 'E-' e 'e-' fazem com que o sinal do expoente seja exibido apenas quando o mesmo for negativo.
'xx'/"xx"	Caracteres entre aspas simples ou duplas, não afetam a formatação.
;	Separa seções para números positivos, negativos e o zero na string de formatação.

Obs.: A localização do '0' extremo esquerdo antes do ponto decimal e do ‘0’ extremo direito após o ponto decimalna string de formatação determinam a faixa de dígitos sempre exibida na string de saída. O número sendo formatado formatado sempre é arredondado para o número de digitos definidos após o ponto decimal (com '0' ou '#'). Se a string de formatação não contém o ponto decimal, o valor sendo formatado será arredondado para o número mais próximo.
Se o número sendo formatado possui mais dígitos a esquerda do separador decimal que os definidos pelos especificadores de dígito na string de formatação, digitos extras serão exibidos antes do primeiro especificador de dígito.
Para definir formatação diferente para valores positivos, negativos, e zero, a string de formatação pode conter entre uma e três seções separadas por ‘;’ (ponto-e-vírgula).

Se a seção de valores positivos estiver vazia, ou se a string de formatação estivar vazia, o valor é formatado usando a formatação geral de números de ponto flutuante com 15 dígitos significativos, correspondendo a uma chamada a FloatToStrF com o formato ffGeneral. Formatação geral de ponto flutuante também é usada quanto o valor tem mais de 18 digitos à esquerda do ponto decimal e a string de formatação não especifica formatação com notação científica.

Exemplos de strings de formatação e valores resultantes:

0	1234	-1234	1	0	
0.00	1234.00	-1234.00		0.50	0.00
#.##	1234	-1234	.5		
#,##0.00	1,234.00		-1,234.00	0.50	0.00
#,##0.00;(#,##0.00)	1,234.00	(1,234.00)	0.50	0.00	
#,##0.00;;Zero	1,234.00		-1,234.00	0.50	Zero
0.000E+00	1.234E+03	-1.234E+03	5.000E-01	0.000E+00	
#.###E-0	1.234E3	-1.234E3	5E-1	0E0	

function Frac(X: Real): Real;

Retorna a parte fracional do parâmetro X

Ex.:

```
var
  R: Real;
begin
  R := Frac(234.567);    { 0.567 }
  R := Frac(-234.567);   { -0.567 }
end;
```

function GetCurrentDir: string;

Retorna uma string contendo o diretório corrente

procedure GetDir(D: Byte; var S: string);

Rerorna o diretório corrente do drive especificado.
O onde D pode ser :

Valor	Drive
0	Corrente
1	A
2	B
3	C

Ex.:

```
var
   s : string;
 begin
   GetDir(0,s); { 0 = drive corrente}
   MessageDlg('Drive e diretório atual: ' + s, mtInformation, [mbOk] , 0);
 end;
```

procedure Inc(var X [; N: Longint]);

Incrementa de uma ou N unidades o parâmetro X

Ex.:

```
var
  IntVar: Integer;
  LongintVar: Longint;
begin
  Inc(IntVar);        { IntVar := IntVar + 1 }
  Inc(LongintVar, 3;  { LongintVar := LongintVar + 3}
end;
```

function IncMonth(const Date: TDateTime; NumberOfMonths: Integer): TDateTime;

Retorna Date acrescido ou decrescido de NumberOfMonths meses.

function InputBox(const ACaption, Aprompt, ADefault: string): string;

Exibe uma Caixa de Entrada onde o usuário pode digitar uma string.
ACaption representa o título do Input Box e APrompt é o título do edit e ADefault representa o valor inicial do Edit.

Ex.:

```
uses Dialogs;
procedure TForm1.Button1Click(Sender: TObject);
var
  InputString: string;
begin
  InputString:= InputBox('Informe', 'Nome do arquivo a ser criado', 'MeuTexto.TXT');
end;
```

function InputQuery(const ACaption, APrompt: string; var Value: string): Boolean;

Semelhante ao InputBox, sendo que retorna True se o usuário fechou a O InputBox com OK e False se fechou com Cancel ou ESC. E Value armazena a string digitada.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  NewString: string;
  ClickedOK: Boolean;
begin
  NewString := 'MeuTexto.TXT';
  Label1.Caption := NewString;
  ClickedOK := InputQuery('Informe, 'Nome do arquivo a ser criado', NewString);
  if ClickedOK then                { NewString contém nova string digitada.}
    Label1.Caption := 'A nova string é ''' + NewString + '''';
end;
```

procedure Insert(Source: string; var S: string; Index: Integer);

Insere uma string em outra a partir da posição Index

Ex.:

```
var
  S: string;
begin
  S := 'Delphi 3 /Server';
  Insert('Client', S, 9);                    { 'Delphi 3 Client/Server' }
end;
```

function IntToHex(Value: Integer; Digits: Integer): string;

Converte o inteiro Value num Hexadecimal (Base 16). Digits indica o número mínimo de dígitos Hexa a serem retornados

Ex.:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit2.Text := IntToHex(StrToInt(Edit1.Text), 6);
end;
```

function IntToStr(Value: Integer): string;

Transforma um Inteiro numa String

Ex.:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Value: Integer;

begin
  Value := 1583;
  Edit1.Text := IntToStr(Value);
end;
```

function IsValidIdent(const Ident: string): Boolean;

Indica se um identificador é válido para o Pascal

function Length(S: string): Integer;

Retorna o número de caracters usados na string S.

Ex.:

```
var
  S: string;
begin
  S := 'Curso de Programação em Delphi 3 - Object Pascal';
  Canvas.TextOut(10, 10, 'Tamanho da String = ' + IntToStr(Length(S)));
end;
```

function MaxIntValue(const Data: array of Integer): Integer;

Retorna o maior inteiro de um vetor

function MaxValue(const Data: array of Double): Double;

Retorna o maior valor de um vetor

function Mean(const Data: array of Double): Extended;

Retorna a média aritmética de um vetor

function MessageDlg(const Msg: string; AType: TMsgDlgType;
AButtons: TMsgDlgButtons; HelpCtx: Longint): Word;

Exibe uma Caixa de Mensagem e obtem uma resposta do usuário.
Onde

Msg : Mensagem a ser exibida.
AType : Tipo da caixa de mensagem

Valor	Significado
mtWarning	Uma caixa de mensagem contendo um ponto de exclamação amarelo.
mtError	Uma caixa de mensagem contendo um símbolo de pare vermelho.
mtInformation	Uma caixa de mensagem contendo um "i" azul.
mtConfirmation	Uma caixa de mensagem contendo um ponto de interrogação verde.
mtCustom	Uma caixa de mensagem sem bitmap.

Obs.: O caption da caixa de mensagem é o nome do arquivo executável da aplicação.

AButtons: Quais botões aparecerão na caixa de mensagem.
Onde:

Valor	Significado
mbYes	Um botão com o texto 'Yes'
mbNo	Um botão com o texto 'No'
mbOK	Um botão com o texto 'OK
mbCancel	Um botão com o texto 'Cancel'
mbHelp	Um botão com o texto 'Help'
mbAbort	Um botão com o texto 'Abort'
mbRetry	Um botão com o texto 'Retry'
mbIgnore	Um botão com o texto 'Ignore'

mbAll Um botão com o texto 'All'
mbYesNoCancel Coloca os botões Yes, No, e Cancel na caixa de mensagem
mbOkCancel Coloca os botões OK e Cancel na caixa de mensagem
mbAbortRetryIgnore Coloca os botões Abort, Retry, e Ignore na caixa de mensagem

MessageDlg retorna o valor do botão selecionado. Os valores de retorno possíveis são:

mrNone	mrAbort	mrYes
mrOk	mrRetry	mrNo
mrCancel	mrIgnore	mrAll

Ex.:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if MessageDlg('Deseja sair da aplicação agora ?',
    mtConfirmation, [mbYes, mbNo], 0) = mrYes then
  begin
    MessageDlg('Tenha um bom dia...ject Pascal application.', mtInformation,
      [mbOk], 0);
    Close;
  end;
end;
```

function MessageDlgPos(const Msg: string; AType: TMsgDlgType;
 AButtons: TMsgDlgButtons; HelpCtx: Longint; X, Y: Integer): Word;

Semelhante a MessageDlg exceto por permitir indicar a posição na qual a janela será exibida

Ex.:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  MessageDlgPos('Sair sem gravar alterações ?',mtConfirmation, mbYesNoCancel, 0, 200,
200);
end;
```

function MinIntValue(const Data: array of Integer): Integer;

Retorna o menor inteiro do vetor

function MinValue(const Data: array of Double): Double;

Retorna o menor valor de um vetor

procedure MkDir(S: string);

Cria um novo diretório

Ex.:
```
uses Dialogs;
begin
  {$I-}
  { Pega o nome do diretório de um controle Tedit }
  MkDir(Edit1.Text);
  if IOResult <> 0 then
    MessageDlg('Não posso criar o diretório.', mtWarning, [mbOk], 0)
  else
    MessageDlg('Diretório criado', mtInformation, [mbOk], 0);
end;
```

function Now: TDateTime;

Retorna a data e a hora corrente

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := 'Data e hora atual é  ' + Str(Now);
end;
```

function Ord(X): Longint;

Retorna a ordem de um valor ordinal

```
uses Dialogs;
type
   Colors = (RED,BLUE,GREEN);
var
  S: string;
 begin
   S := 'BLUE (Azul) tem um valor ordinal de ' + IntToStr(Ord(BLUE)) + #13#10;
   S := 'O código decimal ASCII para "p" é ' + IntToStr(Ord('p'));
   MessageDlg(S, mtInformation, [mbOk], 0);
 end;
```

function Pi: Extended;

Retorna o valor de PI
3.1415926535897932385.

function Pos(Substr: string; S: string): Integer;

Procura por uma sub-string numa string e retorna a posição da primeira ocorrência ou zero se não encontrou

Ex.:
```
var S: string;
begin
  S := '   123.5';
  { Converte espaços para zeros }
  while Pos(' ', S) > 0 do
    S[Pos(' ', S)] := '0';
end;
```

function Power(Base, Exponent: Extended): Extended;

Retorna a Potência.

function Pred(X);

Retorna o predecessor de um ordinal

```
uses Dialogs;
type
   Colors = (RED,BLUE,GREEN);
 var
   S: string;
 begin
   S := 'O predecessor de 5 é ' + IntToStr(Pred(5)) + #13#10;
   S := S + 'O sucessor de 10 é ' + IntToStr(Succ(10)) + #13#10;
   if Succ(RED) = BLUE then
     S := S + 'Nos tipos de Cores, RED (Vermelho) é o predecessor de BLUE (Azul).';
   MessageDlg(S, mtInformation, [mbOk], 0);
 end;
```

function Random [(Range: Integer)];

Retorna um valor Randômico
0 <= X < Range.

```
var
   I: Integer;
 begin
   Randomize;
   for I := 1 to 50 do begin
     { Escreve na janela em posiçoes randomicas }
     Canvas.TextOut(Random(Width), Random(Height), 'Olá!');
   end;
 end;
```

procedure Randomize;

Inicializa o modo Randomico. Deve ser utilizada antes de Random

function RemoveDir(const Dir: string): Boolean;

Remove um diretório retornando True caso tenha conseguido e False caso contrário

procedure Rename(var F; Newname);

Altera o nome de um arquivo.
F é uma variável de arquivo (file). Newname é uma expressão do tipo string ou do tipo Pchar.

Ex.:
```
uses Dialogs;
var
   f : file;
 begin
   OpenDialog1.Title := 'Escolha um arquivo... ';
   if OpenDialog1.Execute then begin
     SaveDialog1.Title := 'Renomear para...';
     if SaveDialog1.Execute then begin
       AssignFile(f, OpenDialog1.FileName);
       Canvas.TextOut(5, 10, 'Renomeando ' + OpenDialog1.FileName + ' para ' +
         SaveDialog1.FileName);
       Rename(f, SaveDialog1.FileName);
     end;
   end;
 end;
```

function RenameFile(const OldName, NewName: string): Boolean;

Renomeia arquivos e retorna o sucesso ou insucesso

Ex.:
```
{ O código a seguir renomeia um arquivo }
if not RenameFile('OLDNAME.TXT','NEWNAME.TXT') then
  ErrorMsg('Erro ao renomear arquivo!');
```

procedure RmDir(S: string);

Remove um diretório

Ex.:
```
uses Dialogs;
begin
   {$I-}
   { Pega o nome do diretório de um controle TEdit }
   RmDir(Edit1.Text);
   if IOResult <> 0 then
     MessageDlg('Não posso remover o diretório', mtWarning, [mbOk], 0)
   else
     MessageDlg('Diretório removido', mtInformation, [mbOk], 0);
 end;
```

function Round(X: Extended): Longint;

Arredonda um número real

function SelectDirectory(var Directory: string; Options: TSelectDirOpts; HelpCtx: Longint):Boolean;

Exibe um Caixa de Dialogo para seleção de Diretório. O Diretório passado para a função aparece como diretório corrente e o diretório escolhido é retonado no mesmo Diretório (Directory). O valor do diretório corrente não é alterado

Os valores abaixo podem ser adicionados ao parametro Options:

Valor	Significado
sdAllowCreate	Uma caixa de exição aparece para permitir ao usuário digitar o nome do diretório que não existe. Esta opção não cria o diretório, mas a aplicação pode acessar o parametro Directory para criar o diretório desejado.
sdPerformCreate	Usado somente quando Options contém sdAllowCreate. Se o usuário entrar com um diretório que não existe, SelectDirectory cria-o.
sdPrompt	Usado quando Options contém sdAllowCreate. Exibe uma caixa de mensagem que informa o usuário quando o diretório digitado não existe e pergunta se deseja criá-lo. Se o usuário selecionar OK, o diretório é criado se Options contém sdPerformCreate. Se Options não contém sdPerformCreate, o diretório não é criado: a aplicação precisa criá-o quando SelectDirectory retornar.

A função retorna True se o usuário selecionar o diretório e clicar em OK, e False se o usuário selecionar Cancel ou fechar a caixa de diálogo sem selecionar um diretório.

Ex.:

```
uses FileCtrl;

procedure TForm1.Button1Click(Sender: TObject);
var
  Dir: string;
begin
  Dir := 'C:\MYDIR';
  if SelectDirectory(Dir, [sdAllowCreate, sdPerformCreate, sdPrompt]) then
    Label1.Caption := Dir;
end;
```

function SetCurrentDir(const Dir: string): Boolean;

Torna Dir o diretório corrente e retorna o sucesso da operação

procedure SetLength(var S: string; NewLength: Integer);

Coloca um novo tamanho para uma string. O efeito é similar ao código abaixo: S[0] := NewLength.

procedure ShowMessage(const Msg: string);

Exibe uma mensagem ao usuário

Ex.:
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage('Olá !');
end;
```

function Sqr(X: Extended): Extended;

Retorna o quadrado de X

function Sqrt(X: Extended): Extended;

Retorna a raiz quadrada de X

function StrComp(Str1, Str2 : PChar): Integer;

Compara duas strings em case sensitivity (diferencia maiúsculas e minúsculas)

Valor de retorno	Condição
<0	if Str1< Str2
=0	if Str1= Str2
>0	if Str1 > Str2

function StringOfChar(Ch: Char; Count: Integer): string;

Retorna uma string contendo Count caracteres Ch

Ex.:
```
S := StringOfChar('A', 10);
{sets S to the string 'AAAAAAAAAA'}
```

function StrToDate(const S: string): TDateTime;

Converte uma string em data

Ex.:
```
procedure TForm1.Button1Click(Sender: TObject);
var
  ADate: TDateTime;
begin
  ADate := StrToDate(Edit1.Text);
  Label1.Caption := DateToStr(ADate);
end;
```

function StrToDateTime(const S: string): TDateTime;

Converte uma string para o formato DateTime

Ex.:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  AdateAndTime: TDateTime;
begin
  AdateAndTime := StrToDateTime(Edit1.Text);
  Label1.Caption := DateTimeToStr(ADateAndTime);
end;
```

function StrToFloat(const S: string): Extended;

Converte uma string num Float

function StrToInt(const S: string): Integer;

Converte uma string num inteiro

Ex.:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S: string;
  I: Integer;
begin
  S := '22467';
  I := StrToInt(S);
  Inc(I);
  Edit1.Text := IntToStr(I);
end;
```

function StrToTime(const S: string): TDateTime;

Converte uma string em hora

Ex.:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ATime: TDateTime;
begin
  ATime := StrToTime(Edit1.Text);
  Label1.Caption := TimeToStr(ATime);
end;
```

function Succ(X);

Retorna o sucessor de um ordinal

Ex.:
```
uses Dialogs;
type
   Colors = (RED,BLUE,GREEN);
 var
   S: string;
 begin
   S := 'O predecessor de 5 é ' + IntToStr(Pred(5)) + #13#10;
   S := S + 'O sucessor de 10 é ' + IntToStr(Succ(10)) + #13#10;
MessageDlg(S, mtInformation, [mbOk], 0);
 end;
```

function Sum(const Data: array of Double): Extended register;

Calcula a soma de dos elementos de um vetor

function SumInt(const Data: array of Integer): Integer register;

Calcula a soma dos elementos de um vetor de inteiros

function Time: TDateTime;

Retorna a hora corrente

Ex.:
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := 'A hora é  ' + TimeToStr(Time);
end;
```

function TimeToStr(Time: TDateTime): string;

Converte hora para string

Ex.:
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := TimeToStr(Time);
end;
```

function Trim(const S: string): string;

Retira os espaços em brancos a esquerda e a direita da string S

function TrimLeft(const S: string): string;

Retira os espaços em brancos a esquerda da string S

function TrimRight(const S: string): string;

Retira os espaços em brancos a direita da string S

function Trunc(X: Extended): Longint;

Retorna a parte inteira de um número

function UpCase(Ch: Char): Char;

Converte o caracter Ch em maiúscula

Ex.:

```
uses Dialogs;
var
   s : string;
   i : Integer;
 begin
   { Pega a string de um controle TEdit }
   s := Edit1.Text;
   for i := 1 to Length(s) do
     s[i] := UpCase(s[i]);
   MessageDlg('Tudo em maiúsculo: ' + s, mtInformation, [mbOk], 0);
 end;
```

function UpperCase(const S: string): string;

Converte a string S em maiúsculas

Ex.:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to ListBox1.Items.Count -1 do
    ListBox1.Items[I] := UpperCase(ListBox1.Items[I]);
end;
```

Componentes - Propriedades, Eventos e Métodos

Propriedades Comuns

Nome	Descrição
Align	Determina como o componente será alinhado no seu container [alNone, alTop, alBottom, alLeft, alRight, alClient]
Caption	Legenda do componente (& indica a tecla de atalho)
Cursor	Desenho que representa o cursor da mouse [crDefault, crNone, crArrow, crCross , crIBeam, crSize, crSizeNESW, crSizeNS, crSizeNWSE, crSizeWE, crUpArrow, crHourGlass, crDrag,

Nome	Descrição
	crNoDrop, crHSplit, crVSplit, crMultiDrag, crSQLWait, crNo, crAppStart, crHelp, crHandPoint]
Name	Nome da instância do componente
Left	Distância em Pixel da borda esquerda do componente até a borda esquerda do FORM
Top	Distância em pixel da borda superior do componente até a borda superior do FORM
Height	Altura em pixel do componente
Width	Largura em pixel do componente
ComponentCount	O número de componentes possuídos por um componente container
Components	Uma matriz de componentes possuídos por um componente container
Color	Indica a cor de fundo do componente
Font	Fonte utilizada no componente
Ctl3D	Define a aparência 3D ou 2D de um componente
Enabled	Define se o componente esta ou não ativo
Visible	Define se o componente esta ou não visível
Hint	String utilizada na exibição de dicas instantâneas
ShowHint	Define se o hint será mostrado automaticamente
PopupMenu	Menu que será acionado com o botão direito do mouse
TabOrder	A ordem de tabulação do componente
TabStop	Indica se o componente será ponto de parada para a tecla TAB
HelpContext	Número utilizado para chamar o help on-line sensível ao contexto

Eventos Comuns

Nome	Descrição
OnChage	O conteúdo do componente é alterado
OnClick	O componente é acionado (Mouse ou Enter)
OnDblClick	Duplo-clique sobre o componente
OnEnter	O componente recebe o foco
OnExit	O componente perde o foco
OnKeyDown	Tecla(s) são pressionada(s) inclusive teclas de controle Parametros : Key = Código da tecla Shift = conjunto que indica a(s) tecla(s) de controle pressionadas (ssShift, ssAlt, ssCtrl, ssLeft, ssRight, ssMiddle, ssDouble)
OnKeyPress	Uma tecla é pressionada, onde Key contém o caracter pressionado
OnKeyUp	Uma tecla é solta

Métodos Comuns

Nome	Descrição
Create	Cria uma nova instância

Nome	Descrição
Destroy	Destrói a instância
Show	Torna o componente visível
Hide	Torna o componente invisível
SetFocus	Coloca o foco no componente
Focused	Determina se o componente tem o foco
BringToFront	Coloca o componente na frente dos outros
SendToBack	Coloca o componente atrás dos outros
ScaleBy	Gradua o componente em determina escala. Ex: Button1.ScaleBy(90,100) altera o tamanho do botão para 90% do tamanho original
SetBounds	Muda a posição e o tamanho do componente (ALeft,ATop,AWidth,AHeigh)

Componentes

Form - Elemento básico no desenvolvimento Delphi formando o alicerce sobre, o qual um aplicativo é construído.

Propriedades	Descrição
Active	Indica quando o formulário esta ativo.
ActiveControl	Determina o controle que receberá o foco quando o formulário é ativado
AutoScroll	Adiciona barras de rolagens automaticamente quando um formulário é redimensionado de forma a cobrir componentes
HorzScrollBar	Adiciona Barra de rolagem Horizontais quando necessário
VertScrollBar	Adiciona Barra de rolagem Verticais quando necessário
BorderIcons	Define quais ícones de controle serão visíveis [biSystemMenu, biMinimize, biMaximize, biHelp]
BorderStyle	Estilo da borda da janela [bsDialog, bsSingle, bsNone, bsSizeable, bsToolWindow, bsSizeToolWin]
FormStyle	Tipo da janela [fsNormalfs, MDIChild, fsMDIForm, fsStayOnTop]
Icon	Ícone da janela
Menu	Indica qual o componente menu do formulário será apresentado
Position	Permite controlar a posição e tamanho dos formulários na execução [poDesigned, poDefault, poDefaultPosOnly, poDefaultSizeOnly, poScreenCenter]
WindowState	Estado da janela (normal, minimizado ou maximizado)

Eventos	Descrição
OnCreate	O formulário é criado
OnShow	Antes de mostrar a janela
OnCloseQuery	É chamada para validar se a janela pode ser fechada
OnClose	Ocorre quando a janela é fechada
OnActivate	Ocorre quando a janela torna-se ativa
OnDeactivate	Ocorre quando a janela perde o foco
OnResize	Ocorre quando a janela muda de tamanho

Métodos	Descrição
Show	Mostra uma janela não-modal
ShowModal	Ativa uma janela modal
Close	Fecha a janela
Refresh	Redesenha a Janela

Button - Componente utilizado para representar ações

Propriedades	Descrição
Cancel	Dispara o evento OnClick do botão quando a tecla ESC é pressionada
Default	Dispara o evento OnClick do botão quando a tecla ENTER é pressionada
ModalResult	Associa o botão a opção de fechamento de um Form modal

BitBtn - Botão contendo uma Legenda e um BitMap, possui as propriedades e métodos do SpeedButton

Propriedades	Descrição
Kind	Seleciona um BitMap padrão para o botão
Style	Indica a aparência do botão (win3.11, win95, winxx)

SpeedButton - Botão contendo um BitMap, normalmente utilizado na construção de barra de ferramentas

Propriedades	Descrição
Glyph	BitMap exibido pelo botão
LayOut	Posição do BitMap no Botão
Margin	Indica o espaço entre a borda do botão e o BitMap
Spacing	Indica o espaço entre o BitMap e o Texto do botão
Down	Estado do botão (Pressionado ou não)
GroupIndex	Indica quais botões pertencerão ao mesmo grupo
AllowAllUp	Permite que todos os botões de um grupo possam estar não pressionados

Métodos	Descrição
Click	Ativa o evento OnClick do botão

Label - Utilizado para exibir rótulos

Propriedades	Descrição
Alignment	Alinhamento do texto no componente
AutoSize	Define se o tamanho do componente será automaticamente ajustado ao tamanho da legenda
WordWrap	Retorno automático
Transparent	Define se o componente será transparente
FocusControl	Nome do componente que receberá o foco
ShowAccelChar	Indica se & será fará ou não parte da legenda

Edit - Utilizado para entrada de dados texto em uma única linha.

Propriedades	Descrição
Text	Armazena a entrada de dados
AutoSelect	Indica se o texto será ou não selecionado quando o componente receber o foco
MaxLength	Número máximo de caracteres permitidos
CharCase	Define se as letras aparecerão em maiúsculo, minúsculo ou normal
PasswordChar	Caracter utilizado para esconder os dados digitados (Senhas)
ReadOnly	Define se será permitido alterar o texto
SelLength SelStart SelText	Comprimento da seleção Inicio da seleção Texto selecionado

MaskEdit - Permite entrada de dados texto em uma linha, utilizando uma máscara de edição. Possui todas as propriedades do componente Edit

Propriedades	Descrição
EditMask	Máscara de edição

A propriedade EditMask consiste em uma máscara de edição permitindo definir quais os possíveis caracteres e a formatação para a propriedade Text.
Essa máscara consiste de três partes separadas por ';'. A primeira parte é a máscara propriamente dita, a segunda parte indica se os caracteres literais serão armazenados na propriedade text (0 - não armazena; 1 - armazena). A terceira parte indica qual o caracter utilizado representar os espaços a serem digitados no texto

Estes são os caracteres especiais que podem compor a máscara de edição

Caracter	Descrição
!	Espaços em branco não aparecerão
>	Todos os caracteres seguintes serão maiúsculos até que apareça o caracter <
<	Todos os caracteres seguintes serão minúsculos até que apareça o caracter >
\	Indica um caracter literal
l	Somente caracter alfabético
L	Obrigatoriamente um caracter alfabético (A-Z, a-z)
a	Somente caracter alfanumérico
A	Obrigatoriamente caractere alfanumérico (A-Z, a-z, 0-9)
9	Somente caracter numérico
0	Obrigatoriamente caracter numérico
c	permite um caracter
C	Obrigatoriamente um caracter
#	Permite um caracter numérico ou sinal de mais ou de menos, mas não os requer.

:	Separador de horas, minutos e segundos
/	Separador de dias, meses e anos

Memo - Permite entrada de dados texto em múltiplas linhas.

Propriedades	Descrição
Alignament	Indica como será o alinhamento do texto
Lines	Armazena as linhas de texto
WantReturns	Define se a tecla ENTER será tratada pelo Formulário ou pelo Memo
WantTab	Define se a tecla TAB será tratada pelo Formulário ou pelo Memo
WordWrap	Indica se a linha digitada será quebrada, automaticamente, de acordo com o tamanho do componente
ScrollBar	Indica se Memo terá barras de rolagem

Métodos	Descrição
Clear	Limpa o memo
ClearSelection	Limpa o texto selecionado no memo

Obs: Muitos componentes possuem propriedades do Tipo TStrings, essa classe permite armazenar e manipular uma lista de Strings. Toda propriedade do tipo TStrings permite acesso indexado aos itens da listas (Propriedade[indíce])

TString

Propriedades	Descrição
Count	Número de linhas

Métodos	Descrição
Add	Adiciona uma nova linha no final da lista
Insert	Insere uma nova linha numa posição especificada
Delete	Apaga uma linha
Clear	Apaga toda a lista
IndexOf	Retorna o índice do item e -1 caso não encontre
LoadFromFile	Carrega de um arquivo texto
SaveToFile	Salva para um arquivo texto

CheckBox - Utilizado para obter informações “lógicas” independentes

Propriedades	Descrição
AllowGrayed	Determina se o checkbox terá duas ou três possibilidades
Checked	Determina se o checkbox está selecionado
State	Estado atual do checkbox (cbUnchecked, cbChecked, cbGrayed)

RadioButton - Utilizado para obter informações lógicas mutuamente exclusivas

Propriedades	Descrição
Checked	Determina se o RadioButton esta selecionado

GroupBox - Utilizados para agrupar componentes, sobre determinado título.

RadioGroup - Componente que agrupa e controla RadioButtons

Propriedades	Descrição
Columns	Número de colunas de botões de rádio
Items	Valores dos botões de rádio. Items[n] acessa o n-ésimo componente
ItemIndex	Item selecionado. (-1 nenhum, começa em 0)

Panel - Utilizados para agrupar componentes e criar barras de ferramentas.

Propriedades	Descrição
BevelInner	Estilo interno da superfície do Panel
BevelOuter	Estilo externo da superfície do Panel
BevelWidth	Distância entre as superfícies externas e internas
BorderStyle	Estilo da Borda
BorderWidth	Largura da borda

ScrollBox - Semelhante ao componente GrupoBox, sendo que, possui barras de rolagem

Propriedades	Descrição
HorzScrollBar	Barra Horizontal (Increment, Tracking e Visible)
VertScrollBar	Barra Vertical (Increment, Tracking e Visible)

Bevel - Define linhas, retângulos e molduras nas janelas.

Propriedades	Descrição
Shape	Tipo de figura a ser desenhada
Style	Define alto e baixo relevo

ListBox - Utilizado para escolher em uma lista grande de opções.

Propriedades	Descrição
Columns	Número de colunas da lista
MultiSelect	Define se será permitida a seleção de múltiplos itens
ExtendedSelect	Define se a seleção poderá ser estendida pelo uso das teclas Shift e Ctrl
IntegralHeight	Define se os itens poderão aparecer parcialmente ou somente por completo
Items	Valores dos itens da lista
ItemIndex	Item selecionado. (-1 não existe item selecionado e o 1º é 0)
Selected	De acordo com o índice indica se um item em particular esta selecionado.
SelCount	Indica quantos itens estão selecionado
Sorted	Define se os itens aparecerão ordenados

ComboBox - Reúne as características de Edit e ListBox.

Propriedades	Descrição
Items	Valores a serem exibidos na lista
DropDownCount	Número de itens visíveis da lista
Text	Conteúdo texto digitado na ComboBox
Style	csDropDown - permite edição e exibe os itens mediante solicitação csDropDownList - não permite edição e mostra itens no edit ao pressionar a 1º letra do item. CsSimple - permite edição e exibe a lista
Sorted	Define se os itens aparecerão ordenados

CheckListBox – Possui toda a funcionalidade do ListBox exibindo uma CheckBox p/ cada item

Propriedades	Descrição
Checked[n]	Retorna true se o item n estiver selecionado
State[n]	Retorna o estado do item n : [cvUnchecked, cbChecked, cbGrayed]

Eventos	Descrição
OnClickChecked	Quando um item é marcado ou desmarcado

ScrollBar - Permite a escolha de um valor numérico dentro de uma faixa.

Propriedades	Descrição
Min	Valor mínimo possível
Max	Valor máximo possível
Position	Posição Atual

Propriedades	Descrição
LargeChange	Incremento da posição quando o click é na barra
SmallChange	Incremento da posição quando o click é na seta
Kind	Se a barra é vertical ou horizontal

Eventos	Descrição
OnEditMask	Permite atribuir uma máscara a célula. A linha e a coluna da célula editada são recebidas pelo evento e mascará é representada pelo parâmetro Value

StringGrid e DrawGrid - Utilizado para entrada ou exibição de dados no formato de tabela.

Propriedades	Descrição
ColCount	Número de colunas
RowCount	Número de linhas
Col	Coluna corrente
Row	Linha Corrente
DefaultColWidth	Largura das colunas
DefaultRowHeight	Altura das linhas
DefaultDrawing	Define se o desenho das células será automático ou manual
FixedCol	Quantidade de Linhas Fixas
FixedRow	Quantidade de colunas fixas
GridLineWidth	Espessura das Linhas divisórias
Options	Define características do Grid. goEditing e goTabs
Cells[col,lin]	Permite acessar as células da Grid, por coluna e linha, sendo que a primeira linha e coluna têm valores 0.

Image - Exibe uma imagem.

Propriedades	Descrição
Picture	Arquivo Bitmap, Ícone ou Windows Metafile exibido
Center	Centraliza a figura no componente
Stretch	Define se o tamanho da figura deve ser ajustada ao do componente
Transparent	Torna o fundo visível ou opaco

Shape - Exibe uma figura geométrica

Propriedades	Descrição
Brush	Cor e preenchimeto da figura
Pen	Cor e preenchimento da borda
Shape	Figura geométrica

Splitter – Permite o redimensionamento de componente em tempo de execução

Propriedades	Descrição
Beveled	Exibe o componente com aparência 3D
MinSize	Tamanho mínimo para os componentes

TabControl - Contem guias, as quais podem ser utilizadas para alterar outros componentes

Propriedades	Descrição
MultiLine	Permite múltiplas linhas para as guias
Tabs	Guias
TabPositon	Posição das “orelhas” em cima ou embaixo
HotTrack	Destaca a “orelha” quando o curso do mouse esta em cima da mesma
TabIndex	Guia ativa
ScrollOpposite	Transfere as “orelhas” das linhas anteriores à selecionada para a outra extremidade do Componente

Eventos	Descrição
OnChange	Quando uma guia é selecionada

PageControl - Contem páginas, as quais, podem possuir diversos componentes. Para inserir uma nova página dê um clique com o botão direito do mouse e escolha NewPage. Cada página criada é um objeto do tipo TTabSheet e possui a seguintes propriedades : Caption, PageIndex e TabVisible, onde PageIndex indica a ordem de apresentação de cada página e TabVisible permite exibir ou ocultar as páginas.

Propriedades	Descrição
MultiLine	Permite múltiplas linhas para as páginas
ActivePage	Página ativa
PageIndex	Índice da página ativa

Eventos	Descrição
OnChange	Quando uma página é selecionada

ImageList – Representa uma coleção de imagens do mesmo tramanho, sua principal função é armazenar imagens para outros componentes, onde cada imagem pode ser refrenciada através de seu índice no vetor de imagens.

Propriedades	Descrição
Count	Número de imagens na lista

RichEdit – Componente para editar texto no formato Rich Text, esse componente permite diferentes formatações para o seu texto, diferenciando-o do componente Memo. Possui todas a propriedades do memo.

Propriedades	Descrição
HideScrollBar	Somente exibe as barras de rolagem quando necessário
HideSelection	Indica se o texto continuara com a indicação de selecionado mesmo que o componente perca o foco
SelAttributes	Formatação do Texto selecionado. Color - Cor dos caracteres Name - Nome da fonte usada Size - Tamanho da Fonte Style - Estilo da Fonte [fsBold,fsItalic, fsUnderline,fsStrikeout]

TrackBar - Componente utilizado para seleção de valores inteiros dentro de uma faixa

Propriedades	Descrição
Orientation	Orientação vertical ou horizontal
Min	valor mínimo
Max	valor máximo
Position	Posição corrente
TickStyle	Estilo de exibição das marcações
TickMarks	Aparência do indicador
PageSize	Determina o incremento que deve ser dado quando as teclas PgUp e PgDn forem pressionadas
LineSize	Determina o incremento que deve ser dado quando as setas forem pressionadas
SelStart	Posição de inicio do preenchimento
SelEnd	Posição de término do preenchimento

ProgressBar - Componente para indicar visualmente o andamento da execução de tarefas

Propriedades	Descrição
Min	valor mínimo
Max	valor máximo
Step	incremento que deve ser dado a propriedade position em cada mudança
Position	Posição corrente

Métodos	Descrição
StepIt	Incrementa Position de Step unidades
StepBy	Incrementa Position de n unidades

UpDown - Utilizado normalmente associado a outro componente para realizar incremento em dados numéricos

Propriedades	Descrição
Associate	Indica o componente associado
AlignButton	Indica o alinhamento do UpDown em relação ao componente associado (udLeft, udRight)
Min	Valor mínimo
Max	Valor máximo
Orientation	Orientação do componente (UdHorizontal, udVertical)
Wrap	Salto do valor mínimo para o máximo e vice-versa
Increment	Incremento dado ao componente associado
ArrowKeys	Indica que o componente recebe os incrementos das teclas de SETAS
Positon	Valor corrente
Thousands	Indica se irá aparecer o separdor de milhar
Wrap	Salto altomático de Max para Min e vice-versa

HotKey - Obtém em tempo de execução teclas de atalho e podem ser utilizadas para definir as teclas de atalho para outros componentes quem tenham a propriedade ShortCut

Propriedades	Descrição
HotKey	Combinação de teclas para a HotKey
InvalidKeys	Especifica as teclas inválidas para modificadores [hcNone , hcShift, hcCtrl, hcAlt, hcShiftCtrl, hcShiftAlt, hcCtrlAlt, hcShiftCtrlAlt]
Modifiers	Teclas modifocadoras [hkShift, hkCtrl, hkAlt, hkExt]

Animate – Componente capaz de exibir um AVI, o qual representa um formato de arquivo multimídia com imagens e sons, mas este componente apenas exibe as imagens.

Propriedades	Descrição
Active	Indica se a animação esta sendo exibida ou não
AutoSize	Ajusta automaticamente o tamanho do componente ao tamanho da imagem
Center	Centraliza a animação
FileName	Nome do arquivo AVi

Propriedades	Descrição
FrameCount	Número de Frames da animação
Repetitions	Número de repetições quando a animação for executada. O valor zero indica repetições indefinidas

DateTimePicket – Componente que permite a seleção visual de uma datas ou horas

Propriedades	Descrição
Time	Hora selecionada
Date	Data Selecionada
DateMode	A forma como a data poderá ser selecionada [dmComboBox, dmUpDown]
DateFormat	Formato da Data [dfShort, dfLong]
Kind	Seleciona o componente para Data ou Hora [dtkDate, dtkTime]
ShowCheckbox	Exibe um CheckBox
Check	Indica se o CheckBox esta selecionado

TreeView - Permite exibição de dados em forma hierárquica. Este componente possui métodos e eventos que permitem controle e modificação da sua estrutura em tempo de execução.

Propriedades	Descrição
Items	Define os itens da hierarquia.
Ident	Recuo dos sus-itens
ShowLines	Determina se haverá uma linha ligando os sub-itens
ShowRoot	Determina se haverá uma linha ligando os itens raízes
ShowButtons	Indica se o botão a esquerda do item será visível
HideSelect	Indica se quando o componente perder o foco a seleção continuará ativa
SortedType	nsNone não é ordenado nsData os iten são ordenados quando os dados são alterados nsText os itens são ordenados quando o Caption é alterado. nsBoth a ordenação e feita em ambos os casos
Selected	Item selecionado. Podemos acessar o conteúdo selecionado através de Select.Text;

ListView - Componente que permite exibir de várias maneiras uma lista de itens.

Propriedades	Descrição
ViewStyle	Determina se os itens devem ser apresentados em colunas com cabeçalhos e sub-itens, verticalmente ou horizontalmente, com ícones grandes ou pequenos
LargeImages	Lista de Imagens (TImagesList) contendo a bitmap's a serem exibidos e somente é usada quando a propriedade ViewStyle é vsIcon
SmallImages	Lista de Imagens (TImagesList) contendo a bitmap's a serem exibidos e somente é usada quando a propriedade ViewStyle é vsSmallIcon
Items	Items a serem exibidos
Columns	Cabeçalhos das colunas da Lista
ShowColumnHeaders	Exibe os cabeçalhos das colunas
ColumnClick	Indica se os cabeçalhos das colunas terão a aparência de botões
IconOptions	Opções de exibição dos ícones quando ViewStyle for vsIcon ou vsSmallIcons Arrangement alinhado no topo ou esquerda do ListView AutoArrange os ícones são alinhados automaticamente WrapText a propriedade caption será quebrada
SortedType	nsNone não é ordenado nsData os iten são ordenados dados são alterados nsText os itens são ordenados quando o Caption é alterada. nsBoth a ordenação e feita em ambos os casos
Selected	Item selecionado. Podemos acessar o conteúdo selecionado através de Select.Captiont;

HeaderControl - Utilizado para exibir barra de títulos ajustáveis e cabeçalhos. Podemos abrir o editor de seções através do botão direito do mouse.

Propriedades	Descrição
Sections[n]	Cabeçalhos do componente, possuindo as seguintes propriedades : Text, With, Min, Max, Style, Alignment, AllowClick e Index. Onde uma seção em particular pode ser acessada através do seu índice
HotTrack	Diferencia o cabeçlho quando o mouse esta sobre este

Eventos	Descrição
OnSectionClick	Clique sobre uma seção
OnSectionResize	A seção tem seu tamanho alterado

StatusBar - Utilizado para criar barra de Status para exibir informações sobre a aplicação. Podemos abrir o editor de paines através do botão direito do mose

Propriedades	Descrição
SimplePanel	Indica se o StatusBar possuirá um ou vários panels
SimpleText	Texto exibido caso SimplePanel igual a TRUE
SizeGrip	Indicador de tamanho padrão do Windows
Panels	Painéis do StatusBar, cada panel permite a exibição de informação. Os painéis são acessados como elementos de um vetor (Panels[n]) e possuem as seguintes propriedades : Text, Width, Style, Bevel, Alignment

ToolBar – Componete utilizado para criar barras de ferramentas. Para adicionar botões e sepradores usamos o botão direito do mouse. Este é um componente container e podemos colocar outros componentes no seu interior, os quais serão automaticamente alinhados. Cada botão adicionado possui as seguinte propriedades : ImageIndex, Style[tbsButton, tbsCheck, tbsDivide, tbsDropDown, tbsSeparator] e Indeterminate

Propriedades	Descrição
Flat	Os botões ficam terão a aparência do Office 97
Images	Componente ImageList que conterá as figuras dos botões
Buttons[n]	Acessa os botões através do índice n

CoolBar – Componete utilizado para criar barras de ferramentas que podem ser arrastadas e reposicionadas. Podemos editar as faixas através do botão direito do mouse.

Propriedades	Descrição
Bands	Armazenas as Faixas (Cool bands)
BandBorderStyle	Indica o estilo da borda entre os CollBands
BitMap	BitMap exibindo no fundo do componente
FixedOrder	Indica se o usuário poderá modificar a ordem dos CollBands
FixedSize	Indica se o usuário poderá modificar o tamanho dos CollBands
ShowText	Indica se o valor da propriedade Text de cada TollBand será exibido ao lado de sua faixa
Vertical	Indica se os CollBand serão verticais
Images	Coleção de desenhos exibidos ao lado de cada CoollBand

Diálogos Comuns

Grupo de caixas de diálogo comuns a muitos programas.
Parte integrante do Windows, são atualizadas com a atualização do Windows.
Necessitam pouca ou nenhuma programação para serem utilizadas. Facilitam a padronização em tarefas comuns. Depois de serem executados os Diálogos Comuns armazenam em suas propriedades as escolhas do usuário.

Método	Descrição
Execute	Ativa a caixa de diálogo e retorna True caso o dialogo comum seja encerrado com o botão OK.

OpenDialog, SaveDialog, OpenPictureDialog e SavePictureDialog - Caixas de diálogo para abrir e salvar arquivo

Propriedades	Descrição
FileName	Nome do arquivo
DefaultExt	Extensão padrão para os arquivos
Filter	Define os tipos de arquivos que podem ser abertos ou salvos
FilterIndex	Número do filtro default
InitialDir	Diretório inicial
Title	Título da janela
Options	Define características da janela de abrir ou salvar

FontDialog - Caixa de diálogo de escolha de fonte.

Propriedades	Descrição
Device	Define se deve utilizar fontes para tela, impressora ou ambos
MinFontSize	Tamanho mínimo da fonte
MaxFontSize	Tamanho máximo da fonte
Options	Define características das fontes

Evento	Descrição
OnApply	Ocorre após o usuário pressionar o botão apply e antes da janela fechar

ColorDialog - Caixa de diálogo para escolha de cor.

Propriedades	Descrição
Options	Define características da caixa de diálogo

PrintDialog - Caixa de diálogo de impressão.

Propriedades	Descrição
Copies	Número de cópias inicial
PrintToFile	Define se a impressão será direcionada para arquivo
PrintRange	Faixa de páginas a ser impressa (todas ou somente o

Propriedades	Descrição
	selecionado)
FromPage	Página inicial
ToPage	Página final
MinPage	Menor número de página que o usuário pode escolher
MaxPage	Maior número de página que o usuário pode escolher
Options	Define características da caixa de diálogo

PrintSetupDialog - Dá acesso à caixa de diálogo de configuração de impressora.

FindDialog e ReplaceDialog - Caixas de diálogo de pesquisa e substituição de texto.

Propriedades	Descrição
FindText	Texto a pesquisar
ReplaceText	Texto a substituir (somente em ReplaceDialog)
Options	Define características da caixa de diálogo

Evento	Descrição
OnFind	Ocorre após o usuário pressionar o botão Find Next
OnReplace	Ocorre quando o usuário pressiona o botão Replace (somente em ReplaceDialog)

Método	Descrição
CloseDialog	Fecha a caixa de diálogo

MainMenu e PopupMenu - Usado para criação de barras de menus, onde o MainManu representa o Menu suspenso presente nos aplicativos e o PopupMenu represnta o menu instatâneo (botão direito do mouse) associado aos componentes através da propriedade PopupMenu.

Propriedades	Descrição
Items	Itens do menu, utiliza o MenuEditor

Eventos	Descrição
OnPopup	Ocorre quando o menu popup é ativado

MenuEditor - Utilizado para definição dos itens e sub-itens do menu
Operações : Inserir, Deletar e Criar SubItems

MenuItem - Item do menu.

Propriedades	Descrição
Caption	Texto do item
Checked	Se o item esta marcado ou não
Visible	Se o item esta visível ou não
Enabled	Se o item esta ativado ou desativado
ShortCut	Tecla de atalho
Break	Indica quebra de coluna
GroupIndex	Indica que o item do menu pertence a um grupo
RadioItem	Indica que o item de menu funcionára como um Radio, ou seja, dentro do mesmo grupo apenas um item estrá selecionado.

Eventos	Descrição
OnClick	Quando o item de menu é selecionado, usado para executar a função do item

Respostas dos Exercícios

Capitulo 1

1) O que é Delphi? É uma ferramenta para o desenvolvimento rápido de aplicativos (RAD), fortemente baseada na criação visual através de componentes e na utilização da Programação Orientada a Objetos através da Linguagem Object Pascal.
2) O que é IDE? Quais os itens que a compõe? A IDE é o Ambiente Integrado de Desenvolvimento. É formado pela Janela Principal (menu do Delphi, SpeedBar e Component Palette), o Object Inspector e o TreeView, o Code Editor (Editor de Código) e um Debugger Integrado.
3) Para que serve o Object Inspector? Ele é responsável por permitir a inspeção e modificação das propriedades e eventos associados a cada objeto (componentes).
4) Quais são os principais arquivos de um projeto? O arquivo com extensão .DPR, que gerencia todo o projeto e é responsável pela criação do aplicativo executável, arquivos .DFM dos formulários (que sempre possuem um arquivo .PAS) e os arquivos .PAS com código fonte em Object Pascal.
5) O que é um evento? São rotinas ou procedimentos que são executados como resposta a uma determinada ação ou acontecimento, com o pressionamento de teclas, um clique ou movimento de mouse, etc.

Capitulo 2

1) O que são Classe e Objeto? Objeto é uma estrutura que armazena tanto dados (informações) quanto funções (rotinas) para processa-los e/ou manipula-los. A Classe é uma definição genérica, uma espécie de “molde” para a criação de um objeto.
2) O que são método e propriedade? Método são funções que manipulas os dados de um objeto, propriedade são as características que um objeto possui.
3) O que é e para que serve o BDE? O BDE é uma engine (“ferramenta”) que fornece acesso a bancos de dados Desktop e Client-Server.
4) Qual a vantagem de se utilizar Alias? Permitir a alteração da base de dados, tanto para um novo local como para uma nova arquitetura ou banco de dados sem ter que modificar o aplicativo (executável).

5) Qual a função do Database Desktop? É a ferramenta utilizada para a criação/manutenção de tabelas de dados.

Capitulo 3

1) Qual a função do Menu Editor? O Menu Editor permite a criação visual de menus pulldown (mainmenu), o ajuste das propriedades dos itens de menu e também o acesso aos eventos dos mesmos.
2) Qual a diferença entre aplicações SDI e MDI? Qual tipo podemos criar com o Delphi? Em uma aplicação SDI as janelas são independentes, numa aplicação MDI temos uma janela principal (pai) que gerencia as outras janelas (filhas). No Delphi podemos criar ambas, no entanto, por padrão são criadas aplicações SDI.
3) Para que servem os componentes da Guia BDE? Eles permitem acessar bancos de dados através da engine BDE.
4) Qual a função dos componentes da Guia Data Controls? Eles permitem construir janelas para visualização e manutenção conectadas diretamente a componentes de acesso a dados, executando as operações básicas de leitura/gravação sem que seja necessário programação extra.
5) Qual a função do componente DataSource? O DataSource liga os componentes Data Controls aos componentes de acesso a dados, das guias BDE, ADO, dbExpress e Interbase.

Capitulo 4

1) Qual é a função propriedade EditMask? Facilitar a digitação de dados através de uma máscara de entrada de dados.
2) Qual é o evento que devemos utilizar para inicializar valores num campo? O evento OnNewRecord.
3) Para que serve a propriedade CharCase? Para especificar se a digitação será normal (da forma como for digitada) ou deverá ser fixada em maiúsculo ou minúsculo.
4) Qual a utilizade do método Locate? Localizar informações nas tabelas de dados.
5) Para que serve a função InputQuery? Cria uma caixa de diálogo para a entrada de informações.

Capitulo 5

1) Para que serve a propriedade Glyph presente em alguns tipos de botão? Ela permite que o botão apresente uma imagem além do texto.
2) Como podemos ocultar os botões do DBNavigator? Desativando as sub-propriedades da propriedade VisibleButtons.
3) Qual é a melhor maneira para se inserir um determinado componente em um Form várias vezes? Pressionando-se a tecla Shift antes de seleciona-lo. Depois clica-se quantas vezes se deseja que o componente seja inserido e libera-se a duplicação clicando no ícone de seta na paleta de componentes.
4) Qual a finalizade de campos LookUp? Exibir informações vindas de outra tabela relacionadas através de um campo da tabela atual.
5) Qual a finalizade de campos Calculated? Criar campos calculados (com fórmulas) que são atualizados automaticamente, quando algum dos seus itens é modificado.

Capítulo 6

1) Quais são os principais métodos de pesquisa do componente Table? São os métodos FindKey / FindNearest e Locate.
2) Como podemos alterar dinamicamente o tamanho de um formulário? Modificando suas propriedades Height (altura) e Width (largura).
3) Qual é a funcionalidade do componente PageControl? Ele permite a criação de páginas que permitem separa itens na interface, como acontece nas caixas de diálogo do Windows de propriedade de vídeo, etc.
4) Para que serve as estruturas de controle de exceção try..finally..end? Serve para criar uma proteção a execução de uma rotina, garantindo que os comandos escritos após o finally serão executados mesmo que ocorra algum erro nos comandos entre o try e o finally.
5) Qual o método utilizado para sincronizar a posição de registro de dois componentes Table configurados para acessar a mesma tabela? É o método GotoCurrent.

Capítulo 7

1) Para que serve o Componente ImageList? É um contêiner (repositório) para o armazenamento de imagens que serão utilizadas por menus, barras de ferramenta, etc.

2) Qual é o Componente responsável pela criação de relatórios? É o QuickReport da guia QReport.
3) Para que serve o componente QRExpr? Serve para a efetuar cálculos, somatória e contagem em relatórios.
4) Como são construídos os relatórios? Através de band's (QRBand's) responsáveis por separar os dados que devem ser exibidos (detail) de outras partes como cabeçalho, rodapé e sumário (totalizações).
5) Para que serve a função WinExec? Executa um programa externo.