Delphi 4

Borland Delph

Delphi 4

Adelize Generini de Oliveira
adelize@relativa.com.br

RELATIVA

Revisão:
Juano Nunez del Prado

Editoração Eletrônica:
Adelize Generini de Oliveira

Capa:
André S. de Sá

Relativa Editora Ltda.

R. Des. Tavares Sobrinho 46
Bom Abrigo - Florianópolis - SC
Tel/Fax: (048) 249-8313
E-Mail: info@relativa.com.br

Índice

1

Apresentando o Delphi 4

Apresentando o Delphi 4

O Delphi, lançado em 94 e atualmente em sua quarta versão, é um dos ambientes visuais de desenvolvimento preferidos por programadores. Baseado na linguagem Object Pascal, uma versão do Pascal orientada a objetos, o Delphi oferece inúmeras ferramentas para tornar o desenvolvimento para Windows, fácil, rápido e seguro.

Entre as características do Delphi podemos citar:

- uma interface gráfica para desenvolvimento, conhecida como IDE. Nesta inteface, através de cliques e arrastos de mouse, você pode criar formulários sofisticados rapidamente, para os mais diversos tipos de aplicações;

- um conjunto de componentes para realizar diversas operações, como entrada de dados, conexão com banco de dados, geração de relatórios, etc. Estes componentes estão agrupados na VCL, e podem originar novos componentes, adaptados as suas necessidades;

- um editor de código que oferece mensagens de erro, com cores diferentes para os comandos padrões, e integrado a um depurador profissional;

- possibilidade de desenvolvimento de aplicações para 16 bits (Windows 3.1) e 32-bits (Windows 95/ NT) ;

- um compilador que gera um programa executável com código nativo, otimizado;

- facilidade e rapidez na criação de aplicativos que manipulam bancos de dados de diversos formatos, como Paradox, Fox Pro, Access, SQL Server ou Oracle.

Características Avançadas do Delphi

A maioria das aplicações construídas em Delphi utilizam apenas uma pequena parte de sua potencialidade. Normalmente são aplicativos mono ou multiusuários, que acessam bancos de dados relacionais. No entanto, o Delphi permite que se construa aplicativos extremamente sofisticados.

Com a crescente diversificação de padrões, seja de protocolos de comunicação, de acesso a banco de dados, a servidores WEB, metodologias de construção de objetos, a tarefa de desenvolver sistemas poderia se tornar extremamente lenta e cansativa. O programador dispenderia a maior parte de seu tempo estudando as novas tecnologias, e como utilizá-las, ao invés de se dedicar a construir aplicações que tornassem sua empresa mais competitiva.

Para evitar isto, o Delphi permite que você utilize todas as novas tecnologias através de assistentes, que o auxiliam na montagem de aplicativos rápida e fácilmente. Por exemplo, construir controles ActiveX, ou componentes COM ou CORBA, ou aplicativos para Internet, são operações realizadas em um só passo com o Delphi, através de Wizards. Desenvolvendo em Delphi, o programador passa a Ter mais tempo para a construção do que realmente importa: a lógica das operações empresariais.

Com o Delphi você pode facilmente:

- acessar as funções da API do Windows;
- criar DLL's;
- criar controles e formulários ActiveX;
- criar aplicativos que trabalhem em threads diferentes;

- criar servidores e clientes OLE;
- criar componentes baseados no modelo COM, DCOM ou CORBA;
- criar aplicações para Internet, que suportem tanto o protocolo ISAPI como NSAPI;
- realizar análises multidimensionais em data warehouses;
- acessar qualquer banco de dados, através de dados nativos ou ODBC;
- construir aplicações multicamadas.

Novidades da Versão 4

Anualmente, a Borland (atualmente Inprise) lança uma nova versão do Delphi. Apesar de parecer exagerado, o aparecimento desenfreado de novas tecnologias contribui para forçar uma atualização no programa. Por exemplo, entre o lançamento da versão 3 e a versão 4, surgiram novos padrões de protocolos, metodologias de desenvolvimento, e até novos sistemas operacionais, como o Windows 98.

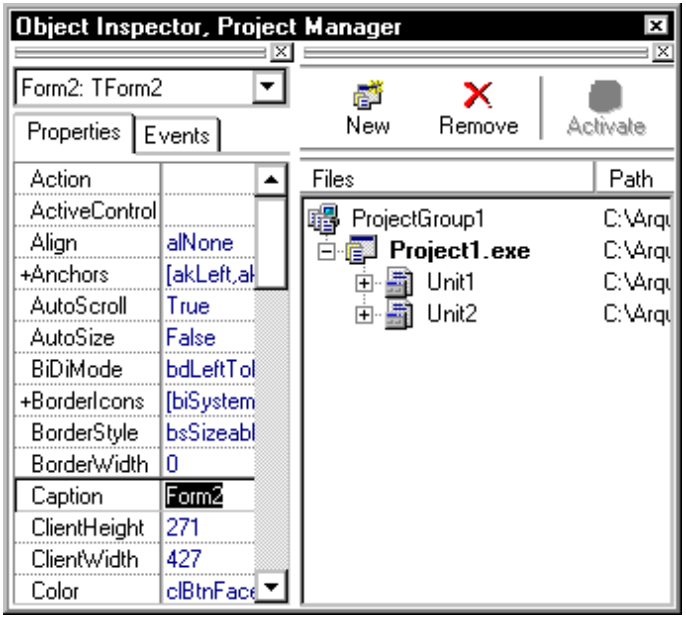
Segundo informativo da Inprise, a empresa não quer apostar em nenhum padrão, como COM ou CORBA, ISAPI ou NSAPI, ActiveX ou Java. Por isso, oferece no Delphi suporte a todas estas tecnologias, o que facilita bastante a vida do programador, que não precisa se definir por nenhum padrão. Por exemplo, um componente criado usando a metodologia COM também pode simultaneamente suportar CORBA.

A versão 4 do Delphi trás novidades em diversas áreas: no ambiente de desenvolvimento, na coleção de componentes, na linguagem Object Pascal, nas ferramentas de depuração, no acesso a banco de dados, no desenvolvimento para

Internet, na criação de componentes e no desenvolvimento de aplicações multicamadas.

Novidades no Ambiente de Desenvolvimento

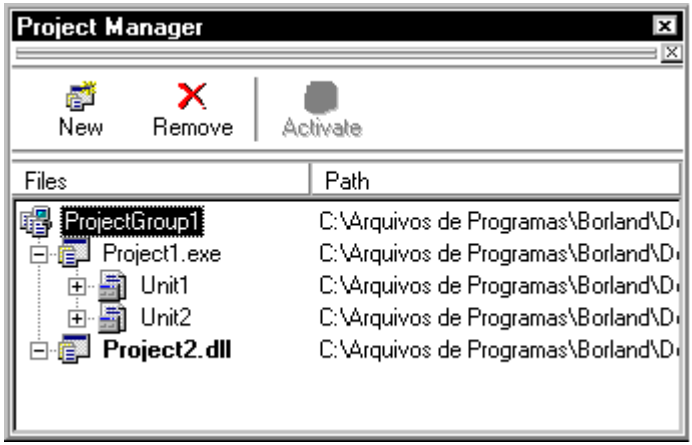
Uma das maiores novidades no ambiente de desenvolvimento é o uso de janelas atracáveis (dockable windows). Com este recurso, você pode atracar, por exemplo, o Project Manager com o Object Inspector:



Eles passarão a ocupar a mesma área de tela, maximizando a visualização do formulário ou código. Além de ocuparem um espaço menor, as janelas atracadas passam a funcionar como uma só janela: ao mover uma delas, a outra (ou outras) automaticamente move-se junto.

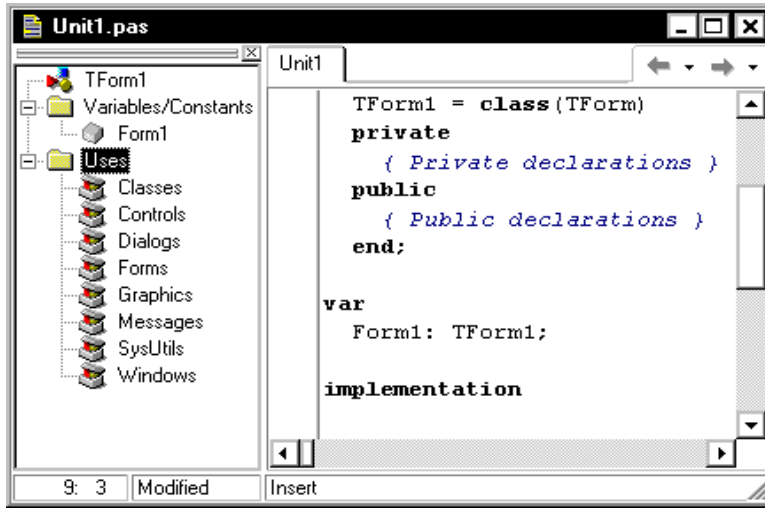
Outra novidade no ambiente de desenvolvimento é o novo gerenciador de projetos (Project Manager). Agora, você poderá combinar projetos para trabalharem em um

único grupo. Dessa forma, você poderá testar uma DLL, por exemplo, junto com o programa executável que a utiliza, estando ambas em um mesmo grupo de projetos:



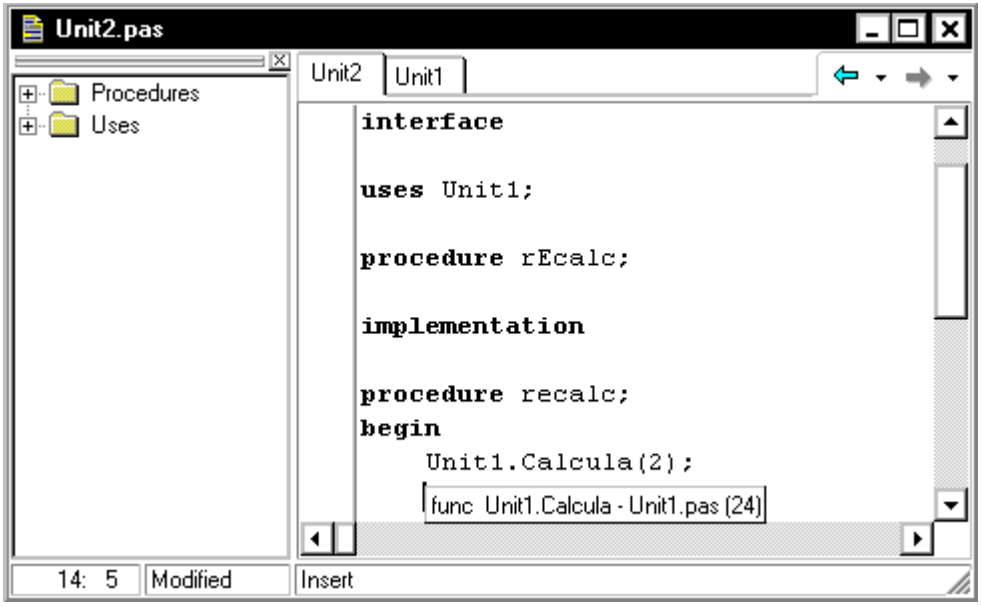
Além disso, todos os projetos do grupo poderão ser compilados simultaneamente, mantendo-os sempre sincronizados.

Foi no Editor de Código, no entanto, que houve mais novidades. A primeira é o Code Explorer, um utilitário atracado à esquerda da Janela de Código que mostra, em forma de árvore, os tipos de dados, propriedades, métodos, variáveis e rotinas globais definidas na unit ativa, além das outras units listadas na cláusula uses:



Além do Code Explorer, o Delphi 4 oferece o AppBrowser, um utilitário que permite que você navegue por todos os elementos do código, através de hiperlinks automáticos. Por exemplo, se você criou em uma unit uma função de nome Calcula, e quer utilizá-la em outra unit, e precisa verificar sua declaração, basta apertar a tecla Ctrl e dara um clique sobre o nome da função. Automaticamente a unit onde ela foi implementada é aberta, com o cursor do mouse sobre o cabeçalho desta.

Se você quiser somente verificar o tipo do identificador e a unit onde foi declarado, use o Tooltip Symbol Insight. Basta passar o mouse sobre o identificador no editor de código. A informação aparece em uma pequena janela:



Outra inovação no editor de código é o ClassCompletion Wizard, um assistente de criação de classes que automatiza a definição de novas classes ao gerar um esqueleto de código para os membros da classe que você declarar.

Novidades na Biblioteca de Componentes

O Delphi 4 oferece, dependendo da versão, até 175 componentes para as mais diversas atividades, desde acesso a banco de dados até análises multidimensionais.

Entre os novos componentes inseridos, podemos citar o Action Lists. Com este componente, múltiplos elementos de interface com o usuário, como botões de barras de ferramentas, menus, etc, podem ser associados a uma ação, permitindo que o programador separe atividades na interface com o usuário da lógica da aplicação, aumentando a reusabilidade e robustez da aplicação.

O componente Control Bar facilita a construção de interfaces semelhantes ao Office 97. O Delphi oferece também componentes para suportar funções do Windows 98, como MonthCalendar, PageScroller, e Flat Scrollbars.

Diversos componentes são oferecidos também para a criação de serviços para o Windows NT. Junto com assistentes e um gerador visual de serviços, criar aplicativos que rodem como serviços no Windows NT ficou bastante simples com esta nova versão do Delphi.

Novidades no Object Pascal

O Delphi 4 introduz novas funções na linguagem Object Pascal:

- arrays dinâmicos oferecem ao desenvolvedor a flexibilidade de utilizar arrays sem precisar especificar o número de elementos, apenas o tipo de dado e a dimensão;

- sobrecarregamento de métodos (method overloading) possibilitam que objetos possuam métodos com o mesmo nome, distinguidos apenas pela diferença entre parâmetros;

- inteiros de 64 bits permitem aplicações mais acuradas, sem erros de arredondamento, sendo ideias para aplicações financeiras, científicas ou de engenharia;

- a variável global TwoDigitYearCenturyWindow é usada pelas funções StrToDate e StrToDateTime para controlar a interpretação de anos com dois dígitos, quando datas forem convertidas.

Novidades nas Ferramentas de Depuração

A versão 4 do Delphi introduz novas funções de depuração, que podem ajudar o programador a encontrar erros no

código mais facilmente. Entre elas temos:

Module View: painel na forma de árvore que mostra informações detalhadas sobre os diferentes módulos de DLL's e EXE's que estão carregados, e o processo corrente que está sendo depurado.

CPU View: uma visão do que está ocorrendo por trás do código fonte, a nível de CPU.

Inspector: o novo inspetor é uma visão inteligente dos elementos de uma aplicação, incluindo arrays, classes, constantes, e ponteiros.

Depuração de Processos múltiplos: com o visualizador de threads, é fácil examinar todas as threads que estão sendo executadas no momento, dando ao programador o poder de definir e mudar o processo corrente.

Depuração de Processos Remotos: agora o programador pode depurar aplicações que estão sendo executadas em qualquer lugar do mundo, desde que haja uma conexão TCP/IP e direito de acesso.

Event logging: a nova versão do Delphi oferece um armazenador de eventos totalmente configurável, permitindo que o programador acompanhe todos os eventos de alto e baixo nível da aplicação.

DataWatch Breakpoints: os programadores podem agora pedir ao Delphi para interromper a execução da aplicação de acordo com alguma mudança de um dado.

Novidades no Acesso a Banco de Dados

Entre as novas características do Delphi relacionada a banco de dados podemos citar:

- suporte completo ao Oracle 8, última versão do

Oracle. Este suporte inclui tabelas aninhadas, tipos de dados abstratos, entre outros;

◆ suporte ao banco de dados Access 97;

◆ possibilidade do uso de agregados (somas, totais) armazenados junto com as tabelas;

◆ agora você pode criar, renomear ou excluir tabelas no próprio ambiente de projeto, através do Object Inspector;

◆ o Visual Query Builder substitui o SQL Builder.

Novidades no Desenvolvimento para Internet

O Delphi 4 oferece aplicações velozes para WEB, através da tecnologia WebBroker. O WebBroker constrói aplicações HTML robustas e distribuíveis, que suportam um grande número de clientes e de dados. Para maior resposta na performance, Delphi acessa diretamente servidores ISAPI e NSAPI.

Você pode centralizar suas informações publicando com a ferramenta WebModules, e use a ferramenta WebDeploy para distribuir aplicações clientes, sem necessidade de configurações, através da Web.

Novidades no Desenvolvimento de Aplicações Distribuídas

O Delphi 4 oferece mais controle sobre aplicações multi-camadas, sendo mais fácil chamar interfaces de servidores.

Oferece também suporte para o uso do Microsoft Transaction Server (MTS), incluindo um assistente que facilita a criação de objetos MTS.

As versões Client/Server e Enterprise incluem suporte para clientes e servidores CORBA. Assistentes facilitam a

criação de servidores CORBA. Você pode criar um servidor que manipule clientes COM e CORBA simultaneamente.

O servidor de aplicações MIDAS oferece uma forma simples dos programadores gerenciar centralizadamente a lógica de suas aplicações. MIDAS distribui aplicações clientes leves que são robustas e seguras.

Novidades na Criação de Componentes ActiveX

O Delphi 4 oferece suporte melhorado a tecnologia ActiveX. Os assistentes agora suportam a eventos para os objetos ActiveX. É oferecido um novo assistente para a criação de objetos COM mais simples.

Componentes do Delphi 4

Além do ambiente de desenvolvimento, o Delphi é composto de diversos utilitários, entre eles:

BDE Administrator: gerencia conexões com banco de dados.

Database Desktop: cria e gerencia bancos de dados.

Database Explorer: visualizador hierárquico de banco de dados.

Image Editor: editor de imagens.

Data Pump: transforma banco de dados (estrutura e dados) de um tipo em outro.

SQL Monitor: monitora comandos SQL.

As Versões do Delphi 4

O Delphi 4 é vendido em três versões: Standard, Professional e Client/Server, sendo a Client/Server a mais completa.

Standard: a mais simples, voltada para usuários iniciantes, inclui recursos elementares para desenvolvimento de aplicações com ou sem acesso a banco de dados.

Professional: além de todos os recursos da versão acima, inclui assistentes para o código, para criação de objetos COM e assistentes e componentes para Internet.

Client/Server: além dos recursos da versão Professional, inclui depuração remota, componentes para análises multidimensionais, assistentes para a criação de servidores WEB, o utilitário SQL Monitor, suporte para Oracle 8 e diversas tecnologias para a construção de aplicativos multi-camadas.

2

O Ambiente do Delphi 4

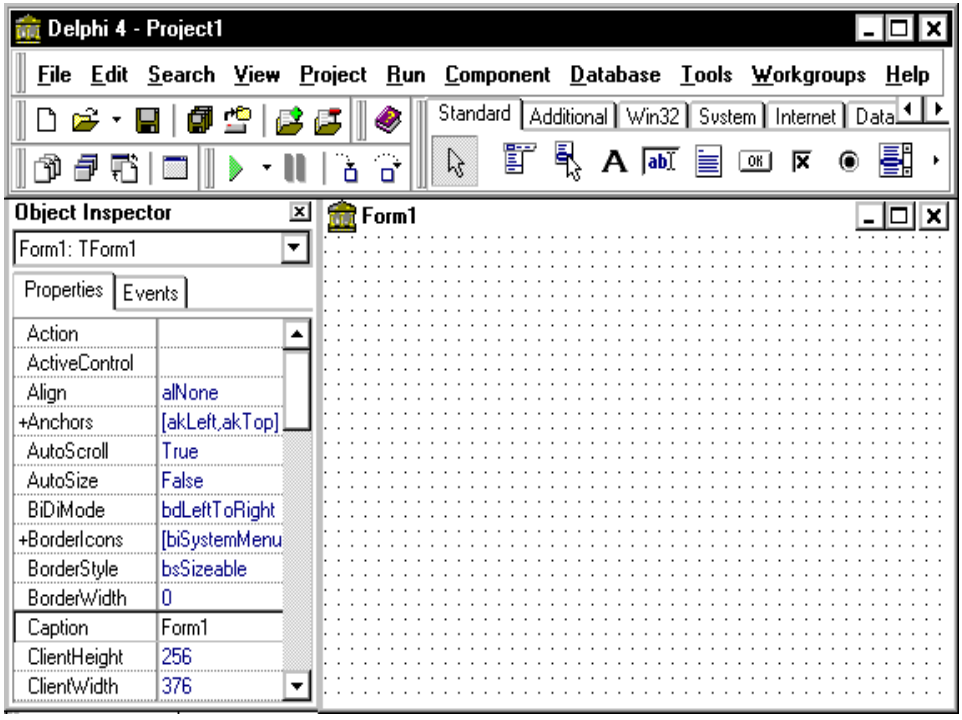
Entrando no Delphi 4

Se você instalou o Delphi 4 sem mudar sua pasta padrão, o programa e seus componentes estarão instalados na pasta Arquivos de Programas, grupo Borland.

Para acessar o Delphi, clique no botão *Iniciar*, selecione *Programas*, escolha a opção *Borland Delphi 4* e clique na opção *Delphi 4*. Você entrará no ambiente de trabalho do Delphi 4.

O Ambiente de Trabalho do Delphi 4

Ao entrar no Delphi 4, você encontrará o ambiente de trabalho mostrado a seguir:



O ambiente de trabalho do Delphi está dividido em 5 partes:

- Barra de Menus
- Barra de Ferramentas
- Paleta de Componentes
- Object Inspector
- Formulário

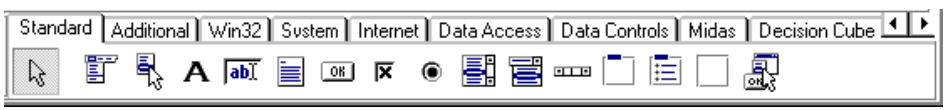
Barra de Menus

Acessa os comandos do Delphi, para atividades como abrir arquivos, compilar, etc.

Barra de Ferramentas

Contém ícones que são atalhos para os comandos do menu. Como veremos, você poderá incluir outros ícones nesta barra, adaptando-a as suas necessidades.

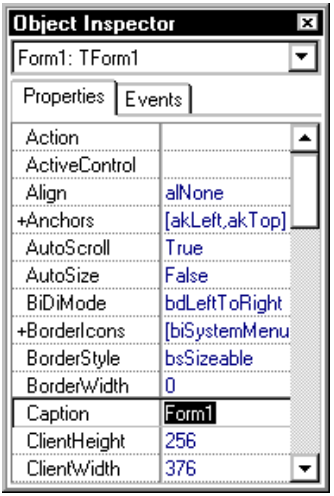
Paleta de Componentes



Contém diversas páginas com grupos de objetos para inserção de botões, caixas de edição, texto, etc, no formulário. Como veremos, você poderá modificar esta

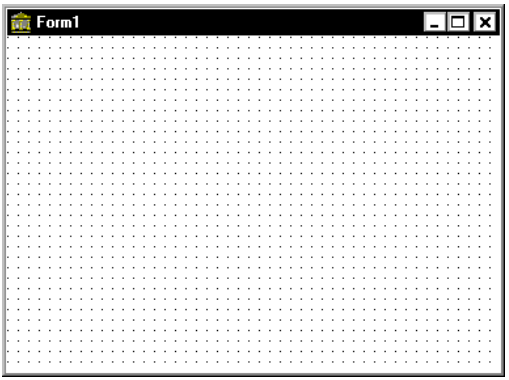
paleta, e também incluir outros componentes em suas páginas.

Object Inspector



Determina propriedades e eventos para componentes e formulários.

Formulário



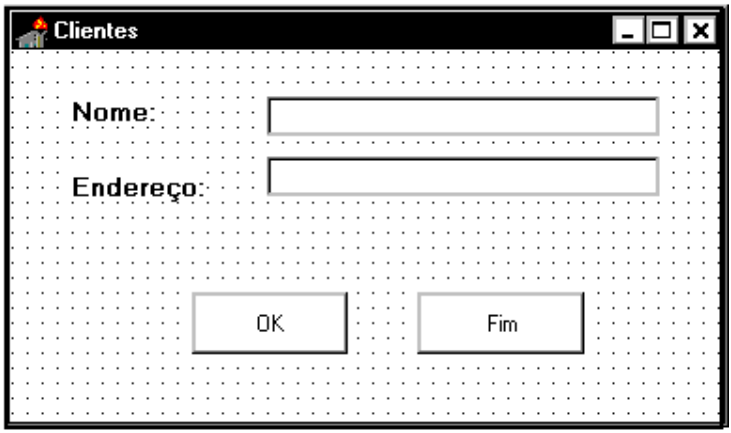
Janela para interface com o usuário, onde são inseridos os componentes.

Delphi 4

3
Formulários e Componentes

Formulários

Você cria formulários para servir como interface de suas aplicações. Cada formulário é uma janela que mostra textos, gráficos ou outros objetos.



Tipos de Formulários

Existem quatro tipos de formulários, de acordo com a propriedade FormStyle:

Normais (fsNormal): são formulários que não possuem vínculos com outros formulários. Uma aplicação composta de formulários desvinculados é uma aplicação SDI (Single Document Interface).

MDIParent (fsMDIForm): são formulários que permitem gerenciar outros formulários, chamados de filhos. Aplicações que usam estes formulários são chamadas aplicações MDI (Multiple Document Interface).

MDIChild (fsMDIChild): são os formulários filhos, gerenciados por um formulário pai (MDIParent), em uma aplicação MDI.

StayOnTop (fsStayonTop): são formulários que ficam sempre visíveis, independente de possuirem ou não o foco.

O padrão do Delphi é a construção de formulários do tipo Normal.

Operações com Formulários

Para mover um formulário, clique sobre a barra de título do formulário selecionado, mantenha o botão do mouse apertado e arraste o formulário para a posição desejada.

Para redimensionar um formulário, clique sobre uma das molduras que envolvem o formulário. O cursor muda para o formato <-||->. Mantenha o botão do mouse apertado e arraste a moldura para a posição desejada.

Como veremos, as propriedades *Top, Height, Width* e *Left* posicionam e dimensionam o formulário, em pixels.

Já a propriedade *Position* determina a posição que o formulário ocupará na tela, quando a aplicação for executada. A opção poScreenCenter centraliza o formulário na tela.

Componentes

Componentes são objetos como caixas de edição, botões e etiquetas, que você coloca em um formulário para interfaciar com o usuário.

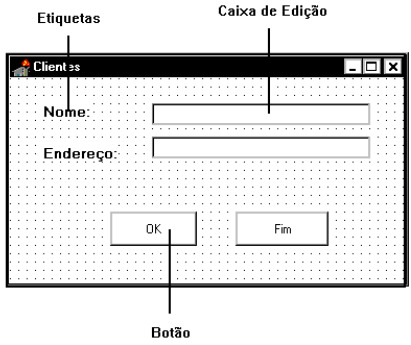
Por exemplo, você pode inserir os seguintes objetos no formulário:

Caixa de Edição (Edit): oferece ao usuário uma área de entrada de dados.

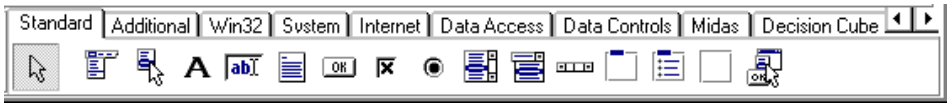
Etiqueta (Label): mostra um texto ao usuário.

Botão (Button): para o usuário clicar e realizar uma tarefa.

Na figura abaixo,vemos um formulário com alguns componentes inseridos:



A Paleta de Componentes



A Paleta de Componentes é composta de várias páginas contendo cada uma grupos de ícones. Cada ícone corresponde a um objeto que você pode inserir no formulário.

A Paleta de Componentes vem com as seguintes páginas:

Standard: engloba os componentes mais utilizados, tais como botões, etiquetas, caixas de edição.

Additional: engloba componentes menos utilizados, tais como botões com bitmap, imagens, ícones, etc.

Win32: engloba componentes padrões do Windows 95 e 98, como TabControl, UpDown, RichEdit.

System: engloba componentes que manipulam o sistema, como temporizador.

Internet: engloba componentes para criação de browsers, navegação pela Internet, etc.

DataAccess: engloba componentes que manipulam bancos de dados.

DataControls: engloba controles para visualização de registros de bancos de dados.

Midas: engloba controles para construção de aplicativos multicamadas.

Decision Cube: engloba componentes para tomada de decisão.

QReport: engloba componentes especificos para criar relatórios, tais como: QuickReport, QRBand, QRLabel, QRDBText, QRMemo, QRShape e outros.

Dialogs: engloba componentes referentes a quadros de diálogo padrão, tais como Abrir, Salvar, etc.

Win 3.1: engloba componentes do Windows 3.1, como Notebook, Outline, Header e TabSet.

ActiveX: engloba componentes utilizados para criação de graficos, GraphicsServer, VCFirstImpression, entre outros.

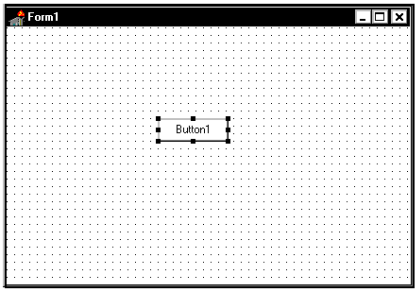
Samples: engloba componentes criados como exemplo, como ColorGrid, Calendar.

Para mudar de página na Paleta de Componentes, clique sobre a aba da página que você quer acessar.

Operações com Componentes

Para inserir um componente no formulário, clique no componente, na Paleta de Componentes, posicione o cursor no formulário, onde você quer inserir o componente e clique e arraste o cursor, criando a área do componente.

Outra opção seria dar um duplo clique sobre o componente. Este é inserido automaticamente no centro do formulário.



novo! A partir de agora, sempre que você passar o mouse sobre o componente, aparecerá uma legenda com o nome e o tipo do componente.

Dica: Para inserir vários componentes do mesmo tipo no formulário, mantenha a tecla Shift apertada quando clicar sobre o componente na Paleta. Dessa forma, você poderá inserir diversos componentes do tipo selecionado no formulário, sem ter que clicar novamente em seu ícone na Paleta.

Os componentes inseridos no formulário podem ser movidos ou redimensionados. Para isto, você deve selecioná-los, clicando sobre estes.

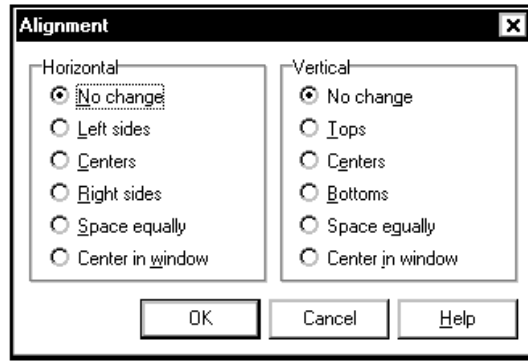
Para movê-los, clique sobre o componente selecionado, mantenha o botão do mouse apertado e arraste o componente para a posição desejada.

Para redimensionar os componentes, clique sobre uma das alças que envolvem o componente, mantenha o botão do mouse apertado, e arraste a alça para a posição desejada.

Para remover um componente do formulário, clique sobre o componente que você quer remover, selecionando-o e

aperte a tecla *DEL*.

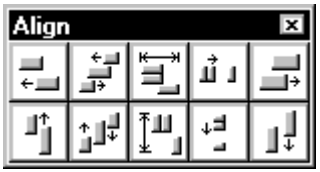
Depois de inserir diversos componentes no formulário, é comum que você queira alinhá-los. Para isso, selecione os componentes que serão alinhados: clique no primeiro componente (que servirá de padrão), aperte a tecla Shift e a mantenha apertada, e clique sobre os demais componentes. Todos serão selecionados. Escolha, no menu *Edit*, a opção *Align* (ou clique com o botão direito do mouse sobre o formulário e no menu que aparece escolha a opção *Align)*. Aparece o quadro Alignment:



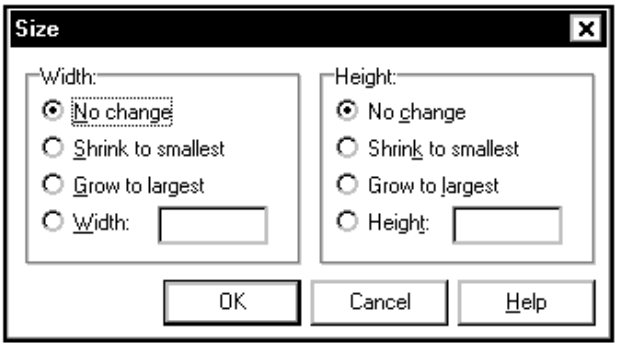
◆ no quadro Horizontal, você define como os componentes serão alinhados na horizontal: *No Change* (sem mudança), *Left Sides* (pela esquerda), *Centers* (pelo centro), *Right sides* (pela direita), *Space equally* (com igual espaçamento), *Center in Windows* (centralizado na janela);

◆ no quadro Vertical, você define como os componentes serão alinhados na vertical: *No Change* (sem mudança), *Tops* (pelo topo), *Centers* (pelo centro), *Bottom* (por baixo), *Space equally* (com igual espaçamento), *Center in Windows* (centralizado na janela).

Através do menu *View*, opção *Alignment Palette*, você pode manter visível uma paleta para alinhamentos:



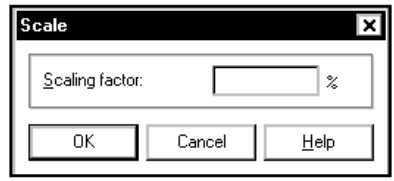
Além de alinhar, você pode redimensionar diversos componentes do formulário para um tamanho padrão. Para isso, selecione os componentes que serão redimensionados: clique no primeiro componente, aperte a tecla *Shift* e a mantenha apertada, e clique sobre os demais componentes. Todos serão selecionados. Escolha no menu *Edit* a opção *Size* (ou clique com o botão direito do mouse sobre o formulário e no menu que aparece escolha a opção *Size)*. Aparece o quadro Size:



◆ no quadro Width, escolha a largura do componente: *No Change* (sem mudança), *Shrink to smallest* (reduza para o menor dos componentes selecionados), *Grow to largest* (cresça para o maior dos componentes selecionados), ou defina na opção *Width* a largura do componente, em pixels;

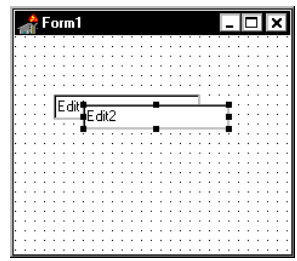
◆ no quadro Heigth, escolha a altura do componente: *No Change* (sem mudança), *Shrink to smallest* (reduza para o menor dos componentes selecionados), *Grow to largest* (cresça para o maior dos componentes selecionados), ou defina na opção *Heigth* a altura do componente, em pixels.

Você pode escolher uma escala, em relação ao tamanho original do componente. Para isto, selecione, no menu *Edit*, a opção *Scale* (ou clique com o botão direito do mouse sobre o formulário e no menu que aparece escolha a opção *Scale)*. Aparece o quadro Scale:



Entre com a escala (entre 25% e 400%) e clique *OK*.

Muitas vezes, um componente quando é inserido no formulário acaba se sobrepondo a outro componente que lá já existia:



Para mudar a ordem de sobreposição de componentes, clique sobre o componente que você quer reordenar, e selecione no menu *Edit* a opção *Send to Back*, se quiser enviá-lo para trás dos demais componentes, ou *Bring to Front*, se quiser enviá-lo para frente dos demais componentes.

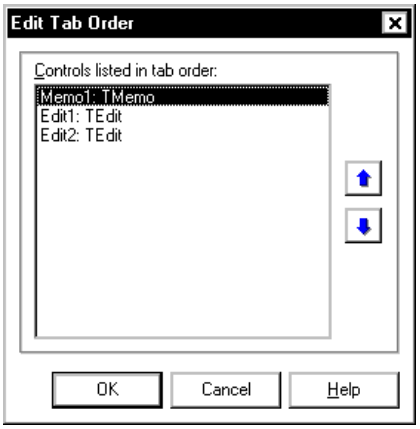
Para congelar os componentes inseridos no formulário, impedindo que eventualmente se estrague seu posicionamento, selecione os componentes a serem congelados, e no menu *Edit*, escolha a opção *Lock Controls*, para congelá-los.

Para descongelar componentes congelados, basta repetir a operação acima.

Mudando a Ordem de Tabs

Quando você executa sua aplicação, pode passar de um componente para outro simplesmente teclando TAB. A ordem de ativação dos componentes a cada TAB corresponde normalmente a ordem de inserção dos componentes no formulário. Mas você pode redefinir esta ordem.

Para isto no menu *Edit*, escolha a opção *Tab Order*. Aparece o quadro abaixo, listando todos os componentes do formulário acessáveis através de um TAB, na ordem de ativação:



Se quiser mudar a ordem de determinado componente, selecione-o e clique em uma das setas à direita, até o componente atingir a ordem desejada.

Propriedades

Os componentes que você insere no formulário, como caixas de edição, quadros de lista, possuem características chamadas propriedades. As propriedades dos componentes estabelecem valores iniciais para características como altura, largura, nome, título, texto, entre outras. O formulário também possui propriedades.

Entre as propriedades comuns dos formulários e componentes temos:

Caption: estabelece o título do formulário ou componente.

Heigth: estabelece a altura do formulário ou componente.

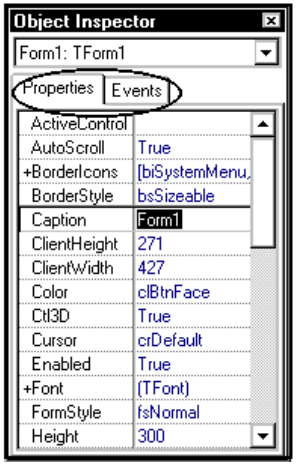
Name: estabelece um identificador para uso no código de sua aplicação.

Width: estabelece a largura do formulário ou componente.

Você determina as propriedades de formulários e componentes durante o projeto de sua aplicação através do Object Inspector. Como veremos posteriormente, estas propriedades também podem ser mudadas durante a execução da aplicação.

O Object Inspector

O Object Inspector (Inspetor de Objetos) é uma ferramenta composta de duas páginas: Properties (Propriedades) e Events (Eventos).



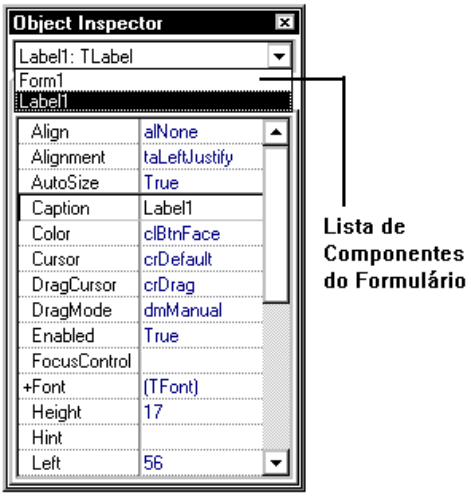
A página Properties permite que você estabeleça as propriedades de formulários e componentes.

A página Properties é composta de duas colunas: a esquerda temos os nomes das propriedades associadas ao componente; a direita temos os valores possíveis para estas propriedades.

Para mudar o valor de uma propriedade:

◆ clique sobre o formulário ou sobre o componente inserido no formulário para o qual você quer mudar a propriedade, selecionando-o (deverão aparecer alças em volta do componente, caso a seleção tenha sido feita corretamente);

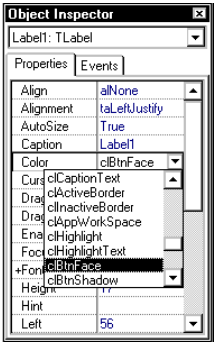
◆ outra forma de selecionar um componente para mudar suas propriedades é clicar na seta da caixa de lista na parte superior do Object Inspector:



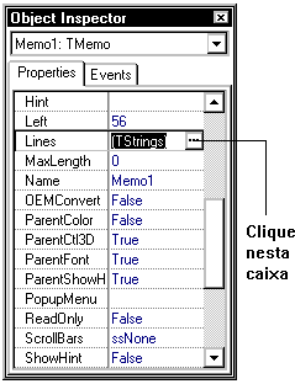
◆ no Object Inspector, selecione a página *Properties*, clicando sobre a aba na parte superior da ferramenta;

◆ procure a propriedade que você quer modificar, utilizando a barra de rolamento;

◆ dê um clique sobre a coluna que conterá o valor da propriedade. Isto ilumina o conteúdo da propriedade. Se não aparecer nada a direita da coluna, você mesmo deverá digitar o contéudo da propriedade.
◆ se, ao clicar na coluna, aparecer uma seta a direita da coluna, basta clicar sobre esta seta para acessar as opções possíveis para a propriedade:

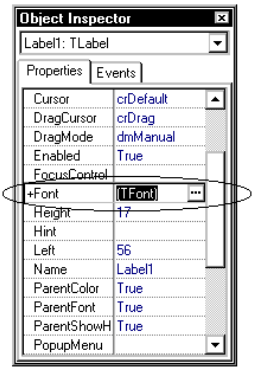


◆ se aparecer uma caixa contendo três pontos [...], basta clicar sobre estes pontos para acessar um quadro de diálogo contendo as opções da propriedade:



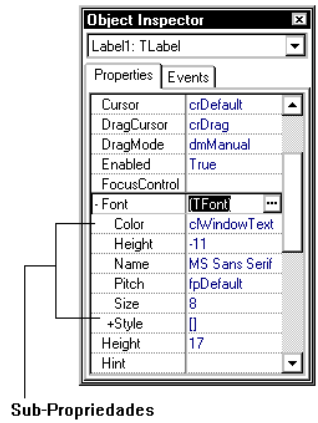
◆ após escolher o conteúdo da propriedade, tecle *Enter*. O formulário ou componente é modificado de acordo com o novo valor da propriedade.

Se a propriedade possuir um sinal de + a sua esquerda, ela engloba sub-propriedades:



Para acessar estas sub-propriedades, basta dar um duplo clique sobre o nome da propriedade. O sinal de + se transforma em um sinal de - (menos), e a lista de sub-propriedades aparece. Para fechar a lista de sub-propriedades, basta dar outro duplo-clique sobre o nome da propriedade.

Por exemplo, para o componente Label, há um sinal de + a esquerda da propriedade Font. Se dermos um duplo clique sobre o nome da propriedade Font, no Object Inspector, aparecerá uma lista de sub-propriedades: Charset, Color, Height, Name, Pitch, Size, Style:



Note que Style também possuei sub-propriedades.

A página Events será vista posteriormente.

Quando trabalhamos com formulários muito grandes, é comum que o formulário fique por cima do Object Inspector, atrapalhando a mudança de propriedades. Se isto acontecer, basta teclar F11 e o Object Inspector aparece.

Outra opção é manter o Object Inspector sempre visível. Para isso, clique com o botão direito sobre uma área em branco do Object Inspector. No menu que aparece, escolha a opção *Stay on Top*. Agora, mesmo que seu formulário utilize a tela inteira, o Object Inspector estará sempre visível.

Delphi 4

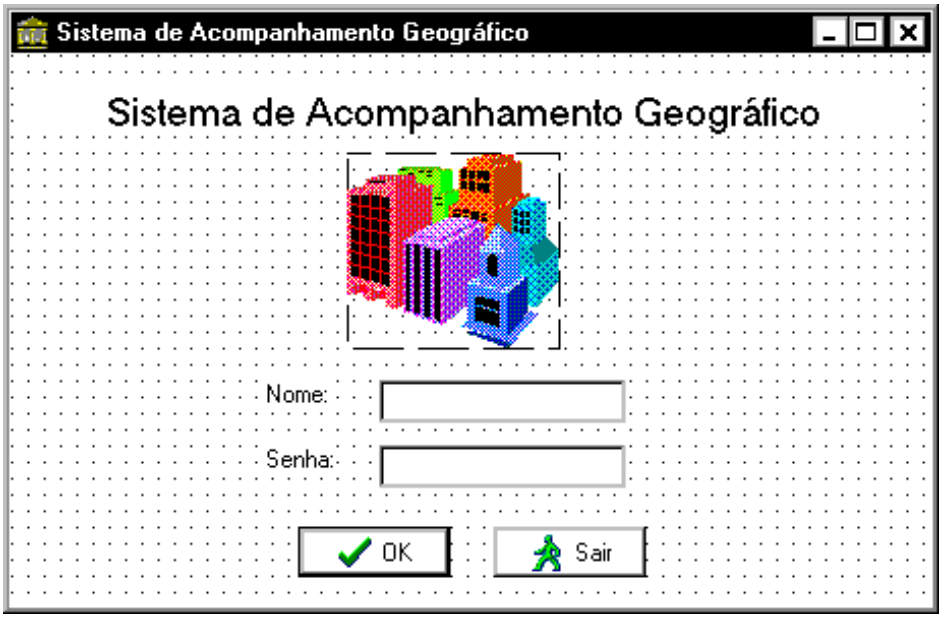
Sistema de Acompanhamento Geográfico

Parte I

Tela Principal

Para acompanhar os conceitos apresentados no livro, vamos criar um Sistema de Acompanhamento Geográfico (SAG), que gerenciará os países existentes atualmente.

Inicialmente, vamos criar um formulário que contenha a tela principal do Sistema:



1) Entre no Delphi, ou escolha, no menu *File*, a opção *New Application*, caso já esteja no Delphi.

2) Mude, através do Object Inspector, a propriedade Caption para *Sistema de Acompanhamento Geográfico*, a propriedade Name: *FrmPrincipal*, a propriedade Position para *poScreenCenter*, a propriedade Heigth para 300 e a propriedade Width para 460.

3) Insira um componente *Label*  , do grupo Standard de componentes, redimensionando-o de acordo com o modelo.

4) Mude, através do Object Inspector, a propriedade Alignment para *taCenter* (para centralizar o texto), a propriedade Caption para *Sistema de Acompanhamento Geográfico,* a propriedade Font / Size para *16 e a* propriedade Name para *LblTitulo.*

5) Insira um componente *Image*  , do grupo de componentes Additional, posicionando-o de acordo com o modelo.

6) Mude, através do Object Inspector, a propriedade Picture para a imagem que você desejar. No nosso exemplo, usamos o arquivo Edificio.wmf, do Microsoft Office. Mude a propriedade Stretch para *True* (para que a imagem se adapte a moldura), e a propriedade Name para *ImgSistema.*

7) Insira um componente *Label*, do grupo Standard, mudando sua propriedade Caption para *Nome:*, e sua propriedade Name para *lblNome*.

8) Insira outro componente *Label*, do grupo Standard, mudando sua propriedade Caption para *Senha:*, e sua propriedade Name para *lblSenha*.

9) Insira um componente *Edit*, do grupo Standard, mudando sua propriedade Text para vazia, e sua propriedade Name para *EditNome*.

10) Insira um componente *Edit*, do grupo Standard, mudando sua propriedade Text para *vazia*, e sua propriedade Name para *EditSenha*.

11) Insira um componente *BitBtn*, do grupo Additional de componentes, posicionando-o de acordo com o modelo.

12) Mude, através do Object Inspector, a propriedade Kind do componente acima para *bkOK*, e a propriedade Name para *btnOK*.

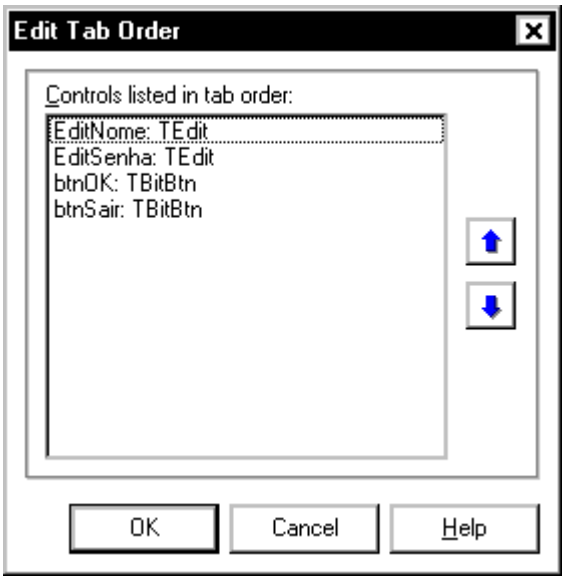
13) Insira um outro componente *BitBtn*, do grupo Additional de componentes, posicionando-o de acordo com o modelo.

14) Mude, através do Object Inspector, a propriedade Kind do componente acima para *bkIgnore*, a propriedade Caption para *Sair*, e a propriedade Name para *btnSair*.

15) Selecione todos os componentes, através do menu *Edit* opção *Select All*.

16) No menu *Edit*, selecione a opção *Align* e, no quadro que aparece, escolha, no quadro de opções *Horizontal*, a opção *Center in window*, para centralizar todos os componentes na janela.

17) Confira, através do menu *Edit*, opção *Tab Order*, se a ordem de tabulação do formulário está correta:



As opções de armazenamento de aplicações serão vistas com mais detalhes posteriormente. No momento, siga os passos abaixo:

1) No menu *File*, escolha a opção *Save Project As*.

2) No quadro Save Unit1 As, selecione uma pasta e salve a unit com o nome *UnitPrincipal*, e tecle *OK*.

3) No quadro Save Project As, salve seu projeto com o nome *SAG*, e tecle *OK*.

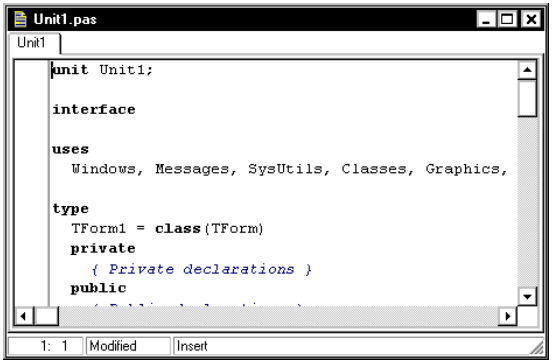
Para executar sua aplicação, clique sobre o ícone  , da Barra de Ferramentas, ou tecle F9.

Após a execução, para voltar ao ambiente de trabalho do Delphi, basta fechar a janela do formulário.

4
Gerenciando o Projeto

O Código do Formulário

A cada formulário da aplicação está associada uma Unit, que você visualiza na Janela de Código:



É nesta unit que você irá colocar as rotinas necessárias para o seu programa. Mesmo que você não tenha necessidade de escrever alguma linha de código para rodar seu programa, o Delphi já inclui algumas linhas default, que são necessárias para a execução do programa.

Units serão vistas com mais detalhes posteriormente.

Projetos

Uma aplicação em Delphi é composta de diversos arquivos: arquivos com código, arquivos binários contendo os formulários, arquivos de recursos, entre outros.

Para gerenciar os arquivos de uma mesma aplicação, usamos o conceito de projeto. Uma aplicação em Delphi na realidade corresponde a um projeto. Quando você abre o projeto de uma aplicação, está abrindo diversos arquivos. Quando você salva o projeto de uma aplicação, está salvando diversos arquivos simultaneamente.

Alguns arquivos são gerados durante o desenho de sua aplicação. Outros são gerados quando sua aplicação é compilada.

Quando você cria uma aplicação em Delphi, gera os seguintes arquivos durante o desenho da aplicação:

**.DPR:* é o arquivo principal do projeto, que lista todos os formulários e units usados na aplicação. Há um arquivo .DPR para cada aplicação.

**.PAS*: arquivos que contém o código. Há um arquivo .PAS para cada formulário.

**.DFM*: arquivos binários referentes aos formulários.

**.RES*: arquivo de recursos, contendo o ícone da aplicação e outros recursos usados pelo projeto.

**.DOF*: arquivo texto contendo os parâmetros do projeto.

**.CFG*: arquivo contendo os parâmetros de configuração do projeto.

**.DSK*: arquivo contendo parâmetros do ambiente de programação para aquele projeto.

**.~DP, *.~DF, *.~PA*: arquivos back-up referentes, respectivamente aos arquivos DPR, DFM e PAS.

Os arquivos abaixos são gerados durante a compilação:

**.DCU*: é o arquivo .PAS compilado. Cada arquivo .PAS gera um arquivo .DCU.

**.EXE*: é o arquivo executável da aplicação.

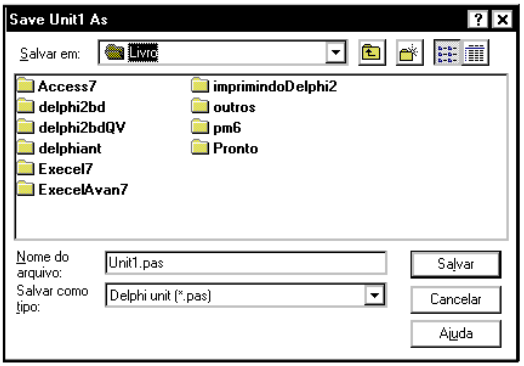
Salvando a Aplicação

Salvar sua aplicação implica em salvar um Projeto. Ao salvar o projeto, você estará salvando os arquivos de formulários e units ligados a sua aplicação.

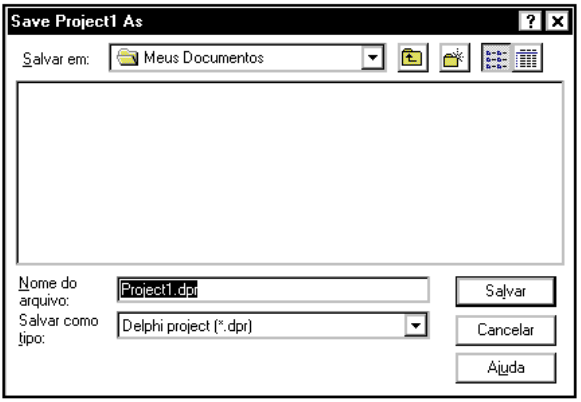
Quando você começa um novo projeto, o Delphi já lhe oferece um nome para a Unit, chamada Unit1, e um nome para o projeto, chamado Project1. Você pode aceitá-los ou,

o que é mais conveniente, pode dar nomes mais apropriados.

Para salvar o projeto pela primeira vez, escolha, no menu *File*, a opção *Save All*, ou clique sobre o ícone  , da Barra de Ferramentas. Aparece o quadro Save Unit1 As:



No quadro acima, dê um nome para a Unit (ele dá o mesmo nome da Unit ao formulário associado), e clique *OK*. Se você tiver mais de uma unit no projeto, este quadro será mostrado novamente, pedindo o nome das demais units. Quando terminar de nomear as units, aparece o quadro *Save Project1 As*:



Dê um nome para seu projeto e clique *OK*.

Dica: Não confunda o nome do arquivo, usado para fins de armazenamento, com a propriedade Name, usada para referência ao formulário a nível de programação.

Para salvar seu projeto posteriormente, basta escolher a opção *Save All* do menu *File*. Caso tenha inserido um novo formulário ou unit, o Delphi pedirá um nome para ele, quando você salvar o projeto novamente.

Salvando um Arquivo

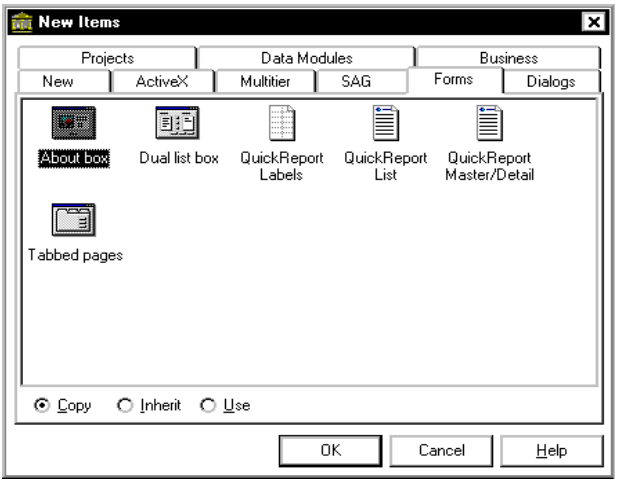
Você pode salvar somente o formulário ou o arquivo de Unit. Para isto, use a opção *Save* do menu *File*, ou clique sobre o ícone  .

Inserindo um Novo Formulário na Aplicação

Para inserir um formulário em branco em sua aplicação, escolha a opção *New Form*, no menu *File*.

Lembre-se: associada ao formulário há sempre uma unit.

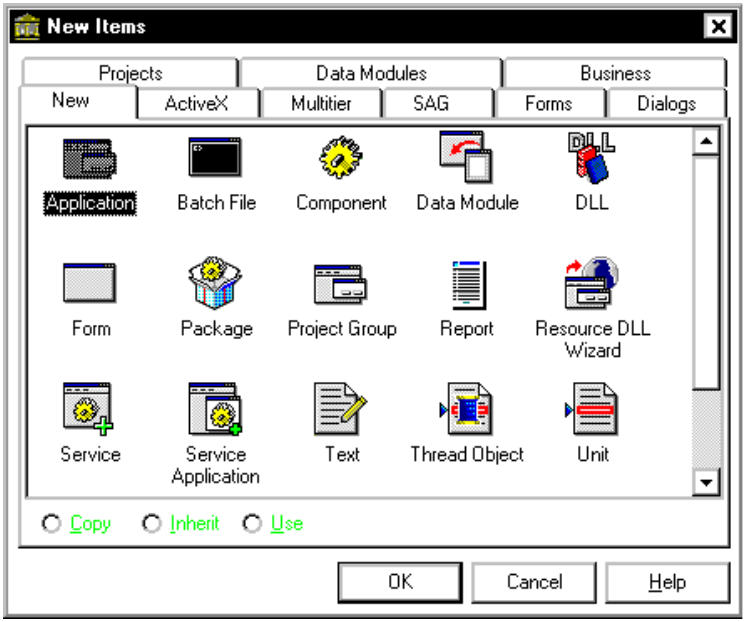
Já se você escolher no menu *File* a opção *New*, guia *Forms*, poderá escolher entre diversos modelos de formulários pré-construídos:



Inserindo uma Nova Unit na Aplicação

Você pode inserir em sua aplicação units sem estarem vinculadas a formulários. Estas units poderiam guardar funções e rotinas usadas pelo resto da aplicação.

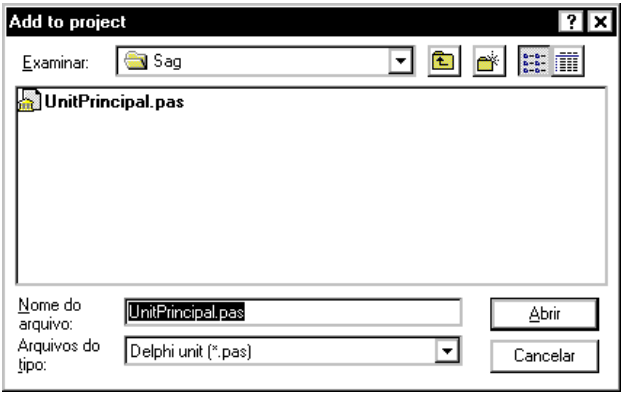
Para inserir uma nova unit em sua aplicação, escolha a opção *New* do menu *File.* Aparece o quadro New Items. Clique na ficha *New* e selecione o ícone *Unit:*



Clique no botão *OK* para inserir a unit no projeto.

Adicionando um Formulário já Existente

Para adicionar um formulário já existente a sua aplicação, escolha, no menu *Project*, a opção *Add to Project*, ou clique no ícone da Barra de Ferramentas. Aparece o quadro abaixo:

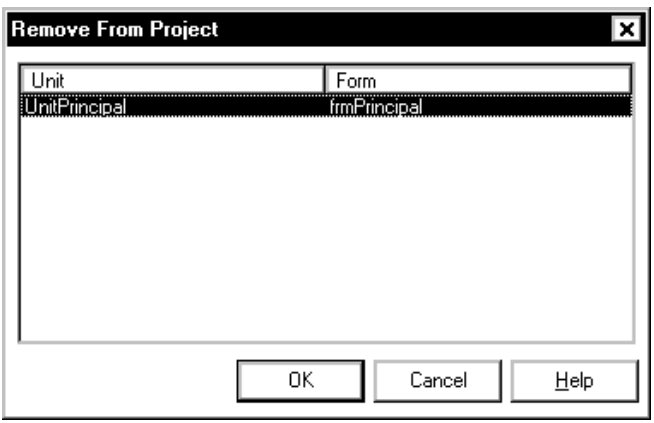


Selecione o arquivo e clique no botão *Abrir*.

Mas cuidado! Se você fizer modificações neste formulário, e ele for usado em outra aplicação, poderá causar transtornos. Para evitar isto, após adicionar o formulário, salve-o através do comando *Save As*, do menu *File*, na pasta de sua aplicação.

Excluindo um Formulário da Aplicação

Para remover uma unit ou formulário com sua respectiva unit de sua aplicação, escolha, no menu *Project*, a opção *Remove From Project...*, ou clique no ícone da Barra de Ferramentas. Aparece o quadro a seguir:



Selecione a unit que você deseja remover e clique no botão *OK*.

Quando você exclui uma unit, ela é dissociada da aplicação, mas continua no disco rígido.

Fechando um Projeto

Para fechar um projeto, escolha, no menu *File*, a opção *Close All*.

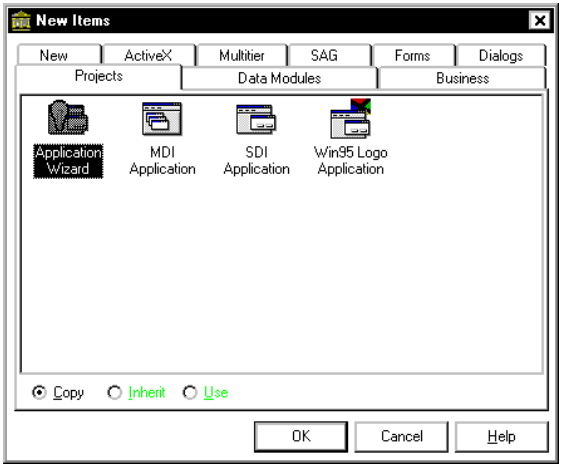
Fechando um Formulário ou Unit

Para fechar um formulário ou unit, escolha, no menu *File*, a opção *Close* .

Começando uma Nova Aplicação

Para criar um novo projeto, escolha, no menu *File*, a opção *New Application*.

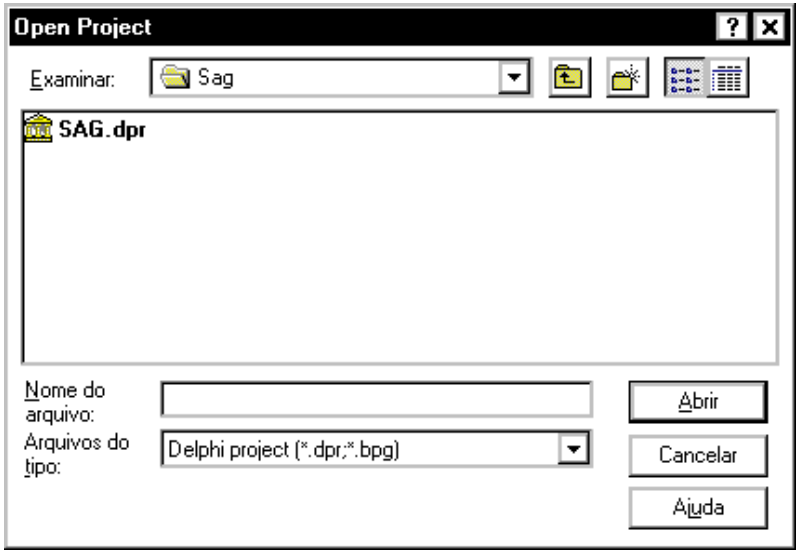
Se você desejar escolher um dos modelos de projetos que o Delphi oferece, basta você clicar na opção *New* do menu *File*. Aparece o quadro *New Items*. Clique na ficha *Projects:*



Selecione o modelo e clique no botão *OK*.

Abrindo um Projeto Existente

Para abrir um projeto já existente, escolha, no menu *File*, a opção *Open Project*, ou clique no ícone , da Barra de Ferramentas. Aparece o quadro abaixo:



Escolha o nome da aplicação a ser recuperada e clique em *Abrir*.

Se você tiver utilizado o projeto recentemente, é provável que ele esteja listado no menu *File*, opção *Reopen*.

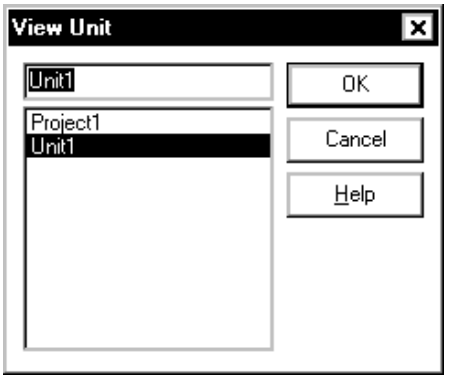
Abrindo um Formulário ou Unit Existente

Para abrir um formulário ou unit já existente, escolha, no menu *File*, a opção *Open*, ou clique sobre o ícone . Escolha o nome do formulário ou unit a ser recuperado e clique no botão *OK*.

Vendo os Formulários e Units do Projeto

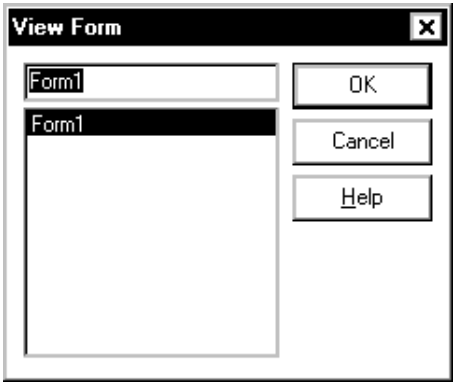
Quando se trabalha com mais de um formulário, é comum termos dificuldades de se acessar um ou outro formulário, já que um se sobrepõe ao outro.

Para escolher uma unit a ser visualizada, escolha no menu *View* a opção *Units*. Aparece o quadro abaixo:



Selecione a unit que você deseja acessar, e clique no botão *OK*.

Para acessar determinado formulário, escolha no menu *View* a opção *Forms*. Aparece o quadro abaixo:



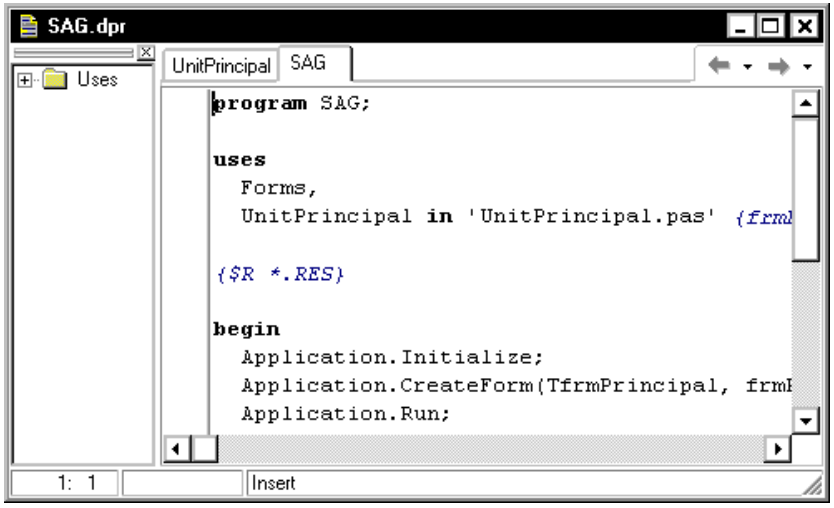
Selecione o formulário que você deseja acessar, e clique no botão *OK*.

O Arquivo Fonte do Projeto

Além de todos os arquivos de units e formulários, um projeto possui um arquivo fonte, que mostra quais formulários e units o compõe. Você não deve modificar este arquivo diretamente, sob pena de impedir a execução da aplicação.

Para ver as linhas de código deste arquivo fonte, selecione no menu *Project* a opção *View Source*.

Veja o arquivo fonte do projeto da aplicação SAG, que contém apenas a unit UnitPrincipal:



Além da lista de units que compõe o projeto, este arquivo contém comandos para inicializar e executar a aplicação.

Trabalhando com Grupo de Projetos

novo! A versão 4 do Delphi introduz a possibilidade de se trabalhar com diversos projetos simultaneamente, em um grupo de projetos.

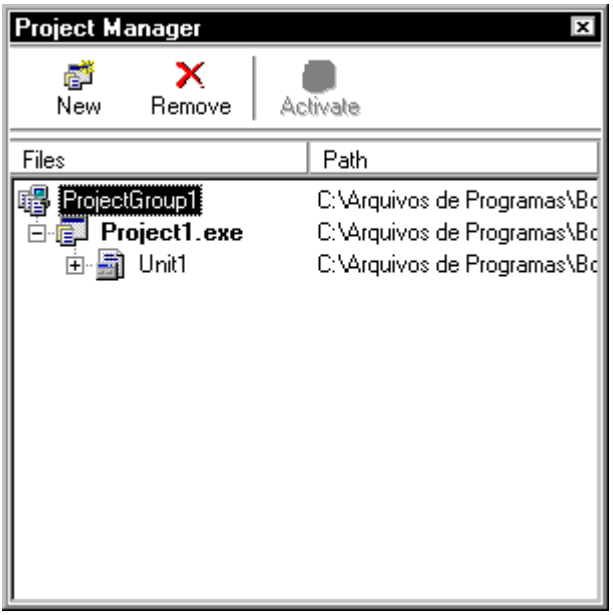
Este recurso facilita bastante a criação de bibliotecas (DLL's) ou controles ActiveX, que precisam ser testados em parceria com outro projeto.

Os projetos que compõe o grupo estão armazenados em um arquivo com extensão .BPG.

As operações com grupos de projetos são feitas a partir do gerenciador de projetos do Delphi, o Project Manager.

Gerenciando Projetos com o Project Manager

O Delphi oferece um gerenciador de projetos chamado *Project Manager*. Você acessa o Project Manager através do menu *View*, opção *Project Manager*:

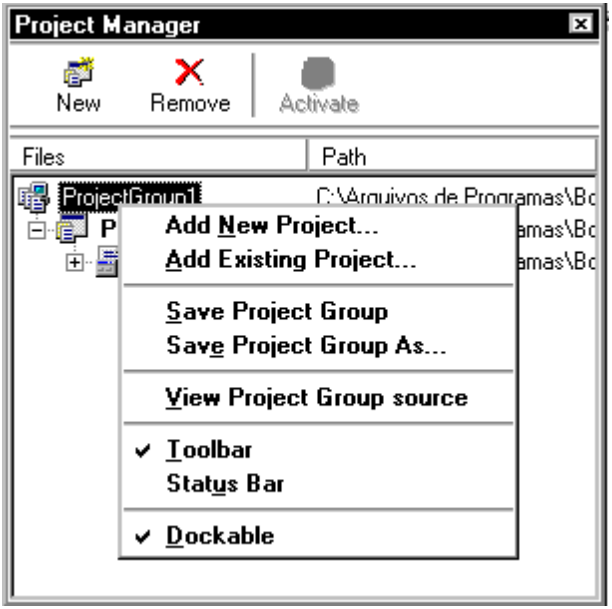


Através do Project Manager, você pode visualizar os projetos de um grupo de projeto, assim como as units e os formulários de cada um dos projetos.

Gerenciando o Grupo de Projetos

Quando você inicia um novo projeto, este já está incluído automaticamente em um arquivo de grupo, com o nome genérico ProjectGroup1. Para salvar o arquivo de grupo, clique com o botão direito sobre o item ProjectGroup1 e selecione a opção *Save Project Group.*

Além da opção acima, você poderá escolher uma das seguintes opções:

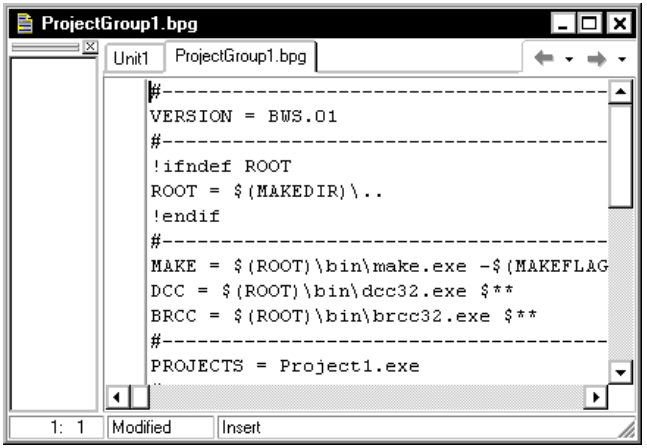


Add New Project: adiciona um novo projeto ao grupo.

Add Existing Project: adiciona um projeto existente ao grupo.

Save Project Group As: salva o arquivo de grupo de projetos com outro nome.

View Project Group Source: vê o fonte do arquivo de grupo de projetos (.BPG):



O Project Manager lista todos os arquivos de um projeto, sendo que o projeto ativo (aquele em que serão inseridos formulários e data modules) está negritado. Para tornar outro projeto ativo, dê um duplo clique sobre ele.

Você pode compilar todos os projetos de um grupo ou apenas determinado projeto. Para compilar todos os projetos, selecione o arquivo de grupo de projetos no Project Manager, e no menu *Project* escolha a opção *Compile All Projects* ou *Build All Projects*. Já para compilar apenas um projeto, selecione-o no *Project Manager* e, no menu *Project* escolha a opção *Compile Project* ou *Build Project*.

Se você clicar com o botão direito do mouse sobre um dos projetos, terá acesso a um menu contendo as opções vistas no início deste capítulo, para adicionar e remover arquivos, fechar o projeto, ver o fonte, etc.

Da mesma forma, se você clicar com o botão direito do mouse sobre uma das units de um dos projetos, terá acesso a um menu para visualizar a unit, removê-la do projeto, etc.

Para fechar o Project Manager, dê um duplo clique sobre o botão de controle da janela.

Delphi 4

Sistema de Acompanhamento Geográfico

Parte II

O Formulário Sobre

A maioria das aplicações profissionais possui um formulário com dados sobre o sistema, como a versão, o nome completo, etc.

Vamos adicionar ao projeto do Sistema de Acompanhamento Geográfico um formulário Sobre.

Para isso, siga os passos abaixo:

1) Abra o projeto SAG.dpr, através do menu *File*, opção *Open Project*.

2) No menu File, escolha a opção New, aba Forms. Selecione a opção About Box. Um formulário Sobre pré construído é oferecido:

3) Mude as seguintes propriedades do formulário:

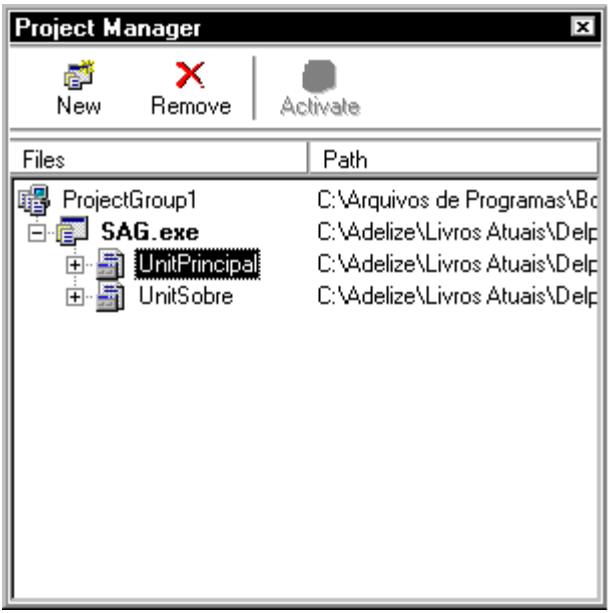
Componente	Propriedade	Para
AboutBox	Caption	Sobre
ProductName	Caption	Sistema de Acompanhamento Geográfico
Version	Caption	Versão 1.0
Copyright	Caption	Copyright Advanced Editora
Comments	Caption	Exemplo do livro Delphi 4

Veja como fica o formulário:

4) Salve o novo formulário, através do menu *File*, opção *Save*. Dê a este formulário o nome UnitSobre.

5) Salve o projeto, através do menu *File*, opção *Save All*.

6) Visualize o Project Manager, através do menu *View*, opção *Project Manager*. Note que os dois formulários estão listados:

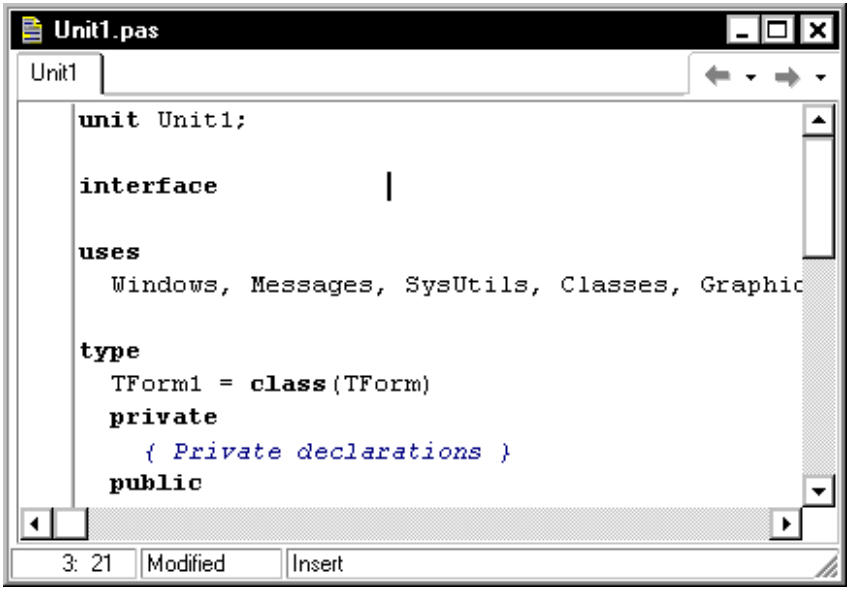

Project Manager
New
Remove
Activate
Files
Path
ProjectGroup1
C:\Arquivos de Programas\Bc
SAG.exe
C:\Adelize\Livros Atuais\Delp
UnitPrincipal
C:\Adelize\Livros Atuais\Delp
UnitSobre
C:\Adelize\Livros Atuais\Delp

5

Iniciando a Construção do Código

Unit

Como vimos, a cada formulário está associada uma janela contendo código, chamada Unit:



A Unit conterá declarações de variáveis, subrotinas e funções referentes ao formulário e aos componentes inseridos dentro do formulário.

Mesmo que não escrevamos nenhuma linha de código adicional (como foi o caso dos exemplos feitos até agora), o Delphi acrescenta umas linhas de código default na Unit, essenciais para que a aplicação execute.

Estas linhas de código estão em Object Pascal, que é a linguagem de programação base do Delphi. Você não precisa ser um exímio programador em Pascal para criar aplicações em Delphi. Isto porque o Delphi já acrescenta as linhas de código necessárias quando você acrescenta um componente a seu formulário. Você só precisa escrever linhas de código caso sua aplicação responda a algum evento.

Orientação a Objetos

Por ser baseado no Object Pascal, o Delphi permite que se construa aplicações orientadas a objetos.

Em linhas gerais, aplicações orientadas a objetos se baseiam no conceito de *classe*. A Classe é um tipo de dado, contendo atributos e serviços. O *objeto* é uma variável de determinada classe.

Por exemplo, um formulário nada mais é do que um objeto da classe Formulário (ou, para usar a terminologia do Delphi, da classe TForm). Contém atributos (são os componentes inseridos neste) e serviços (são os métodos, funções pré-definidas que serão vistas mais adiante).

Os componentes também são objetos de determinada classe. Por exemplo, o componente Edit é um objeto da classe TEdit, possuindo atributos e serviços próprios.

O Delphi já oferece diversas classes (como TForm, que representa os formulários, TButton, que representa os botões, etc), que dão origem aos objetos, mas você pode criar novas classes. Nos exemplos utilizados neste livro, são utilizadas as classes já oferecidas pelo Delphi.

Não se assuste! Você não precisa dominar a metodologia de programação orientada a objetos para desenvolver uma aplicação em Delphi. O uso das classes já oferecidas pelo Delphi é automático, como veremos adiante.

Além das classes relacionadas a componentes, o Delphi também oferece outras classes:

- a classe TApplication possui atributos e serviços que dizem respeito a aplicação como um todo;
- a classe TScreen representa a tela da aplicação;
- a classe TPrinter representa a impressora.

Estrutura da Unit

Uma Unit possui uma estrutura padrão, com diversas seções e cláusulas:

```
unit  nome_da_unit

interface
   uses Windows, Messages, SysUtils, Classes,
     Graphics, Controls, Forms, Dialogs, ...;

   type
   ...
   const
   ...
   var
   ...

implementation
   uses
   ...
   type
   ...
   const
   ...
   var
   ....

initialization


finalization


end.
```

Cabeçalho da Unit

Declara o nome da unit. Quando você inicia um projeto, o Delphi oferece uma unit chamada Unit1. Você pode mudar seu nome a partir do momento que salvar o projeto, ou a própria unit. O nome dado ao arquivo .PAS será o nome da unit que aparecerá no cabeçalho da unit, e na aba abaixo do formulário.

Seção Interface

Nesta seção são declaradas outras units, funções, procedures, variáveis e constantes de uso público. Ou seja, os objetos declarados aqui poderão ser acessados por outras units da aplicação, ou mesmo por outras aplicações.

A seção Interface possui as seguintes cláusulas:

Cláusula Uses

Contém as units que são acessadas por esta unit.

No início do projeto, o Delphi já inclui no corpo do programa as units padrões (Windows, Messages, etc). Quando você inclui um componente no formulário, o Delphi também automaticamente insere a unit em que ele está declarado.

Mas algumas vezes você mesmo tem que incluir units na cláusula uses: quando seu formulário chama outro formulário, a unit referente a este último deve ser acrescentada a claúsula uses.

Cláusula Type

Declara variáveis de tipos definidos pelo usuário (data types).

Por default, o Delphi sempre cria nesta seção um objeto TForm1 da classe TForm. Conforme você vai acrescentando componentes a seu formulário, estes vão sendo declarados nesta seção, como atributos do objeto TForm1. Da mesma forma, conforme você vai chamando procedures de eventos, estas vão sendo declaradas nesta seção, como serviços do objeto TForm1.

Não se preocupe. O Delphi faz isso automaticamente para você. Quando você apaga um componente do formulário,

o Delphi apaga automaticamente as declarações que criaram o objeto.

Cláusula Const

Declara as constantes que poderão ser acessadas por outras units.

Cláusula Var

Declara as variáveis utilizadas pela aplicação que poderão ser acessadas por outras units. O Delphi já declara automaticamente uma variável com o nome do seu formulário, do tipo TForm1.

Seção Implementation

Contém os corpos das funções e procedures declaradas na seção Interface.

Na seção Implementation você pode declarar units, variáveis, constantes, funções e procedures de uso privado daquela unit. Da mesma forma que na seção Interface, estas declarações são feitas nas cláusulas uses, var, type e const.

Um alerta. Como vimos, se sua unitA utilizar dados de uma unitB, você deverá declarar a unitB na cláusula uses da seção interface da unitA. Mas se a unitB também utilizar dados da unit A, e você declarar a unitA na cláusula uses da seção interface da unitB, ocorrerá um erro de referencia circular. Para evitar isto, faça a declaração da unitA na cláusula uses da seção *implementation* da unitB.

Seção Initialization

Inicializa variáveis. Quando sua aplicação é iniciada,

todas as variáveis inicializadas na seção Initialization são jogadas para a memória. (Esta seção é opcional).

Seção Finalization

A seção Finalization é a contra partida da seção Initialization e é lida quando a aplicação termina. (Esta seção é opcional).

Exemplo de Unit

Quando você inclui um formulário em branco em sua aplicação, a unit associada a este formulário já possui o seguinte código:

```
unit Unit1;

interface

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

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

var
  Form1: TForm1;

implementation

{$R *.DFM}

end.
```

Quando você acrescenta em seu formulário um componente ListBox e um componente Button, sua unit é automaticamente atualizada, incluindo a criação dos

objetos ListBox1 e Button1 na seção interface, cláusula type:

```
unit Unit1;

interface

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

type
     TForm1 = class(TForm)
     Button1: TButton;
     ListBox1: TListBox;
  private
      { Private declarations }
  public
      { Public declarations }
  end;

var
   Form1: TForm1;

implementation

{$R *.DFM}

end.
```

O Editor de Código

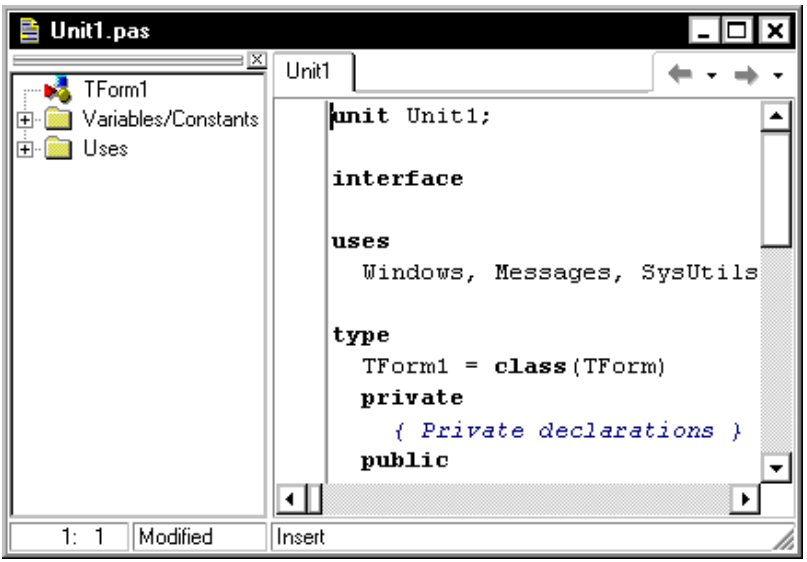
As linhas de código são inseridas na Unit através do Editor de Código.

Entrando no Editor de Código

O Editor de Código é chamado ao se clicar sobre a barra de *status*, na parte inferior do formulário, ou através do menu *View*, opção *Toggle Form/Unit,* que alterna entre o formulário e a unit.

Você pode também clicar sobre o ícone  , da Barra de Ferramentas, para alternar entre a unit ligada ao formulário, e o formulário.

Entramos, então, no Editor de Código:



Adicionando Linhas de Código

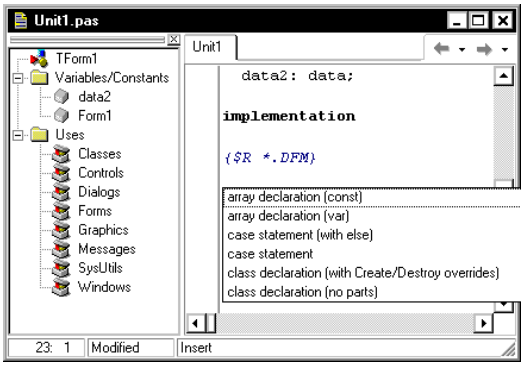
Para adicionar linhas de código, basta clicar sobre o local onde a linha será inserida e digitá-las.

Se você quiser inserir comentários, use as chaves { } entre os comentários. Na tela anterior a expressão *Private declarations* é um comentário, pois está entre chaves.

No painel na parte inferior do Editor de Código o Delphi oferece algumas mensagens de erro, como quando, por exemplo, esquecemos de colocar o ponto e vírgula (;) após a linha.

Entre os recursos que o Delphi oferece para facilitar a edição do código temos:

Code Template Wizard: um assistente para comandos comuns de programação que você pode rapidamente inserir no seu código. Para acessar modelos de código no Editor, pressione Ctrl+J:



novo! *AppBrowser:* utilitário que permite que você navegue por todos os elementos do código, através de hiperlinks automáticos. Por exemplo, se você criou em uma unit uma função de nome Calcula, e quer utilizá-la em outra unit, e precisa verificar sua declaração, basta apertar a tecla Ctrl e dara um clique sobre o nome da função. Automaticamente a unit onde ela foi implementada é aberta, com o cursor do mouse sobre o cabeçalho desta.

Code Completion Wizard: com este assistente, após digitar um objeto ou nome de classe, seguido de um ponto, é exibida uma lista pop-up dos métodos, propriedades e eventos válidos. Se digitar um comando de atribuição, basta pressionar Ctrl+Espaço para mostrar uma lista pop-up de argumentos para a variável.

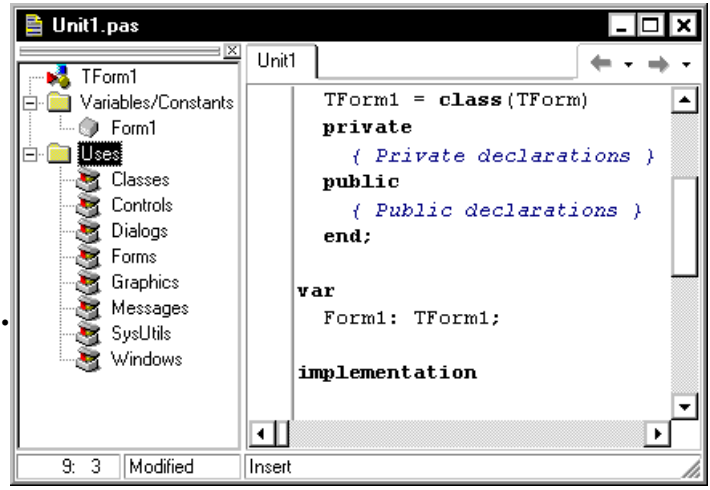
novo! *Tooltip Symbol Insight:* se você quiser verificar o tipo de determinado identificador e a unit onde este foi declarado, basta passar o mouse sobre o identificador no editor de código. A informação aparece em uma pequena janela.

Code Parameter Wizard: com este assistente, quando você digita um método seguido de um parênteses, um menu pop-up mostra uma lista dos argumentos requeridos para aquele método.

novo! *ClassCompletion Wizard*: assistente de criação de classes que automatiza a definição de novas classes ao gerar um esqueleto de código para os membros da classe que você declarar.

O Explorador de Código (Code Explorer)

novo! O Code Explorer é um utilitário atracado à esquerda do Editor de Código e mostra, em forma de árvore, os tipos de dados, propriedades, métodos, variáveis e rotinas globais definidas na unit ativa, além das outras units listadas na cláusula uses:



Para alternar entre o Code Explorer e o Editor de Código, pressione Ctrl+Shift+E.

Dica: Para procurar por uma classe, propriedade, método, variável ou rotina no Explorador de Código, basta digitar seu nome.

Quando você dá um duplo clique sobre um item da árvore do Code Explorer, o cursor se move para a implementação do mesmo no Editor de Código. Para adicionar ou renomear um item, clique com o botão direito do mouse sobre o nó apropriado e escolha respectivamente *New* ou *Rename* no menu que aparece.

Saindo do Editor de Código

Para sair do Editor de Código e voltar ao formulário basta clicar sobre o formulário, ou sobre o ícone  , da Barra de Ferramentas, acessando o formulário ligado a Unit.

Cuidado! Se você fechar o Editor de Código, dando um duplo clique sobre seu botão de controle, você encerrará o Delphi. Se não tiver salvo sua aplicação, o Delphi pedirá para salvá-la.

Eventos

Os programas desenvolvidos em Delphi são orientados a eventos. Eventos são ações normalmente geradas pelo usuário: clicar o mouse, apertar uma tecla, etc. Os eventos podem ser também gerados pelo Windows.

Existem eventos associados ao formulário, e a cada componente inserido neste. Por exemplo:

- ao formulário está ligado o evento onCreate, que ocorre quando mostramos o formulário na tela;
- ao componente botão está ligado o evento onClick, que ocorre quando damos um clique com o mouse sobre o botão.

Eventos Comuns a Formulários e Componentes

Alguns eventos, ligados tanto a formulários quanto a componentes, estão listados a seguir:

OnClick: ocorre quando o usuário clica no objeto.

OnDblClick: ocorre quando o usuário dá um duplo clique no objeto.

OnKeyDown: ocorre quando o usuário pressiona uma tecla enquanto o objeto tem o foco.

OnKeyUp: ocorre quando o usuário solta uma tecla enquanto o objeto tem o foco.

OnKeyPress: ocorre quando o usuário dá um clique em uma tecla ANSI.

OnMouseDown: ocorre quando o usuário pressiona o botão do mouse.

OnMouseUp: ocorre quando o usuário solta o botão do mouse.

OnMouseMove: ocorre quando o usuário move o ponteiro do mouse sobre o objeto.

Respondendo a Eventos

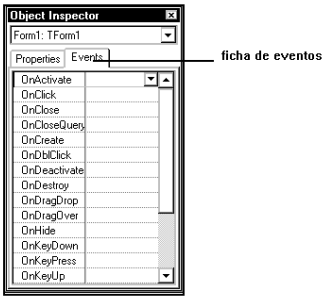
Cada evento está associado a uma procedure, aonde você deve inserir as linhas de código que envolvem este evento. Por exemplo, o evento OnClick, que é gerado ao clicarmos no botão BtnCalcula, cria a procedure:

```
procedure TForm1.BtnCalcula Click(Sender
     : TObject)
begin

end;
```

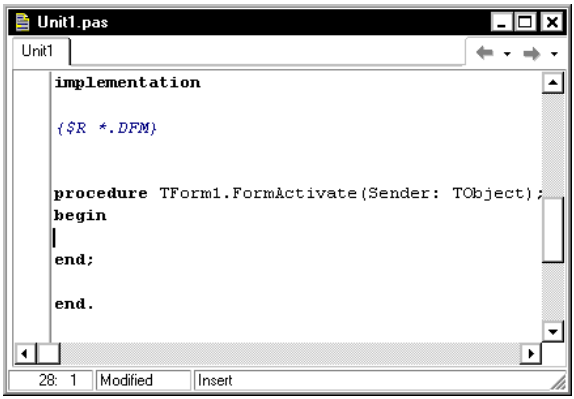
onde *TForm1* é o objeto TForm que contém o botão BtnCalcula, e *Sender* é um objeto TObject que representa o componente que deu origem ao evento.

Se você quiser inserir uma rotina que trate um determinado evento de um componente, siga os seguintes passos:

- clique sobre o componente;
- no Object Inspector, selecione a página *Events*, clicando sobre a ficha correspondente:



- dê um duplo clique sobre o evento para o qual você quer inserir o código (clique na coluna de valores deste evento);
- você entra no Editor de Código, onde é inserida a procedure correspondente ao evento, dentro da seção implementation da unit. Por exemplo:



- escreva as linhas de código dentro desta procedure, entre as palavras *begin* e *end*.

Normalmente, um evento dá origem a:

- uma mudança em propriedades de determinados componentes. Por exemplo, ao evento clicar em um botão *Limpa*, podemos modificar a propriedade Text de um componente Edit para vazia;

- uma chamada para um método, associado a algum componente. Por exemplo, ao darmos Enter após digitarmos um texto em uma caixa de edição, o programa pode executar o método Add em um ListBox, acrescentando o texto digitado à lista;

Veremos cada um destes tópicos posteriormente. É importante ressaltar que um evento também pode dar origem a um novo evento. Além disso, um evento também pode gerar uma chamada para uma função definida pelo programador.

Alteração de Propriedades como Resposta a Eventos

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

componente.propriedade

Por exemplo, para modificar a propriedade Text de uma caixa de edição Edit1 para 'Bom Dia', faça:

```
Edit1.Text := 'Bom Dia';
```

Dica: Ao terminar uma linha de comando, você deve acrescentar um ponto e vírgula (;)

Se a propriedade do componente possuir sub-propriedades, para acessá-la, utilize a seguinte sintaxe:

componente.propriedade.sub-propriedade

Por exemplo, para modificar a sub-propriedade Name, referente a propriedade Font, de uma caixa de edição Edit1, para Arial, faça:

```
Edit1.Font.Name := 'Arial';
```

Métodos

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

Alguns métodos são descritos a seguir:

Show: mostra um formulário.

Hide: esconde um formulário.

Print: imprime o conteúdo de um formulário na impressora.

SetFocus: estabelece o foco para um formulário ou componente.

Chamada de Métodos como Resposta a Eventos

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

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

componente.método

Por exemplo, clicar em um botão pode dar origem ao evento Show de um outro formulário Form2, mostrando este novo formulário na tela:

```
Form2.Show;
```

Eventos Compartilhados por Diversos Componentes

Muitas vezes, diferentes componentes podem dar origem ao mesmo evento. Por exemplo, em uma calculadora, o evento Click gera um código semelhante para cada botão clicado.

O Delphi oferece um atalho para que você não precise escrever código para cada um destes componentes separadamente.

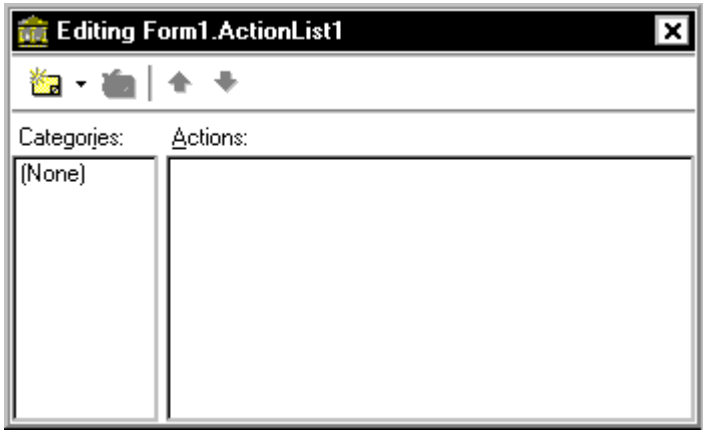
Para compartilhar um evento entre componentes:

- escreva o código para o primeiro componente;
- clique no segundo componente que compartilhará o código, e acesse a página *Events* do Object Inspector;
- na coluna do evento que será compartilhado, clique sobre a seta a direita da coluna, e escolha a procedure referente ao evento do primeiro componente;
- repita esta operação para os demais componentes que compartilharão o código do primeiro componente.

O Componente Action Lists

O componente Action Lists permite que você centralize a resposta a comandos do usuários, oferecendo controle sobre componentes que respondem a comandos de botões e menus.

Após adicionar um componente Action List – que fica na página Standard da paleta de componentes - em seu formulário ou data module, dê um duplo clique para mostrar o editor de ações, que permite que você adicione, exclua e rearrume ações:



Para inserir uma ação, clique com o botão direito do mouse e escolha *New Action* ou *New Standard Action*. No Object Inspector, determine as propriedades para cada ação. A propriedade Name identifica a ação, e as demais propriedades correspondem as propriedades dos controles clientes.

Para excluir uma ação da lista, clique sobre ela com o botão direito e escolha a opção *Delete*.

Diversos controles, como TToolButton, TSpeedButton, TMenuItem e TButton possuem uma propriedade chamada Action. Quando você estabelece esta propriedade para uma das ações existentes em seu componente Action List, os valores das propriedades da ação são copiadas para o controle. Todas as propriedades e eventos em comum com a ação são dinamicamente ligadas ao controle. Por exemplo, suponha que você possua um botão e um item de menu para salvar. Quando clicados, chamariam o método Save. Você faria:

```
Ao clicar botão/menu
   Salvar
```

Agora, você cria uma procedure chamada Salvar, contendo código para salvar. Em um Action List, cria uma ação de

nome SalvarCmd, e associa ao evento onExecute desta procedure Salvar. Depois, no botão e item de menu, associa a propriedade Action a ação SalvarCmd. Note que automaticamente o Delphi associa ao evento onClick do botão ou item do menu a procedure Salvar.

Note que, a partir do momento em que você associa a propriedade Action de um controle a um objeto Taction, você pode modificar o valor de uma determinada propriedade deste TAction e automaticamente a propriedade respectiva do controle também se modifica. Por exemplo, se você quiser desabilitar determinada ação, basta fazer:

```
TAction1.Enabled := False
```

Automaticamente todos os controles cuja propriedade Action seja igual a TAction1 ficarão desabilitados.

Compilando e Executando a Aplicação

Como vimos, para executar sua aplicação (ou seja, compilá-la e ligá-la às bibliotecas necessárias), escolha a opção *Run* do menu *Run*, ou clique no ícone  .

Quando quiser voltar ao modo de projeto (design mode, o modo de construção do programa), basta fechar o formulário, dando um duplo clique no menu de controle. Ou, caso não consiga fechar a aplicação por ocorrer algum erro, utilize a opção *Program Reset* do menu *Run*.

Normalmente, se durante a execução do programa ocorrer algum erro, o Delphi mostra a unit onde ocorreu o erro. Você pode consertar o código e mandar executar a aplicação novamente.

O Delphi já cria automaticamente um arquivo *.Exe*, quando você manda executar sua aplicação. Este arquivo possui o mesmo nome do projeto.

O menu *Project* oferece também algumas opções:

Compile: compila somente as units alteradas.

Build: recompila todas as units, alteradas ou não.

Sintax Check: faz uma verificação no código, procurando por erros de sintaxe.

Information: oferece informações sobre a aplicação, como o número de linhas e o tamanho do código.

Delphi 4

Sistema de Acompanhamento Geográfico

Parte III

O Código do Formulário Principal

Vamos construir o código para o formulário principal. Este código estará ao botão Sair, e fechará a aplicação.

Para isso, siga os passos abaixo:

1) Abra o projeto *SAG.dpr*.

2) No Project Manager, selecione a unit UnitPrincipal, e visualize o formulário.

3) Selecione o botão Sair, e, no Object Inspector, vá para a página Events.

4) Dê um duplo clique sobre o evento OnClick. Você entra no editor de código, com a estrutura da procedure relativa ao evento Click já construída:

```
procedure TfrmPrincipal.BtnSairClick(Sender:
TObject);
begin

end;
```

5) Insira então o código para fechar a aplicação, através do método Close do formulário:

```
procedure TfrmPrincipal.BtnSairClick(Sender:
TObject);
begin
     FrmPrincipal.Close;
end;
```

6) Salve o projeto, através do menu File, opção Save All, e execute-o, através do menu *Run*, opção *Run*.

7) Clique no botão Sair, e a aplicação é encerrada.

Delphi 4

6

Visão Geral do Object Pascal

Neste capítulo vamos mostrar conceitos básicos da linguagem Object Pascal, na qual se baseia o Delphi. Com estes conceitos, como tipos de dados, variáveis e constantes, operadores, você poderá construir o código de sua aplicação.

Tipos de Dados

Tipos Simples

Definem conjuntos de valores ordenados. Se dividem em dois grupos: ordinais e reais.

Entre os tipos ordinais temos:

Integer: representa um subconjunto de todos os números. Veja os tipos de dados integer oferecidos pelo Object Pascal:

Tipo	Região de Valores	Espaço na memória
Integer	–2147483648..2147483647	signed 32-bit
Cardinal	0..4294967295	unsigned 32-bit
Shortint	–128..127	signed 8-bit
Smallint	–32768..32767	signed 16-bit
Longint	–2147483648..2147483647	signed 32-bit
Int64	–2^63..2^63–1	signed 64-bit
Byte	0..255	unsigned 8-bit
Word	0..65535	unsigned 16-bit
Longword	0..4294967295	unsigned 32-bit

Char: assume caracteres do teclado como valores.

Boolean: assume apenas True (verdadeiro) e False (falso) como valores.

Enumerated: define um conjunto ordenado de valores, como os dias da semana, nomes, etc.

Subrange: definem um subconjunto de valores de um outro conjunto ordinal.

O tipo real define um conjunto de números que podem ser representado com notação de pontos flutuantes.Entre os tipos reais temos:

Tipo	Região	Dígitos Significativos	Tamanho
Real48	2.9 x 10^–39 .. 1.7 x 10^3811–12		6 bytes
Single	1.5 x 10^–45 .. 3.4 x 10^387–8		4 bytes
Double	5.0 x 10^–324 .. 1.7 x 10^308	15–16	8 bytes
Extended	3.6 x 10^–4951 .. 1.1 x 10^4932	19–20	10 bytes
Comp	–2^63+1 .. 2^63 –1	19–20	8 bytes
Currency	–922337203685477.5808 .. 922337203685477.5807	19–20	8 bytes

Tipo String

É formado por um conjunto de elementos do tipo Char. Existem os seguintes tipos de strings:

Tipo	Tamanho Máx.	Memória	Uso
ShortString	255 caracteres	2 a 256 bytes	compatibilidade com versões antigas
AnsiString	~2^31 caracteres	4 bytes a 2GB	É o padrão
WideString	~2^30 caracteres	4 bytes a 2GB	Servidores COM

A palavra reservada **string** funciona como um tipo genérico de string.

Outros Tipos de Dados

Além dos tipos de dados do Object Pascal, o Delphi oferece outros tipos de dados, como *TDateTime:* usado para armazenar data e hora.

O Delphi permite que você crie seus próprios tipos de dados através da palavra reservada *type*.

Variáveis

Declarando Variáveis

Você precisa dar a uma variável tanto um nome quanto um tipo. Os nomes de variáveis podem ter até 40 caracteres, e precisam começar com uma letra.

As variávies são declaradas na cláusula **var** da unit.

Uma vez que você tenha determinado o nome da variável, poderá declarar seu tipo da seguinte forma:

```
var
   nome: string;
   num: integer;
   x,y: real;
```

Atribuindo Valores a Variáveis

Para atribuir valor a uma variável, você utiliza o operador de atribuição (:=). Por exemplo, para atribuirmos o valor 4 a uma variável *num*, faríamos:

```
num := 4;
```

Variáveis globais (que serão vistas a seguir) podem ser declaradas e inicializadas simultaneamente:

```
var
   nome: string = 'Adelize';
```

Variáveis Globais

Variáveis globais são declaradas fora de sub-rotinas e funções, e podem ser usadas por todas as sub-rotinas, funções e métodos da unit, ou mesmo por outras units da aplicação.

Para criar variáveis globais, você deve declará-las na cláusula var da seção *interface* da Unit:

```
interface

(...)

var
   nome: string;
```

Caso a sua unit utilize uma variável global declarada na Unit2, por exemplo, você deve acrescentar a Unit2 na lista de units usadas, na cláusula *uses* da seção *interface*:

```
interface

uses
   Windows,..., Unit2
```

Você pode declarar variáveis globais somente para a unit onde ela é criada, ou seja, variáveis que podem ser usadas por todas as procedures, funções e métodos da unit, mas não por outras units. Para declarar variáveis com escopo da unit, faça a declaração na cláusula *var* da seção *implementation* da unit:

```
implementation

var
   nome: string;
```

Variáveis Locais

São declaradas dentro de sub-rotinas, funções e métodos. Só são válidas dentro do bloco, e são destruídas quando a sub-rotina ou função termina.

Para criar variáveis locais, crie uma cláusula *var* dentro da sub-rotina ou função:

```
procedure Calculo;
var
   A : integer;
begin
   A := 4;
end;
```

Constantes

São identificadores que não alteram seu valor durante a execução do programa. Podem ser dos mesmos tipos descritos para variáveis, sendo declarados na cláusula *const*:

```
const
   num = 350;
   nome = 'Delphi';
```

Note que:

- você usa o operador = para atribuir valores à constantes;
- para trabalhar com constantes do tipo string, você deverá colocar seu conteúdo entre aspas simples.
- as demais constantes serão classificadas de acordo com o que foi digitado. Por exemplo, um número 6500000000 é muito grande para ser do tipo integer, logo só pode ser do tipo longint.

Palavras Reservadas

As palavras abaixo são reservadas pelo ObjectPascal, não podendo ser usadas para nomes de variáveis ou constantes:

and	as
asm	array
begin	case
class	const
constructor	destructor

dispinterface	div
do	downto
else	end
except	exports
file	finalization
finally	for
function	goto
if	implementation
in	inherited
inline	initialization
interface	is
label	library
mod	nil
not	object
of	or
packed	procedure
program	property
raise	record
repeat	set
shl	shr
string	then
to	try
type	unit
until	uses
var	while
with	xor

Operadores e Expressões

Uma expressão pode ser uma simples constante ou variável, ou a combinação de variáveis e constantes usando operadores.

Os operadores podem ser: aritméticos, relacionais, lógicos, string e funcionais.

aritméticos: +, -, *, /, ^, div, mod

relacionais: =, <, >, <=, >=, <>

lógicos: and, or, not, xor (para or exclusivos)

de strings: o sinal de + concatena strings. Por exemplo, em:

```
nome1 := 'Rosa';
nome2 := ' Souza';
nome3 := nome1 + nome2;
```

o resultado de nome3 será Rosa Souza.

funcionais: são aqueles que chamam funções. Por exemplo:

numero := abs(num);

faz com que à variável *numero* seja igual ao valor absoluto da variavel *num*.

Criando Procedures e Funções

Você pode criar suas próprias procedures e funções, para organizar o código da aplicação de forma a facilitar sua leitura e manutenção, ou mesmo para reaproveitar procedures e funções em outras aplicações.

A diferença principal entre procedures e funções é que funções retornam valores, ao passo que procedures nada retornam.

Uma procedure ou função é composta de duas partes:

- um cabeçalho;
- um bloco de código.

Criando o Cabeçalho da Procedure ou Função

O cabeçalho identifica a procedure ou função e lista os parâmetros que a rotina usa, se usar. Uma procedure começa com a palavra reservada *procedure*. Uma função inicia com a palavra reservada *function*.

Os parâmetros são listados entre parênteses. Cada parâmetro possui um identificador e um tipo, e são separados entre si por um ponto e vírgula (;):

```
procedure ValidaEntrada(num: integer; nome: string);
```

As funções possuem também um tipo para o valor de retorno. Este tipo é especificado no final do cabeçalho, separado deste por um sinal de dois pontos (:):

```
function CalculaSoma(num1, num2: integer):longint;
```

Escrevendo o Corpo da Procedure ou Função

O corpo da procedure ou função pode conter declarações de variáveis, constantes ou novos tipos de dados.

A parte dos comandos inicia com a palavra reservada *begin* e termina com a palavra reservada *end*.

```
procedure ValidaEntrada(num: integer; nome:
   string);
begin
   if (num =0) or (nome = ' ') then
     ShowMessage('Entrada Inválida!');
end;
```

Se for uma função, o valor de retorno desta pode ser atribuído à uma variável com o nome da função, ou à variável *Result*.

```
function CalculaSoma(num1, num2: integer):
   longint;
begin
   CalculaSoma := num1 + num2;
end;
```

Posicionando a Procedure ou Função no Código

Se você quiser que a procedure ou função seja acessável por outra unit além daquela onde está declarada, coloque o cabeçalho da procedure ou função na seção *interface* da unit, e o corpo na seção *implementation*:

```
unit Unit1

interface

uses
...

type
....

var
...

procedure ValidaEntrada(num: integer; nome:
string);

implementation

procedure ValidaEntrada(num: integer; nome:
string);
begin
   if (num =0) or (nome = ' ') then
     ShowMessage('Entrada Inválida!');
end;

end.
```

Chamando a Procedure ou Função

Para chamar a procedure ou função, basta digitar o nome da procedure ou função, seguida dos parâmetros, caso haja:

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

Comandos Condicionais

Com um comando condicional, você poderá forçar o programa a executar um bloco de operações de acordo com uma condição.

Dica: Para digitar a sintaxe de comandos condicionais corretamente, basta, no editor de código, teclar Ctrl+J, e escolher o comando desejado.

If ... then

Existem dois tipos de sintaxes para o comando if...then:

```
if expressão then comandos1;
```

ou

```
if expressão then comandos1 else comandos2;
```

O exemplo abaixo estabelece um valor para a variável *nome*, de acordo com o valor da variável *opcao*:

```
if  opcao = 1 then
   nome := 'Alyssa'
else
   nome := 'Jonata';
```

Não coloque um ponto e vírgula (;) antes de um comando *else*.

Se o bloco de comandos a ser realizado contiver mais de uma linha, deverá estar contido entre as palavras reservadas begin e end:

```
if  opcao = 1 then
   begin
     nome := 'Alyssa';
     idade := 10;
   end
else
   nome := 'Jonata';
```

Case ... of

Existem dois tipos de sintaxes para o comando case...of:

```
case expressão of
     sequencias de constantes 1: comando1;
     sequencias de constantes 2: comando2;
end;
```

ou

```
case expressão of
     sequencias de constantes 1: comando1;
     sequencias de constantes 2: comando2;
else
     sequencia de comandos 3;
end;
```

Note que *expressão* deve ser do tipo ordinal (integer, char, boolean, etc).

O exemplo abaixo determina um valor para o componente TxtNome, de acordo com o valor da variável Tecla:

```
case Tecla of
   13 : TxtNome.Text := 'Serial';
   14..16 : TxtNome.Text := 'Advanced';
else
   TxtNome.Text := 'Delphi';
end;
```

Se o bloco de comandos a ser realizado contiver mais de uma linha, deverá estar contido entre as palavras reservadas begin e end:

```
case Tecla of
   13 :
   begin
     TxtNome.Text := 'Serial';
     TxtCEP.Text := '88080-700';
   end;
   14..16 : TxtNome.Text := 'Advanced';
else
   TxtNome.Text := 'Delphi';
end;
```

Comandos de Laços

Laços permitem que você execute uma sequência de

comandos repetidamente, usando uma condição de controle ou variável para determinar quando a execução termina. O Object Pascal possui três tipos de comandos de laço: repeat, while e for.

Dica: Para digitar a sintaxe de comandos de laço corretamente, basta, no editor de código, teclar Ctrl+J, e escolher o comando desejado.

While ... do

Este laço repete um bloco de comandos enquanto determinada condição for verdadeira.

Sintaxe:

```
while expressão do
     comando;
```

O exemplo abaixo soma 1 a variável total, enquanto a variável num for diferente de 0:

```
while num<>0 do
   total := total + 1;
```

Se o bloco de comandos a ser realizado contiver mais de uma linha, deverá estar contido entre as palavras reservadas *begin* e *end*:

```
while num<>0 do
begin
   total := total + 1;
   num := num +1;
end;
```

Você pode usar os comandos *Break* e *Continue* para controlar o fluxo do comando while. O comando Break termina o comando no qual ele é chamado, enquanto o comando Continue executa o próximo comando.

For ... do

Com essa estrutura, você poderá repetir um bloco de comandos um número determinado de vêzes.

Sintaxe:

```
for var1 := expressão1 to expressão2 do
     comando;
```

Note que var1 tem que ser do tipo ordinal.

Dica: Ao invés da palavra to, você pode usar a palavra downto. Com downto, você parará no passo anterior a expressão2.

O exemplo abaixo incrementa a variável *Valor*, enquanto a variável *i* estiver no intervalo de 1 a 100:

```
For i:=1 to 100 do
   Valor := Valor + i;
```

Se o bloco de comandos a ser realizado contiver mais de uma linha, deverá estar contido entre as palavras reservadas *begin* e *end*.

Além disso, você pode usar os comandos *Break* e *Continue* para controlar o fluxo do laço *for*.

Repeat ... until

Laço usado para se executar um grupo de comandos repetidas vezes, até determinada condição.

Sintaxe:

```
repeat
   comando1;
   comando2;
until condição;
```

Comandos de Manipulação de Strings

O Delphi oferece diversas funções para manipulação de strings:

Comando	Descrição
Copy	Retira um substring do meio de uma string.
Length	Determina a largura da string.
Pos	Permite que se procure um determinado substring em um string.
UpperCase	Substitui todas as letras minúsculas de uma string por letras maiúsculas.
LowerCase	Substitui todas as letras maiúsculas de uma string por letras minúsculas.
IntToStr	Converte um número do tipo integer para uma string.
FloatToStr	Converte um número do tipo real em uma string.
StrToInt	Converte uma string representando um número integer em um número.
StrToFloat	Converte uma string representando um número real em um número do tipo extended.
Str	Converte um valor integer ou real para uma string, armazenando o resultado em uma variável.
Val	Retira o valor numérico de um string.

Comandos de Manipulação de Data e Hora

O Delphi oferece inúmeras funções para manipulação de data e hora:

Comando	Descrição
DateTimeToStr	Converte uma variável do tipo TDateTime em uma string.
DateToStr	Converte uma variável do tipo TDateTime em uma string.

TimeToStr	Converte uma variável do tipo TDateTime em uma string.
StrToDate	Convete uma string em um valor do tipo TDateTime.
Date	Retorna a data corrente.
Time	Retorna a hora corrente.
Now	Retorna a data e hora corrente.

Delphi 4

Sistema de Acompanhamento Geográfico

Parte IV

O Formulário de Menu

Vamos acrescentar ao Sistema de Acompanhamento Geográfico um novo formulário, que conterá o menu de nossa aplicação. No momento, acrescentaremos apenas o formulário, sem o menu.

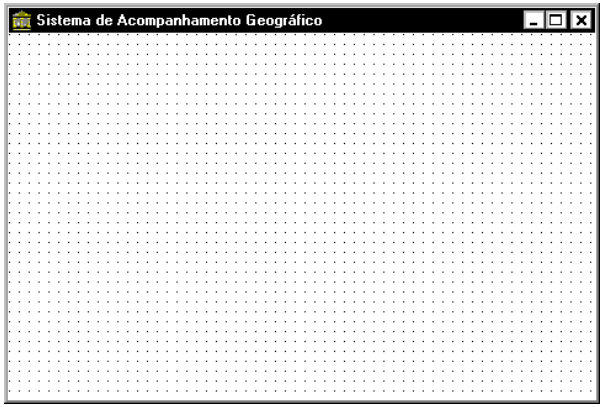
Para isso, siga os passos abaixo:

1) Abra o projeto SAG.dpr.

2) No menu *File*, escolha a opção *New Form*. Um novo formulário é adicionado ao projeto.

3) Modifique as seguintes propriedades:

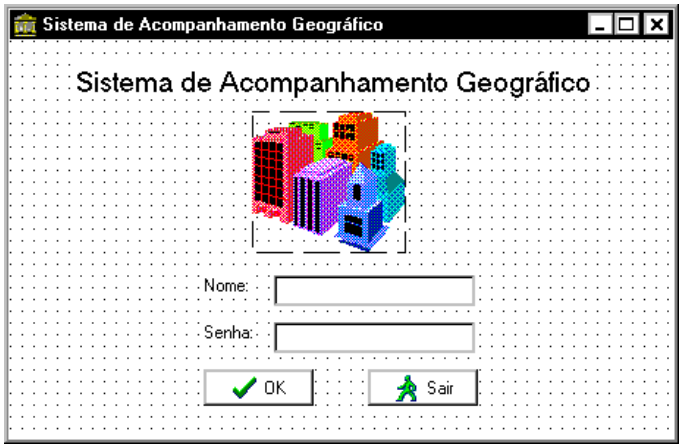
Propriedade	Valor
Caption	Sistema de Acompanhamento Geográfico
Name	frmMenu
Position	poScreenCenter



4) Salve o projeto, através do menu *File* opção *Save All*. Dê o nome de *UnitMenu* para o novo formulário.

O Código do Botão OK do Formulário Principal

Para acessarmos o formulário de menu criado acima, é necessário que entremos com um nome e senha no formulário principal, clicando então no botão OK:



Vamos criar um código que avalia a senha digitada, ao clicar do botão OK, e o número de vezes que se tentou entrar no aplicativo.

Para isso, siga os passos abaixo:

1) Através do Project Manager, ative o formulário principal, de nome UnitPrincipal.

2) Dê um duplo clique no botão OK. Você entra no editor de código, com a procedure referente ao evento Click já pré-construída:

```
procedure TfrmPrincipal.BtnOKClick(Sender:
TObject);
begin

end;
```

3) Entre então com o seguinte código:

```
procedure TfrmPrincipal.BtnOKClick(Sender:
       TObject);
var
   senha : string;
begin
     senha := EditSenha.Text;
     if (senha <> 'ADV') then
        ShowMessage('Senha inválida!')
     else
        frmMenu.Show;
     VerificaNumeroTentativas;
end;
```

4) Inicialmente criamos uma cláusula *var* dentro da procedure, e declaramos a variável *senha*, do tipo string. Depois, atribuímos à variável senha o valor entrado na caixa de edição EditSenha. Avaliamos então a variável senha. Se for diferente de ADV (senha sugerida), uma mensagem é exibida para o usuário, através do comando *ShowMessage* (que será visto com mais detalhes posteriormente). Caso o usuário entre com a senha ADV, o formulário de menu (frmMenu) é mostrado, através do método *Show*.

5) Após a avaliação da senha, chamamos a procedure VerificaNumeroTentativas, que, como o nome diz, verifica quantas vezes o usuário já tentou acertar a senha para entrar no aplicativo. O usuário só poderá errar três vezes. No quarto erro, aparece uma mensagem, e o aplicativo é fechado. Para construir a procedure VerificaNumeroTentativas, digite seu cabeçalho na seção interface do formulário:

```
interface

(...)

procedure VerificaNumeroTentativas;

var
  frmPrincipal: TfrmPrincipal;
```

6) Vamos criar na seção implementation uma cláusula var, para declarar a variável *NumTentativas*, do tipo integer, inicializando-a com o valor 0:

```
implementation

{$R *.DFM}

var
   NumTentativas : integer = 0;
```

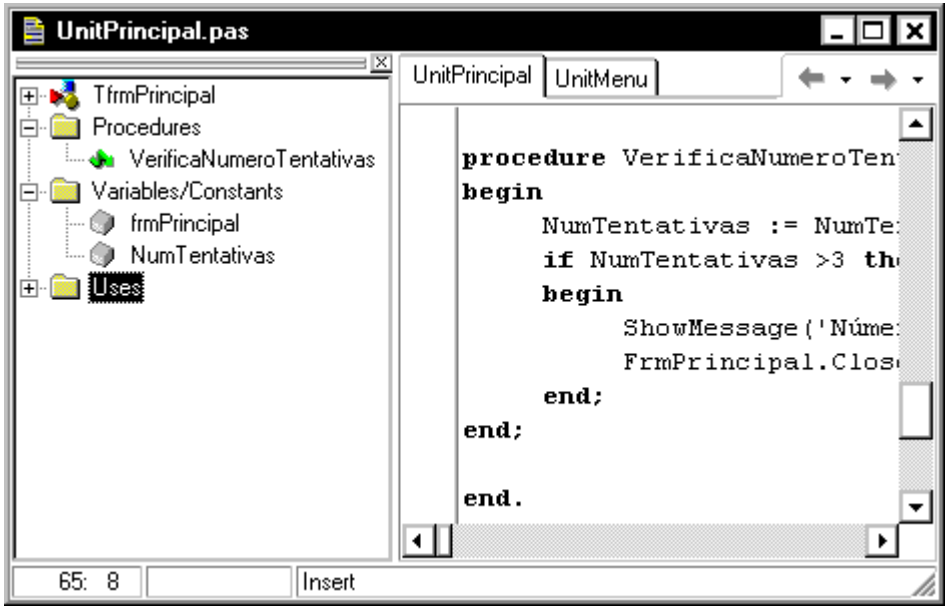
7) Vamos então implementar a procedure VerificaNumeroTentativas, na seção implementation:

```
implementation

(...)

procedure TfrmPrincipal.BtnOKClick(Sender:
   TObject);
(...)

procedure VerificaNumeroTentativas;
begin
     NumTentativas := NumTentativas + 1;
     if NumTentativas >3 then
     begin
          ShowMessage('Número máximo de tentativas
            alcançado. O sistema será fechado!');
          FrmPrincipal.Close;
     end;
end;
```

8) Inicialmente, incrementamos o valor de NumTentativas, depois o avaliamos. Se for maior do que três, uma mensagem é exibida para o usuário, e o formulário principal é fechado, através do método *Close*, o que encerra o aplicativo.

9) Salve o projeto, através do menu *File*, opção *Save All*.*Delphi 4* ◆ **108108** ◆ *Delphi 4*

10) Veja no Explorador de Código, as procedures e variáveis globais que já criamos:



11) Execute o aplicativo, e teste o uso da senha, e a entrada no formulário de menu.

Delphi 4

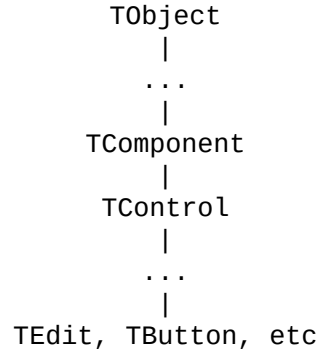
7

Explorando Formulários e Componentes

Introdução

Como vimos, uma aplicação é composta de formulários e componentes, como caixas de edição, etiquetas, etc. Tanto o formulário quanto os componentes oferecidos pelo Delphi possuem diversas propriedades, assim como métodos e eventos associados a eles. Muitas destas propriedades, métodos e eventos são comuns a formulários e componentes, e serão vistos neste capítulo.

A existência de propriedades, métodos e eventos comuns se deve a metodologia de criação destes componentes, orientada a objetos. O Delphi possui uma classe base, TObject, de onde são derivadas as demais classes, entre elas TComponent, de onde se originam todos os componentes do Delphi. De TComponent é criado TControl, ligado aos componentes visuais:



Portanto, todos os componentes visuais do Delphi herdam as propriedades e métodos de TObject, TComponent e TControl, e das demais classes que os deram origem.

Além das propriedades comuns, formulários e componentes também possuem propriedades, métodos e eventos específicos, que os diferenciam, e também serão vistos neste capítulo.

Propriedades Comuns

Entre as propriedades comuns a formulários e componentes podemos citar:

Action

novo! Determina a ação associada ao evento de clicar no componente.

Align

Determina como o componente se alinhará dentro do formulário. Você pode escolher *alNone* (mantém o componente onde você o colocou; é o valor default), *alTop* (move o componente para o topo do formulário e o redesenha para que tenha a largura do formulário, mantendo sua altura), *alBottom*, *alRight*, *alLeft* e *alClient*.

Anchors

novo! Determina como o componente está ancorado a seu componente pai (aquele que o contém, por exemplo, um formulário é o pai de uma caixa de edição).

BevelEdges, BevelInner, BevelKind, BevelOuter, BevelWidth

novo! Manipulam o formato das margens do controle.

BorderWidth

novo! Determina a largura da borda do controle.

Caption

Oferece um identificador para o usuário.

Color

Modifica a cor de fundo do componente. Para mudar a cor de fundo, dê um duplo clique na coluna de valor da propriedade. Aparece a caixa de diálogo Colors, permitindo que você faça a modificação de cor:

DockClientCount, DockClients, DockManager, DockSite, UseDockManager

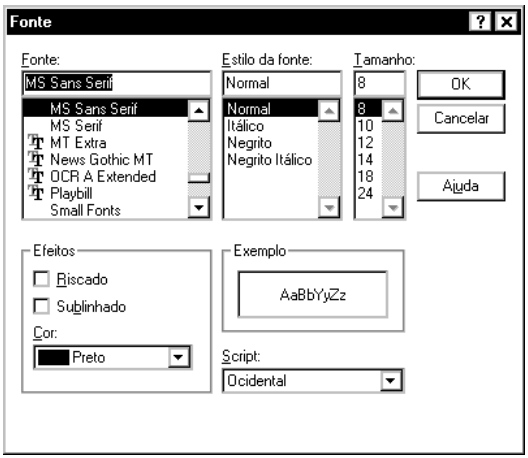
novo! Oferecem informações sobre o atracamento do controle.

Enabled

Controla se o componente responde ou não a eventos feitos pelo mouse, teclado ou tempo. Se for igual a *True*, o componente responde aos eventos. Se for igual a *False* (ou seja, desabilitado), o componente não responde a eventos. Componentes desabilitados aparecem no formulário mais claros.

Font

Modifica a fonte, estilo e cor do texto inserido no componente. Quando você dá um duplo clique na coluna de valor da propriedade, aparece o quadro Fonts, permitindo que você faça as modificações necessárias:



Height

Estabelece a altura do componente.

Hint

Estabelece o texto da legenda do componente. A legenda aparece quando o mouse passa por cima do componente.

Left

Estabelece a distância entre o formulário e o lado esquerdo do componente.

Name

Estabelece um identificador para o componente.

PopupMenu

Associa o componente a um menu Popup. Você deve escrever o nome do menu Popup. A criação de menus popup será vista com mais detalhes posteriormente.

ShowHint

Determina se o componente deve mostrar a legenda quando o ponteiro do mouse passar por cima de si. Se ShowHint for igual a *True*, a legenda aparece. Se ShowHint for igual a *False*, a legenda não aparece.

TabOrder

Indica a posição do componente na ordem de tabs de seu proprietário, ou seja, a ordem em que o componente recebe o foco quando o usuário pressiona a tecla Tab. O primeiro componente tem TabOrder igual a 0. Você pode verificar a ordem de todos os componentes através da opção *Tab Order* do menu *Edit*.

TabStop

Se for igual a *True*, o usuário pode dar um tab para se mover para o componente.

Tag

Associa um valor inteiro ao componente, que você pode utilizar no código de sua aplicação.

Top

Estabelece a distância entre o formulário e o lado superior do componente.

Visible

Se for igual a *True*, o componente será mostrado quando o programa for executado. Se for igual a *False*, o componente não será mostrado. Para escolher o estado do componente, basta clicar sobre a coluna de valor da propriedade. Aparece um quadro de lista contendo os valores *True* e *False*. Clique sobre o valor escolhido.

Width

Estabelece a largura do componente.

Métodos Comuns

Entre os métodos comuns a formulários e componentes podemos citar:

AssignTo

Copia as propriedades de um controle para outro.

BeginDrag

Inicia o arrastar (drag) do controle.

BringToFront

Coloca o componente ou formulário na frente de todos os outros objetos dentro de seu formulário.

CanFocus

Determina se um controle pode receber o foco.

ChangeScale

Reposiciona e redimensiona o controle de acordo com uma determinada taxa.

Create

Cria uma instancia do componente.

DoDock

Manipula o atracar de um controle em outro.

Dragging

Determina se um controle está sendo arrastado.

EndDrag

Termina a operação de arrastar o controle.

Hide

Esconde o controle.

Focused

Esta função é usada para determinar se um controle enjanelado tem o foco e portanto, é o controle ativo (ActiveControl). Retorna *True* se o componente possui o foco.

GetSiteInfo

Oferece informações sobre o controle atracado.

ManualDock

Atraca o controle a outro.

ManualFloat

Desatraca o controle.

Realign

Realinha os controles dentro do controle os contém.

Refresh

Este método apaga qualquer imagem que estiver sobre a tela e depois redesenha todo o componente. Para redesenhar sem apagar primeiro, escolha o método Repaint.

Repaint

Este método força o controle a redesenhar sua imagem sobre a tela, mas sem apagar o que já está lá.

ReplaceDockedControl

Substitui um controle atracado por outro.

ScaleBy

Redimensiona o controle e todos os controles que este contém, para uma determinada taxa.

ScrollBy

Rola o conteúdo de um controle.

SelectFirst

Seleciona o primeiro controle filho do controle, que pode ser acessado por um Tab.

SelectNext

Seleciona o próximo controle filho do controle, que pode ser acessado por um Tab.

SendToBack

Este método coloca o componente atrás de todos os outros componentes dentro do formulário.

SetBounds

Este método atribui os valores passados como parâmetro em SetBounds para as propriedades de limite do componente Left, Top, Width e Height.

SetFocus

Este método dá o foco ao controle.

Show

Torna um formulário ou componente visível por estabelecer a propriedade Visible deste igual a *True*.

Eventos Comuns

Além dos eventos específicos para cada componente, existem eventos comuns a quase todos os componentes.

OnCanResize


Ocorre quando o componente é redimensionado.

OnChange

Ocorre quando o componente muda de conteúdo.

OnClick

Ocorre quando o usuário clica sobre o componente.

OnConstrainedResize


Ocorre quando o componente é redimensionado de acordo com suas restrições.

OnDblClick

Ocorre quando o usuário dá um duplo clique sobre o componente.

OnDockDrop


Ocorre quando outro componente é atracado a este componente.

OnDockOver


Ocorre quando outro componente atracável é arrastado sobre este componente.

OnDragDrop

Ocorre quando o componente é largado, após ser arrastado.

OnDragOver

Ocorre quando um componente é arrastado sobre outro.

OnEndDock

Ocorre logo após o componente ser atracado a outro.

OnEndDrag

Ocorre quando um componente deixa de ser arrastado.

OnEnter

Ocorre quando o componente se torna ativo (ou seja, tem o foco).

OnExit

Ocorre quando o componente perde o foco.

OnGetSiteInfo

Retorna informações sobre o atracamento do controle.

OnKeyDown

Ocorre quando o usuário pressiona uma tecla.

Possui a seguinte lista de parâmetros:

(Sender: TObject; var Key: Word; Shift: TShiftState)

onde *Sender* é o componente que originou o evento; *Key* é uma variável do tipo word que determina qual tecla foi apertada; *Shift* determina se Ctrl, Alt ou Shift foram

apertados também (em uma combinação Alt+a, por exemplo).

A variável Key pode assumir, entre outros valores:

VK_F1: indica que a tecla funcional F1 foi apertada.

VK_MULTIPLY: indica que a tecla de multiplicação (*) foi apertada.

Veja os demais valores que a variável Key pode assumir no Help do Delphi.

O parâmetro Shift pode assumir, entre outros valores:

ssShift: indica que a tecla Shift foi apertada.
ssCtrl: indica que a tecla Ctrl foi apertada.
ssAlt: indica que a tecla Alt foi apertada.

Veja os demais valores que o parâmetro Shift pode assumir no Help do Delphi.

OnKeyUp

Ocorre quando o usuário solta uma tecla pressionada.

Possui a seguinte lista de parâmetros:

(Sender: TObject; var Key: Word; Shift: TShiftState)

onde *Sender* é o componente que originou o evento; *Key* é uma variável do tipo word que determina qual tecla foi solta; *Shift* determina se Ctrl, Alt ou Shift foram soltos também (em uma combinação Alt+a, por exemplo).

A variável Key pode assumir, entre outros valores:

VK_F1: indica que a tecla funcional F1 foi solta.

VK_MULTIPLY: indica que a tecla de multiplicação (*) foi solta.

Veja os demais valores que a variável Key pode assumir no Help do Delphi.

O parâmetro Shift pode assumir, entre outros valores:

ssShift: indica que a tecla Shift foi solta.
ssCtrl: indica que a tecla Ctrl foi solta.
ssAlt: indica que a tecla Alt foi solta.

Veja os demais valores que o parâmetro Shift pode assumir no Help do Delphi.

OnKeyPress

Ocorre quando o usuário pressiona uma tecla simples (padrão ANSI). Ou seja, não vale para teclas funcionais, ou combinações de teclas, por exemplo.

Possui a seguinte lista de parâmetros:

(Sender: TObject; var Key: char)

onde *Sender* é o componente que originou o evento; *Key* é uma variável do tipo char que determina qual tecla foi pressionada.

OnMouseDown

Ocorre quando o usuário pressiona o botão do mouse.

Possui a seguinte lista de parâmetros:

(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer)

onde *Sender* é o componente que originou o evento; *Button* indica que o botão do mouse foi apertado; *Shift* indica se as teclas Alt, Shift ou Ctrl foram apertadas, e X e Y indicam as coordenadas do componente onde o botão do mouse foi pressionado.

O parâmetro Button pode assumir, entre outros valores:

mbRight: indica que o botão direito do mouse foi pressionado.
mbLeft: indica que o botão esquerdo do mouse foi pressionado.
mbMiddle: indica que o botão central do mouse foi pressionado.

O parâmetro Shift pode assumir, entre outros valores:

ssShift: indica que a tecla Shift foi apertada.
ssCtrl: indica que a tecla Ctrl foi apertada.
ssAlt: indica que a tecla Alt foi apertada.

Veja os demais valores que o parâmetro Shift pode assumir no Help do Delphi.

OnMouseUp

Ocorre quando o usuário solta o botão do mouse.

Possui a seguinte lista de parâmetros:

(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer)

onde *Sender* é o componente que originou o evento; *Button* indica que botão do mouse foi solto; *Shift* indica se as teclas Alt, Shift ou Ctrl foram soltas, e X e Y indicam as coordenadas do componente quando o botão do mouse é solto.

O parâmetro Button pode assumir, entre outros valores:

mbRight: indica que o botão direito do mouse foi solto.
mbLeft: indica que o botão esquerdo do mouse foi solto.
mbMiddle: indica que o botão central do mouse foi solto.

O parâmetro Shift pode assumir, entre outros valores:

ssShift: indica que a tecla Shift foi solta.
ssCtrl: indica que a tecla Ctrl foi solta.
ssAlt: indica que a tecla Alt foi solta.

Veja os demais valores que o parâmetro Shift pode assumir no Help do Delphi.

OnMouseMove

Ocorre quando o usuário passa o cursor do mouse sobre o componente.

Possui a seguinte lista de parâmetros:

(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer)

onde *Sender* é o componente que originou o evento; *Button* indica que botão do mouse foi apertado; *Shift* indica se as teclas Alt, Shift ou Ctrl foram apertadas, e X e Y indicam as coordenadas onde o mouse está, quando este é movido.

O parâmetro Button pode assumir, entre outros valores:

mbRight: indica que o botão direito do mouse foi pressionado.
mbLeft: indica que o botão esquerdo do mouse foi pressionado.
mbMiddle: indica que o botão central do mouse foi pressionado.

O parâmetro Shift pode assumir, entre outros valores:

ssShift: indica que a tecla Shift foi apertada.
ssCtrl: indica que a tecla Ctrl foi apertada.
ssAlt: indica que a tecla Alt foi apertada.

Veja os demais valores que o parâmetro Shift pode assumir no Help do Delphi.

OnMouseWheel

Ocorre quando a roda do mouse é rotacionada.

OnMouseWheelDown

Ocorre quando a roda do mouse é rotacionada para baixo.

OnMouseWheelUp

Ocorre quando a roda do mouse é rotacionada para cima.

OnResize

Ocorre quando o componente é redimensionado.

OnStartDock

Ocorre quando se inicia o atracar de um componente a outro.

OnStartDrag

Ocorre quando se inicia o arrastar de um componente.

OnUndock

Ocorre quando outro componente é desatracado deste componente.

Explorando Formulários

Você usa formulários para interfaciar com o usuário. No formulário são inseridos os componentes.

O formulário, representado no Delphi pela classe *TForm*, é uma janela, e portanto, possui os atributos de uma janela (menu de controle, botões de maximizar e minimizar, barra de título, bordas redimensionáveis).

Propriedades Específicas

Entre as propriedades específicas de TForm podemos citar:

ActiveControl: especifica que componente do formulário tem o foco.

AutoScroll: determina se as barras de rolamento irão aparecer no formulário quando este não for largo o bastante para mostrar todos os componentes que contém. Se AutoScroll for igual a *True*, as barras de rolamento aparecem sempre que necessário (por exemplo, se você redimensionar o formulário).

BorderIcons: suas sub-propriedades determinam se o formulário terá menu de controle, botões de maximizar e minimizar, e se mostrará o Help em um menu popup. Se alguma destas sub-propriedades for igual a *False*, ela não aparecerá.

BorderStyle: determina o estilo da borda do formulário: *bsDialog* (janela não redimensionável, com borda padrão); *bsSingle* (janela não redimensionável, com borda linear); *bsNone* (janela não redimensionável, sem borda, botões e Menu de Controle); *bsSizeable* (janela padrão), entre outras.

DefaultMonitor: determina em que monitor será mostrado o formulário, em uma aplicação de múltiplos monitores.

novo! *Floating:* determina se o formulário está atracado a outra janela.

FormState: determina o estado do formulário (sendo criado, visível, etc).

FormStyle: determina o tipo do formulário: *fsMDI form* (formulário pai), fs*MDIchild* (formulários filhos), *fsNormal* (formulários sem relacionamento entre si) e *fsStayonTop* (sempre no topo).

HelpFile: determina o arquivo de help associado ao formulário.

HorizScrollBar, *VertScrollBar*: adiciona barras de rolamento. São propriedades com sub-propriedades, para customizar as barras de rolamento.

Icon: determina o ícone que aparecerá quando o formulário for minimizado.

Menu: determina o menu associado ao formulário.

ModalResult: determina o valor de retorno de um formulário modal.

Position: determina a posição do formulário na tela: *poDesigned* (aparece na tela com a posição, largura e altura que foi estabelecidas no projeto); *poDefault* (aparece na tela com a posição, largura e altura estabelecidas pelo Delphi); *poDefaultPosOnly* (aparece na tela com a largura e altura que foi estabelecida no projeto, mas com posição determinada pelo Delphi); *poDefaultSizeOnly* (aparece na tela com a largura e altura estabelecidas pelo Delphi, e posição estabelecida noe projeto); *poScreenCenter* (o formulário fica com o tamanho estabelecido no projeto, mas é posicionado no centro da tela).

Window State: estabelece o estado inicial do formulário: *wsNormal* (nem maximizado nem

minimizado); *wsMaximized* (o formulário aparece maximizado); *wsMinimized* (o formulário aparece minimizado).

Métodos Específicos

Entre os métodos específicos de TForm podemos citar:

Close: fecha o formulário, terminando a aplicação se este for o formulário principal.

CloseQuery: determina se o formulário pode ser fechado.

DefocusControl: retira o foco de determinado componente do formulário.

FocusControl: estabelece o foco em determinado componente do formulário.

GetFormImage: retorna um bitmap do formulário.

Hide: esconde o formulário.

Print: imprime o formulário.

ScrollinView: rola o formulário para que determinado componente fique visível na tela.

Show: mostra o formulário na tela.

ShowModal: mostra o formulário como um formulário modal.

Eventos Específicos

Entre os eventos específicos de TForm temos:

onActivate: ocorre quando um formulário se torna ativo.

onClose: ocorre quando o formulário for fechado.

onCloseQuery: ocorre quando o formulário for fechado, possuindo uma variável booleana que verifica se o formulário pode ser fechado.

onCreate: ocorre quando o formulário é criado. Quando um formulário é criado, estes eventos ocorrem na ordem listada: onCreate, onActivate, onPaint.

onDeactivate: ocorre quando o formulário perde o foco.

onDestroy: ocorre quando um formulário é destruído.

onHelp: ocorre quando um formulário recebe um pedido de help.

onHide: ocorre quando o formulário é escondido.

onKeyPress: ocorre quando uma tecla simples (ou seja, diferente de combinação de teclas ou teclas funcionais) é pressionada.

onPaint: ocorre quando o formulário recebe o foco ou se torna visível.

onResize: ocorre sempre que um formulário é redimensionado enquanto a aplicação está rodando.

OnShow: ocorre exatamente antes do formulário se tornar visível.

Formulário Modal

Formulário modal é aquele que tem que ser fechado antes que a aplicação possa continuar.

Para mostrar um formulário modal, use o método *ShowModal* do formulário. Com este método, o formulário mostrado deverá ser fechado para que a aplicação possa continuar.

Enquanto o formulário modal estiver aberto, sua propriedade *ModalResult* é igual a 0. Se o usuário tentar dar o foco para outro formulário, o valor de ModalResult será passado para o método ShowModal que chamou o formulário, e a operação não será completada.

Quando o usuário fechar o formulário modal de alguma forma, o valor de sua propriedade *ModalResult* passa para um valor diferente de zero. Este valor será recebido pelo método ShowModal, e outro formulário poderá ter o foco.

Inserindo Texto no Formulário

Label

O componente Label, do grupo de componentes Standard, mostra um texto em uma área determinada. É utilizado normalmente para etiquetar um outro componente.

O texto contido no componente Label é determinado pela propriedade *Caption*.

Entre as propriedades específicas do componente Label temos:

Alignment: modifica o alinhamento do texto inserido no componente. Você pode escolher à esquerda (*taLeftJustify*), à direita (*taRightJustify*) ou ao centro (*taCenterJustify*).

AutoSize: se for igual a *True*, o tamanho do componente irá se modificar conforme o texto for sendo inserido. O default é *True*.

Canvas: representa a superfície do componente.

Layout: determina o alinhamento vertical do componente. Você poderá escolher *tlTop, tlCenter* ou *tlBottom.*

Transparent: determina se a etiqueta é transparente (ou seja, usa a cor de fundo do que estiver a seu fundo, só o texto aparece). Se for igual a *True*, a etiqueta é transparente.

WordWrap: se for igual a *True*, permite que o texto inserido no componente vá para a linha de baixo, caso alcance a margem direita.

TStaticText

Além do componente Label, o Delphi oferece o componente TStaticText, do grupo de componentes Additional, um controle enjanelado que também mostra texto em um formulário.

Criando uma Área de Entrada de Dados

O Delphi oferece diversos componentes que, inseridos no formulário, permitem que o usuário entre dados. Entre eles temos: Edit, Memo, MaskEdit e RichEdit.

Edit

O componente Edit, do grupo de componentes Standard, é utilizado para que o usuário possa entrar informações.

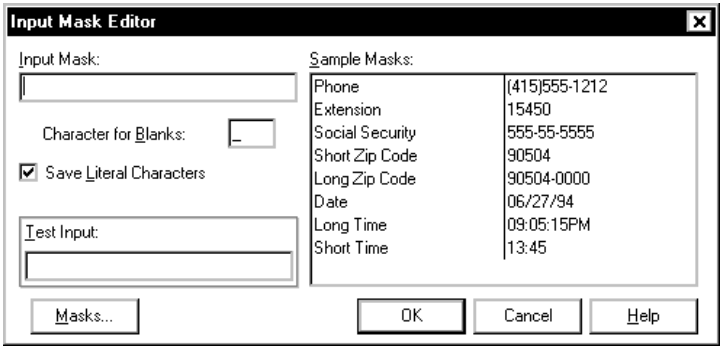
O texto inserido no componente Edit é armazenado na propriedade *Text*.

MaskEdit

O componente MaskEdit, do grupo de componentes Additional, é semelhante ao componente Edit, só que o usuário só pode entrar determinados valores, definidos na propriedade EditMask do componente.

O texto inserido no componente MaskEdit é definido pela propriedade *Text*.

A propriedade *EditMask* representa a máscara usada para limitar a entrada de dados. No Object Inspector, quando você dá um duplo clique sobre o botão (...), na coluna à direita da propriedade EditMask, aparece um quadro de diálogo com diversas máscaras para você escolher:



Você também pode criar outras máscaras neste quadro, clicando sobre o botão Masks. A propriedade *isMasked* define se o controle possui uma máscara ou não. A propriedade *EditText* define o texto entrado no controle, após a formatação da máscara.

O Objeto TStrings

O Delphi oferece para o programador o objeto TStrings, que representa uma lista de strings. Este objeto possui diversas propriedades e métodos, que você pode utilizar para manipular listas de strings.

Diversos componentes possuem propriedades que se referem a listas de strings. Por exemplo, o contéudo de um quadro de lista (ListBox), determinado pela propriedade Items do ListBox, está contido em um objeto TStrings. Para manipular o conteúdo do quadro de lista (por exemplo, para inserir uma linha), você usará as propriedades e métodos do objeto TStrings.

Entre as propriedades do objeto TStrings temos:

Count: contém o número de elementos da lista de strings.

Strings[Index]: acessa determinada linha da lista. Para acessar a primeira linha, por exemplo, substitua o parâmetro Index por 0: String[0].

Entre os métodos do objeto TStrings temos:

Add(String): insere uma string no componente.

Delete(index): remove a string de índice index do componente.

Exchange(Index1,Index2): troca a string da posição Index1 pela string da posição Index2, e vice-versa. Index1 e Index2 são integer.

Insert(index,s): insere uma string na posição estabelecida por index, onde index é integer e s é string.

LoadFromFile(arquivo): recupera uma lista de string de um arquivo.

Move(PosCur,NovaPos): move a string da posição PosCur para a posição NovaPos. PosCur e NovaPos são integer.

SaveToFile(arquivo): salva uma lista de strings para um arquivo.

Com estas propriedades e métodos, você pode manipular listas de strings, que serão usadas em diversos componentes.

Memo

 O componente Memo, do grupo de componentes Standard, é semelhante ao componente Edit, só que

permite a entrada ou visualização dos dados em múltiplas linhas.

A propriedade *Lines* contém as linhas de texto do componente Memo. Esta propriedade é um objeto TStrings, portanto, usa suas propriedades e métodos. Por exemplo, para se referir a décima linha do componente Memo1, faça:

```
Memo1.Lines.Strings[10]
```

Para adicionar a string 'Bom Dia!', faça:

```
Memo1.Lines.Add('Bom Dia!');
```

A propriedade *ScrollBars* determina se o componente terá ou não barras de rolamento. A propriedade *WordWrap*, se for igual a *True*, permite que o texto inserido no componente vá para a linha de baixo, caso alcance a margem direita.

RichEdit

O componente RichEdit, do grupo de componentes Win32, é semelhante ao componente Memo, só que o usuário pode formatar de forma mais completa o texto inserido.

O texto inserido no componente RichEdit é definido pela propriedade *Lines*. A propriedade *Paragraphs* contém informações sobre a formatação do parágrafo selecionado. A propriedade *PlainText* determina se o texto inserido pode ser formatado ou não.

A propriedade *ScrollBars* determina se o componente terá ou não barras de rolamento. A propriedade *WordWrap*, se for igual a *True*, permite que o texto inserido no componente vá para a linha de baixo, caso alcance a margem direita.

Use o método *Print* para imprimir o conteúdo do componente RichEdit.

Outras Propriedades, Métodos e Eventos dos Componentes de Entrada de Dados

Entre as demais propriedades dos componentes Edit, MaskEdit, Memo e RichEdit temos:

AutoSelect: se for igual a *True*, o texto no componente é automaticamente selecionado quando o usuário acessar o componente. O default é *True*.

AutoSize: se for igual a *True*, o tamanho do componente irá se modificar conforme o texto for sendo inserido. O default é *True*.

novo! *CanUndo:* se for igual a True, indica que o componente contém mudanças que podem ser desfeitas.

Charcase: determina como o texto digitado aparecerá na tela. As opções são: *ecUpperCase* (em letras maiúsculas), *ecLowerCase* (em letras minúsculas) ou *ecNormal* (como o usuário entrar).

MaxLength: determina o número máximo de caracteres que podem ser inseridos. O default é 0, que significa que não há limite.

Modified: se for igual a *True*, significa que o texto inserido no componente mudou.

PasswordChar: indica o caracter que deve ser mostrado na tela no lugar dos caracteres reais digitados pelo usuário.

ReadOnly: se for igual a *True*, o componente só poderá mostrar o texto, que não poderá ser modificado.

SelLength: retorna o comprimento em caracteres do texto selecionado do componente.

SelStart: retorna a posição inicial da parte selecionada do texto do componente.

SelText: contém a parte selecionada do texto do componente.

Entre os métodos específicos dos componentes Edit, Memo, MaskEdit e RichEdit temos:

Clear: apaga todo o texto do componente.

ClearSelection: apaga o texto selecionado no componente.

ClearUndo: apaga todo o texto carregado no buffer, de forma que nenhuma mudança possa ser desfeita.

CopyToClipBoard: copia o texto selecionado para a área de transferência.

CutToClipBoard: copia o texto selecionado para a área de transferência, e então apaga o texto do controle.

PasteFromClipboard: cola o texto da área de transferência no controle.

SelectAll: seleciona todo o texto do controle.

Undo: copia de volta do buffer todas as mudanças para lá transferidas.

Iniciando uma Ação com Botões

O Delphi oferece diversos componentes que, se pressionados pelo usuário, geram determinada ação. Entre estes componentes temos: Button e BitBtn.

Button

O componente Button, do grupo de componentes Standard, é utilizado para que o usuário possa iniciar uma ação.

O título do botão é dado pela propriedade *Caption*.

BitBtn

Use o componente BitBtn, do grupo de componentes Additional, para mostrar algum desenho em um botão, tornando-o mais comunicativo.

A propriedade *Glyph* determina o desenho que aparecerá no componente BitBtn.

A propriedade *Kind* define o tipo de botão. Há vários botões com bitmaps pré-desenhados que você pode usar em seu formulário, mudando a propriedade Kind. Os botões com bitmaps pré-desenhados são:

Valor	Tipo
bkCancel	botão Cancel
bkCustom	botão adaptável
bkHelp	botão Help
bkNo	botão No
bkOK	botão OK
bkYes	botão Yes

A propriedade *Layout* determina a posição do desenho em relação ao botão. Já a propriedade *Margin* determina a distância em pixels da borda da imagem à borda do botão. A propriedade *Spacing* determina a distância em pixels entre o texto e a imagem do botão. E a propriedade *NumGlyphs* determina o número de imagens que o botão conterá. Cada imagem aparecerá de acordo com o estado do botão (pressionado ou não pressionado, por exemplo).

Outras Propriedades, Métodos e Eventos

Entre as demais propriedades dos componentes Button e BitBtn temos:

Cancel: se esta propriedade for igual a *True*, qualquer hora que o usuário pressionar Esc, equivalerá a execução do evento OnClick do botão,ou seja, equivalerá a apertar o botão.

Default: indica se o botão é o botão default. Se esta propriedade for igual a *True*, toda vez que o usuário pressionar Enter, equivalerá a execução do evento OnClick, ou seja, equivalerá a apertar o botão.

ModalResult: Permite que o clicar do botão feche um formulário modal. Veja maiores informações sobre formulários modais no capítulo sobre Formulários.

Entre os demais métodos dos componentes Button e BitBtn podemos citar:

Click: simula um clicar no botão, executando o código associado ao evento onClick deste.

Juntando Botões em Barras de Ferramentas

SpeedButton

 Use o componente SpeedButton, do gurpo de componentes Additional, para criar Barras de Ferramentas. SpeedButtons são botões com um desenho e normalmente nenhum texto.

A propriedade *Glyph* determina o desenho que aparecerá no componente SpeedButton. A propriedade *Caption* determina o texto que aparecerá no componente SpeedButton. A propriedade *GroupIndex* especifica quais SpeedButtons trabalham juntos como um grupo. O valor

0 indica que o botão não pertence a nenhum grupo. A propriedade *Down* indica se o componente está pressionado ou não.

Já a propriedade *MouseInControl* determina se o cursor está sobre o controle ou não.

A propriedade *Layout* determina a posição do desenho em relação ao botão. Já a propriedade *Margin* determina a distância em pixels da borda da imagem à borda do botão. A propriedade *Spacing* determina a distância em pixels entre o texto e a imagem do botão. E a propriedade *NumGlyphs* determina o número de imagens que o botão conterá. Cada imagem aparecerá de acordo com o estado do botão (pressionado ou não pressionado, por exemplo).

A propriedade *Transparent* determina o tipo de fundo do botão.

ToolBar


Este componente, do grupo de componentes Win32, cria barras de ferramentas, onde são inseridos botões e outros controles, arranjando-os em linhas e automaticamente ajustando seus tamanhos e posições.

Para gerenciar as imagens que serão mostradas nos botões, insira um componente ImageList, também do grupo Win32, e através do ImageList Editor, armazene as imagens. Associe então o ImageList à propriedade *Images* do componente ToolBar.

Para inserir um novo botão, clique com o botão direito sobre o componente, e escolha a opção *New Button*. Os botões inseridos são objetos do tipo ToolButton. A propriedade *ImageIndex* determina qual imagem armazenada no ImageList será mostrada no botão.

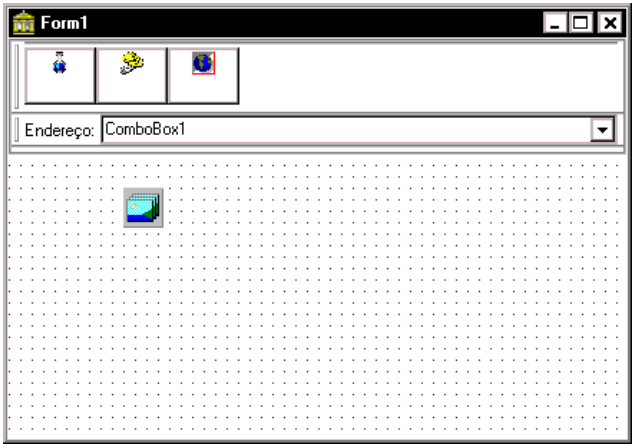
As propriedades *ButtonWidth* e *ButtonHeight* do componente ToolBar determinam o tamanho dos botões.

A propriedade *ShowCaptions* determina se o título dos botões, determinado pela propriedade *Caption* de cada botão, será mostrado ou não.

Quanto aos botões, a propriedade *Down* mantém o botão está pressionado ou não. A propriedade *Style* determina o tipo de botão: *tbsButton* (botão comum), *tbsCheck*, *tbsDivider* (barra de divisão); *tbsDropDown* (botão que oferece uma lista de outros botões); *tbsSeparator* (espaço separador).

CoolBar

Este componente, do grupo de componentes Win32, é semelhante ao anterior, só que a barra pode conter outros componentes que não botões. Além disso, a barra pode ser movida ou redimensionada, assim como se agrupar a outras.



Após inserir o componente CoolBar no formulário, você deve inserir os componentes desejados. Por exemplo, você pode inserir um ComboBox, ou uma ToolBar. Para cada componente inserido é criado uma banda (na verdade, um objeto CoolBand), que você gerencia através do Bands Editor, acessado através de um clique com o botão direito

sobre o componente CoolBar. Cada banda pode ter um texto, dado pela propriedade *Text*.

ControlBar

 Este componente, do grupo Additional de componentes, gerencia o layout dos componentes de uma ToolBar.

Selecionando Opções

CheckBox

 Use o componente CheckBox, do grupo de componentes Standard, para mostrar ao usuário uma caixa que ele pode marcar ou desmarcar, com um texto que acompanha esta caixa.

A propriedade *Caption* representa o texto associado ao componente. A propriedade *Checked* determina se o componente está selecionado. Se for igual a *True*, está selecionado, quando se executa a aplicação. Se for igual a *False*, não está selecionado.

A propriedade *Alignment* modifica o alinhamento do texto inserido no componente. Você pode escolher a esquerda (taLeftJustify), ou a direita (taRightJustify).

A propriedade *State* determina os vários estados que uma caixa de checagem pode ter. Pode estar marcada (cbChecked), desmarcada (cbUnchecked) ou cinza (cbGrayed).

RadioButton

 Use o componente RadioButton, do grupo de componentes Standard, para apresentar um conjunto de

opções mutuamente exclusivas para o usuário (ou seja, o usuário só pode selecionar uma das opções do conjunto).

Você deve inserir os botões de rádio em um componente como GroupBox ou Panel, para agrupá-los.

A propriedade *Caption* representa o texto associado ao componente. A propriedade *Checked* determina se o componente está selecionado. Se for igual a *True*, está selecionado, quando se executa a aplicação. Se for igual a *False*, não está selecionado.

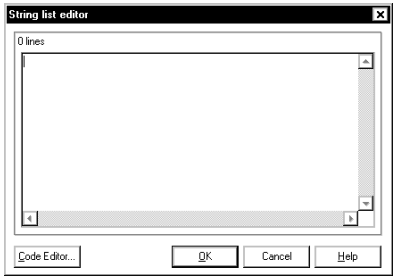
A propriedade *Alignment* modifica o alinhamento do texto inserido no componente. Você pode escolher a esquerda (taLeftJustify), ou a direita (taRightJustify).

RadioGroup

Use o componente RadioGroup, do grupo de componentes Standard, para criar um quadro de grupo que contém RadioButtons.

A propriedade *Columns* define o número de colunas do RadioGroup. Se esta propriedade for igual a 1, os botões de rádio inseridos no RadioGroup aparecerão um abaixo do outro.

A propriedade *Items* contém os textos dos RadioButtons que aparecem no RadioGroup. Você pode inserir estes textos através do Object Inspector, dando um clique sobre o botão (...) na coluna à direita da propriedade Items:



No quadro de diálogo que aparece, insira o título do primeiro botão de rádio, tecle Enter, insira o título do segundo botão de rádio, e assim sucessivamente. Quando terminar, clique no botão OK.

Se você quiser manipular os RadioButtons durante a execução do programa (por exemplo, inserindo novos botões ou apagando algum), deve saber que a propriedade Items corresponde a um objeto TStrings, portanto, utiliza suas propriedades e métodos para manipular a lista de RadioButtons.

Por exemplo, para adicionar os botões "Masculino" e "Feminino" em um RadioGroup durante a programação, faça:

```
RadioGroup1.Items.Add('Masculino');
RadioGroup1.Items.Add('Feminino');
```

A propriedade *ItemIndex* corresponde ao número de ordem do RadioButton selecionado no RadioGroup. Se nenhum RadioButton estiver selecionado, o valor de ItemIndex é -1.

Listando os Dados

ListBox

Use este componente, do grupo de componentes Standard, para mostrar uma lista de itens e permitir que o usuário selecione um item desta lista.

A propriedade *Columns* define o numero de colunas em um ListBox. A propriedade *IntegralHeight* controla a forma como o ListBox aparece no formulario.

A propriedade *Selected* indica quais itens da lista estão selecionados, se algum estiver. Use com a seguinte sintaxe

Lista.Selected[Índice]. Retorna *True* se o item com este índice estiver selecionado.

A propriedade *SelCount*Conta o número de itens que estão selecionados quando o valor da propriedade MultiSelect está igual a *True*. Se nenhum item está selecionado, o valor da propriedade SelCount é igual a 1

A propriedade *TopIndex* representa o índice do item da lista que aparece no topo do quadro. E a propriedade *MultiSelect* determina se o usuário pode selecionar mais de um elemento de cada vez na lista.

ComboBox

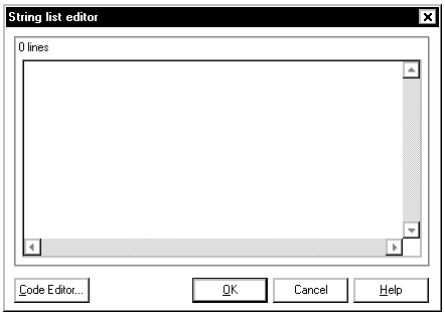
 Este componente, também do do grupo de componentes Standard, é semelhante ao ListBox, só que o usuário também pode entrar um novo item, ao invés de selecionar um item da lista.

Um evento importante ligado ao componente ComboBox é o evento onChange, que ocorre quando o usuário altera o texto, o que ele só pode fazer em componentes ComboBox.

A propriedade *DropDownCount* determina o tamanho da lista drop-down de um ComboBox. Por default, cabem 8 elementos sem precisar de rolamento. A propriedade *Text* corresponde ao texto inserido na caixa de edição de um ComboBox.

Inserindo Dados nas Listas

Para entrar dados em componentes ListBox ou ComboBox durante o projeto, use a propriedade *Items*, dando um clique no botão a sua direita. Aparece o editor de strings:



Entre com o texto do primeiro item da lista, e tecle Enter, entre com o texto do segundo item da lista, e tecle Enter, e assim sucessivamente, até o fim da lista. Quando acabar, clique no botão OK.

Você também pode inserir dados na lista durante a execução do programa. A propriedade Items corresponde a um objeto TStrings, portanto, utiliza suas propriedades e métodos para manipular a lista de strings. O objeto TStrings é visto em mais detalhes no capítulo 11.

Por exemplo, para adicionar os itens "Segunda" e "Terça" em um ListBox ou ComboBox durante a programação, faça:

```
ListBox1.Items.Add('Segunda');
ListBox1.Items.Add('Terça');
```

Referindo-se ao Item Selecionado

O valor da propriedade ItemIndex é o número de ordem do item selecionado na lista do componente. Se nenhum item é selecionado, o valor de ItemIndex é -1. Para se referir ao elemento selecionado na lista, faça:

```
ListBox1.Items.Strings[ListBox1.ItemIndex]
```

Propriedades, Métodos e Eventos dos Componentes ListBox e ComboBox

Entre as propriedades dos componentes ListBox e ComboBox temos:

ItemHeight: é a altura de um item no componente, quando a propriedade Style for igual a *IsOwnerDrawFixed*.

Sorted: indica se a lista estará ordenada alfabeticamente.

Style: determina o estilo do componente. Escolha *lbStandard* para criar quadros em que cada string tenha altura padrão (20 pixels); *lbOwnerDrawFixed* para criar quadros em que os itens tenham a altura especificada na propriedade ItemHeight; *lbOwnerDrawVariable* para criar quadros em que os itens possam ter alturas variáveis.

Entre os métodos dos componentes ListBox e ComboBox temos:

Clear: apaga todo o texto do componente.

Entre os eventos específicos dos componentes ListBox e ComboBox temos:

onDrawItem: ocorre sempre que uma aplicação precisa redesenhar um item em uma caixa de lista ou combo (por exemplo, quando se rola o combo ou lista). Se a propriedade Style for igual a *lbOwnerDrawFixed*, este item é desenhado utilizando o valor da propriedade ItemHeight. Se a propriedade Style for igual a *lbOwnerDrawVariable*, o item é desenhado utilizando-se a altura obtida quando o evento onMeasureItem ocorre.

onMeasureItem: ocorre sempre que uma aplicação precisa redesenhar um item em uma caixa de lista ou combo, e a propriedade Style for igual a lbOwnerDrawVariable. Quando este evento ocorre, a altura do item na caixa é medida. Após este evento, ocorre o evento OnDrawItem.

OutLine

 O componente OutLine, do grupo de componentes Win 3.1, lista dados de forma hierárquica.

Cada item de um componente Outline é chamado de nó. Os nós são acessados através da propriedade *Items*. Por exemplo, Items[1] refere-se ao primeiro item da árvore de dados.

Para adicionar um item ao componente Outline, use o método *Add*. Para inserir um item, use o método *Insert*. Por exemplo, para inserir o item 'Dias' no componente Outline, faça:

```
Outline1.Items.Add('Dias');
```

Para adicionar um item de nível inferior, use o método *AddChild*. Por exemplo, para inserir o item 'Segunda' abaixo do item 'Dias', faríamos:

```
Outline1.AddChild(Outline1.SelectedItem,
'Segunda');
```

Note o item abaixo do qual será inserido um sub-item deve estar selecionado. A propriedade *SelectedItem* especifica o item selecionado na lista.

Para remover um item, use o método *Delete*. Para inserir um item, use o método *Insert*.

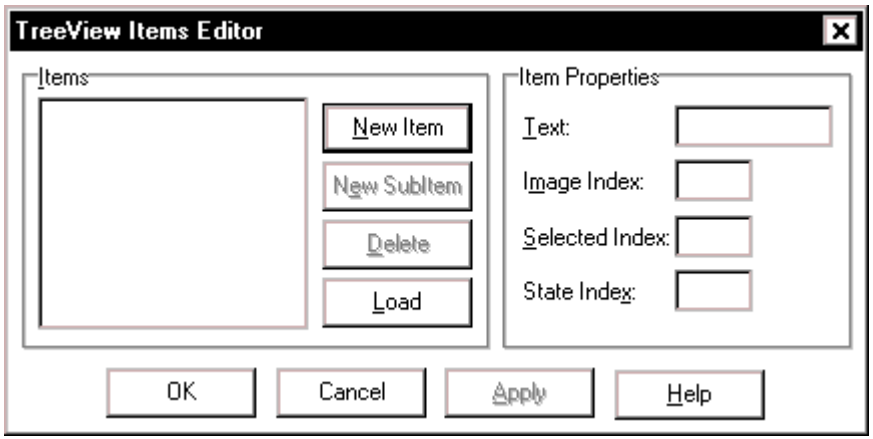
A propriedade *OutlineStyle* determina o tipo de imagens usados no Outline.

TreeView

 Este componente, do grupo de componentes Win32, lista dados de forma hierárquica em uma árvore. Ao contrário do componente Outline, você pode adicionar uma imagem a cada nó da árvore hierárquica.

Cada item de um componente TreeView é chamado de nó. Os nós são acessados através da propriedade *Items*. Por exemplo, Items[1] refere-se ao primeiro item da árvore de dados. Cada item do componente TreeView é na realidade um objeto da classe TTreeNodes.

Para adicionar um item ao componente TreeView, use a propriedade *Items*. Quando você dá um duplo clique sobre esta propriedade, aparece o quadro abaixo:



Clique sobre o botão *New Item* e, à direita, na caixa Text, digite o texto do primeiro item. Clique novamente sobre o botão *New Item* e digite na caixa Text o texto do segundo item.

Se quiser inserir um sub-item, ilumine o item e clique no botão *New Subitem*, digitando na caixa Text o texto do subitem.

Quando terminar de inserir itens e subítens, clique no botão OK.

No código, você poderá usar os métodos da classe TTreeNodes, para trabalhar com itens e subitens: *AddChildFirst*, *AddChild*, *AddChildObjectFirst*, *AddChildObject*, *AddFirst*, *Add*, *AddObjectFirst*, *AddObject* e *Insert*. Para remover um item, use o método *Delete*.

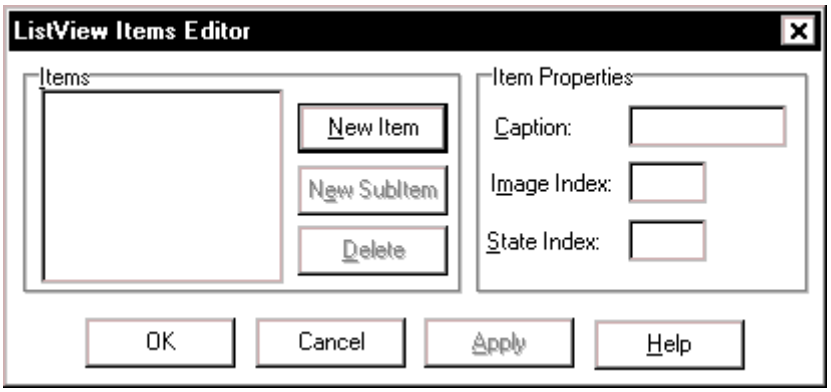
A propriedade *Images* determina o a lista de imagens associada ao componente TreeView.

ListView

Use este componente, do grupo de componentes Win32, para listar dados de forma hierárquica em uma lista.

Cada item de um componente ListView é acessado através da propriedade *Items*. Por exemplo, Items[1] refere-se ao primeiro item da lista. Cada item do componente ListView é na realidade um objeto da classe TListItems.

Para adicionar um item ao componente ListView, use a propriedade *Items*. Quando você dá um duplo clique sobre esta propriedade, aparece o quadro abaixo:



Clique sobre o botão *New Item* e, à direita, na caixa Text, digite o texto do primeiro item. Clique novamente sobre o botão *New Item* e digite na caixa Text o texto do segundo item.

Se quiser inserir um sub-item, ilumine o item e clique no botão *New Subitem*, digitando na caixa Text o texto do subitem.

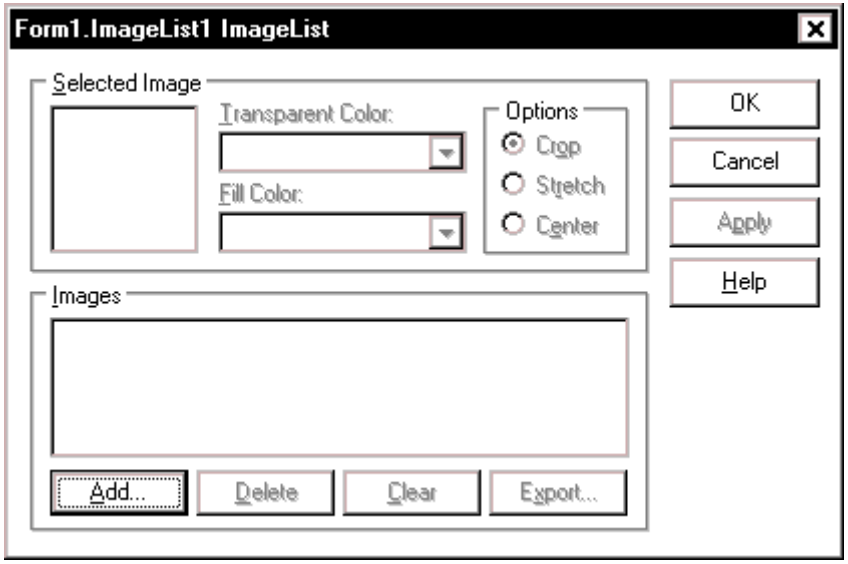
Quando terminar de inserir itens e subítens, clique no botão OK.

No código, para adicionar um item ao componente ListView, use os métodos da classe TListItems *Add* e *Insert*. Para remover um item, use o método *Delete*.

ImageList

Use este componente, do grupo de componentes Win32, para armazenar uma lista de imagens que serão usadas posteriormente em componentes Toolbar, TreeView, entre outros.

Para armazenar as imagens, dê um clique com o botão direito no componente. Aparece o quadro abaixo:



Clique no botão *Add*, para adicionar as imagens, e no botão OK para encerrar.

TCheckListBox

 Este componente, do grupo Additional, é semelhante a um ListBox, só que cada item possui uma caixa de checagem a seu lado.

Emoldurando os Dados

GroupBox

 Use este componente, do grupo Standard, para agrupar outros controles em um formulário. Normalmente é utilizado para agrupamento de RadioButtons.

A propriedade *Caption* determina um identificador para o grupo.

Panel

 Use este componente, do grupo Standard, para colocar um painel no formulário onde outros componentes podem ser inseridos. O componente Panel é usado normalmente para Barras de Ferramentas, de Atalhos, e Barras de Status.

A propriedade *Alignment* modifica o alinhamento do texto inserido no componente. A propriedade *BevelInner* determina o estilo da moldura interna. A propriedade *BevelOuter* determina o estilo da moldura externa.

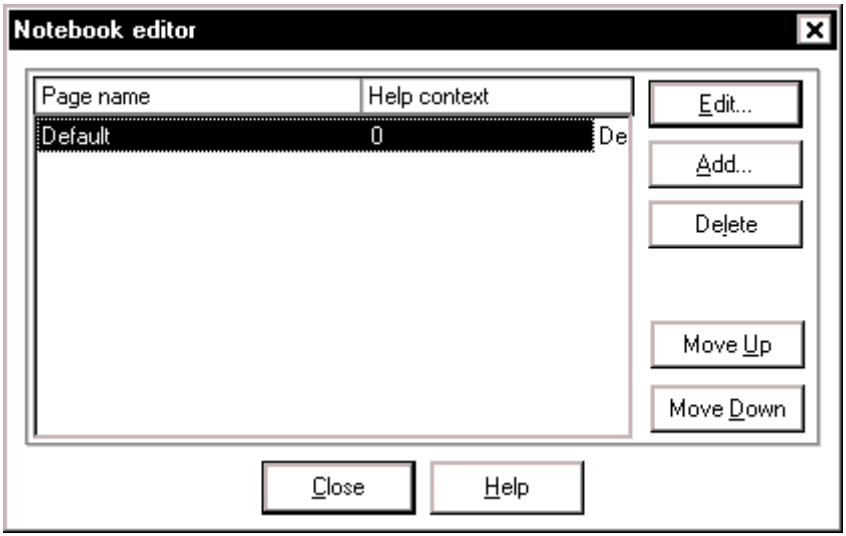
A propriedade *BorderStyle* especifica o estilo de borda. Você pode optar por uma borda simples (Single) ou por nenhuma borda (None). A propriedade *BorderWidth* determina a largura da borda em pixels.

Criando Páginas no Formulário

TabbedNotebook

Use este componente, do grupo Win 3.1, para criar múltiplas páginas no formulário, acessadas por abas (tabs).

A propriedade *Pages* adiciona páginas ao TabbedNotebook, cada uma com um nome. Para criar a lista de páginas, no Object Inspector dê um duplo clique na coluna à direita da propriedade Pages. Aparece o Pages Editor:



Clique em *Edit* para mudar o nome da página Default. Clique em *Add* para adicionar novas páginas. Clique em *Close* quando terminar.

A propriedade *ActivePage* mostra o nome da página ativada. A propriedade *PageIndex* mostra o índice da página ativada. Se você constrói um TabbedNotebook com três páginas, a primeira terá índice 0, a segunda índice 1, e a terceira índice 2.

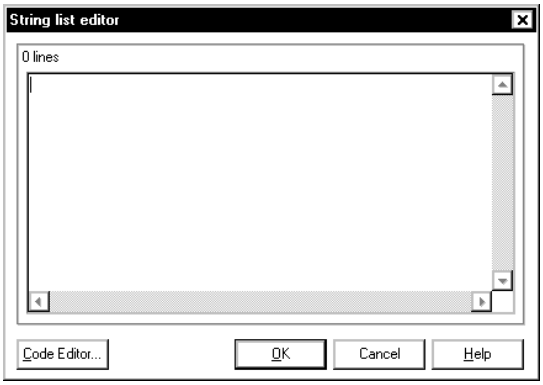
A propriedade *Align* posiciona a página na área cliente. A propriedade *TabsPerRow* determina o número de abas que podem aparecer em uma linha no topo do componente.

O método *SetTabFocus(Index: Integer)* muda a página ativa do componente. O parâmetro Index é o valor da propriedade PageIndex da página, que indica sua posição no array criado na propriedade Pages.

TabControl


Use o componente TabControl, do grupo Win32, para apresentar ao usuário opções de tabs (abas) que ele pode clicar.

A propriedade *Tabs* cria uma lista de tabs, ou seja, dos textos que as abas conterão. Quando você dá um duplo clique sobre esta propriedade, aparece o quadro abaixo:



Entre com a lista de abas, e clique no botão OK.

A propriedade *TabIndex* mostra o índice do tab corrente. O método *SelectNext(Direção: Boolean):* seleciona o próximo tab, rolando a estrutura de abas se necessário. O valor do parâmetro Direção determina se a aba à direita ou à esquerda será selecionada. Se Direção for igual a *True*, a aba da direita será selecionada.

PageControl

 Use o componente PageControl, do grupo Win32, para ter múltiplas páginas, uma acima da outra. Você pode deixar o usuário selecionar as páginas utilizando abas, menus, ou caixas de listas.

Para inserir páginas no componente, clique com o botão direito sobre o componente e escolha a opção *New Page*. Na realidade, você cria então um componente TabSheet, cuja propriedade *Caption* determina o identificador da aba.

A propriedade *ActivePage* determina a página ativada. A propriedade *Align* posiciona a página na área cliente.

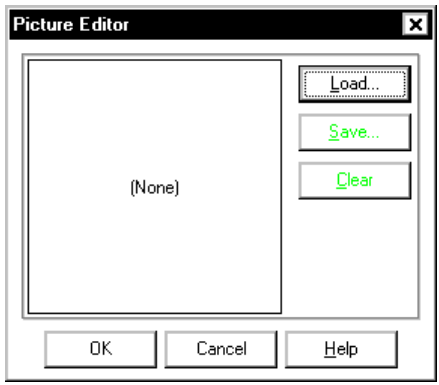
O evento *onChange* ocorre logo depois que uma nova página se torna ativa.

Inserindo Gráficos

Image

 Use o componente Image, do grupo Additional, para mostrar uma imagem gráfica em um formulário. Este componente suporta os seguintes formatos de figura: bitmaps (.BMP), metafiles (.WMF), icons (.ICO) e JPEG (padrão da Internet).

Para adicionar uma imagem em seu formulário, no projeto deste, adicione um componente Image a seu formulário, e dê um duplo clique no componente imagem. O Picture Editor aparece:



Escolha o botão *Load*. Na caixa de diálogo Abrir, selecione a imagem que você quer mostrar, então clique OK. O Picture Editor mostra a imagem selecionada. Clique OK para aceitar a imagem e sair do Picture Editor. A imagem aparece no formulário. Esta imagem passa a ser o valor da propriedade *Picture* do componente Image.

Para enquadrar a imagem no tamanho do componente Image, mude a propriedade *Stretch* para *True*.

A propriedade *Center* determina se a imagem será centralizada no componente. Se for igual a *True*, a imagem será centralizada (valor default).

Shape


Use o componente Shape, do grupo Additional, para adicionar formatos gráficos como círculos, quadrados, etc, a seus formulários.

A propriedade *Brush* muda a cor de preenchimento do formato. A propriedade *Pen* muda o estilo de borda do formato.

A propriedade *Shape* muda o formato do componente. O componente pode assumir um dos seguintes formatos: *stCircle* (círculo), *stEllipse* (elipse), *stRectangle* (retângulo),

stRoundRect (retângulo com as pontas arredondadas), *stRoundSquare* (quadrado com as pontas arredondadas), *stSquare* (quadrado).

PaintBox


Este componente, do grupo System, oferece uma área retangular no formulário para desenho, impedindo que o usuário desenhe fora desta área. É útil na construção de Editores de Desenho, como o Paintbrush.

A superfície de desenho do componente é representada pela propriedade *Canvas*.

Controlando o Tempo

Timer


Use o componente Timer, do grupo Additional, para garantir que uma determinada ação ocorra após um período de tempo.

Use a propriedade *Interval* para determinar de quanto em quanto tempo determinada ação ocorrerá. No evento *OnTimer*, defina a ação a ocorrer quando o período de tempo definido na propriedade Interval passar.

ProgressBar


Use o componente ProgressBar, do grupo Win32, para acompanhar a execução de uma procedure dentro da aplicação.

Use a propriedade *Position* para determinar a posição atual da marcação. Use a propriedade *Step* para determinar de quanto em quanto mudará a marcação.

No evento *OnTimer*, defina a ação a ocorrer quando o período de tempo definido na propriedade Interval passar.

Inserindo Tabelas

StringGrid

Este componente, do grupo Additional, permite que você entre dados do tipo string em uma tabela.

DrawGrid


Este componente, do grupo Additional, permite que você entre dados de diversos tipos em uma tabela.

Propriedades dos Componentes StringGrid e DrawGrid

Entre as propriedades dos componentes StringGrid e DrawGrid temos:

Cols: acessa as strings de determinada coluna.

Rows: acessa as strings de determinada linha.

Col: especifica a coluna da célula que tem o foco.

Row: especifica a linha da célula que tem o foco.

ColCount: determina o número de colunas na grade.

RowCount: determina o número de linhas na grade.

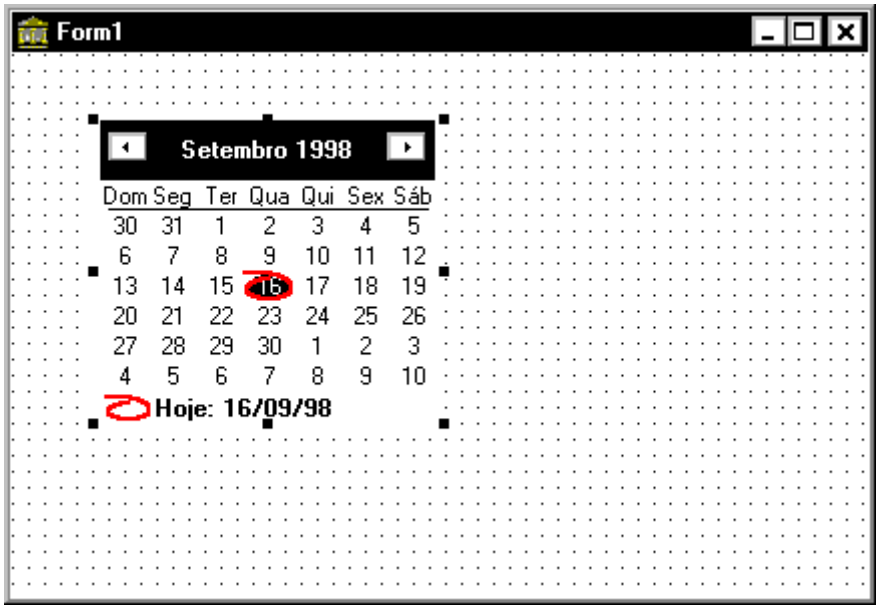
Controles do Windows 98

O Delphi 4 já oferece controles típicos do Windows 98, para você deixar seus aplicativos com um

estilo bastante atual. Estre os componentes oferecidos pelo Delphi podemos citar: MonthCalendar e PageScroller.

MonthCalendar

Este componente, da página de componentes Win32, oferece um calendário para que o usuário selecione uma data, ou uma região de datas:



A propriedade *Date* determina a data que está marcada no calendário.

PageScroller

Este componente, da página de componentes Win32, oferece uma área de visualização para janelas compridas, como barras de ferramentas. Com a ajuda deste controle, a barra pode ser rolada, horizontal ou verticalmente.

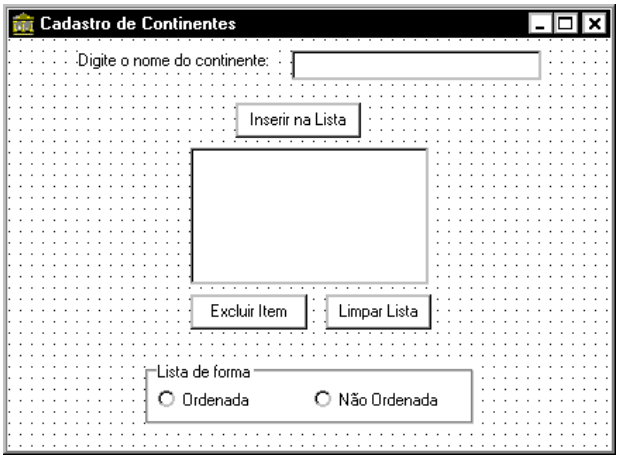
Delphi 4

Sistema de Acompanhamento Geográfico

Parte V

O Formulário de Cadastro de Continentes

Vamos acrescentar ao Sistema de Acompanhamento Geográfico um novo formulário, que gerenciará um cadastro de continentes, e terá o seguinte formato:



Para isso, siga os passos abaixo:

1) Abra o projeto SAG.dpr.

2) No menu *File*, escolha a opção *New Form*. Um novo formulário é adicionado ao projeto.

3) Modifique as seguintes propriedades:

Propriedade	Valor
Caption	Cadastro de Continentes
Name	frmCadastroContinentes
Position	poScreenCenter

4) Insira no formulário um componente Label. Modifique a propriedade Caption para *Entre com o nome do continente:*

5) Ao lado da etiqueta, insira um componente Edit, modificando as seguintes propriedades:

Propriedade	Valor
Name	EditContinente
Text	*vazia*

6) Insira no formulário um componente Button, mudando as seguintes propriedades:

Propriedade	Valor
Caption	Inserir na Lista
Name	BtnInserir

7) Insira um componente ListBox, abaixo do botão, mudando sua propriedade Name para *ListaContinentes*.

8) Insira no formulário um outro componente Button, abaixo do ListBox. Mude as seguintes propriedades:

Propriedade	Valor
Caption	Excluir Item
Enable	False
Name	BtnExcluirItem

9) Insira no formulário um outro componente Button, ao lado do anterior. Mude as seguintes propriedades:

Propriedade	Valor
Caption	Limpar Lista
Name	BtnLimparLista

10) Abaixo dos botões, insira um componente RadioGroup, mudando as seguintes propriedades:

Propriedade	Valor
Caption	Lista de forma
Name	RGOrdenaLista
Columns	2
Items	Ordenada, Não Ordenada

Vamos montar o código:

1) Ao clicarmos no botão Inclui na Lista, o continente digitado na caixas de edição é inserido no ListBox, através do método *Add*. Depois, limpamos a caixa de edição através do método *Clear*, e mandamos o foco para ela, através do método *SetFocus*:

```
procedure TfrmCadastroContinentes.BtnInserirClick(
Sender: TObject);
begin
   ListaContinentes.Items.Add(
     EditContinente.Text);
   EditContinente.Clear;
   EditContinente.SetFocus;
end;
```

2) Ao clicarmos em um item da lista o botão Excluir Item é habilitado, através da propriedade *Enabled*:

```
procedure
TfrmCadastroContinentes.ListaContinentesClick(
Sender: TObject);
begin
     BtnExcluirItem.Enabled := True;
end;
```

3) Ao clicarmos no botão Excluir Item, o continente selecionado no ListBox é excluído da lista, através do método *Delete*:

```
procedureTfrmCadastroContinentes.BtnExcluirItemClick(
     Sender: TObject);
begin
     ListaContinentes.Items.Delete(
       ListaContinentes.ItemIndex)
end;
```

4) Ao clicarmos no botão Limpar Lista, todo o contéudo do ListBox é excluído:

```
procedure
TfrmCadastroContinentes.BtnLimparListaClick(Sender:
TObject);
begin
     ListaContinentes.Clear;
end;
```

5) Ao clicarmos na opção Ordenada, do Radio Group, a lista será ordenada alfabéticamente, através da propriedade Sorted. A partir deste momento, todos os itens inseridos na lista serão pré-ordenados. Se clicarmos no botão Não Ordenada, os itens serão ordenados conforme forem entrados:

```
procedure
TfrmCadastroContinentes.RGOrdenaListaClick(Sender:
TObject);
begin
     if RGOrdenaLista.ItemIndex = 0 then
        ListaContinentes.Sorted := True
     else
        ListaContinentes.Sorted := False;
end;
```

Vamos salvar a aplicação:

1) No menu *File*, escolha a opção *Save All*.

2) Dê o nome *UnitCadastroContinentes* para novo formulário.

Para testar este formulário, você deve modificar as opções do projeto para que o formulário FrmCadastroContinentes seja o formulário principal. Para isso, selecione no menu *Project* a opção *Options*, aba *Forms*. Na caixa de lista Main Form, selecione o formulário *FrmCadastroContinentes*.

Delphi 4

8

Trabalhando com Menus

Menus

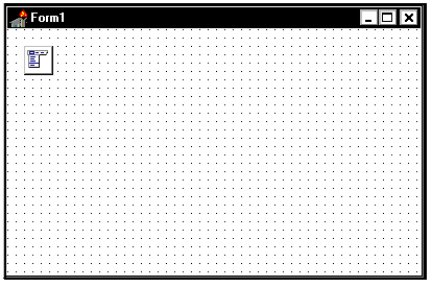
O Delphi lhe oferece uma forma rápida de incluir menus em suas aplicações. Você simplesmente digita os menus diretamente na janela do Gerador de Menus. Você pode clicar e arrastar os itens do menu para rearranjá-los em seu projeto.

Você não precisa executar o programa para ver os resultados: seu menu está sempre visível no formulário, com a aparência que terá durante a execução do programa.

Abrindo o Gerador de Menus

Para abrir o Gerador de Menus, siga os passos abaixo:

◆ selecione o componente *MainMenu*  , do grupo Standard da Paleta de Componentes, e adicione-o ao formulário:

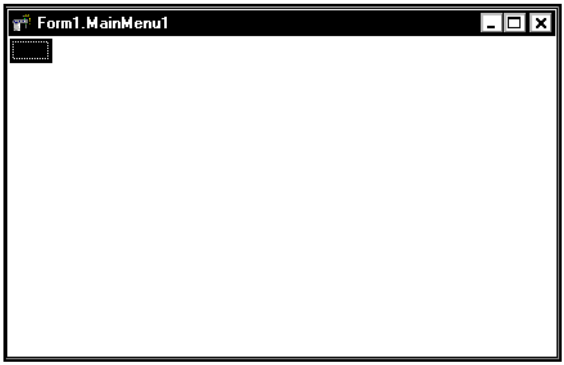


◆ mantendo o componente selecionado, escolha um dos métodos abaixo:

◆ no formulário, dê um duplo clique no componente; ou

◆ no Object Inspector, selecione a propriedade Items, e dê um duplo clique na coluna de valor.

A janela do Gerador de Menus aparece:



Adicionando Itens ao Menu

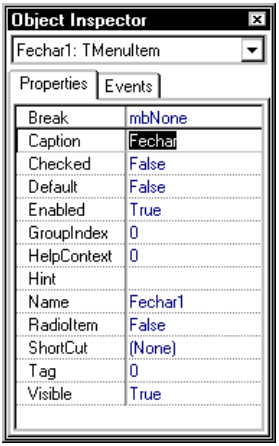
Para adicionar itens ao menu:

- após abrir a janela do Gerador de Menus, o primeiro item do menu (em branco) está iluminado, assim como a propriedade Caption no Object Inspector;
- na propriedade *Caption*, digite na coluna do valor um título para este item do menu;
- aperte a tecla *Enter*. O cursor se move para o lado. Use as teclas para mover-se para o lado (inserindo novos itens na barra de menus) ou para baixo (incluindo subitens);
- entre os novos títulos na propriedade Caption, repetindo os passos acima.

Para especificar uma tecla de atalho, coloque um "&" na frente da letra apropriada. Por exemplo, para utilizar a tecla de atalho A para o comando Abrir, digite na propriedade Caption "&Abrir".

Para adicionar uma barra separadora, digite um hífen (-) na propriedade Caption do item do menu.

Note que, cada vez que você cria um item de menu, é criado no Object Inspector um objeto do tipo TMenuItem:



O nome deste objeto é dado adicionando-se o nome do item de menu dado por você a um determinado número. Na tela acima, o objeto TMenuItem foi identificado como Fechar1, já que o item tem a denominação Fechar, e é o primeiro item com este nome.

Como veremos, as propriedades destes itens de menu podem ser modificadas, através do Object Inspector.

Inserindo um Item no Menu

Para inserir um novo item no interior do menu já construído:

- posicione o cursor sobre um item do menu;
- aperte a tecla *Ins*;
- aparece um espaço em branco;
- clique sobre a propriedade Caption e entre com o nome do item do menu.

O novo item de menu é inserido a esquerda do item selecionado, se estiver na barra de menu; os itens de menu na lista são inseridos acima do item selecionado.

Excluindo um Item do Menu

Para excluir um item do menu:

- posicione o cursor sobre o item do menu;
- aperte a tecla *Del*.

Dica: Você não pode apagar o demarcador que aparece abaixo ou ao lado do item do menu. Este demarcador não aparecerá em seu menu, quando você executar seu programa.

Teclas de Atalho

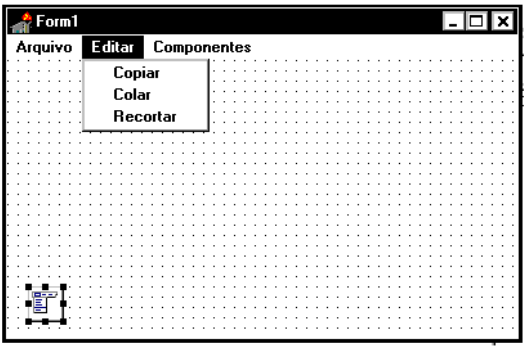
As teclas de atalho permitem ao usuário realizar uma ação sem acessar o menu diretamente, digitando uma combinação de teclas.

Para especificar uma tecla de atalho para um item do menu:

- selecione na lista na parte superior do Object Inspector o item de menu (representado, como vimos, como um objeto TMenuItem);
- na propriedade ShortCut selecione as teclas de atalho, escolhendo uma combinação na lista oferecida.

Criando Menus Edentados

Você pode criar menus com vários níveis de subitens. São os menus edentados.



Para criar um sub-menu:

- selecione o item do menu ao lado do qual você quer criar um sub-menu;
- pressione *Ctrl+setaDireita* para criar o demarcador do sub-menu;
- digite o nome do item do submenu;
- pressione a tecla *Enter* para criar o próximo item do sub-menu;
- repita os passos acima até terminar seu sub-menu;
- pressione *Esc* para retornar ao item superior.

Movendo Itens do Menu

Durante o projeto de seu menu, no Gerador de Menus, você pode mover qualquer item de menu para um lugar diferente da barra de menus, ou para outro menu, simplesmente arrastando os itens para os novos lugares. Enquanto você os está arrastando, o cursor muda de formato para indicar se você pode posicionar o cursor na nova posição.

Quando você move um item de menu para um lugar diferente na barra de menus, todos os subitens se movem juntos.

Quando você move um item de menu para fora da barra de menus, seus subitens se tornam um sub-menu.

Vendo o Menu

Você pode ver o menu no formulário durante o projeto, sem precisar rodar sua aplicação. Para isto:

- feche a janela do Gerador de Menus; ou
- clique no formulário; ou
- do Object Inspector, selecione o formulário na lista de seleção de objetos, clicando sobre a seta que fica ao lado da caixa que contém o nome dos objetos.

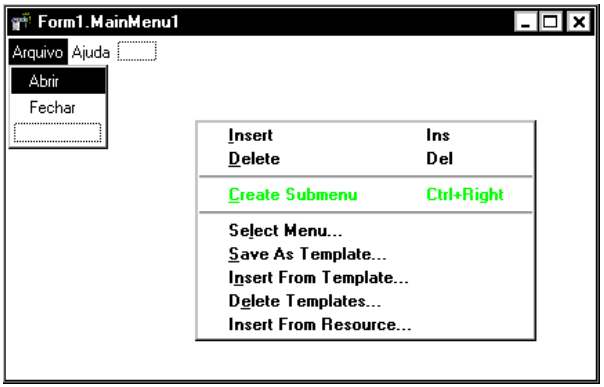
O menu aparece exatamento como estará quando você executar o programa.

Para retornar à janela do Gerador de Menus, no formulário, dê um duplo clique sobre o componente MainMenu.

O Menu Pop-up do Gerador de Menus

O Gerador de Menus contém um menu rápido do tipo pop-up, que oferece acesso facilitado a maioria dos comandos do Gerador de Menus.

Para acessá-lo, clique no botão direito do mouse, estando dentro do Gerador de Menus.



Através deste menu rápido você pode inserir e deletar itens de menu, criar submenus, selecionar ou apagar um modelo, e salvar seu menu como modelo.

Desabilitando Itens de Menu

Para desabilitar itens de menu, não permitindo que o usuário o utilize, basta modificar a propriedade Enabled do item do menu para *False*, através do Object Inspector.

Propriedades Específicas do Componente MainMenu

Entre as propriedades específicas do componente MainMenu temos:

AutoMerge: determina se os menus dos demais formulários serão agrupados ao do formuláiro principal. O default é *False* (sem agrupamento).

Items: contém os itens de um menu principal. Para se referir ao primeiro item de um menu chamado Menu1, faça:

```
Menu1.Items[0]
```

Métodos Específicos do Componente MainMenu

Entre os métodos específicos do componente MainMenu temos:

Merge(Menu): agrupa um outro menu principal ao do formulário atual. O parâmetro Menu é o menu que você quer agrupar.

Unmerge(Menu): reverte a junção de dois menus em um. O parâmetro menu é o menu associado que você não quer mais que seja agrupado.

Propriedades Específicas do MenuItem

Os itens do menu (objetos da classe TMenuItems) também possuem propriedades e métodos específicos:

Break: permite que você quebre um menu muito longo em colunas.

Checked: determina se o item está selecionado. Se *true*, o item está selecionado. Disponível só durante a execução.

Count: contém o número de subitens do item do menu.

Items: especifica os sub-itens do item do menu.

ShortCut: determina as teclas de atalho utilizadas para o acesso do usuário pelo teclado.

Métodos Específicos do Componente MenuItem

Entre os métodos específicos do componente MainMenu temos:

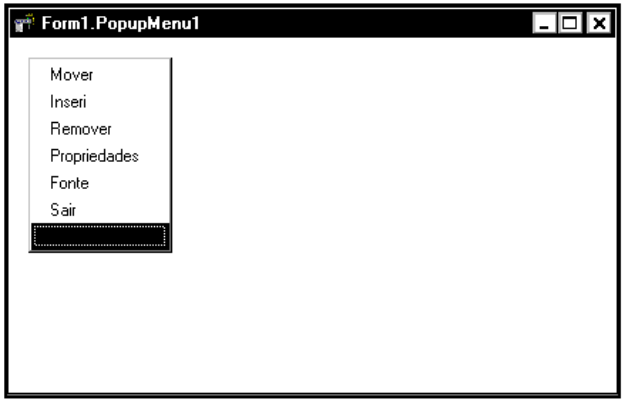
IndexOf(Item): determina a posição do item do menu dentro do menu, retornando um valor inteiro.

Add(Item): adiciona um sub-item ao item de menu, no fim da lista.

Menus Pop-up

O Componente PopupMenu


O componente PopupMenu, da página Standard da Paleta de Componentes, permite a criação de menus que aparecem quando o usuário clica o botão direito do mouse.



Criando Menus Pop-up

Criar um menu pop-up é semelhante a criar um menu principal:

- dê um duplo clique no componente PopUpMenu, da Barra de Componentes Standard, inserindo-o no formulário;
- dê um duplo clique sobre o componente inserido no formulário;
- digite o nome do primeiro item, na propriedade Caption do Object Inspector;
- tecle *Enter*;
- digite o nome dos demais itens.

Para tornar um menu Popup disponível para o componente a que ele diz respeito, atribua o nome do menu pop-up a propriedade PopupMenu do formulário ou componente.

Propriedades Específicas

Entre as propriedades específicas do componente PopupMenu temos:

Alignment: determina a posição do menu pop-up em relação ao cursor do mouse. Você pode escolher *paLeft* (o canto superior esquerdo do menu aparece sob o pointeiro do mouse); *paCenter* (o canto superior central do menu aparece sob o pointeiro do mouse) ou *paRight* (o canto superior direito do menu aparece sob o pointeiro do mouse). O valor default é *paLeft*.

Autopopup: determina se o menu pop-up aparecerá quando o usuário clicar o botão direito do mouse. O default é *True*, ou seja, o menu aparecerá.

Items: especifica os itens do menu pop-up.

PopupComponent: é o nome do componente que chamou o menu pop-up. Disponível só durante a execução.

Métodos Específicos

Entre os tópicos específicos do componente PopUpMenu temos:

Popup(X,Y: integer): mostra um menu pop-up nas coordenadas estabelecidas por X e Y.

Delphi 4

Sistema de Acompanhamento Geográfico

Parte VI

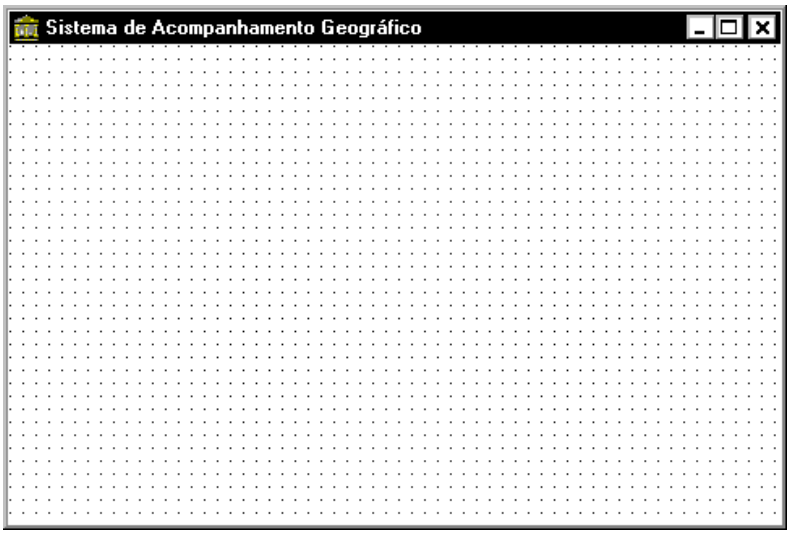
Construindo o Menu

Vamos acrescentar o menu e uma Toolbar ao Sistema de Acompanhamento Geográfico. Isto será feito no formulário frmMenu, criado anteriormente.

Para isso, siga os passos abaixo:

1) Abra o projeto SAG.dpr.

2) No Project Manager, dê um duplo clique sobre a unit correspondente ao formulário: UnitMenu. O formulário é mostrado:

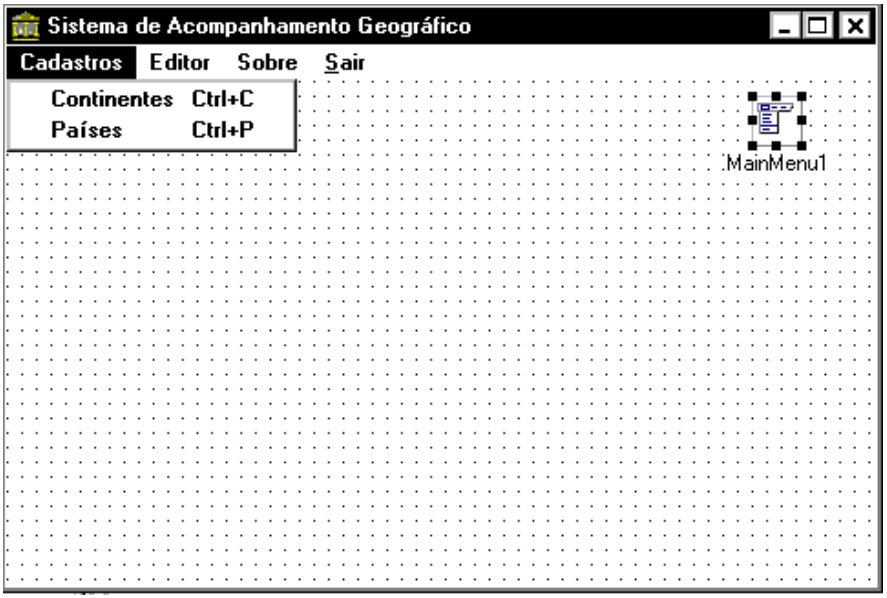


3) Insira um componente MainMenu no formulário. Através do Editor de Menus, insira os seguintes itens, com suas respectivas propriedades:

Tipo	Propriedade	Valor
Item	Caption	Cadastros
	Name	IM_Cadastros

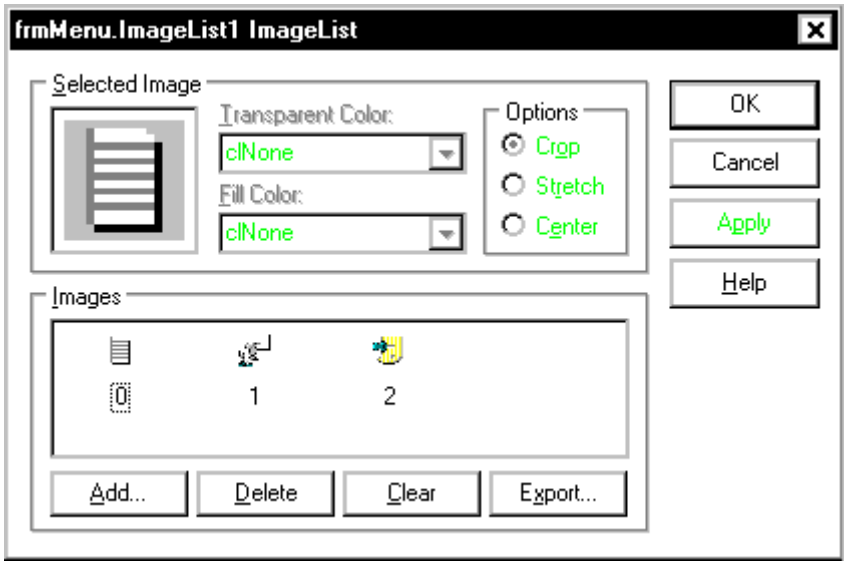
Tipo	Propriedade	Valor
SubItem	Caption	Continentes
	Name	IM_Continentes
	ShortCut	Ctrl+C
SubItem	Caption	Países
	Name	IM_Paises
	ShortCut	Ctrl+P
Item	Caption	Editor
	Name	IM_Editor
Item	Caption	Sobre
	Name	IM_Sobre
Item	Caption	&Sair
	Name	IM_Sair

4) Saia do Editor de Menus e veja como está seu menu:



Agora, vamos inserir uma Toolbar, que conterá botões de atalho para os cadastros de continentes e países, e para o editor. Inicialmente devemos criar uma lista das imagens que serão mostradas nos botões. Para isso, usaremos o componente ImageList.

5) Insira no formulário um componente ImageList. Dê um duplo clique sobre o componente, e insira três imagens a sua escolha para colocar nos botões, a primeira para o Cadastro de Continentes, a segunda para o Cadastro de Países, e a terceira para o Editor:

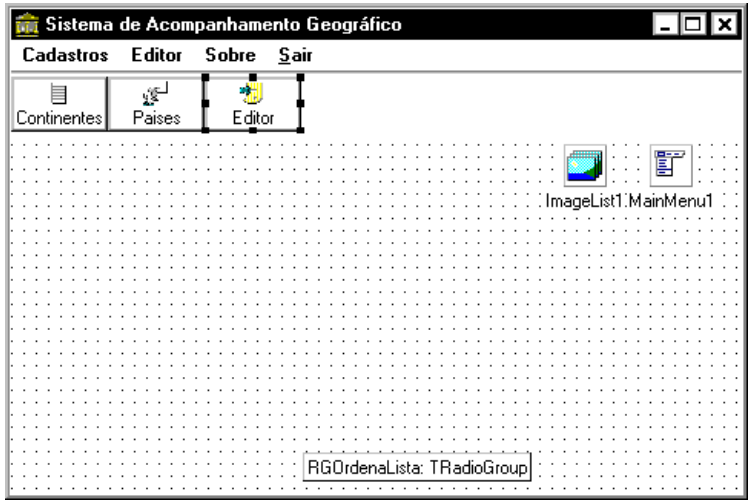


6) Insira um componente Toolbar, alinhado na parte superior do formulário. Mude então as seguintes propriedades:

Propriedade	Valor
Images	ImageList1
ButtonHeight	36
ButtonWidth	65
ShowCaptions	True

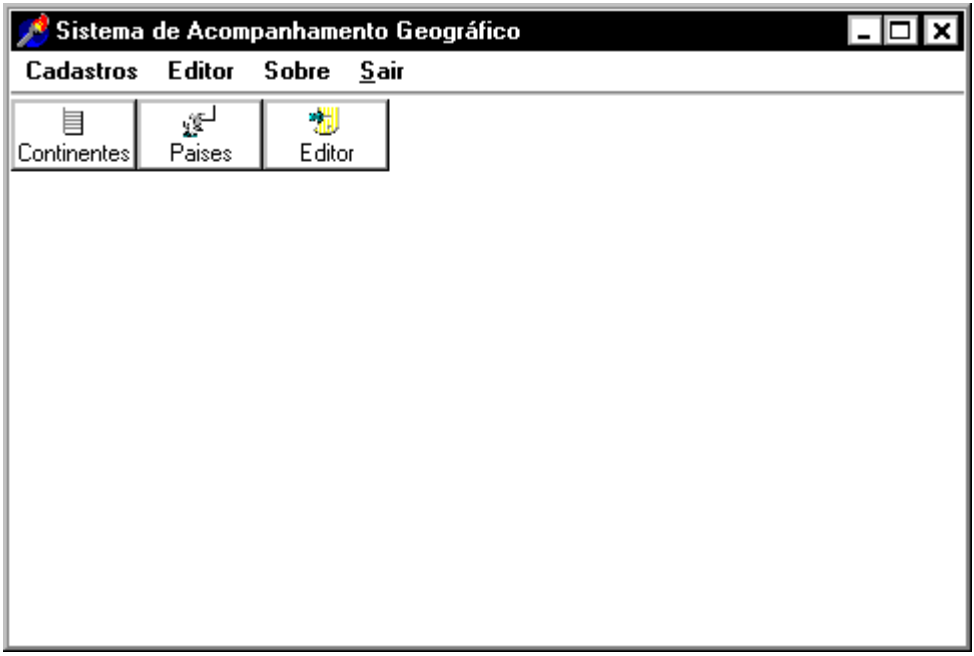
7) Clique com o botão direito do mouse sobre o componente e escolha a opção *New Button*. Um botão é inserido na barra. Repita esta operação duas vezes. Mude as seguintes propriedades dos botões:

Propriedade	Valor
Caption	Continentes
ImageIndex	0
Hint	Abre o formulário de Continentes
ShowHint	True
Caption	Países
ImageIndex	1
Hint	Abre o formulário de Países
ShowHint	True
Caption	Editor
ImageIndex	2
Hint	Abre o Editor
ShowHint	True



8) Salve o projeto, através do menu *File* opção *Save All*.

9) Para executar o projeto e ver o formulário de menu na tela, lembre-se de colocá-lo como formulário principal, através do menu *Project*, opção *Options*, aba *Forms*. No quadro Main form, escolha o formulário *frmMenu*. Após executar o aplicativo, veja como está o formulário do menu:



10) Ainda não há código associado aos botões ou itens de menu. A associação será feita na última parte da aplicação.

Delphi 4

9

Quadros de Diálogo

A Função MessageDlg

Você monta um quadro de mensagem através da função MessageDlg. A função MessageDlg só permite um diálogo restrito; você coloca uma mensagem em uma janela, e o usuário fica restrito a comunicar-se de volta por meio de botões.

Sintaxe:

```
MessageDlg(Mensagem: string, Tipo: TMsgDlgTypes, Botões: TMsgDlgButtons, HelpContext: Longint) : Word;
```

Onde:

Mensagem: é a mensagem que você quer mostrar, entre aspas simples.

Tipo: indica o ícone que você deseja no quadro. Alguns tipos seriam:

mtWarning: um ponto de exclamação amarelo.

mtError: um sinal de 'pare' vermelho.

mtInformation: um 'i' azul.

mtConfirmation: um ponto de interrogação verde.

mtCustom: sem bitmap. O título do quadro de mensagem é o nome do arquivo executável da aplicação.

Botões: indica os botões que você quer no quadro. Os nomes dos botões devem vir entre colchetes, separados por vírgula. Alguns botões seriam:

- mbYes
- mbNo
- mbOK
- mbCancel
- mbHelp
- mbAbort
- mbRetry
- mbIgnore
- mbAll

Além desses valores para botões, temos também três grupos de botões:

mbYesNoCancel: conjunto com botões Yes, No e Cancel.

mbOkCancel: conjunto com botões OK e Cancel.

mbAbortRetryIgnore: conjunto com botões Abort, Retry e Ignore.

Os nomes destes grupos não precisam vir entre colchetes, na definição da função.

HelpContext: é o número do help associado a mensagem. Se não houver, coloque 0.

Esta função retorna um valor do tipo word, correspondendo ao nome do botão selecionado. As respostas possíveis são:

- mrNone
- mrYes
- mrNo
- mrOK
- mrCancel
- mrHelp
- mrAbort

- mrRetry
- mrIgnore
- mrAll

A Função MessageDlgPos

O Delphi oferece também a função MessageDlgPos. Esta função é semelhante a MessageDlg, mas permite escolher a posição onde será mostrado o quadro de mensagem.

Sintaxe:

```
MessageDlgPos(Mensagem: string; Tipo: TMsgDlgTypes; Botões: TMsgDlgButtons; HelpContext: Longint; X, Y: integer) : Word;
```

Onde:

X, Y: correspondem as coordenadas do canto superior esquerdo do quadro de mensagem.

Os demais parâmetros são semelhantes aos da função MessageDlg, vista anteriormente.

ShowMessage

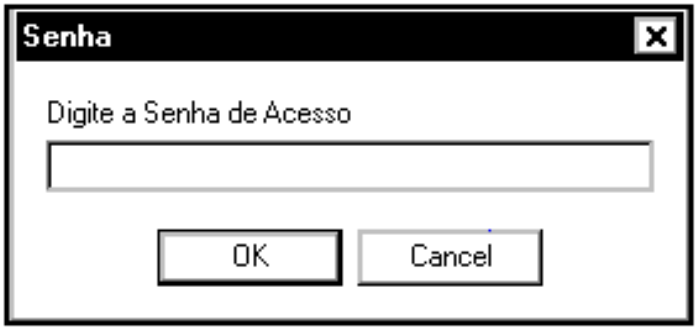
Esta procedure mostra um quadro de mensagem simples, contendo apenas a mensagem e um botão de OK.

Sintaxe:

```
ShowMessage(const Msg: string);
```

Msg é a string com a mensagem. O nome do arquivo executável da aplicação é o título do quadro de mensagem.

A Função InputBox



Você cria quadros de entrada através da função InputBox(). Esta função fornece um quadro padrão, que permite a entrada de dados.

Sintaxe:

InputBox (título, prompt, default: string) : string;

Onde:

título: é a legenda que queremos dar ao quadro de entrada;

prompt: é o aviso que mostramos para indicar que tipo de entrada é desejada;

default: é o string original que aparece inicialmente na Caixa de Edição do quadro (ou seja, é o que é retornado caso o usuário não faça nenhuma entrada);

A função InputBox retorna um valor do tipo string, correspondendo ao texto digitado na caixa de edição do quadro.

Quadros de Diálogo Padrões

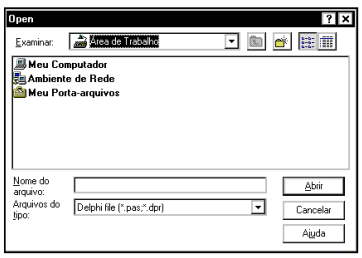
O Delphi oferece diversos componentes que representam os quadros de diálogo padrões do Windows:

Componente	Quadro
OpenDialog	Abrir
SaveDialog	Salvar
OpenPictureDialog	Abrir Figura
SavePictureDialog	Salvar Figura
FontDialog	Fontes
ColorDialog	Cores
PrintDialogo	Imprimir
PrintSetupDialog	Configurar Impressão
FindDialog	Encontrar
ReplaceDialog	Substituir

Estes componentes podem ser inseridos em seu formulário como os demais componentes. Estão agrupados na página *Dialogs*, do conjunto de componentes.

Por serem componentes, possuem propriedades e métodos associados, e respondem a eventos. Mudando algumas propriedades, você pode adaptar os quadros padrões para atender as suas necessidades.

Quadro de Diálogo Abrir (OpenDialog)



Para adaptar este quadro de diálogo, você pode modificar as seguintes propriedades:

DefaultExt: determina uma extensão default para arquivos que aparecerão na lista de arquivos.

FileEditStyle: determina se irá ter um Quadro de Lista (fsEdit) ou Quadro Combo (fsComboBox) para o usuário selecionar o arquivo.

FileName: armazena o nome do arquivo escolhido pelo usuário.

Filter: determina um filtro para os arquivos que aparecerão na lista de arquivos, estabelecendo as extensões destes. Para modificar esta propriedade, use a seguinte *Sintaxe:*

'Tipo de Arquivo|*.Extensão do Arquivo'

onde Tipo de Arquivo descreve o tipo de extensão do arquivo (por exemplo, Texto), e Extensão do Arquivo é uma combinação de três caracteres (por exemplo, TXT).

Files: contém os arquivos selecionados, se a subpropriedade ofAllowMultiselection, da propriedade Options, estiver com valor True. Disponível só em tempo de programação.

FilterIndex: determina qual das máscaras especificadas na propriedade Filter é a default.

InitialDir: determina de qual diretório serão listados os arquivos.

HistoryList: contém a lista de arquivos que foram digitados na caixa de nome de arquivo, quando esta caixa é do estilo combo. Você armazena estes arquivos em uma variável do tipo TStringList, criando um histórico dos arquivos abertos ou salvos, e, quando o quadro de diálogo aparecer, atribui a propriedade List o objeto TStringList.

Options: determina as opções que estão disponíveis para o quadro de diálogo.

Por exemplo, você pode escolher *ofAllowMultiSelect*, para permitir que o usuário selecione mais de um arquivo; *ofCreatePrompt*, que mostra um quadro de mensagem caso o usuário tente abrir um arquivo que não exista. Veja as demais opções no Help do Delphi.

Siga a seguinte sintaxe*:*

```
Quadro.Options := [ofAllowMultiSelect,
ofCreatePrompt];
```

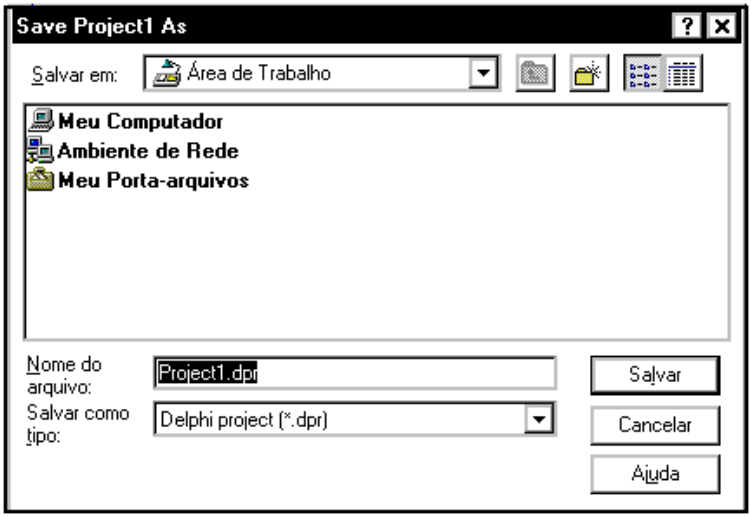
Title: determina o título do quadro de diálogo.

Para mostrar o quadro de diálogo, utilize o método *Execute*:

```
Quadro.Execute
```

Este método retorna *True* se o quadro for mostrado, e um arquivo for selecionado.

Quadro de Diálogo Salvar (SaveDialog)



Para adaptar este quadro de diálogo, você pode modificar as seguintes propriedades:

DefaultExt: determina uma extensão default para arquivos que aparecerão na lista de arquivos.

FileEditStyle: determina se irá ter um Quadro de Lista (fsEdit) ou Quadro Combo (fsComboBox) para o usuário selecionar o arquivo.

FileName: armazena o nome do arquivo escolhido pelo usuário.

Filter: determina um filtro para os arquivos que aparecerão na lista de arquivos, estabelecendo as extensões destes. Para modificar esta propriedade, use a seguinte *Sintaxe:*

'Tipo de Arquivo|*.Extensão do Arquivo'

onde Tipo de Arquivo descreve o tipo de extensão do arquivo (por exemplo, Texto), e Extensão do Arquivo é uma combinação de três caracteres (por exemplo, TXT).

FilterIndex: determina qual das máscaras especificadas na propriedade Filter é a default.

InitialDir: determina de qual diretório serão listados os arquivos.

HistoryList: contém a lista de arquivos que foram digitados na caixa de nome de arquivo, quando esta caixa é do estilo combo. Você armazena estes arquivos em uma variável do tipo TStringList, criando um histórico dos arquivos abertos ou salvos, e, quando o quadro de diálogo aparecer, atribui a propriedade List o objeto TStringList.

Options: determina as opções que estão disponíveis para o quadro de diálogo. Por exemplo, você pode escolher ofAllowMultiSelect, para permitir que o

usuário selecione mais de um arquivo; ofCreatePrompt, que mostra um quadro de mensagem caso o usuário tente abrir um arquivo que não exista. Veja as demais opções no Help do Delphi. Siga a seguinte *Sintaxe:*

```
Quadro.Options := [ofAllowMultiSelect,
ofCreatePrompt];
```

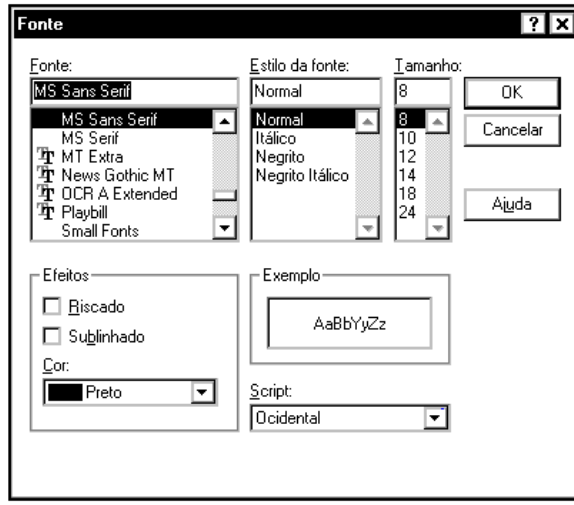
Title: determina o título do quadro de diálogo.

Para mostrar o quadro de diálogo, utilize o método *Execute*:

```
Quadro.Execute
```

Este método retorna *True* se o quadro for mostrado, e um nome de arquivo for digitado.

Quadro de Diálogo Fontes (FontDialog)



Para adaptar este quadro de diálogo, você pode modificar as seguintes propriedades:

Device: determina que saída é afetada pela mudança de fonte: a tela (fdScreen), a impressora (fdPrinter) ou ambas (fdBoth).

Font: é a fonte retornada pelo usuário quando usa o quadro de diálogo Fonte. É atribuido a uma variável TFont. Para armazenar o nome da nova fonte, faça:

Quadro.Font.Name.

Quando você usa a propriedade Font de um componente, esta recupera todas as modificações feitas no quadro Fonte.

MaxFontSize/MinFontSize: determina o maior e menor tamanho de fonte possível. O default é 0, que não limita tamanhos.

Options: determina algumas opções, tais como fdEffects, para mostrar a caixa de checagem de Efeitos (Effects) e lista de cores (Color list), no quadro de diálogo. Veja as demais opções no Help do Delphi.

Siga a seguinte sintaxe*:*

Quadro.Options := [fdEffects, fdAnsiOnly];

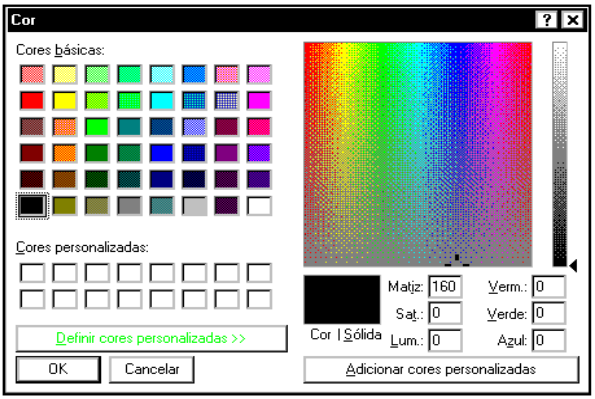
Para mostrar o quadro de diálogo, utilize o método *Execute*:

```
Quadro.Execute
```

Este método retorna *True* se o quadro for mostrado, e uma fonte for selecionada.

Quadro de Diálogo Cores (ColorDialog)

Para adaptar este quadro de diálogo, você pode modificar as seguintes propriedades:



Color: define a cor escolhida pelo usuário. Durante o projeto, você pode determinar a cor que estará marcada quando o usuário acessar o Quadro de Cores.

Options: determina algumas opções, tais como cdFullOpen, para mostrar as opções de cores customizadas, no quadro de diálogo. Veja as demais opções no Help do Delphi.

Siga a seguinte sintaxe*:*

```
Quadro.Options :=
[cdFullOpen, cdPreventFullOpen];
```

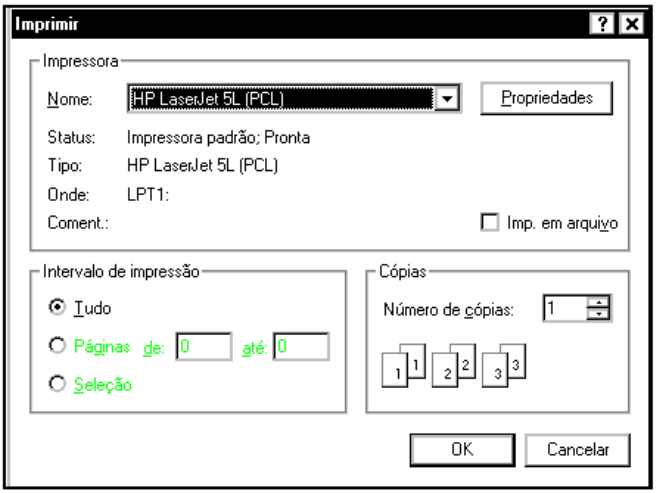
Para mostrar o quadro de diálogo, utilize o método *Execute*:

```
Quadro.Execute
```

Este método retorna *True* se o quadro for mostrado, e uma cor for selecionada.

Quadro de Diálogo Imprimir (PrintDialog)

Para adaptar este quadro de diálogo, você pode modificar as seguintes propriedades:



Collate: determina se a caixa de checagem Collate está marcada, ou seja, se Collate está selecionada.

Copies: determina o número de páginas que o usuário quer imprimir. O valor digitado no projeto é o valor default na caixa de edição Copies.

FromPage/ToPage: determina a página inicial e final que o usuário quer imprimir.

MinPage/MaxPage: define os números mínimo e máximo que o usuário pode especificar nas caixas Pages From e To

Options: determina algumas opções, tais como poPageNums, que mostra os botões de rádio Pages: From e To; poPrintToFile que adiciona uma caixa de checagem Print to File. Veja as demais opções no Help do Delphi.

Siga a seguinte sintaxe:

```
Quadro.Options := [poPageNums,
poPrintToFile];
```

PrintRange: especifica o que aparece com default em Print Range.

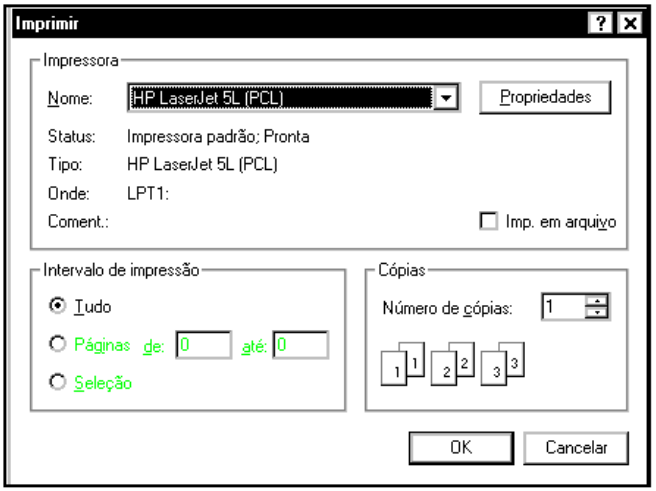
PrintToFile: determina se o usuário escolheu imprimir o trabalho em arquivo. Se True, o usuário marcou a caixa de checagem Print to File, e quer imprimir para arquivo.

Para mostrar o quadro de diálogo, utilize o método Execute:

```
Quadro.Execute
```

Este método retorna *True* se o quadro for mostrado e o usuário clicar no botão OK.

Quadro de Diálogo Configurar Impressão (PrinterSetupDialog)

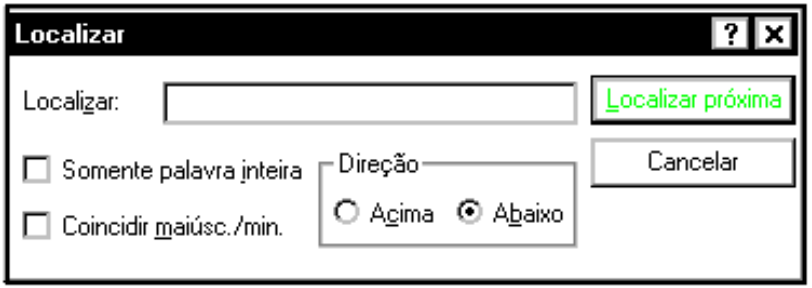


Para mostrar o quadro de diálogo, utilize o método *Execute*:

```
Quadro.Execute
```

Este método retorna *True* se o quadro for mostrado e o usuário clicar sobre o botão OK.

Quadro de Diálogo Encontrar (FindDialog)



Para adaptar este quadro de diálogo, você pode modificar as seguintes propriedades:

FindText: contém a string que o usuário quer encontrar. Você pode especificar um valor no projeto, que aparecerá na caixa de edição Find do Quadro de Diálogo.

Options: determina algumas opções, tais como frDown, que determina a direção da procura da posição do cursor até o fim do arquivo. Veja as demais opções no Help do Delphi.

Siga a seguinte sintaxe*:*

```
Quadro.Options := [frDown,frFindNext];
```

Position: determina onde o formulário aparece na tela.

Para mostrar o quadro de diálogo, utilize o método *Execute*:

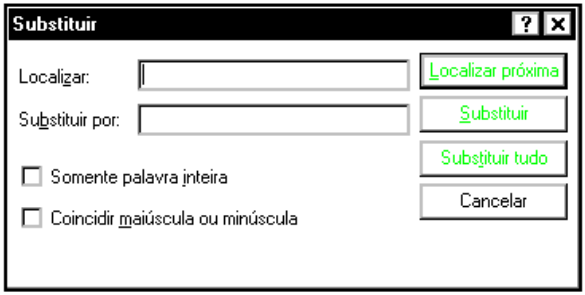
```
Quadro.Execute
```

Este método retorna *True* se o quadro for mostrado, e uma palavra escolhida para ser procurada.

Use o método *CloseDialog* para fechar o quadro:

```
Quadro.CloseDialog
```

Quadro de Diálogo Substituir (ReplaceDialog)



Para adaptar este quadro de diálogo, você pode modificar as seguintes propriedades:

FindText: contém a string que o usuário quer encontrar. Você pode especificar um valor no projeto, que aparecerá na caixa de edição Find do Quadro de Diálogo.

Options: determina algumas opções, tais como frDown, que determina a direção da procura da posição do cursor até o fim do arquivo. Veja as demais opções no Help do Delphi.

Siga a seguinte sintaxe:

```
Quadro.Options := [frDown,frFindNext];
```

ReplaceText: contém a string pela qual o usuário substituirá a palavra a procurar quando esta for encontrada.

Para mostrar o quadro de diálogo, utilize o método *Execute*:

```
Quadro.Execute
```

Este método retorna *True* se o quadro for mostrado, e uma palavra escolhida para ser substituída.

Use o método *CloseDialog* para fechar o quadro:

```
Quadro.CloseDialogs
```

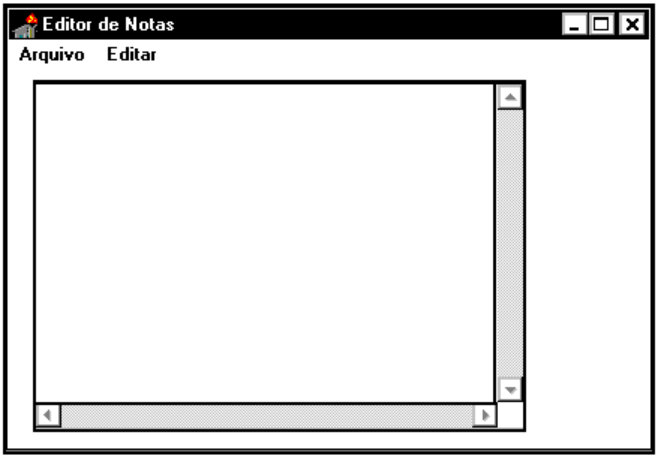
Delphi 4

Sistema de Acompanhamento Geográfico

Parte VII

O Editor de Notas

Vamos acrescentar ao Sistema de Acompanhamento Geográfico um Editor de Notas, para inserção e gravação de texto:



1) Abra a aplicação SAG.dpr.

2) Insira um novo formulário, através do menu *File*, opção *New Form*.

3) Mude as seguintes propriedades:

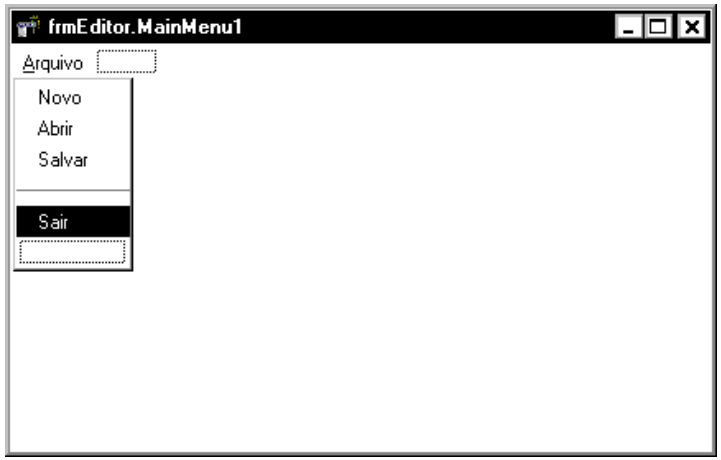
Propriedade	Valor
Caption	Editor de Notas
Name	frmEditor
Position	*poScreenCenter*

4) Insira no formulário um componente MainMenu. Dê um duplo clique sobre o componente, entrando no Gerador de Menus.

5) Entre então com os seguintes itens, mudando suas propriedades de acordo com o quadro abaixo:

Item	Propriedade	Valor
Item	Caption	&Arquivo
	Name	ÌM_Arquivo
Sub-item	Caption	Novo
	Name	IN_Novo
Sub-item	Caption	Abrir
	Name	IM_Abrir
Sub-item	Caption	Salvar
	Name	IM_Salvar
Sub-item	Caption	-
	Name	IM_Separador
Sub-item	Caption	Sair
	Name	IM_Sair

Seu menu estará assim:

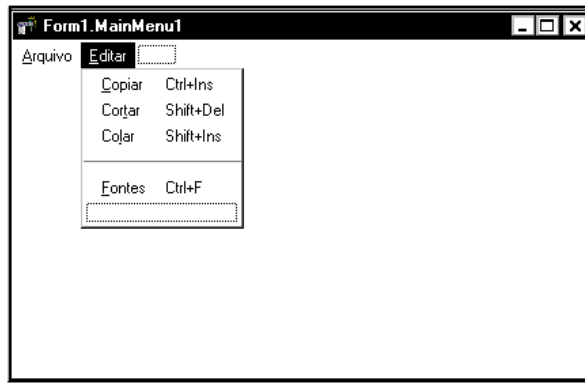


6) Clique sobre o marcador ao lado do item Arquivo, do menu.

7) Entre então com os seguintes itens, mudando suas propriedades de acordo com o quadro abaixo:

Item	Propriedade	Valor
Item	Caption	&Editar
	Name	ÌM_Editar
Sub-item	Caption	Copiar
	Name	IN_Copiar
	ShortCut	Ctrl+Ins
Sub-item	Caption	Recortar
	Name	IM_Recortar
	ShortCut	Shift+Del
Sub-item	Caption	Colar
	Name	IM_Colar
	ShortCut	Shift+Ins
Sub-item	Caption	-
	Name	IM_Separador
Sub-item	Caption	Fonte
	Name	IM_Fonte
	ShortCut	Ctrl+F

Seu menu estará assim:



8) Clique sobre o formulário, para ver o menu pronto.

9) Insira no formulário um componente Memo.

10) Mude a propriedade ScrollBars para *ssBoth* (ambas), e a propriedade Name para *MemoEditor*.

11) Insira no formulário um componente OpenDialog, do grupo Dialogs de componentes.

12) Insira no formulário um componente SaveDialog, do grupo Dialogs de componentes.

13) Insira no formulário um componente FontDialog, do grupo Dialogs de componentes.

Vamos montar o código:

1) Quando o usuário entra na aplicação, utilizamos o método Clear para limpar o conteúdo do Memo. Para criar este código, selecione o formulário, e clique sobre o evento onCreate, na página Events do Object Inspector. Entre então com as linhas abaixo:

```
procedure TfrmEditor.FormCreate(Sender: TObject);
begin
     MemoEditor.Clear;
end;
```

2) Quando o usuário clica na opção Novo do menu, utilizamos o método Clear para limpar o conteúdo do Memo. Para inserir este código, dê um clique sobre o item Novo do menu, no formulário, e entre com as linhas abaixo:

```
procedure TfrmEditor.IM_NovoClick(Sender:
TObject);
begin
     MemoEditor.Clear;
end;
```

3) Quando o usuário clica na opção Abrir do menu chamamos o método *Execute* para o Quadro de Diálogo Abrir. Se a operação for concluída, através do método LoadFromFile associamos as linhas do Memo (MemoEditor.Lines) ao conteúdo do arquivo

disposto na propriedade Filename do quadro de diálogo Abrir. Para inserir este código, dê um clique sobre o item Abrir do menu, no formulário, e entre com as linhas abaixo:

```
procedure TfrmEditor.IM_AbrirClick(Sender:
TObject);
begin
   if OpenDialog1.Execute then
     MemoEditor.Lines.LoadFromFile(
       OpenDialog1.Filename);
end;
```

4) Quando o usuário clica na opção Salvar executamos o método Execute para o quadro de diálogo Salvar. Se a operação for concluída, através do método *SaveToFile* associamos as linhas do Memo ao conteúdo do arquivo disposto na propriedade Filename do quadro de diálogo Salvar. Para inserir este código, dê um clique sobre o item Salvar do menu, no formulário, e entre com as linhas abaixo:

```
procedure TfrmEditor.IM_SalvarClick(Sender:
TObject);
begin
   if SaveDialog1.Execute then
    MemoEditor.Lines.SaveToFile(SaveDialog1.FileName);
end;
```

5) Quando o usuário clica na opção Sair do menu fechamos a aplicação. Para inserir este código, dê um clique sobre o item Sair do menu, no formulário, e entre com as linhas abaixo:

```
procedure TfrmEditor.IM_SairClick(Sender:
TObject);
begin
     Close;
end;
```

Para o código de Cortar, Copiar e Colar, vamos utilizar a variável *Texto*, tipo string, para armazenar o texto selecionado. Esta variável é declarada na cláusula var da

seção interface da Unit:

```
var
   frmEditor: TfrmEditor;
   texto : string;
```

1) Quando o usuário clica na opção Copiar do menu, jogamos o texto iluminado para a variável *Texto*, através do método SelText. Para inserir este código, dê um clique sobre o item Copiar do menu, no formulário, e entre com as linhas abaixo:

```
procedure TfrmEditor.IM_CopiarClick(Sender:
TObject);
begin
     Texto := MemoEditor.SelText;
end;
```

2) Quando o usuário clica na opção Cortar do menu, jogamos o texto iluminado para a variável *Texto*, através do método SelText. Segundo, substituímos o texto iluminado por espaço em branco. Para inserir este código, dê um clique sobre o item Cortar do menu, no formulário, e entre com as linhas abaixo:

```
procedure TfrmEditor.IM_RecortarClick(Sender:
TObject);
begin
     Texto := MemoEditor.SelText;
     MemoEditor.SelText := '';
end;
```

3) Quando o usuário clica na opção Colar do menu, atribuimos a posição selecionada o valor da variável *Texto*. Para inserir este código, dê um clique sobre o item Colar do menu, no formulário, e entre com as linhas abaixo:

```
procedure TfrmEditor.IM_ColarClick(Sender:
TObject);
begin
     MemoEditor.SelText := Texto;
```

```
end;
```

4) Quando o usuário clica na opção Fonte do menu executamos o método Execute para o quadro de diálogo Fonte. Se a operação for concluída, associamos a propriedade Font do Memo à propriedade Font do quadro de diálogo Fonte. Para inserir este código, dê um clique sobre o item Fonte do menu, no formulário, e entre com as linhas abaixo:

```
procedure TfrmEditor.IM_FonteClick(Sender:
TObject);
begin
   if FontDialog1.Execute then
     MemoEditor.Font := FontDialog1.Font;
end;
```

Vamos salvar a aplicação:

1) No menu *File*, escolha a opção *Save All*. Dê o nome *UnitEditor* para o formulário.

Para testar o projeto, você precisa tornar o formulário do Editor o formulário principal. Para isso, vá no menu *Project*, opção *Options*, aba *Forms* e, na caixa *Main Form*, selecione o formulário *frmEditor*.

Delphi 4

10

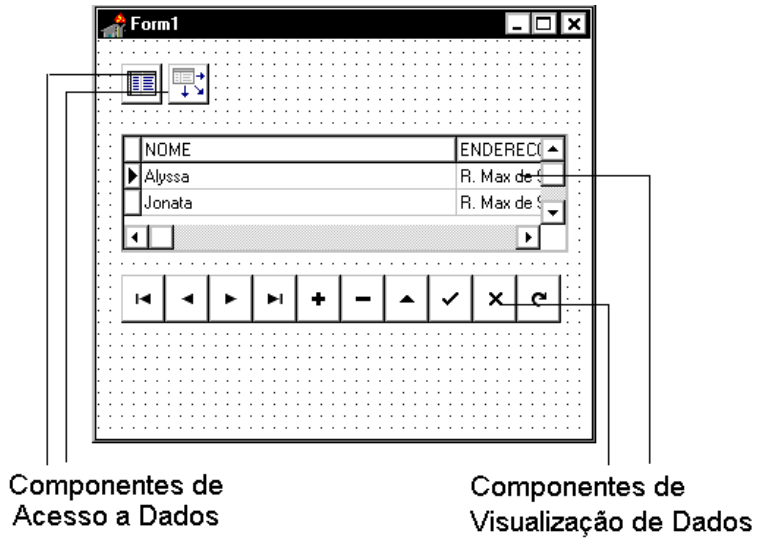
Aplicações com Bancos de Dados

Acessando um Banco de Dados com o Delphi

Desenvolver uma aplicação em Delphi ligada a um banco de dados é bastante simples. Em resumo, são necessários três passos:

- inserir no formulário da aplicação componentes que acessarão as tabelas, consultas e campos: são os componentes de acesso a banco de dados;
- inserir no formulário da aplicação componentes que mostrarão os dados: são os componentes de visualização dos dados;
- inserir no formulário da aplicação um componente especial, que fará a ponte entre os componentes que acessam os dados e os componentes que os mostram.

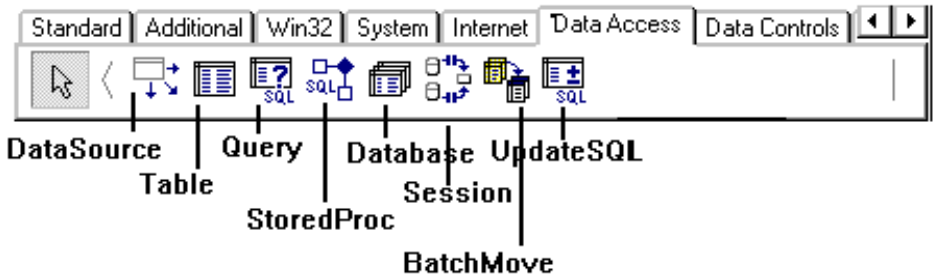
Veja um formulário típico de acesso a dados:



A Paleta de Componentes do Delphi oferece duas páginas de componentes de banco de dados: a página Data Access e a página Data Controls.

Componentes de Acesso a Dados

A página *Data Access* contém componentes que acessam os dados:



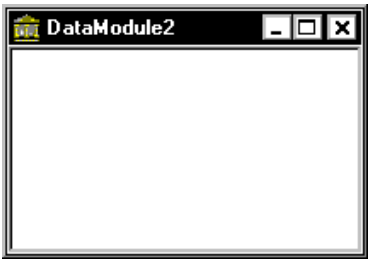
Estes componentes especificam o banco de dados, tabelas e registros a serem acessados. Como exemplo, podemos citar o componente Table (tabela), ou o componente Query (pesquisa). O componente DataSource também pertence a esta página, e faz a ligação entre os componentes de acesso a dados e os componentes de visualização de dados.

Componentes de Visualização de Dados

A página *Data Controls* contém os componentes de visualização de dados:

Estes componentes mostram valores das tabelas, ou passam seus valores para estas, permitindo ao usuário folhear, editar ou entrar dados nas tabelas. Como exemplo podemos citar o componente DBEdit (caixa de edição), o componente DBGrid (grade) e o componente DBNavigator, que permite a navegação entre os dados.

Data Module

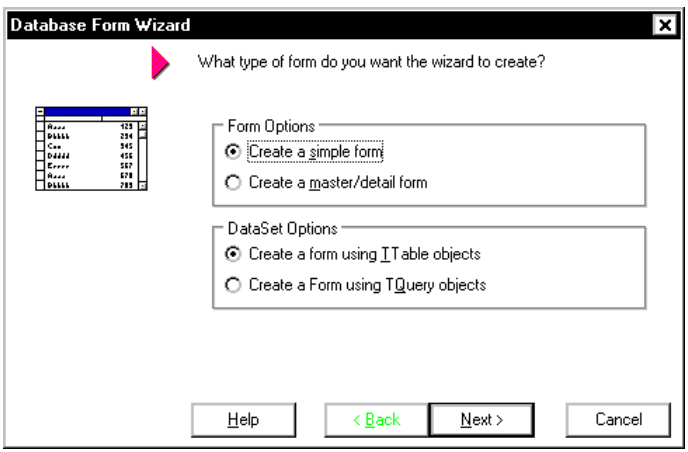


Na versão 1 do Delphi, cada formulário que manipulava uma tabela precisava, em seu interior, no mínimo, de um componente Table e um componente DataSource. Se você tivesse em sua aplicação dois formulários que utilizassem a mesma tabela, teria que colocar estes componentes em ambos os formulários.

A partir da versão 2 você pode centralizar estes componentes de acesso a dados em um único Data Module, e seus formulários se referenciarão a este Data Module somente.

Database Form Wizard

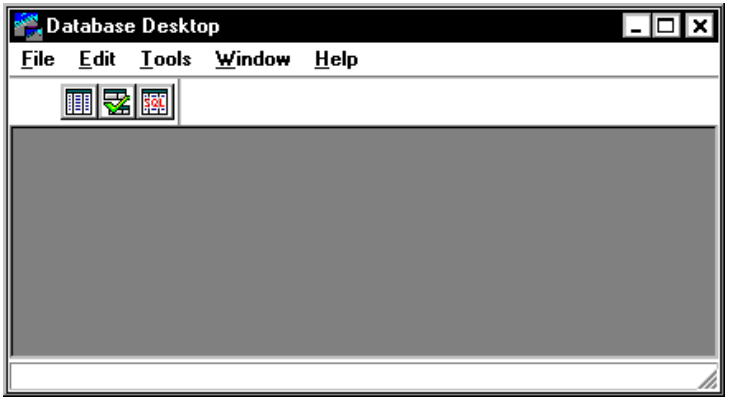
O Delphi oferece um mecanismo de criação de formulários que manipulam bancos de dados: é o *Database Form Wizard*:



Esta ferramenta automatiza diversas tarefas necessárias para a criação de formulários de entrada de dados ligados a uma tabela. O Database Form Wizard automaticamente insere os componentes de banco de dados no formulário, conecta-os ao banco de dados, e conecta-os aos componentes de visualização de dados.

Database Desktop

O Delphi oferece também o *Database Desktop*, uma ferramenta para criar, consultar, reestruturar, indexar, modificar e copiar tabelas de bancos de dados do tipo Paradox, dBASE, FoxPro, Access, entre outras:



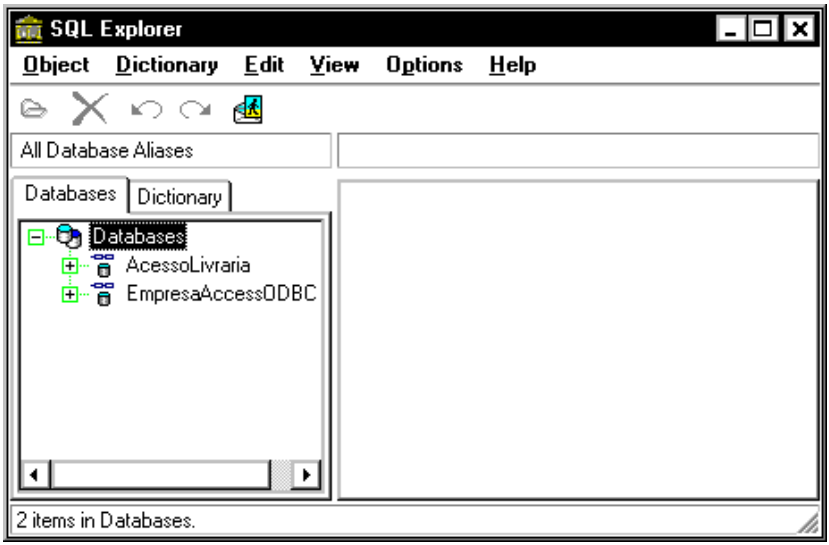
Quick Report

O Delphi 4 oferece um conjunto de componentes, na página QReport da Paleta de Componentes:

Com estes componentes, você pode criar relatórios de forma prática e rápida, contendo cabeçalhos, rodapés, títulos e imagens.

DataBase Explore

A versão 4 do Delphi apresenta o utilitário *Database Explorer* (ou *SQL Explorer*, na versão Client/Server), que visualiza hierarquicamente os bancos de dados registrados em seu computador:



Através deste utilitário, você pode visualizar os bancos de dados, as tabelas e índices destes, e os campos das tabelas. Pode visualizar não só a estrutura como os dados, editando-os inclusive, se necessário.

Relacionamento entre Delphi, Utilitários e Bancos de Dados

Uma aplicação desenvolvida em Delphi utilizando banco de dados é construída utilizando-se os recursos do próprio Delphi (componentes de acesso e visualização de dados), e dos utilitários acima descritos (Database Desktop, Database Form Wizard). Através dos componentes, a aplicação se comunica com o Borland Database Engine

(BDE), um conjunto de funções para acesso a banco de dados, que por sua vez se comunica com o banco de dados.

Tipos de Bancos de Dados Acessáveis

Quando você instala o Delphi, são instalados automaticamente drivers para acesso a bancos de dados locais como Paradox, dBASE, FoxPro e Access.

Caso você precise acessar dados de outros bancos de dados, deverá utilizar o utilitário de configuração do BDE (BDE Administrator) para criar uma conexão com este banco de dados.

O Delphi 4 acessa os seguintes tipos de bancos de dados:

- Paradox (extensão .DB);
- dBASE (extensão .DBF);
- Access 97(extensão .MDB);
- arquivos ASCII (extensão .TXT);
- Local Interbase Server (extensão .GDB);
- Bancos de Dados SQL: Oracle, Sybase, Microsoft SQL Server, Informix, Interbase, através da ODBC ou do SQL Links (versão cliente/servidor, somente).

11

Criando Formulários que Acessam Banco de Dados

Componentes de Acesso a Dados

Componentes de acesso a dados são componentes que acessam tabelas, consultas e campos de um banco de dados. Estão dispostos na página Data Access, da Paleta de Componentes:

 *DataSource*: liga os componentes de acesso a dados aos componentes de visualização de dados.

 *Table*: acessa uma tabela do banco de dados.

 *Query*: cria uma consulta relacionada a um banco de dados.

 *StoredProc*: permite que uma aplicação acesse procedures armazenadas em um banco de dados cliente/servidor.

 *Database*: realiza a conexão com um banco de dados, principalmente aqueles mantidos no servidor. A cada banco de dados corresponde um componente Database.

 *Session:* provê controle global sobre os bancos de dados manipulado pela aplicação, como por exemplo, quantos e quais bancos de dados estão ativos.

 *BatchMove*: copia a estrutura de uma tabela ou seus dados. Pode ser usado para mover uma tabela de um formato para outro.

UpdateSQL: para armazenar consultas feitas em bancos de dados nos quais o usuário pode apenas ler os dados. Posteriormente, quando a atualização for permitida, os dados armazenados no componente serão passados para o banco de dados cliente/servidor.

Os componentes de acesso a dados inseridos no formulário só aparecem durante o projeto, e não durante a execução da aplicação.

Componente Table

Especifica o banco de dados ou tabela a ser acessada.

Entre as propriedades mais importantes do componente Table, temos:

DatabaseName: para arquivos dBASE ou Paradox, especifica o caminho do banco de dados a ser acessado. Para outros bancos de dados, especifica o nome do banco de dados a ser acessado, ou o alias (apelido) para este, criado no utilitário de administração do BDE.

TableName: para arquivos dBASE ou Paradox, especifica o banco de dados a ser acessado. Para outros bancos de dados, especifica a tabela do banco de dados a ser acessada.

IndexName: especifica o índice a ser utilizado com a tabela. Para utilizar o índice primário do Paradox, deixe esta propriedade em branco.

Active: se for igual a *True*, torna a tabela ativa, mostrando os dados nos componentes de visualização ainda no projeto. Se for igual a *False*, desativa a tabela, e os dados não são mostrados nem durante a execução da aplicação.

Componente Query

O componente Query cria um conjunto de dados (dataset) usando comandos SQL, ou seja, cria uma pesquisa relacionada a determinada tabela .

Ao contrário do componente Table, o componente Query não usa a propriedade *TableName* para especificar a tabela do banco de dados a ser acessada. Esta tabela é especificada através da propriedade *SQL*.

Entre as propriedades mais importantes do componente Query, temos:

DatabaseName: para arquivos dBASE ou Paradox, especifica o caminho do banco de dados a ser acessado. Para outros bancos de dados, especifica o nome do banco de dados a ser acessado, ou o alias (apelido) para este, criado no utilitário de administração do BDE.

SQL: especifica o comando SQL de acesso a tabela do banco de dados.

Active: se for igual a True, torna a pesquisa ativa. Se for igual a False, desativa a pesquisa.

O componente Query pode retornar dois tipos de conjuntos de dados, de acordo com o valor estabelecido na propriedade *RequestLive*:

Live: podem ser editados pelo usuário.

Read Only: não podem ser editados pelo usuário.

Se a propriedade RequestLive for igual a True, os dados retornados pela consulta poderão ser editados. O padrão é o retorno de dados não editáveis.

Componente DataSource

 O componente DataSource é uma ponte entre componentes do tipo Table ou Query e um ou mais componentes de visualização de dados (Data Controls).

Entre as propriedades mais importantes do componente DataSource, temos:

Dataset: nome do componente de acesso a dados (Table ou Query) associado ao DataSource.

Componente Database

 O componente Database não é necessário para o acesso a banco de dados, mas permite um maior controle sobre aplicações cliente/servidor. Se você não cria um componente Database, e sua aplicação abre uma tabela de um banco de dados, o Delphi cria um componente TDatabase "virtual".

Componente TField

Ao contrário dos componentes acima, o componente TField não está presente na página de componentes Data Access. Componentes TField estão relacionados aos campos (fields) da tabela acessada. Por padrão, componentes TField são criados automaticamente em tempo de projeto quando você estabelece a propriedade Active de uma tabela ou pesquisa para *True.*

Apesar de não ser visível no formulário, o componente TField é importante porque provê uma ligação direta com os campos de um banco de dados. Este componente possui propriedades que especificam o tipo de dado, valor corrente, formato visual, formato de edição e outras características. Também oferece eventos, como OnValidate, que podem ser usados para validação de dados.

Componentes de Visualização de Dados

Componentes de visualização de dados (Data Controls components) são componentes que mostram os dados referentes a uma determinada tabela ou pesquisa, criando uma interface para o usuário. Estão dispostos na página *Data Controls*, da Paleta de Componentes:

 *DBGrid*: mostra os campos de uma tabela em uma grade.

 *DBNavigator*: insere botões para navegação pela tabela.

 *DBText*: mostra um campo da tabela em uma etiqueta (label).

 *DBEdit*: mostra um campo da tabela em uma caixa de edição (Edit).

 *DBMemo*: mostra dados de um campo memo.

 *DBImage*: mostra imagens armazenadas em uma tabela.

 *DBListBox*: mostra valores de um campo em um quadro de lista (ListBox).

 *DBComboBox*: mostra valores de um campo em quadro combo (ComboBox), permitindo sua edição.

 *DBCheckBox*: mostra um campo booleano da tabela em um botão de checagem.

 *DBRadioGroup*: mostra valores de um campo em um conjunto de botões de radio (RadioGroup).

 *DBLookupListBox*: mostra dados de uma outra tabela (associada através de um campo comum) em um quadro de lista.

 *DBLookupComboBox:* mostra dados de uma outra tabela (associada através de um campo comum) em um quadro combo, permitindo sua edição.

 *DBRichEdit*: é uma caixa de edição que pode mostrar texto formatado (rich text).

 *DBCtlrGrid*: mostra os dados de uma tabela na forma de painéis que você pode adaptar. Você cria um painel com os dados de um registro dispostos de uma determinada forma, e o componente reproduz este formato para os demais registros.

 *DBChart*: mostra uma série de dados.

Estes componentes podem tanto mostrar os dados quanto, se necessário, aceitar modificações e retorná-los para o banco de dados.

Componente DBGrid

 O componente DBGrid permite que você visualize os dados de uma tabela ou pesquisa em uma grade, mostrando todos os campos ou apenas aquelas estabelecidos no Editor

de Campos (acessado através de um duplo clique no componente Table ou Query, como vimos).

Entre as propriedades mais importantes do componente DBGrid, temos:

DataSource: define o nome do componente DataSource associado à tabela ou pesquisa a ser visualizada.

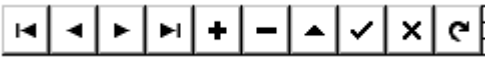
Options: controla o comportamento e a aparência da grade, permitindo ou não, por exemplo, a edição de dados, a visualização dos nomes dos campos, o redimensionamento das colunas, etc.

DefaultDrawing: controla como o Delphi desenha células individuais em uma grade.

Componente DBNavigator

 Este componente permite ao usuário navegar por entre um conjunto de dados, manipulando-os se necessário.

Isto é feito através de um conjunto de botões que permite ao usuário ir para o primeiro ou último registro, para o próximo registro ou para o registro anterior, inserir um novo registro, editar, gravar ou deletar um registro, cancelar modificações ou refazer a visualização dos dados:



Entre as propriedades mais importantes do componente DBNavigator, temos:

DataSource: define o nome do componente DataSource associado à tabela ou pesquisa que você quer navegar.

VisibleButtons: define quais botões do componente DBNavigator estarão visíveis.

ShowHints: se for igual a *True*, exibe a legenda dos botões, identificando-os para o usuário.

Componente DBText

 Este componente mostra os dados de um campo em uma etiqueta (label).

Entre as propriedades mais importantes do componente DBText, temos:

Datasource: define o nome do Datasource associado à tabela de onde sairão os dados.

Datafield: define o nome do campo da tabela a que o componente está associado.

Autosize: permite que o componente se redimensione, para mostrar dados de diversos tamanhos.

Componente DBEdit

 Este componente mostra o valor corrente de um campo em uma caixa de edição, permitindo que o usuário modifique seu valor.

Entre as propriedades mais importantes do componente DBEdit, temos:

Datasource: define o nome do Datasource associado à tabela de onde sairão os dados.

Datafield: define o nome do campo da tabela a que o componente está associado.

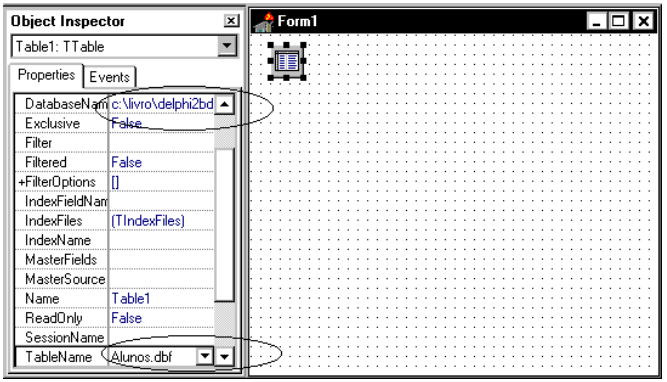
Montando um Formulário Simples

Para montar um formulário simples manualmente, que mostre os dados em uma grade, siga os passos abaixo:

- escolha a opção *New Application*, do menu *File;*

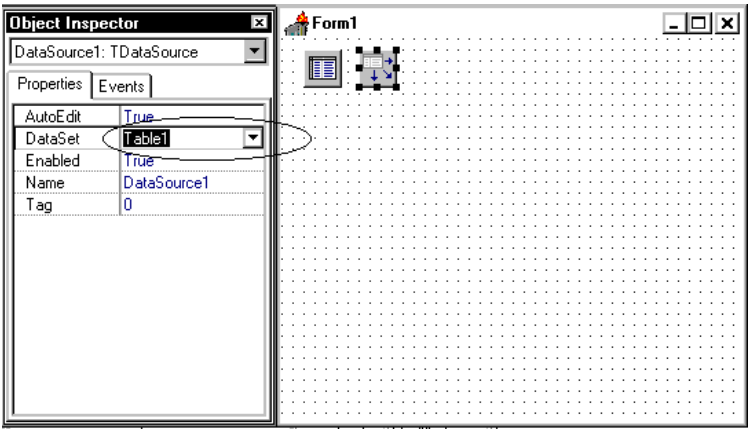
◆ no formulário em branco que aparece, insira um componente Table, da página de componentes DataAccess;

◆ defina as propriedades *DatabaseName*, com o caminho do banco de dados associado; *TableName*, com a tabela do banco de dados a ser acessada; *Active,* igual a *True*, para tornar a tabela ativa:



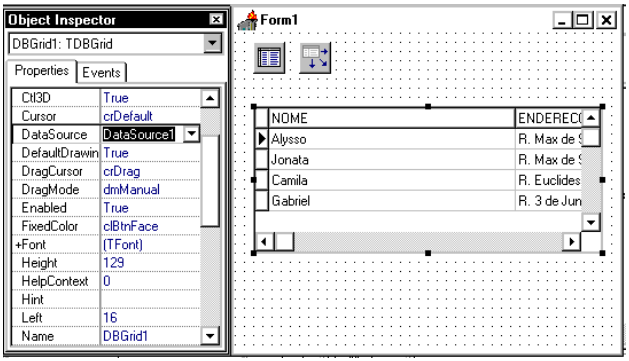
◆ insira um componente DataSource, da página de componentes DataAccess;

◆ defina a propriedade *Dataset* com o nome do componente Table inserido:



◆ insira um componente DBGrid, da página de componentes DataControls;

◆ defina a propriedade *DataSource* com o nome do componente DataSource inserido:

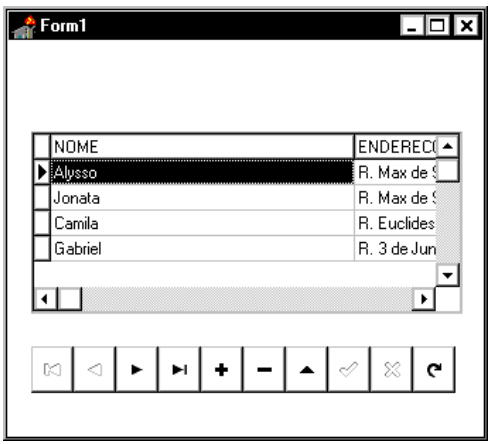


◆ insira um componente DBNavigator, da página de componentes DataControls;

◆ defina a propriedade *DataSource* com o nome do componente DataSource inserido:

◆ salve o projeto através do comando *Save All*, do menu *File;*

◆ execute seu projeto, teclando F9.



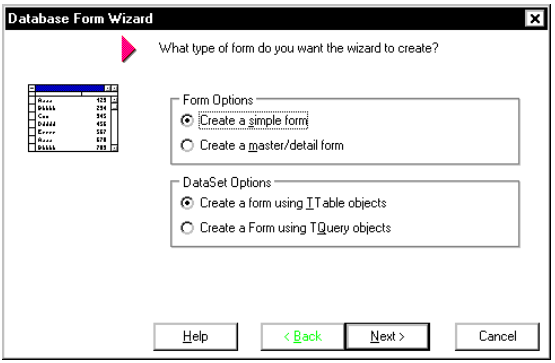
O Database Form Wizard

O Delphi oferece uma ferramenta que automatiza a criação de formulários ligados a bancos de dados. É o Database Form Wizard.

Como veremos, esta ferramenta facilita seu trabalho inserindo automaticamente os componentes de banco de dados no formulário, conectando-os ao banco de dados, e conectando-os aos componentes de visualização de dados. Você pode também posteriormente modificar o formulário criado pelo Database Form Wizard, adaptando-o as suas necessidades.

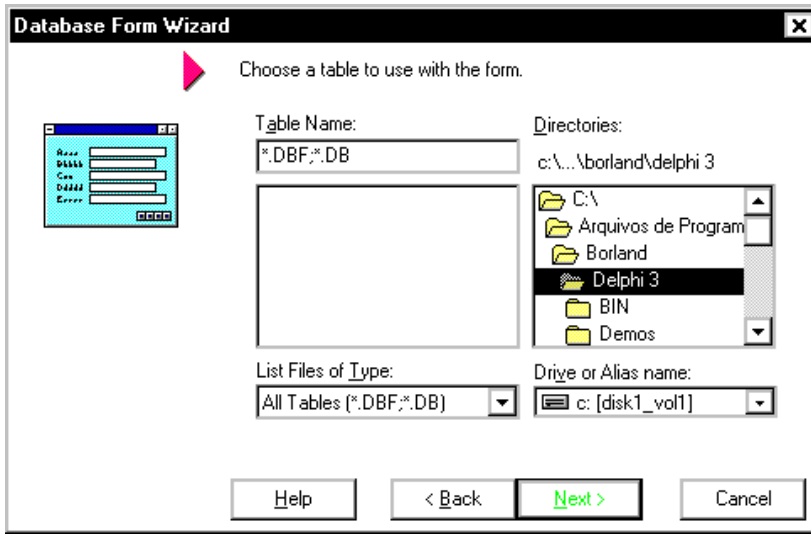
Para acessar o Database Form Wizard:

◆ escolha, no menu *Database*, a opção *Form Wizard;*

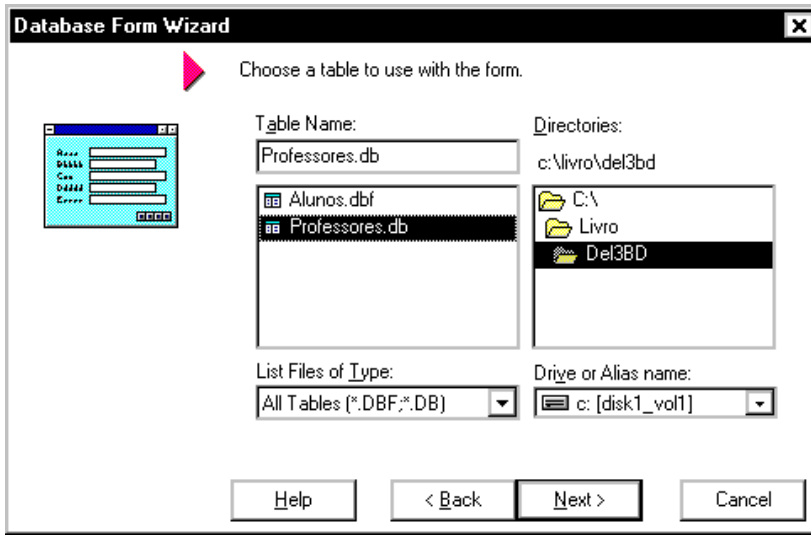


◆ no quadro Form Options, escolha o botão *Create a simple form;*

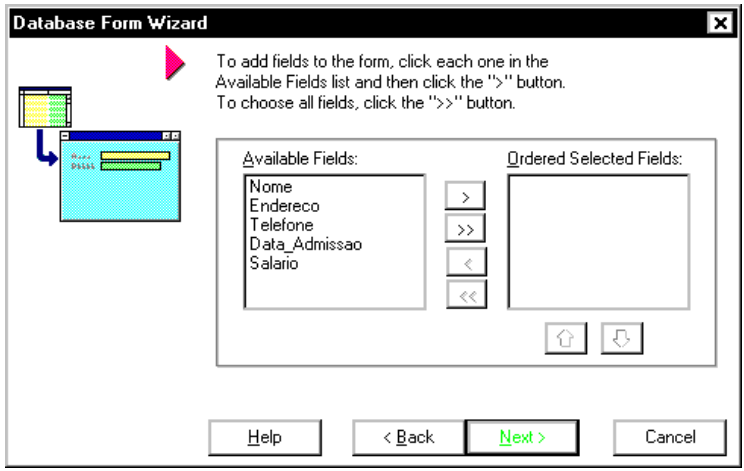
◆ no quadro DataSet Options, defina se você quer criar o formulário baseado em uma tabela (opção *Create a form using TTable objects*) ou baseado em uma pesquisa (opção *Create a form using TQuery objects*) e clique em *Next*;



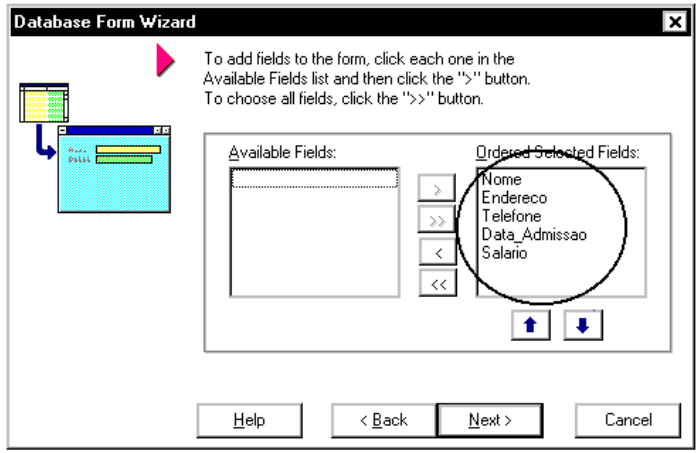
◆ escolha o banco de dados de onde se originará o formulário;



◆ e clique no botão *Next*;

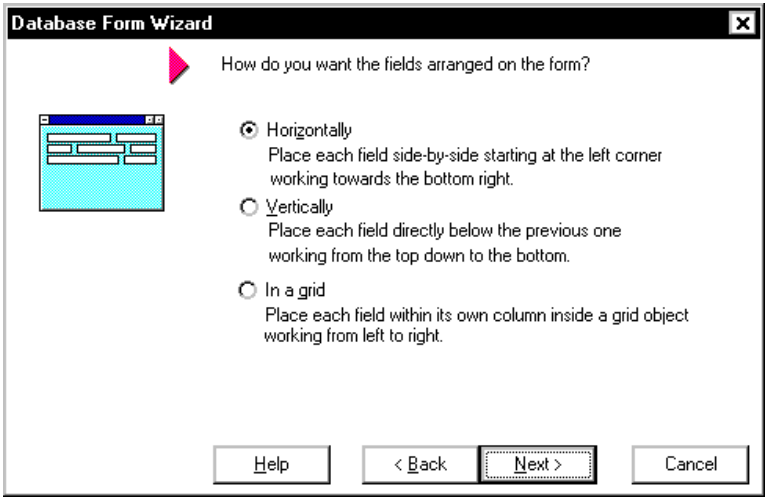


◆ selecione os campos que você usará, clicando no nome deste e no botão > (ou no botão >> para selecionar todos);

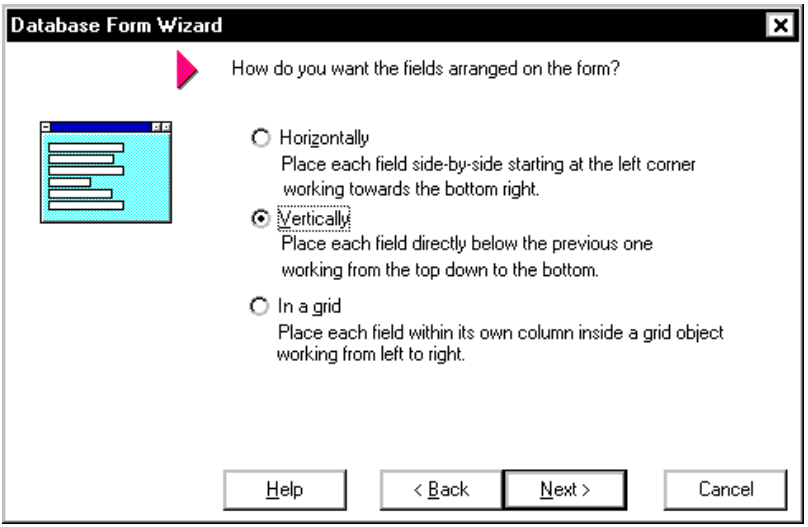


◆ clique novamente no botão *Next*;

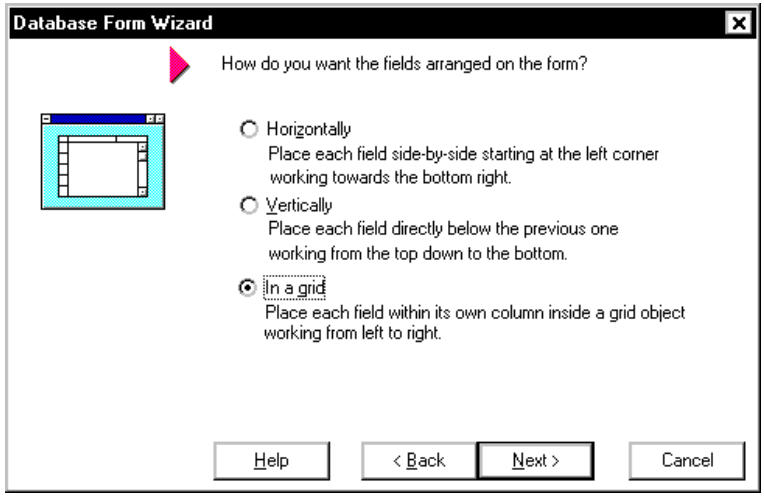
◆ escolha a forma como os dados serão mostrados: *Horizontal* (os campos serão mostrados um ao lado do outro);



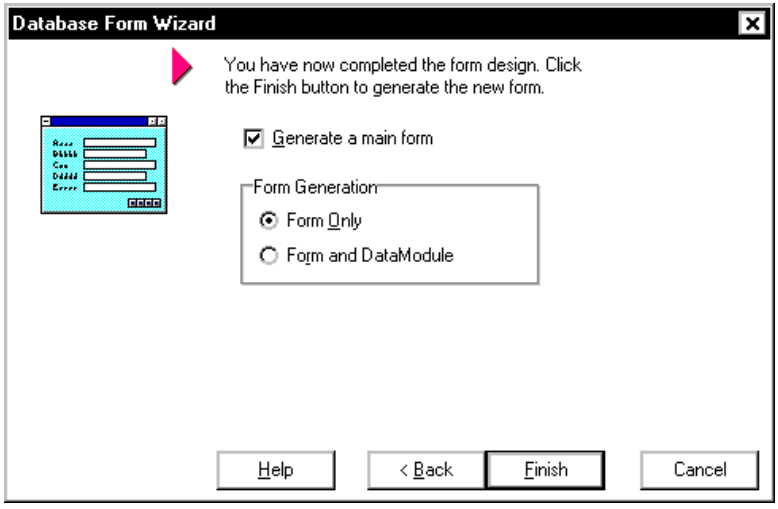
◆ ou *Vertical* (os campos serão mostrados um embaixo do outro);



◆ ou *In a Grid* (os dados serão inseridos em uma grade, em colunas);



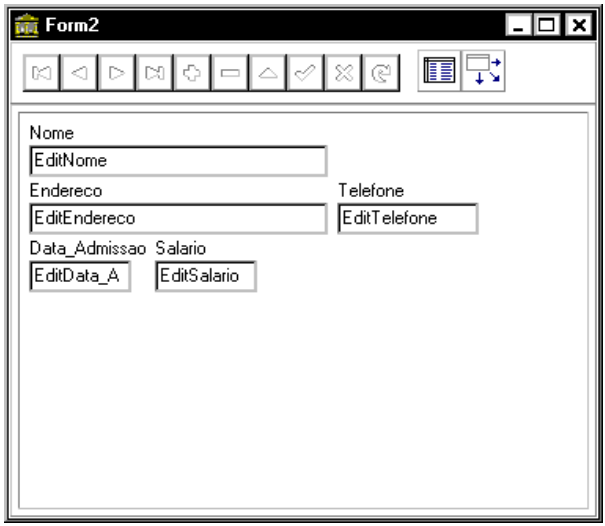
◆clique no botão no *Next*;



◆ na caixa Form Generation, escolha a opção *Form Only* se você desejar criar somente o formulário, ou escolha a opção *Form and DataModule* se quiser criar um formulário com um Data Module;

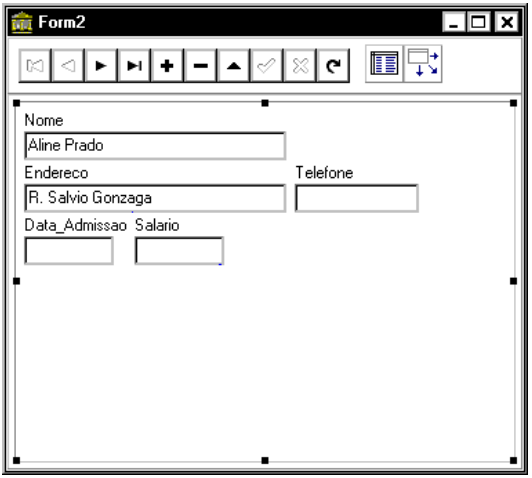
◆ marque a opção *Generate a main form* se quiser que o formulário gerado seja o principal de sua aplicação;

◆ clique no botão *Finish* e o DataBase Form Wizard cria o formulário para você automaticamente:



Este formulário possui dois componentes de acesso a dados: o componente Table (ou Query) e o componente DataSource . Além disso possui o componente DBNavigator , que insere os botões para navegação pelos dados. Isto é, clicando nos botões do componente DBNavigator você poderá ir para o primeiro registro ou ir para o último registro, poderá ver um registro anterior ou posterior ,poderá apagar um determinado registro, salvá-lo, inserir um novo registro, entre outras opções.

Se você mudar a propriedade *Active* do componente Table associado ao formulário para *True*, as caixas de edição mostrarão os dados da tabela:


Form2
Nome
Aline Prado
Endereco
Telefone
R. Salvio Gonzaga
Data_Admissao
Salario

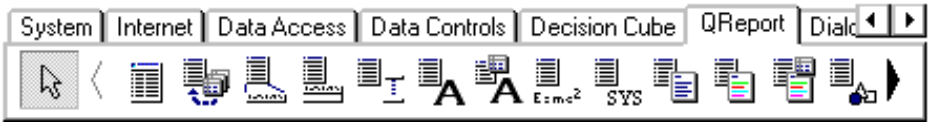
Delphi 4

12

Criando Relatórios Ligados a Bancos de Dados

O QuickReport

O Delphi oferece um grupo de componentes para a criação de relatórios: é o grupo de componentes QReport.



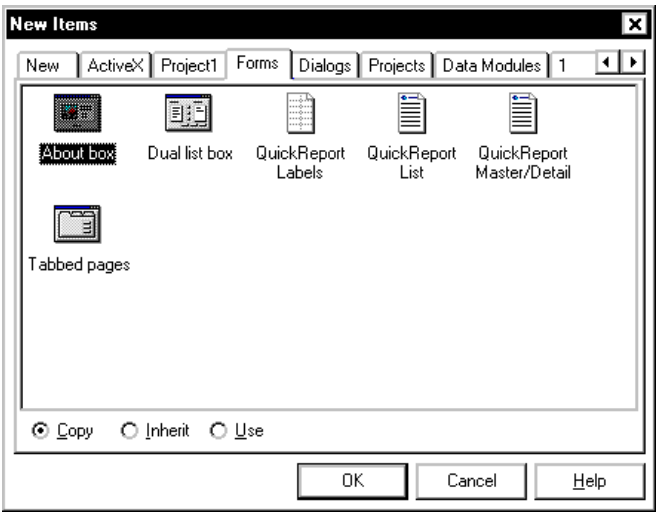
Com estes componentes você pode criar relatórios bastante sofisticados, com agrupamentos e cálculos.

Relatórios Oferecidos pelo Delphi

Você cria os relatórios do QuickReport em um formulário.

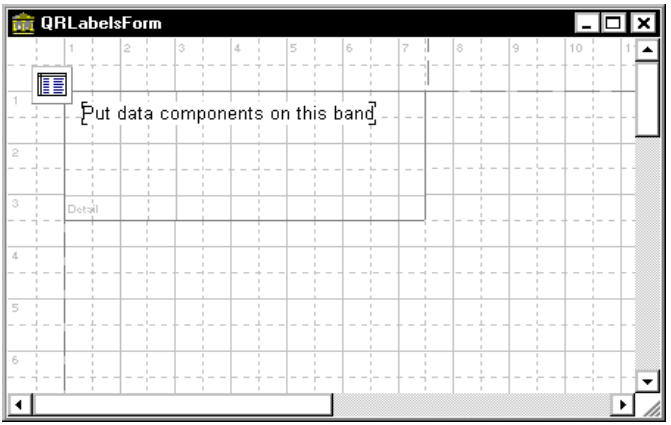
Para facilitar seu trabalho, o Delphi já oferece alguns modelos de formulários com relatórios pré-construídos, que você só precisa adaptar às suas necessidades.

Estes modelos são acessados através do menu *File*, opção *New*, página *Forms*:

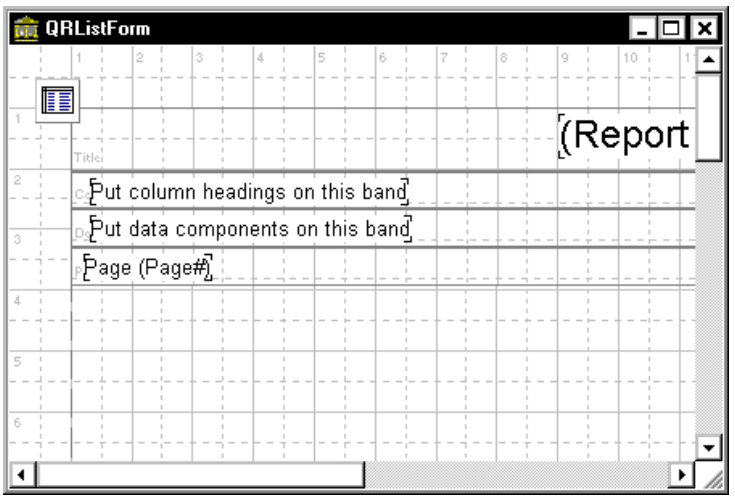


Escolha uma das seguintes opções:

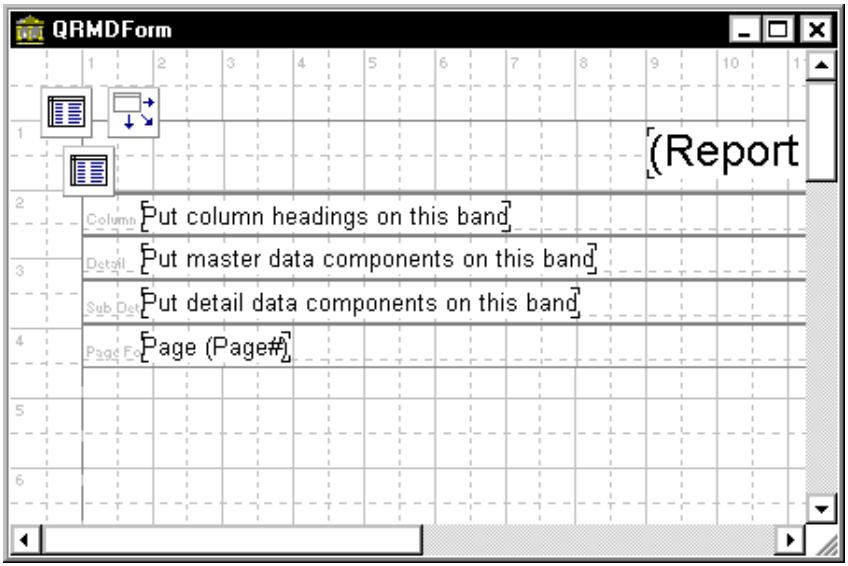
QuickReport Labels: para geração de relatórios de etiquetas:



QuickReport List: para geração de listagens:



QuickReport Master/Detail: para geração de relatórios envolvendo duas tabelas associadas:

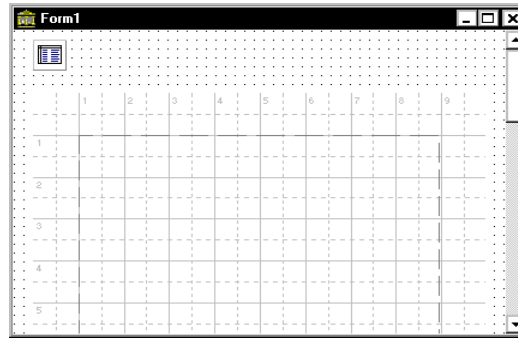


Como veremos, você também pode criar relatórios em um formulário em branco, inserindo manualmente suas seções de cabeçalho, corpo e rodapé.

Estrutura de um Relatório QuickReport

Os relatórios são normalmente associados à tabelas ou consultas, cujos dados serão listados. Para ligar o relatório à tabela, você deve inserir no formulário onde será montado o relatório, além dos componentes Table ou Query, um componente QuickRep, da página de componentes QReport:

Mude a propriedade *DataSet* deste para o nome do componente Table ou Query inserido no formulário.



Os relatórios do QReport são divididos em seções (bands), entre elas: cabeçalho (Page Header), título (Title), título das colunas (Column Headers), corpo (Detail), sumário (Summary) e rodapé (Page Footer):

Title: corresponde ao título do relatório. Esta seção é impressa automaticamente na primeira página do relatório.

PageHeader: corresponde ao cabeçalho. Esta seção é impressa automaticamente no início de cada páginda do relatório.

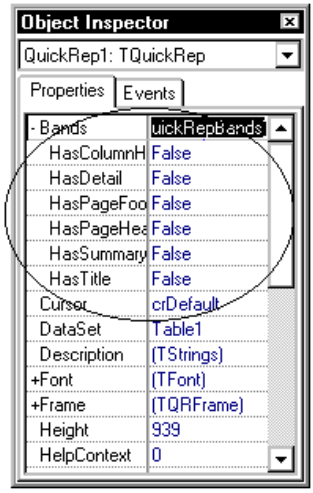
Detail: corresponde ao corpo do relatório. É impressa uma vez para cada registro de sua tabela.

PageFooter: corresponde ao rodapé do relatório. Esta seção é impressa automaticamente ao final de cada página do relatório.

Summary: corresponde ao resumo do relatório, normalmente onde são mostrados os cálculos referentes a somas, contagens, etc. Esta seção é impressa apenas no final do relatório.

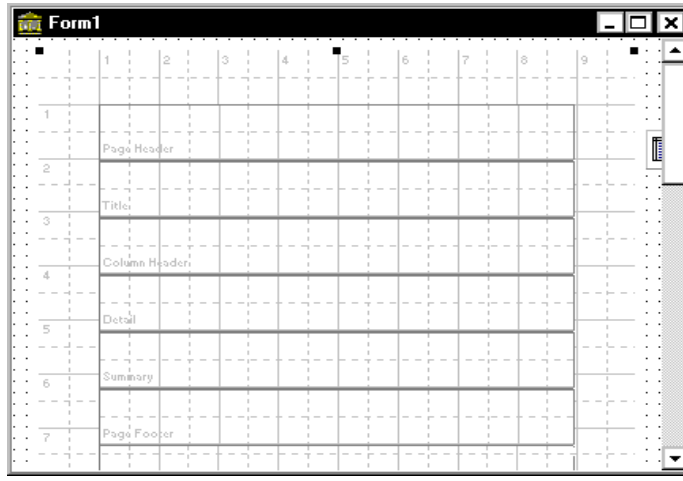
ColumnHeader: corresponde ao título das colunas. Será impresso no topo de cada coluna.

Para criar estas seções em seu relatório, dê um duplo clique na propriedade Bands, do componente QuickRep, e uma lista de seções se abre:



Para inserir uma seção no relatório, basta mudar o valor da propriedade referente à seção para True. Por exemplo, para inserir uma seção Title, mude o valor da propriedade HasTitle para True.

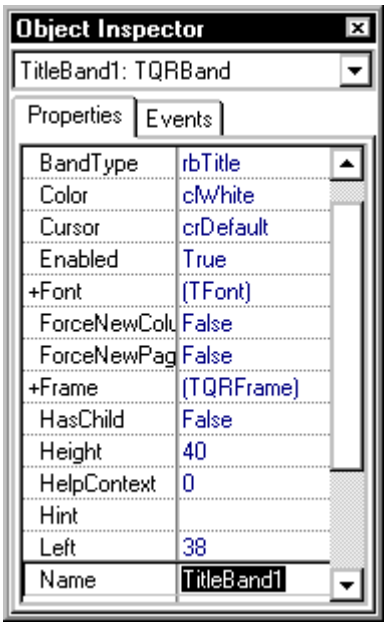
O relatório abaixo possui todas as seções:



Para criar o relatório, você deve primeiro criar estas seções no formulário, e depois, sobre estas seções inserir os dados que constarão do formulário.

Cada uma das seções inseridas é na realidade um objeto do tipo TQRBand, possuindo portanto propriedades e eventos próprios. Para verificar as propriedades referentes a determinada seção, basta clicar sobre a seção, e suas propriedades aparecerão no Object Inspector.

No quadro abaixo, o Object Inspector mostra as propriedades da seção Title:



Você utiliza os demais componentes da página QReport para criar o relatório, entre eles:

QRLabel: para inserir texto no relatório.

QRDBText: para inserir campos de uma tabela no relatório.

QRExpr: para inserir cálculos, como somatórios, totalizações, no relatório.

QRSysData: para inserir informações como a data atual, número de página, etc.

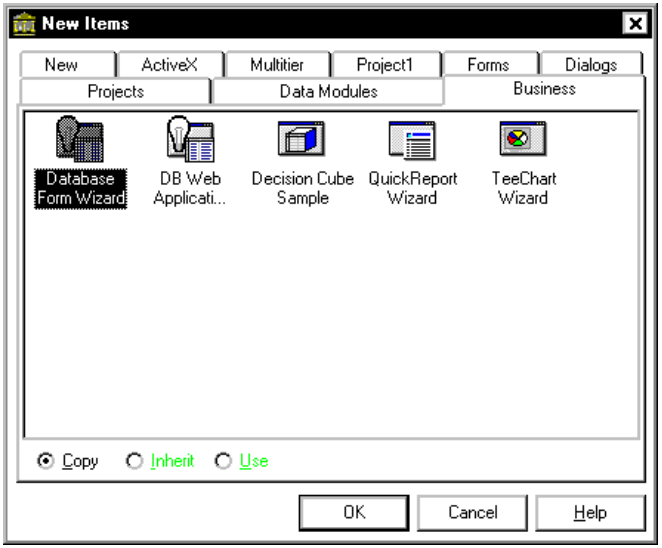
QRShape: para inserir objetos gráficos como linhas, quadros, etc.

QRImage: para inserir imagens no relatório.

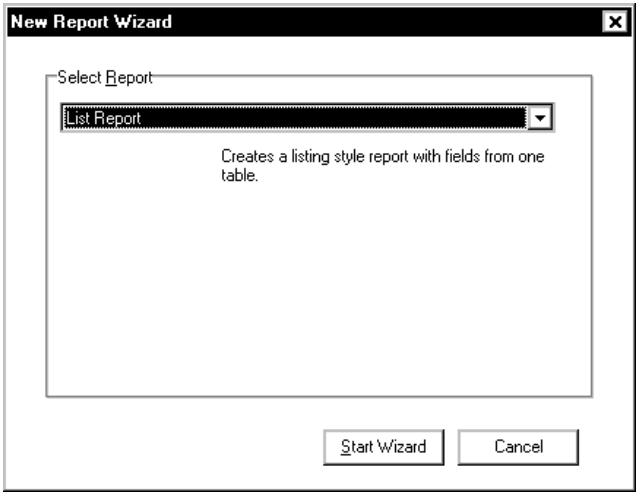
Criando um Relatório com o Assistente

O Delphi oferece um assistente que o auxilia na criação de relatórios. Para utilizá-lo, siga os passos abaixo:

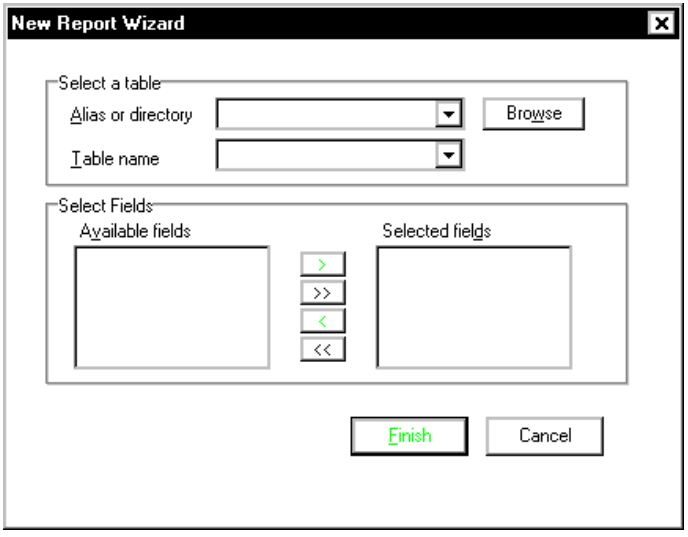
- entre no menu File, opção New, página Business:



- escolha a opção *QuickReport Wizard*:



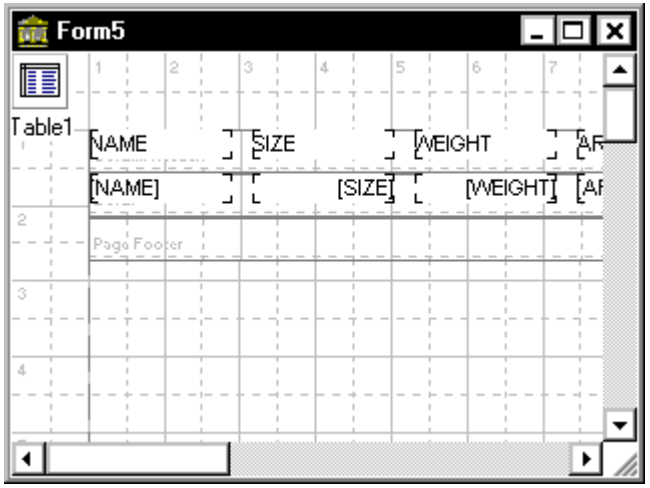
◆ clique no botão *Start Wizard*:



◆ na caixa *Alias or directory*, escolha a pasta onde está seu banco de dados;

◆ na caixa *Table name*, escolha a tabela que dará origem ao relatório;

◆ selecione os campos que aparecerão no relatório e clique no botão *Finish*. Um novo formulário é criado:



Para acessar de um outro formulário o formulário contendo o relatório, use o método *Preview* do componente QuickReport.

Delphi 4

Sistema de Acompanhamento Geográfico

Parte VIII

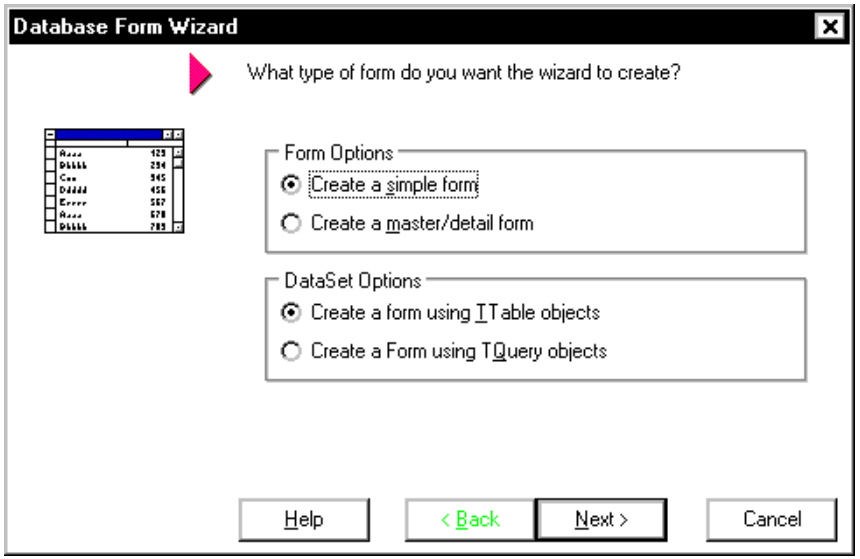
Criando o Formulário de Cadastro de Países

Vamos criar agora o formulário que cadastra países. Para isso, utilizaremos o assistente do Delphi para criação de formulários que manipulam bancos de dados, o Database Form Wizard. Usaremos o banco de dados Country.db, que é instalado em seu micro quando você instala o Delphi.

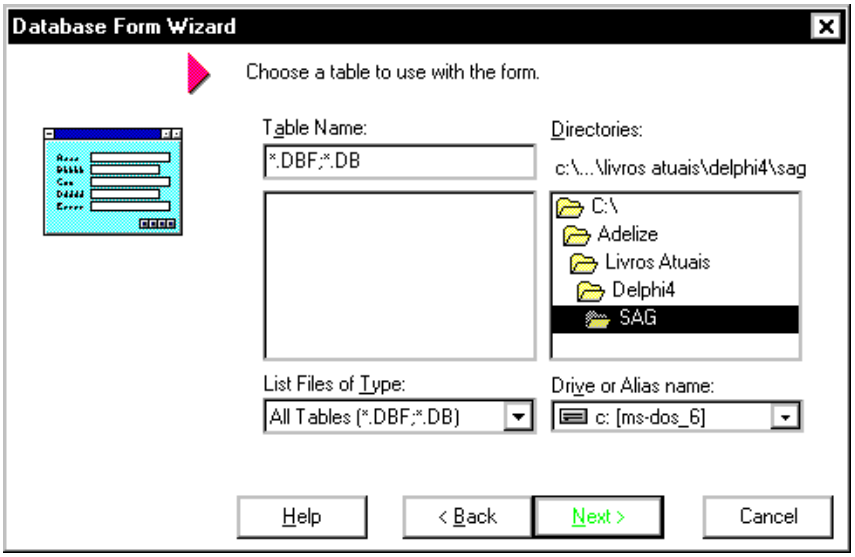
Siga os passos abaixo:

1) Abra o projeto SAG.dpr.

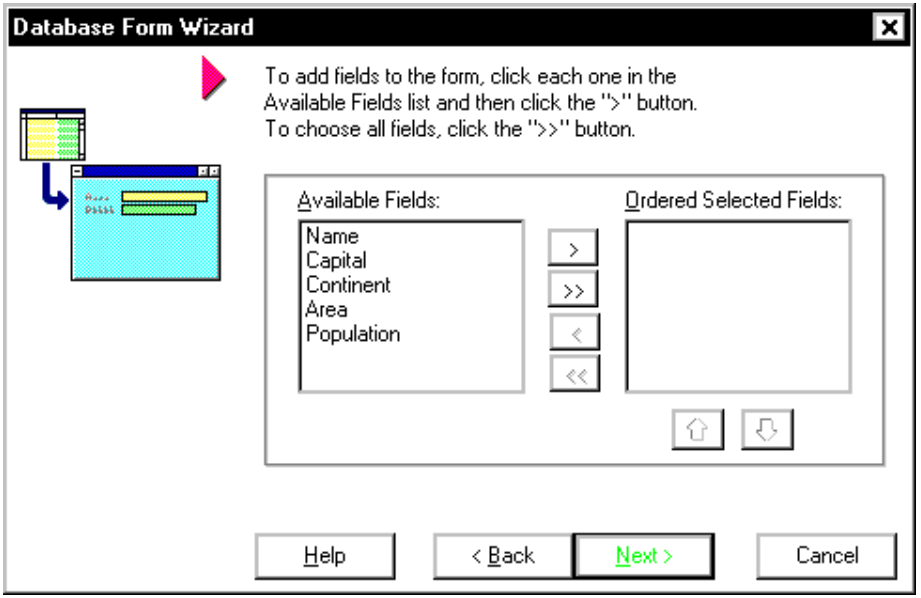
2) No menu *Database*, escolha a opção *Form Wizard*:



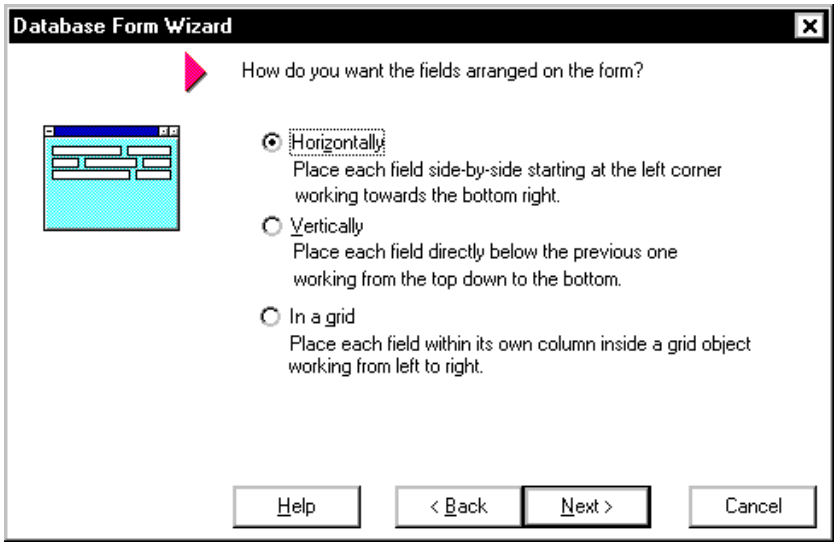
3) Na caixa *Form Options*, escolha a opção *Create a Simple Form*. Na caixa *DataSet Options*, escolha a opção *Create a Form using TTable Objects*, e clique no botão Next:



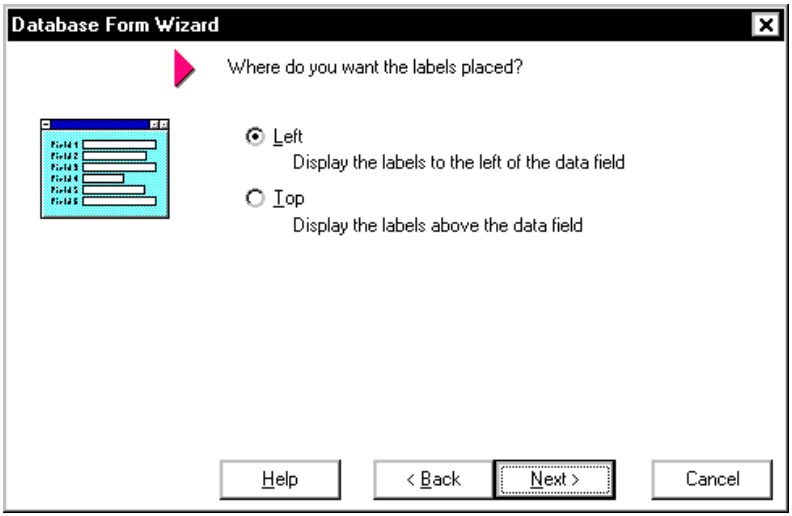
4) Na caixa *Alias or Drive Name*, escolha o alias para os bancos de dados de exemplo do Delphi, *DBDemos*. Na caixa *Table Name*, escolha o banco de dados *Country.db*, e clique no botão Next:



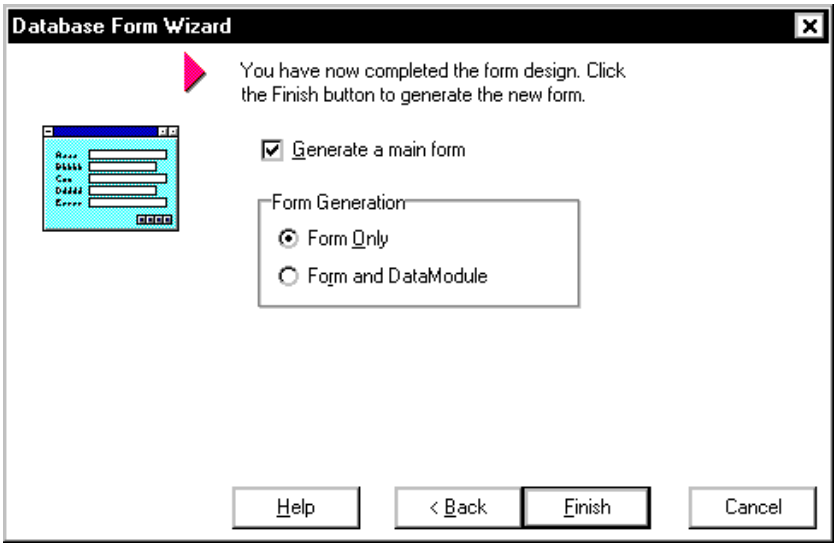
5) Selecione todos os campos, e clique no botão Next:



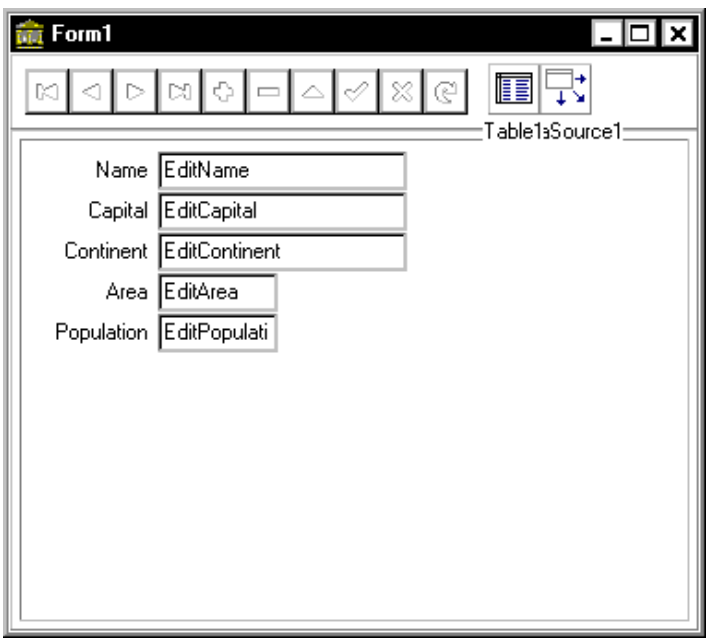
6) Escolha a opção *Vertically*, e clique no botão Next:



7) Escolha a opção *Left*, e clique no botão Next:

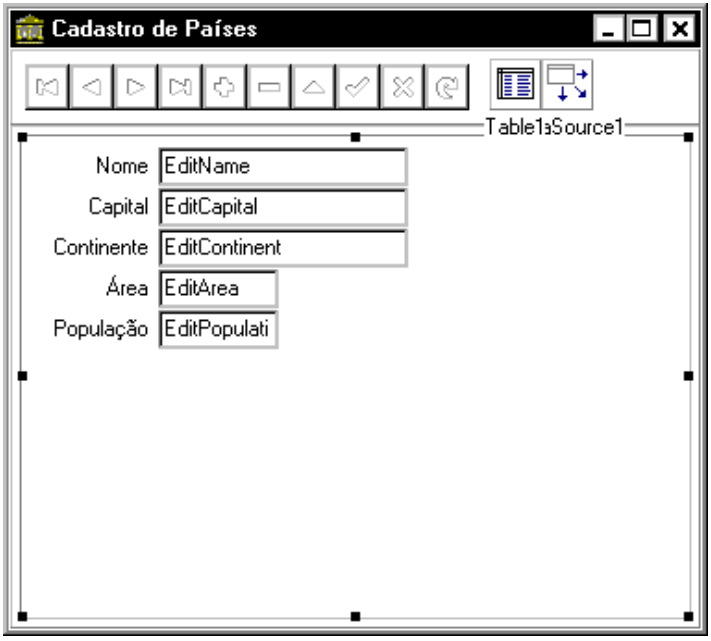


8) Desmarque a opção *Generate a Main Form*, marque a opção *Form Only*, e clique no botão *Finish*:



9) Mude a propriedade Caption do formulário para *Cadastro de Países*, e a propriedade Name para *FrmCadastroPaises*.

10) Mude as etiquetas para os valores Nome, Capital, Continente, Área e População.



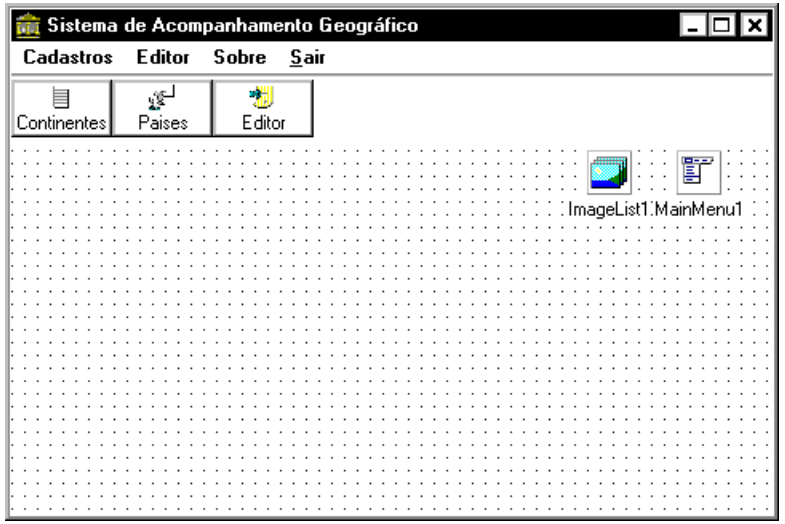
11) Salve o projeto, através do menu File, opção Save All. Dê o nome UnitCadastroPaises para a nova unit.

Associando o Formulário de Menu aos Demais Formulários

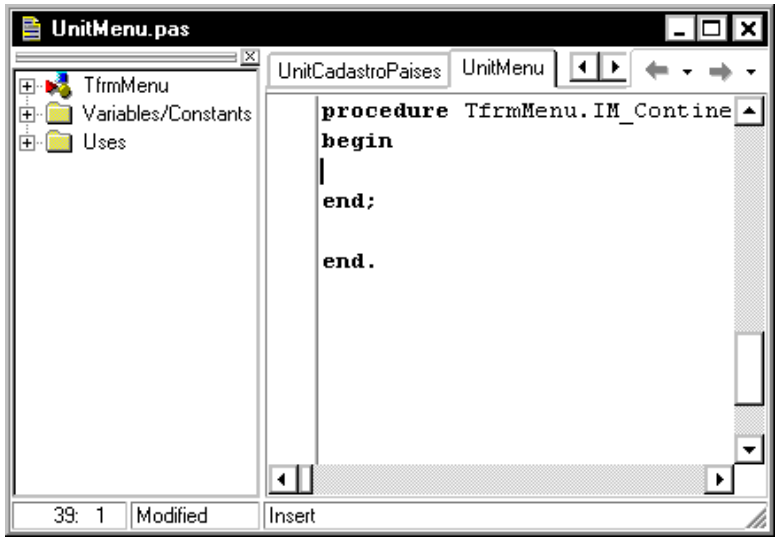
Quando criamos o formulário contendo o menu principal de nosso sistema, e a barra de atalhos, não associamos nenhum código aos itens de menu ou botões. Faremos isto agora, que todos os formulários de nosso aplicativo estão construídos.

Siga os passos abaixo:

1) Através do Project Manager, torne o formulário frmMenu (referente a UnitMenu) ativo.



2) Tecle F12, e visualize a unit referente ao formulário de menu:



3) Na cláusula Uses da seção Interface, acrescente o nome das units que serão manipuladas pelo formulário de menu: *UnitCadastroContinentes, UnitCadastroPaises, UnitEditor, UnitSobre*:

```
unit UnitMenu;

interface

uses
Windows, Messages, SysUtils, Classes, Graphics,
Controls, Forms, Dialogs,Menus, ActnList,
ComCtrls, ToolWin, ImgList,
UnitCadastroContinentes,UnitCadastroPaises,
UnitEditor, UnitSobre;
```

4) Tecle F12 para visualizar o formulário. Clique sobre o menu Cadastros, opção Continentes. Você passa para a unit. Entre então com o seguinte código:

```
procedure TfrmMenu.IM_ContinentesClick(Sender:
TObject);
begin
     FrmCadastroContinentes.Show;
end;
```

5) Volte para o formulário e clique sobre o menu Cadastros, opção Países. Você passa para a unit. Entre então com o seguinte código:

```
procedure TfrmMenu.IMPaisesClick(Sender: TObject);
begin
     FrmCadastroPaises.Show;
end;
```

6) Volte para o formulário e clique sobre o menu Editor.Você passa para a unit. Entre então com o seguinte código:

```
procedure TfrmMenu.IMEditorClick(Sender: TObject);
begin
     FrmEditor.Show;
end;
```

7) Volte para o formulário e clique sobre o menu Sobre.Você passa para a unit. Entre então com o seguinte código:

```
procedure TfrmMenu.IMSobreClick(Sender: TObject);
begin
     AboutBox.Show;
end;
```

8) Finalmente, volte para o formulário e clique sobre o menu Sair.Você passa para a unit. Entre então com o seguinte código:

```
procedure TfrmMenu.IMSairClick(Sender: TObject);
begin
     FrmMenu.Hide;
end;
```

9) Agora, vamos associar o clicar dos botões ao mesmo código do clique no item de menu correspondente. Para isso, selecione o botão Continentes, e no Object Inspector, página Events, abra a lista ao lado do evento onClick. Escolha então a procedure *IM_ContinentesClick*, que foi construída no item 4.

10) Repita a operação para o botão Países, associando seu evento Click a procedure *IMPaisesClick*.

11) Finalmente, repita a operação para o botão Editor, associando seu evento Click a procedure *IMEditorClick*.

12) No menu *Project*, selecione a opção *Options*, aba *Forms*, e na caixa *Main Form*, selecione o formulário principal, *frmPrincipal*.

13) Salve a aplicação, através do menu *File* opção *Save All*.

14) Execute a aplicação. Lembre-se que no formulário de login o nome do usuário pode ser deixado em branco, e a senha é ADV.

Delphi 4

13

Operações Avançadas

Movendo Dados com Drag-and-Drop

O Windows oferece, em diversos aplicativos, a possibilidade de mover dados através de operações de drag-and-drop, ou seja, arrastando-e-soltando os dados.

Por exemplo, você pode mover arquivos de um diretório para outro no Gerenciador de Arquivos simplesmente clicando sobre o arquivo e arrastando-o para o novo diretório. Note que enquanto o arquivo estiver sendo arrastado você mantém o botão do mouse selecionado. Só o solta quando o arquivo estiver posicionado sobre o novo diretório.

O Delphi permite que você realize operações de drag-and-drop entre componentes, de forma que você possa transferir dados com um simples movimentar do mouse.

Inúmeras propriedades, métodos o eventos estão relacionados a esta operação.

Propriedades Ligadas a Operações Drag-and-Drop

DragCursor

Estabelece o formato do cursor quando este estiver sobre o componente que aceitará um objeto sendo arrastado.

DragMode

Determina a possibilidade do componente realizar uma operação de drag-and-drop. Há duas opções: *dmAutomatic* permite o movimento do componente. *dmManual* não permite.

Métodos Ligados a Operações Drag-and-Drop

BeginDrag

Sintaxe:

Componente.BeginDrag (Imediate: Boolean)

Este método inicia o arrastar de um controle. Se o parâmetro Immediate é igual a *True*, o ponteiro do mouse muda para o formato estabelecido na propriedade DragCursor do componente, e o arrastar se inicia imediatamente. Se Imediate é igual a *False*, o ponteiro do mouse não se modifica até que o usuário mova o ponteiro a uma determinada distância (5 pixels).

Sua aplicação precisará chamar o método BeginDrag para começar a arrastar o componente somente quando a propriedade DragMode do componente estiver com o valor *dmManual*.

Dragging

Sintaxe:

Componente.Dragging

Este método especifica se um componente está sendo arrastado. É uma função, retornando *True* se estiver sendo arrastado.

EndDrag

Sintaxe:

Componente.EndDrag (Drop: Boolean);

Este método pára o arrastar de um componente, que não poderá mais ser arrastado. Se o parâmetro Drop está em *True*, o objeto sendo arrastado é inserido, e o arrastar termina. Se o parâmentro Drop está em *False*, o objeto não é inserido, e o arrastar é cancelado.

Eventos Ligados a Operações Drag-and-Drop

OnDragDrop

Ocorre quando o usuário solta um objeto que está sendo arrastado para cima do componente.

Possui a seguinte lista de parâmetros:

(Sender, Source : TObject; X, Y : integer)

onde *Sende*r é o componente que originou o evento (ou seja, o componente onde é solto o que está sendo arrastado); *Source* é o componente que está sendo arrastado; X e Y são as coordenadas.

OnDragOver

Ocorre quando o usuário arrasta um objeto sobre outro.

Possui a seguinte lista de parâmetros:

(Sender, Source : TObject; X, Y : integer; State: TDragState; var Accept : Boolean)

onde *Sender* é o componente que originou o evento (ou seja, o componente onde está sendo passado o que está sendo arrastado); *Source* é o componente que está sendo arrastado; X e Y são as coordenadas; State é o estado do evento; *Accept* permite ou não que o que está sendo arrastado seja solto sobre o componente.

Você deve estabelecer um valor para Accept. Se quiser que o componente aceite qualquer objeto sendo arrastado, faça:

Accept := *True*;

Se quiser que o componente só aceite objetos Etiquetas (label) sendo arrastados, faça:

Accept := Source is TLabel;

OnEndDrag

Ocorre quando o arrastar do objeto termina, seja porque foi solto sobre outro componente, seja porque a operação foi cancelada.

Possui a seguinte lista de parâmetros:

(Sender, Target : TObject; X, Y : integer)

onde *Sender* é o componente que originou o evento (ou seja, o componente que foi arrastado); *Target* é o componente destino; X e Y são as coordenadas.

Se o componente for solto com sucesso, Target será:

Target <> nil;

onde *nil* é uma palavra reservada que indica uma constante tipo ponteiro que não aponta para nada.

Criando Aplicações Multimídia

Uma aplicação multimídia envolve texto, gráficos, sons e imagens, enriquecendo a interação do usuário com o aplicativo.

O Delphi permite que você crie aplicações multimídia através do componente MediaPlayer, da página Systems da Paleta de Componentes.

O Componente MediaPlayer

Este componente é um conjunto de botões que controla um dispositivo multimídia como um CD-ROM, ou uma placa de som.

Os botões apresentados pelo componente MediaPlayer são os seguintes:

Play: faz funcionar o MediaPlayer.

Pause: pausa o funcionamento ou a gravação.

Stop: para o funcionamento ou a gravação.

Next: vai para o próximo track.

Prev: vai para o track anterior.

Step: move-se para frente um número de quadros.

Back: move-se para trás um número de quadros.

Record: inicia a gravação.

Eject: ejeta o dispositivo.

Quais botões o componente MediaPlayer irá mostrar em sua aplicação é definido na propriedade *VisibleButtons* deste.

Quando o usuário clica em um destes botões dá início à operação correspondente. Estas operações também podem ser acessadas através de métodos do componente MediaPlayer, tornando o controle do dispositivo independente do usuário.

Por exemplo, para que o próprio aplicativo dê início a um determinado som ou imagem, faça:

```
MediaPlayer1.Play;
```

O tipo de dispositivo multimídia é especificado pela propriedade *DeviceType*. Se o dispositivo armazena a mídia em um arquivo, o nome do arquivo é especificado pela propriedade *FileName*. Se a propriedade DeviceType for igua a *dtAutoSelect*, o componente MediaPlayer determina o tipo de dispositivo de acordo com a extensão do arquivo exposto na propriedade FileName.

Para habilitar ou desabilitar determinados botões do componente durante a execução da aplicação, use a propriedade *EnabledButtons*.

Para abrir um dispositivo multimídia, use o método *Open*. Para fechar um dispositivo, use o método *Close*.

Delphi 4

14

Depurando Seu Aplicativo

Introdução

Para ajudar a revisar seu código, o Delphi oferece:

◆ *checagem automática de sintaxe:* quando você escreve um comando errado, o Delphi alerta-o, oferecendo a grafia correta;

◆ *ferramentas de depuração*: permitem a análise do código passo a passo.

Erros na Programação

Durante a programação podem ocorrer três tipos de erros:

Erros de Compilação: ocorrem quando você compila seu programa.

Erros de Execução: quando você executa seu programa.

Erros Lógicos: apresentam resultado diferente do esperado.

Para corrigir estes erros, você pode usar as ferramentas de depuração.

Ferramentas de Depuração

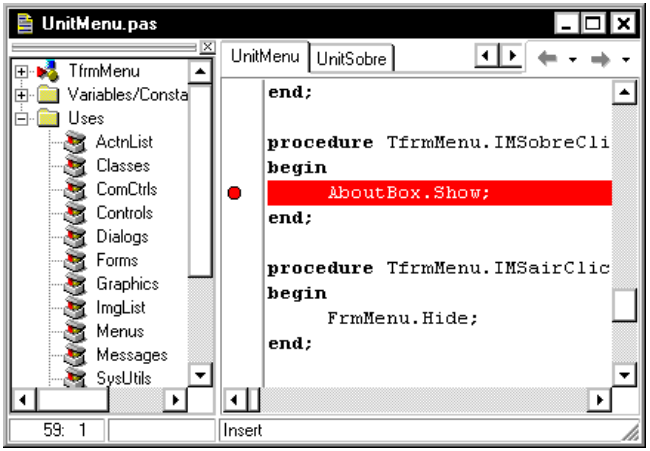
Você pode acessar as ferramentas de depuração através do menu *Run* ou de diversos ícones presentes na Barra de Ferramentas.

Criando um BreakPoint

Você pode iniciar a depuração a partir do início da aplicação ou a partir de determinada linha de seu código. Neste caso, você deve marcar esta linha com um breakpoint, ou seja, um marcador a partir do qual o programa será visto passo-a-passo.

Para criar um breakpoint em alguma linha do código:

◆ clique sobre a linha que terá o breakpoint, e clique sobre ela com o botão direito do mouse. Depois selecione, no menu que aparece, a opção Debug e o comando *Toggle Breakpoint*. Uma moldura vermelha é posta sobre esta linha, e um sinal de Stop a sua esquerda:



◆ outra forma de inserir um breakpoint é clicar na extrema esquerda da linha.

Se quiser retirar o breakpoint, basta clicar novamente na extrema esquerda da linha de código selecionada.

Executando o Programa Passo a Passo

Se quiser executar seu programa passo a passo, para encontrar eventuais erros, clique sobre o ícone  , ao invés de clicar sobre o ícone  .

Se você quiser executar o programa passo a passo apenas após determinado ponto, siga os passos abaixo:

◆ crie o breakpoint;

◆ execute o programa, clicando sobre o ícone  ;

◆ quando o programa encontra um breakpoint, interrompe a execução, e mostra a Unit que contém o breakpoint, iluminando-o;

◆ continue passo a passo clicando sobre o ícone  , que roda uma linha de código por vez, entrando inclusive as funções ou rotinas chamadas pela rotina onde você está; ou sobre o ícone  , que roda uma linha de código por vez, mas não depura as funções ou sub-rotinas chamadas.

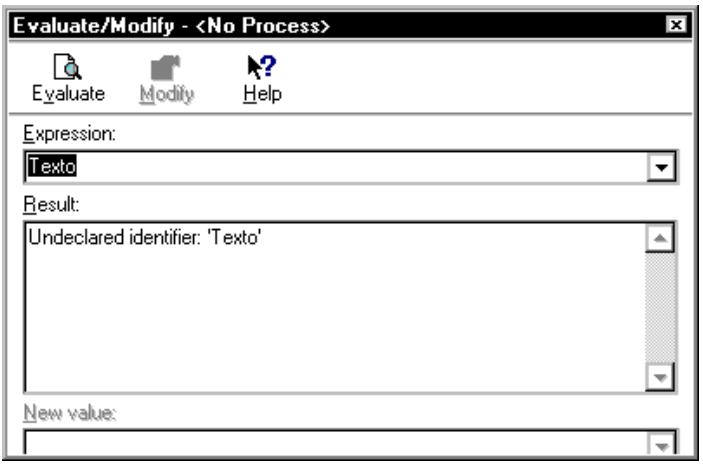
Acompanhando Valores de Variáveis

Para ver o valor de determinada variável em qualquer momento da depuração, siga os passos abaixo:

◆ ilumine, na Unit, a variável;

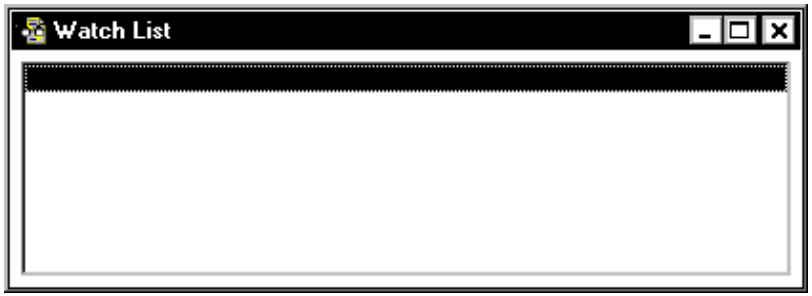
◆ escolha, no menu *Run*, a opção *Evaluate/Modify*;

◆ aparece o quadro Evaluate/Modify, que mostra o valor da variável naquele momento:



No quadro Expression, aparece a variável selecionada (se não sair o nome completo, escreva-o nesta caixa). Clique no botão *Evaluate* para verificar o valor da variável no momento. Se quiser, escolha um novo valor para a variável, na caixa New value, e clique no botão *Modify*.

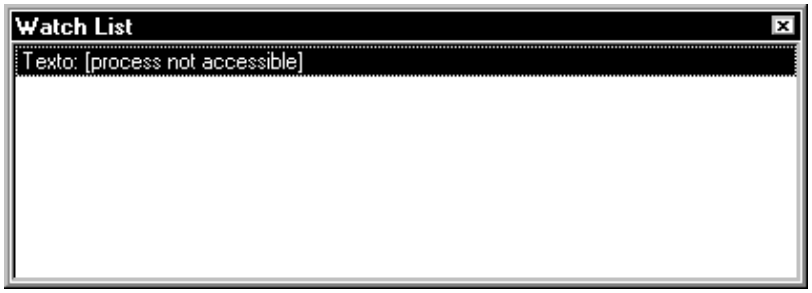
Já na janela Watch List, você pode acompanhar a mudança de valores de variáveis, conforme o programa vai sendo executado. Esta janela pode ser chamada através do menu *View*, opção *Debug Windows,* sub-opção *Watches*.



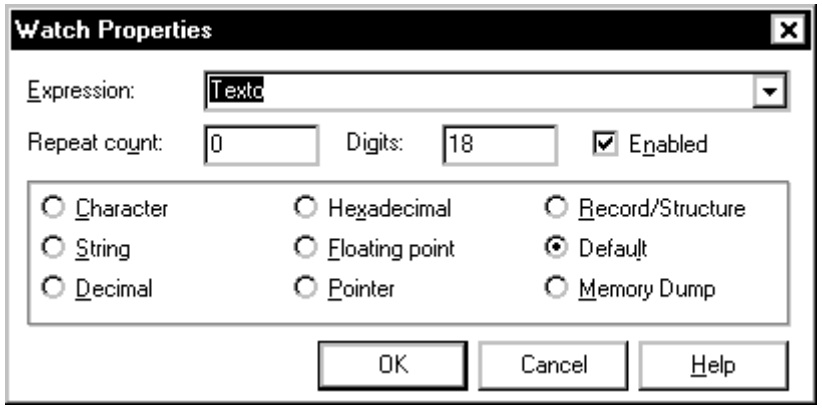
Para escolher as variáveis a serem acompanhadas:

◆ ilumine, na Unit, a variável;

◆ dê um clique sobre a variável com o botão direito do mouse, e escolha no menu que aparece, a opção *Debug,* e o item *Add Watch at Cursor*. Aparece o quadro abaixo:



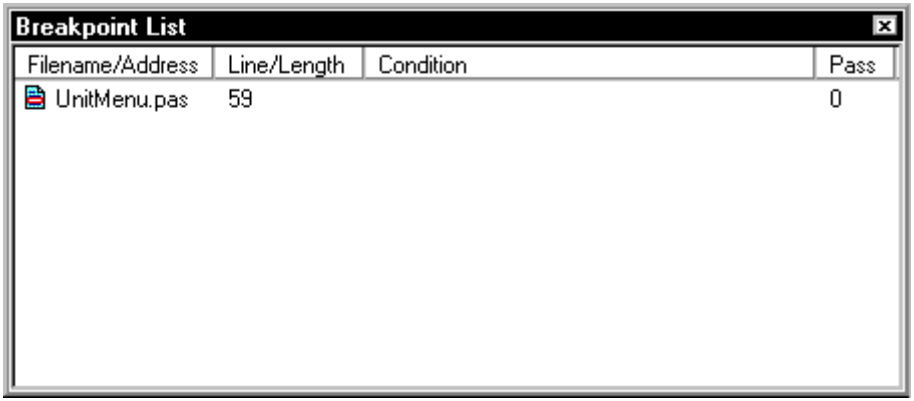
✔ se quiser modificar algo, clique com o botão direito do mouse sobre a janela, e escolha a opção Edit Watch. Aparece o quadro abaixo:



Quando você executar o aplicativo passo a passo, a janela de depuração acima mostrará o valor da variável inserida, a medida que esta vai sendo avaliada.

Lista de BreakPoints

Você pode ver uma lista com os breakpoints criados em sua aplicação. Para isto, escolha, no menu *View*, a opção *Debug Windows,* item *Breakpoints*. Aparece a lista de breakpoints:



Outras Ferramentas de Depuração

A versão 4 do Delphi introduz novas funções de depuração, que podem ajudar o programador a encontrar erros no código mais facilmente. Entre elas temos:

Module View: painel na forma de árvore que mostra informações detalhadas sobre os diferentes módulos de DLL's e EXE's que estão carregados, e o processo corrente que está sendo depurado. Acessado atraves do menu *View*, opção *Debug Windows*, item *Modules*.

CPU View: uma visão do que está ocorrendo por trás do código fonte, a nível de CPU. Acessada atraves do menu *View*, opção *Debug Windows*, item *CPU*.

Depuração de Processos múltiplos: com o visualizador de threads, é fácil examinar todas as threads que estão sendo executadas no momento, dando ao programador o poder de definir e mudar o processo corrente. Acessado atraves do menu *View*, opção *Debug Windows*, item *Threads*.

Event logging: a nova versão do Delphi oferece um armazenador de eventos totalmente configurável, permitindo que o programador acompanhe todos os eventos de alto e baixo nível da aplicação. Acessado atraves do menu *View*, opção *Debug Windows*, item *Event Logs*.

DataWatch Breakpoints: os programadores podem agora pedir ao Delphi para interromper a execução da aplicação de acordo com alguma mudança de um dado.

15
Configurando o Projeto

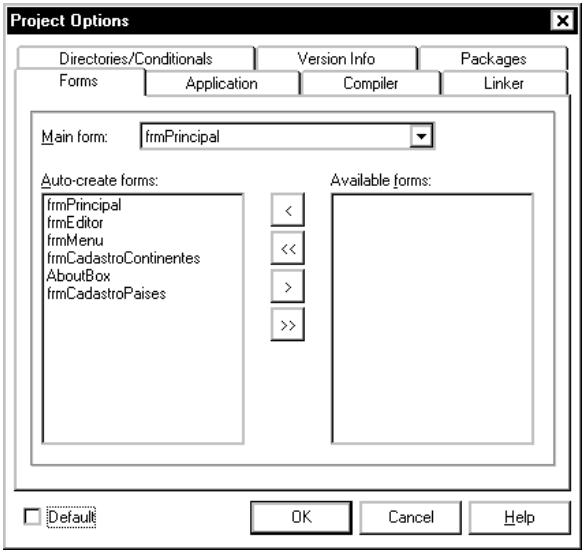
Introdução

Você pode mudar diversas opções de configuração de seu projeto, através do menu *Project*, item *Options*.

Quando você acessa esta opção, o Delphi oferece algumas páginas de configuração. Para escolher a página, clique sobre a aba em sua parte inferior.

Definindo o Formulário Principal

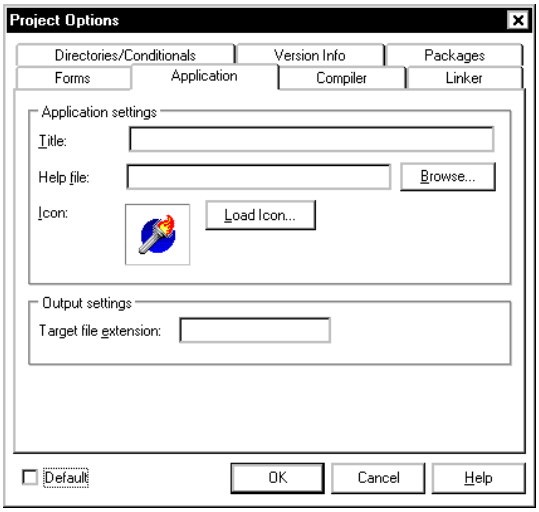
A página *Forms* define, na caixa Main form, qual será o formulário principal de sua aplicação. Clique sobre a seta à direita da caixa Main form para mudar o formulário principal:



Definindo Atributos da Aplicação

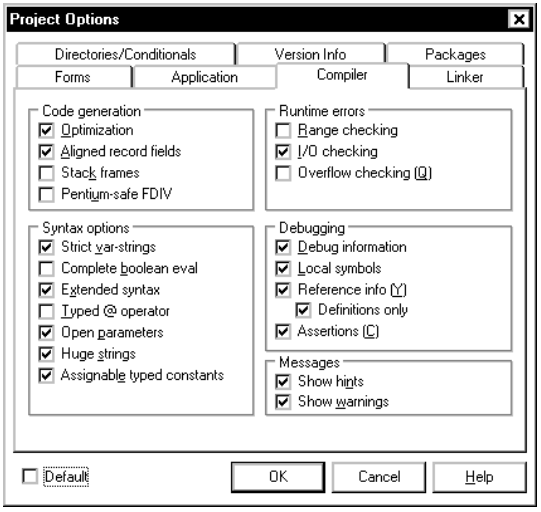
A página *Application* muda opções referente a sua aplicação em uso, como o nome da aplicação, dado pela caixa Title, o nome do arquivo de Help (dado pela

caixa Help file), e o ícone que representará a aplicação (dado pela opção Icon).



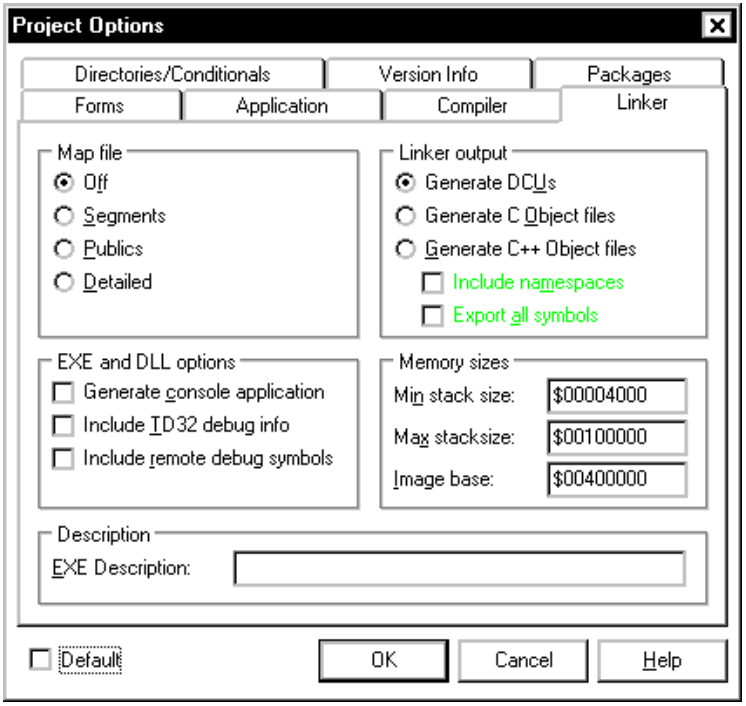
Definindo Opções de Compilação

A página *Compiler:* muda opções de compilação, como checagem de sintaxe, etc.



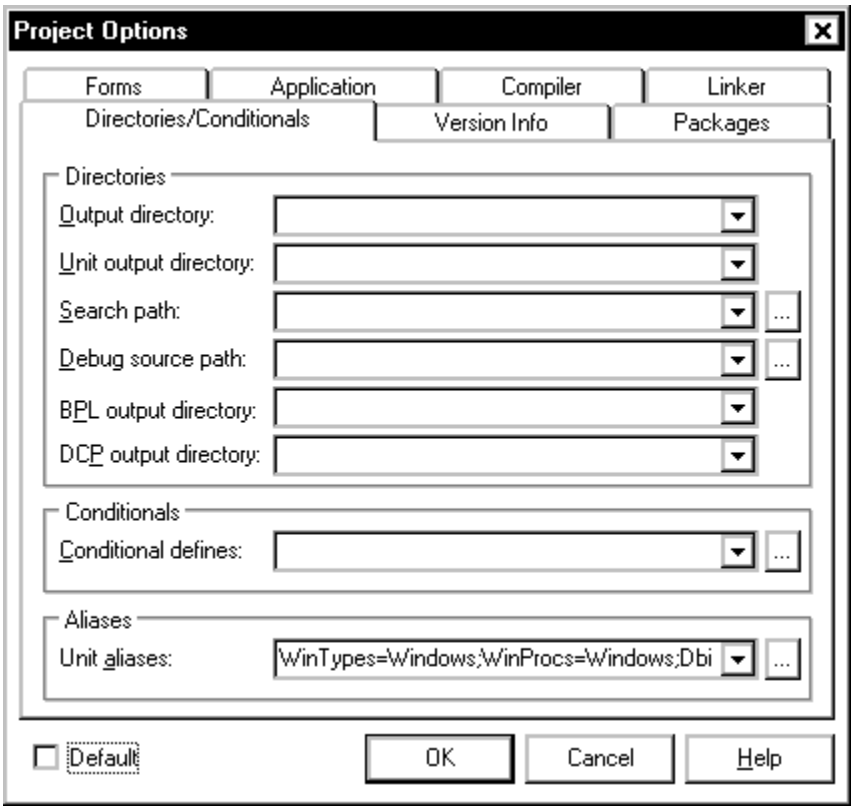
Definindo as Opções de Link-Edição

A página *Linker* define opções de link-edição.



Definindo a Localização de Arquivos

A página Directories/Conditionals permite que você especifique a localização dos arquivos necessários para compilar, link-editar e distribuir sua aplicação.


Project Options
Forms
Application
Compiler
Linker
Directories/Conditionals
Version Info
Packages
Directories
Output directory:
Unit output directory:
Search path:
Debug source path:
BPL output directory:
DCP output directory:
Conditionals
Conditional defines:
Aliases
Unit aliases:
WinTypes=Windows;WinProcs=Windows;Dbi
Default
OK
Cancel
Help

16

Configurando o Ambiente deTrabalho

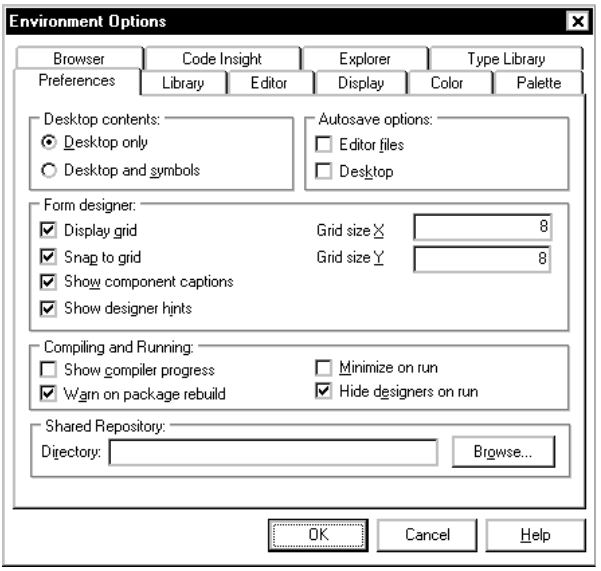
Introdução

Você pode mudar diversas opções de configuração de seu ambiente, através do menu *Tools*, opção *Environments Options*.

Quando você acessa esta opção, o Delphi oferece algumas páginas de configuração. Para escolher a página, clique sobre a aba em sua parte inferior.

Definindo as Preferências

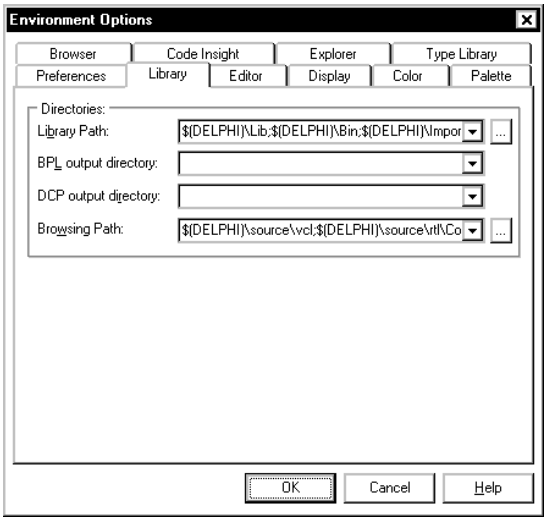
A página *Preferences* determina diversas opções para configurar seu ambiente de trabalho.



No quadro Autosave Options, marque a opção de autosalvamento do editor e da área de trabalho. Marque a opção *Display grid* se quiser visualizar as grades no formulário. Marque a opção *Show Compiler Progress* para visualizar o quadro de compilação.

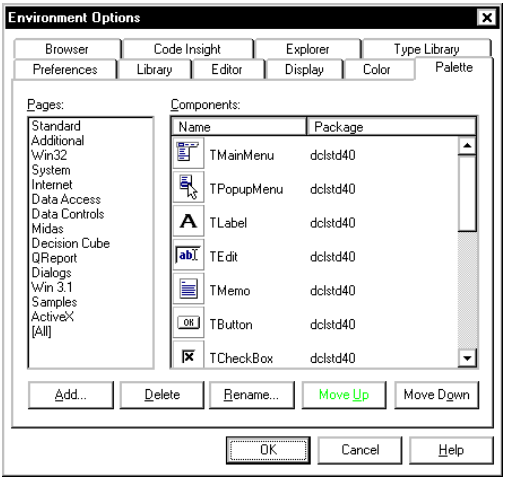
Definindo a Biblioteca de Componentes

A página *Library* controla aspectos da biblioteca de componentes.



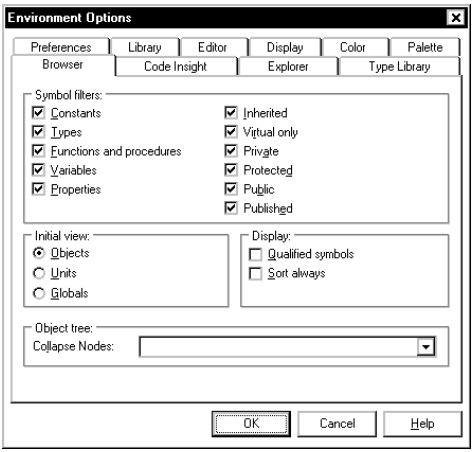
Configurando a Paleta de Componentes

A página *Palette* configura a Paleta de Componentes.



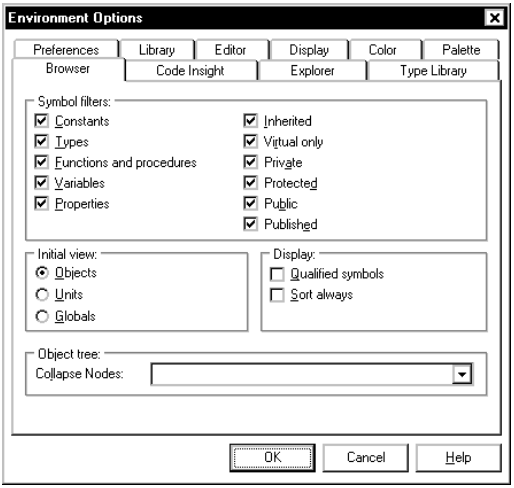
Configurando o Browser

A página *Browser* configura como e com que símbolos o Object Browser é mostrado.

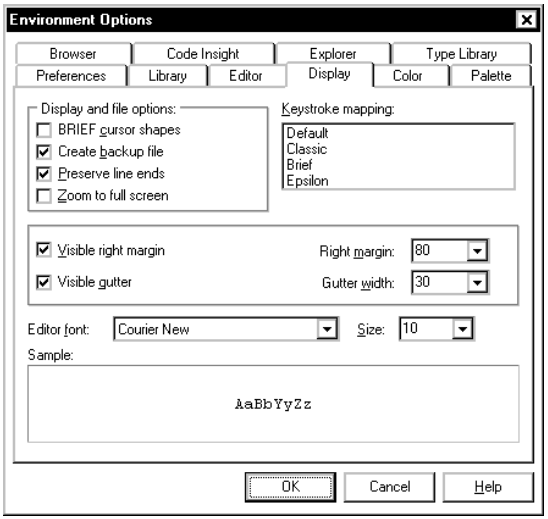


Configurando o Editor de Código

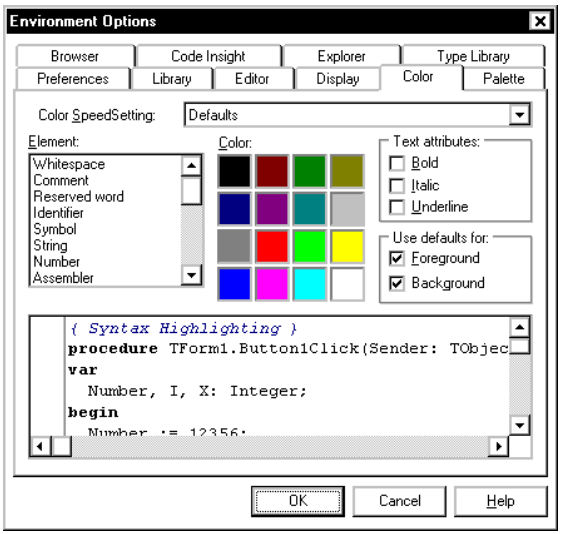
A página *Editor* configura o Editor de Código, mudando, por exemplo, o espaço percorrido pelo cursor quando você tecla Tab.



A página *Display:* configura a visualização do Editor de Código. Você poderá mudar fontes, tamanho da fonte, margens, etc.

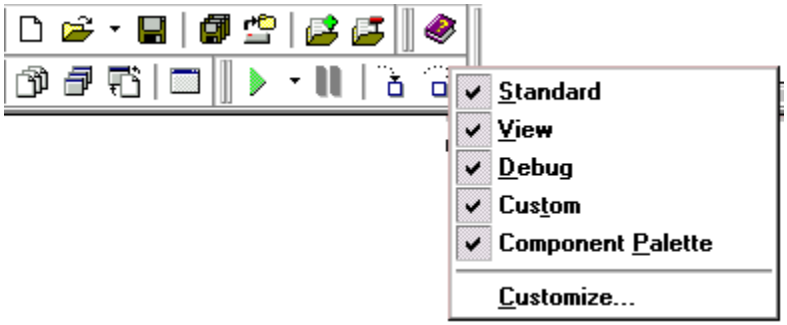


A página *Color* configura as cores usadas no Editor de Código para breakpoints, comandos, palavras reservadas, etc.

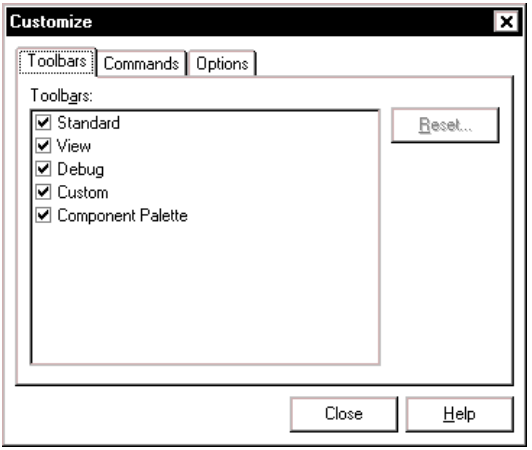


Configurando a Barra de Ferramentas

Você pode configurar a Barra de Ferramentas do seu ambiente de trabalho com uma grande facilidade. Clicando com o botão direito do mouse sobre a Barra de Ferramentas, aparecerá um menu Pop-up com algumas opções:

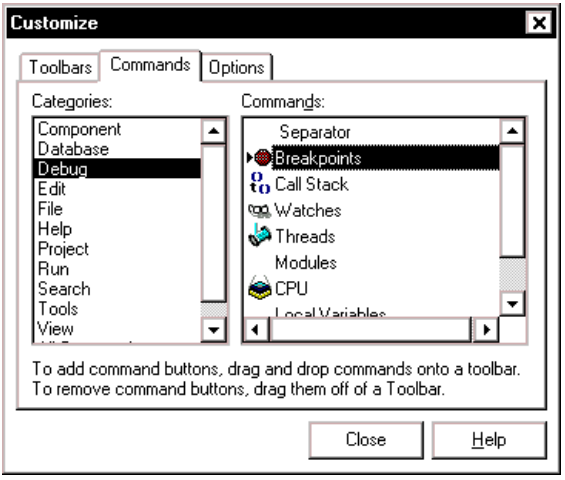


As primeiras opções definem quais barras de ferramentas estão visíveis. Através da opção Customize você pode configurar as barras de ferramentas:



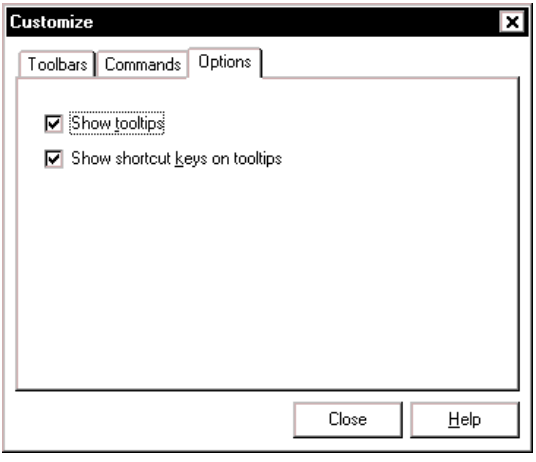
A página Toolbar permite que você escolha que barras estarão visíveis.

A página Commands permite que você adicione ou exclua ícones da barra:



Para adicionar um ícone, basta arrastá-lo da caixa Commands para uma das barras de ferramentas. Para excluir um ícone de uma barra, bastaa arrastá-lo para o quadro Customize.

A página Options permite que você decida se as legendas para os ícones serão mostradas. Para isso, marque a opção *Show Tooltips*:



Caso você não goste da modificação e queira voltar a configuração anterior, basta você clicar no botão *Reset*, na página Toolbars. Clique no botão *Close* para fechar o quadro de diálogo.

Configurando a Paleta de Componentes

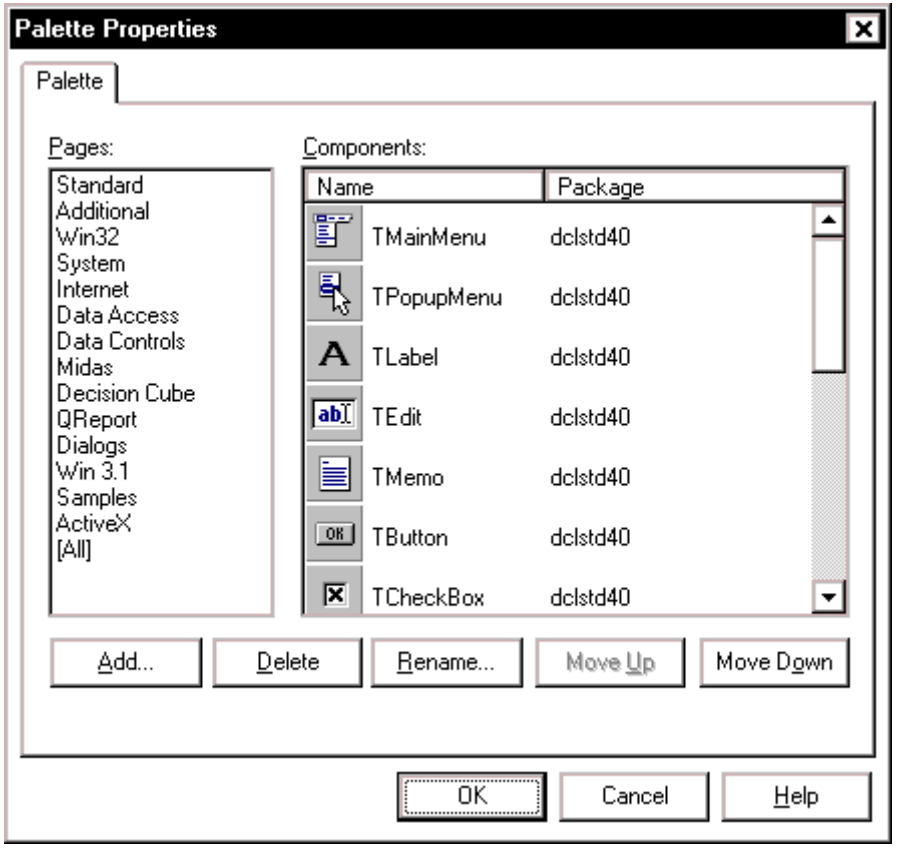
Você pode configurar a Paleta de Componentes do seu ambiente de trabalho com uma grande facilidade. Clicando com o botão direito do mouse sobre a Paleta de Componentes, aparecerá um menu Pop-up com algumas opções:

Show Hints: se esta opção estiver marcada, ao posicionar o mouse sobre um dos componentes da paleta você terá acesso a um legenda de identificação.

Hide: ao clicar nesta opção você estará retirando a Paleta de Componentes de seu ambiente de trabalho. Para fazer com que a Paleta de Componentes volte ao ambiente de trabalho, basta clicar na opção *Component Pallete* do menu *View*.

Help: clicando nesta opção aparecerá o quadro de diálogo *Delphi Help*. Neste quadro você poderá pedir esclarecimentos sobre qualquer um dos componentes, basta clicar sobre o nome da ficha de componentes desejada e após clicar sobre o componente.

Properties: clique nesta opção para configurar sua paleta, mudando a ordem das páginas, por exemplo. Aparecerá o quadro de diálogo *Palette Properties:*



Na caixa *Pages* você deve selecionar a página desejada; automaticamente aparecerão os componentes desta página na caixa *Components*. Você deve selecionar o componente desejado e clicar no botão:

Add: para adicionar na página.

Delete: para excluir da página.

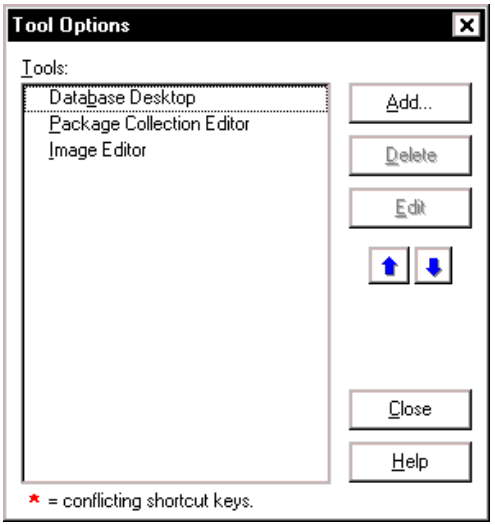
Rename: para renomear o componente.

Caso você não goste da modificação e queira voltar a configuração anterior, basta você clicar no botão *Reset Defaults*. Se você tiver alguma duvida clique no botão *Help*, aparecerá o quadro de diálogo *Delphi Help*, lhe oferecendo alguns esclarecimentos sobre o assunto. Clique no botão *Close* para fechar o quadro de diálogo. E clique no botão Ok para confirmar uma operação.

Acrescentando Utilitários ao Menu Tools

Através da opção *Configure Tools* do menu *Tools*, você pode acrescentar os utilitários do Delphi no menu Tools. Isto facilita seu trabalho pois você não perderá seu tempo em procurá-los dentro de pastas e mais pastas. Vamos ver agora como fazer para acrescentar, editar ou excluir itens no menu Tools:

◆clique na opção *Configure Tools*, do menu *Tools*. Aparece o quadro de diálogo Tools Options:



◆selecione uma das opções da caixa Tools e clique no botão *Delete*, você estará excluindo a opção selecionada;

◆selecione uma das opções da caixa Tools e clique no botão *Edit*. Aparecerá o quadro de diálogo Tools Properties para que você possa editar informações sobre o item selecionado;

◆clique no botão *Add* para adicionar um determinado item no menu;

◆clique no botão *Close*, para fechar o quadro de diálogo;

◆clique no botão *Help*, para obter uma ajuda sobre determinados itens do quadro de diálogo.