1. IDE

IDE – Ambiente Integrado de Desenvolvimento

A figura a seguir mostra o menu principal (*MainMenu*), a barra de ferramentas (*ToolBar*) e a paleta de componentes do Delphi (*Component Palette*).

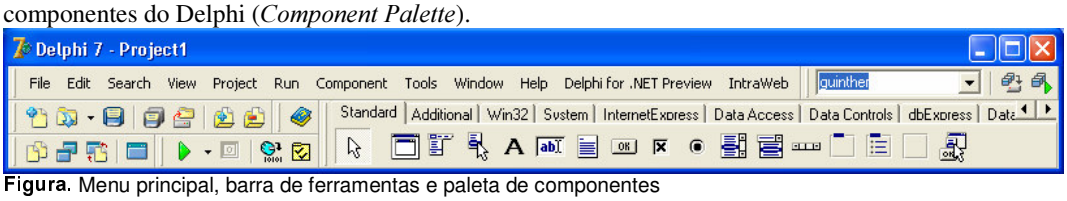


Figura. Menu principal, barra de ferramentas e paleta de componentes

Você pode configurar a barra de ferramentas adicionando e retirando itens. Para isso dê um clique de direita sobre ela e escolha *Customize*.

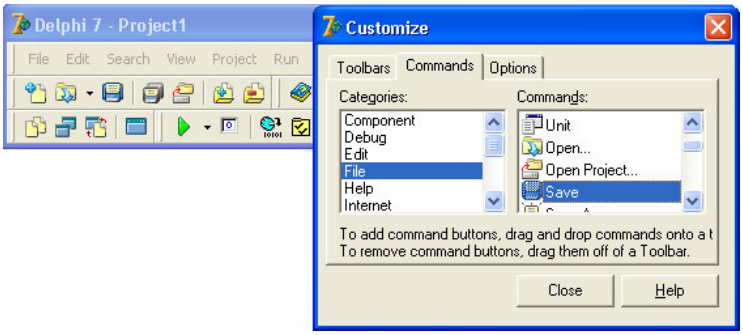


Figura. Configurando a Toolbar do Delphi

Você pode configurar a paleta de componentes dando um clique de direita sobre ela e escolhendo a opção *Properties*.

Object Inspector

Utilizado para definir propriedades e manipuladores de eventos para componentes.

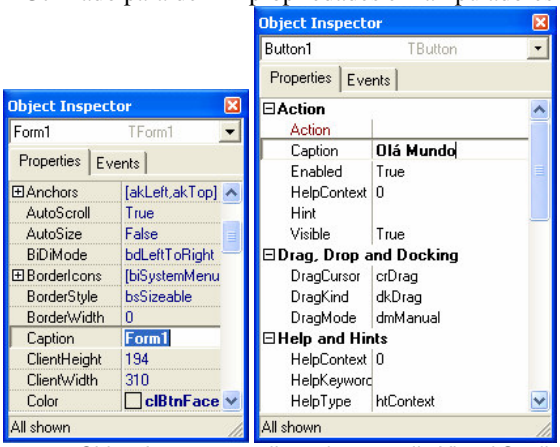


Figura. Object Inspector, estilo padrão e estilo Visual Studio

No Delphi 6/7 suporta ainda a opção *Expand Inline*, onde o *Object Inspector* expande em árvore as propriedades do componente relacionado.

Você pode personalizar o *Object Inspector* dando um clique de direita sobre ele e escolhendo a opção *Properties*. Você pode agrupar propriedades por categoria, ocultar grupos de propriedades, configurar cores, estilos, etc.

[g1] Comentário: Começar com um pequeno histórico, do Pascal ao Delphi, Borland, orientação a objetos x programação estruturada, falar das versões do Delphi

[g2] Comentário: Falar que Delphi deixou de ser só uma IDE / ferramenta e agora é também o nome da linguagem, ou seja, "desenvolvemos na linguagem Delphi", e não mais Object Pascal

[g3] Comentário: Fazer uma introdução à programação visual, mostrando objetos, formulários, exemplo de cadastro / navegação, comparar com Pascal e a linguagem estruturada

[g4] Comentário: Dar sugestões de bons livros, introdutórios e profissionais, colocar lista de sites, componentes, etc.

[g5] Comentário: Mostrar teclas de atalho do O.I., F11. Fechar e abrir o O.I.

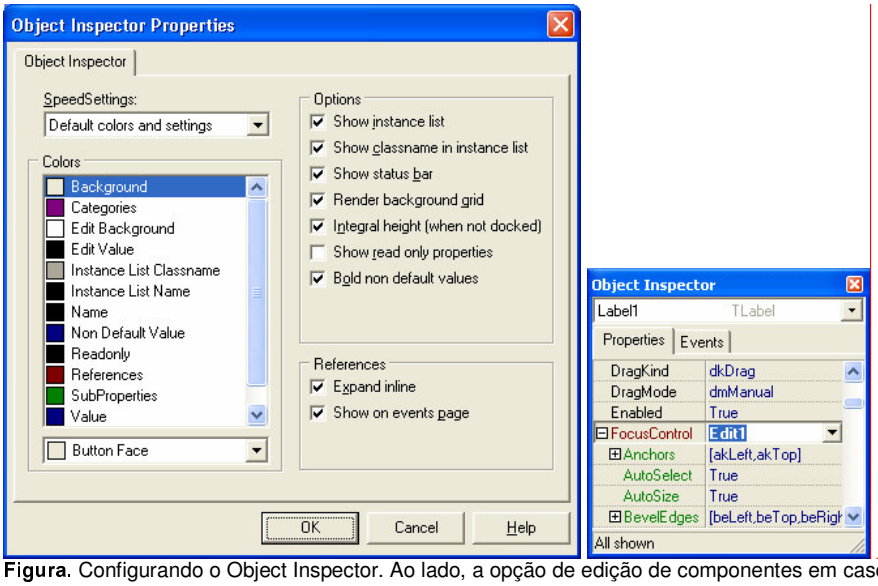


Figura. Configurando o Object Inspector. Ao lado, a opção de edição de componentes em cascata

[g6] Comentário: Mostrar propriedades do tipo componente, agrupar por categoria

Form Designer e Code Editor

Editor do formulário e editor de código.

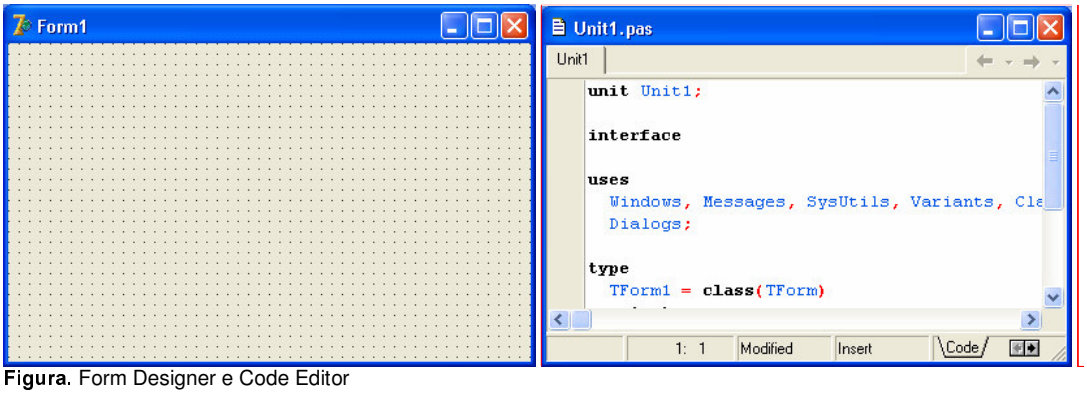


Figura. Form Designer e Code Editor

[g7] Comentário: Mostrar técnicas de uso do Editor, como navegação, múltiplas páginas, trocar ordem, etc.

[g8] Comentário: Mostrar teclas de atalho do editor, F12. Fechar e abrir o form editor

Propriedades

Use o *Object Inspector* para mudar as propriedades de um componente no *Form Designer*.

Figura. Configurando as propriedades de um componente

[g9] Comentário: Important e – selecionar vários componentes e configurar as propriedades comuns de uma só vez

Eventos

Incluindo um manipulador de evento:

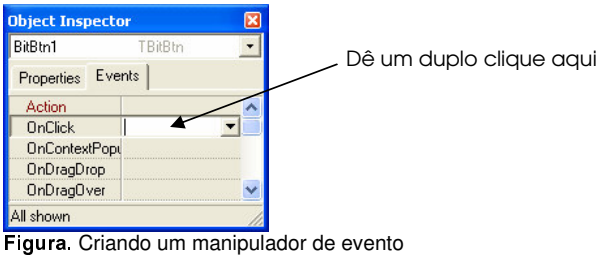


Figura. Criando um manipulador de evento

Atenção: Ao incluir um manipulador de evento, a maioria dos desenvolvedores iniciantes tende a escolher um método na caixa de seleção ao lado do nome do evento, ao invés de dar dois cliques na área branca para incluir um novo. Com isso, um componente utilizará o **mesmo** manipulador de outro componente.

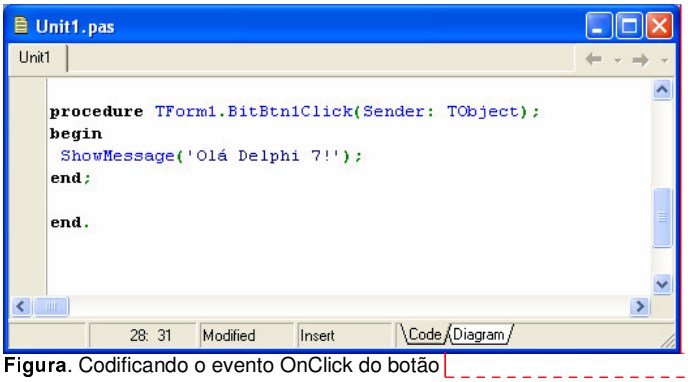


Figura. Codificando o evento OnClick do botão

Atenção: Para limpar um manipulador de evento retire apenas o código que você digitou (entre o *begin* e o *end* e seção *var*), e salve ou rode a aplicação.

Configuração do Ambiente

Abra o menu *Tools|Environment Options*.

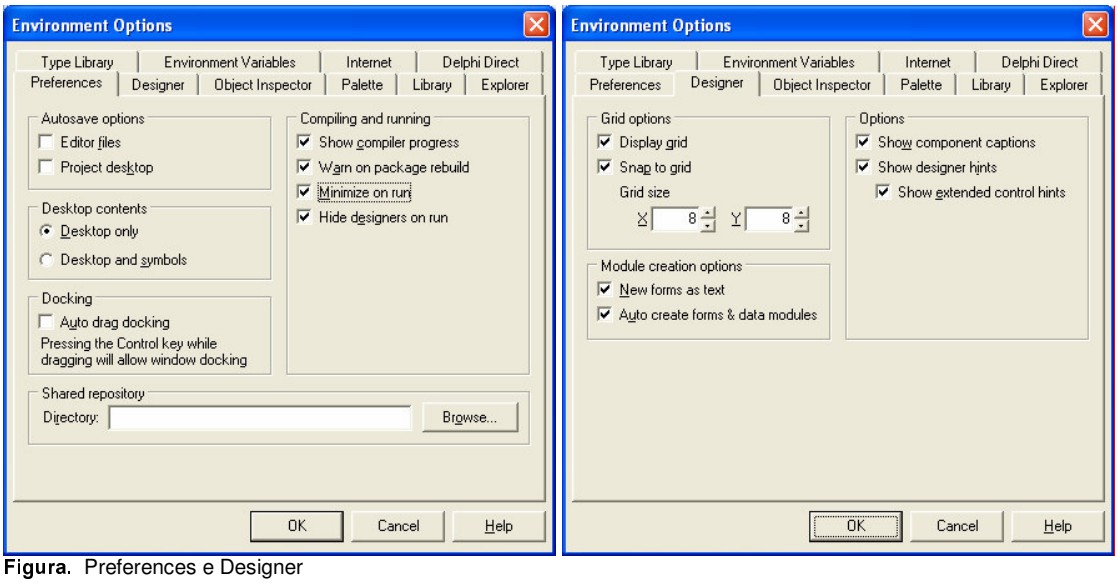


Figura. Preferences e Designer

[g10] Comentário: Exercício – todos devem fazer o "Olá Mundo", observar o formulário, ver como é feita a compilação, o que é gerado (.EXE), etc.

[g11] Comentário: Colar um botão e incluir um manipulador. Depois copiar o botão e mostrar como retirar o manipulador do segundo e inserir um novo. Mostrar que ao salvar ou rodar a aplicação os manipuladores não implementados são retirados. Retirar um manipulador e mostrar o erro, e resolver

[g12] Comentário: Mostrar principais propriedades, Hint, cor, posição, alinhamento, etc.

[g13] Comentário: Falar do shared repository brevemente, que é usado no SIE

Configuração do Editor

Clique em *Tools|Editor Options*.

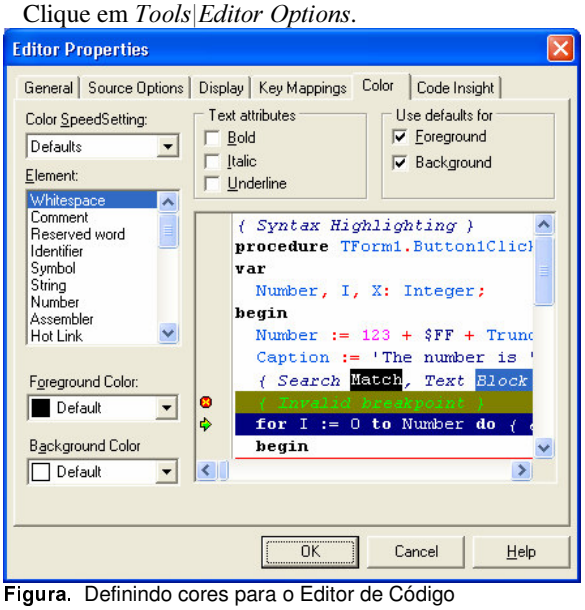


Figura. Definindo cores para o Editor de Código

Atenção: As opções do depurador serão vistas no módulo 3

To-Do List

Lista de tarefas a serem feitas em uma unit ou formulário. Ao clicar no item do *To-Do list*, o Delphi abre a unit e posiciona o cursor onde o código dever ser colocado. Pode ser exportado para HTML.

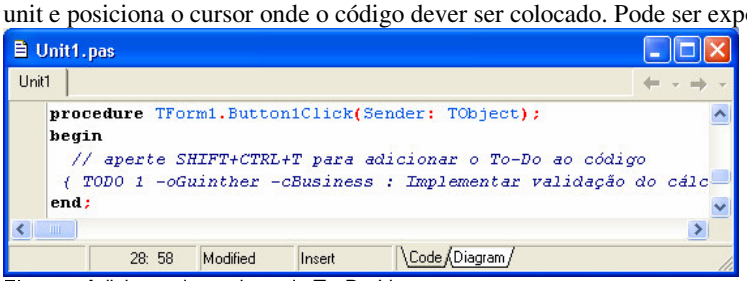


Figura. Adicionando um item de To-Do List

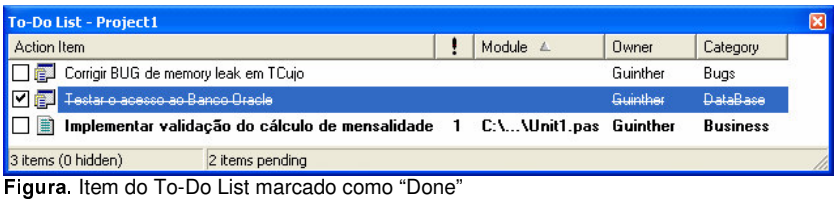
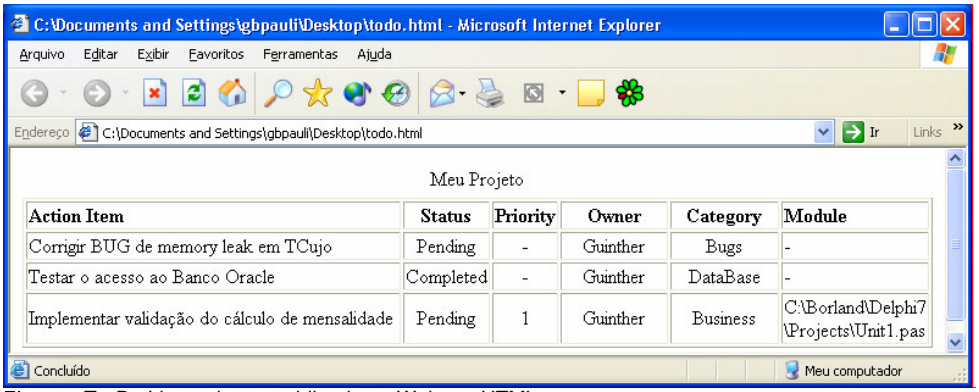


Figura. Item do To-Do List marcado como "Done"



[g14] Comentário: Itens a fazer vinculados ao designer ficam gravados no arquivo ".todo"

Figura. To-Do List pode ser publicado na Web em HTML

Object Repository

Repositório de objetos, *wizards*, *templates*, formulários, *DataModules*, etc.

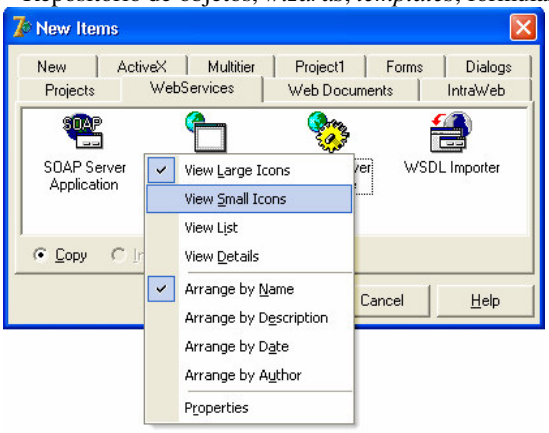


Figura. Object Repository

Você pode adicionar um formulário ao *Object Repository* para que possa ser usado/compartilhado por outras aplicações ou outros desenvolvedores.

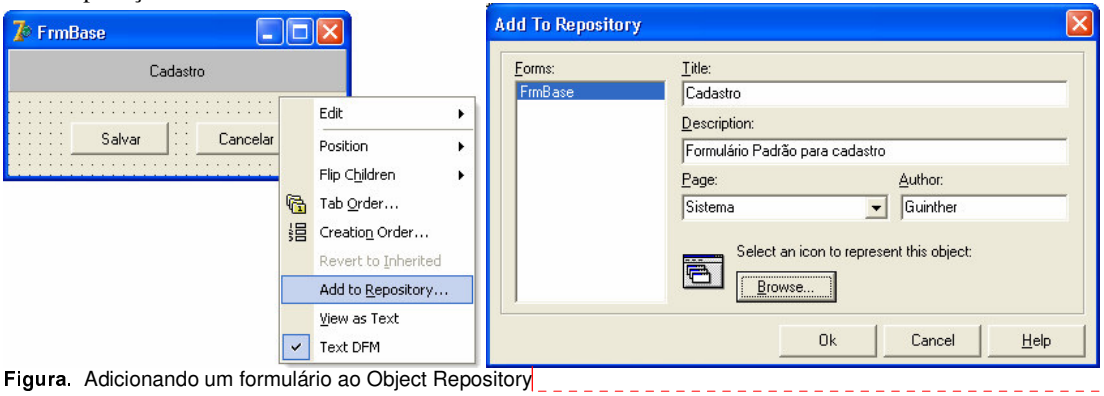


Figura. Adicionando um formulário ao Object Repository

[g15] Comentário: Falar que esses formulários podem ser compartilhados por vários desenvolvedores, como no SIE (shared repository)

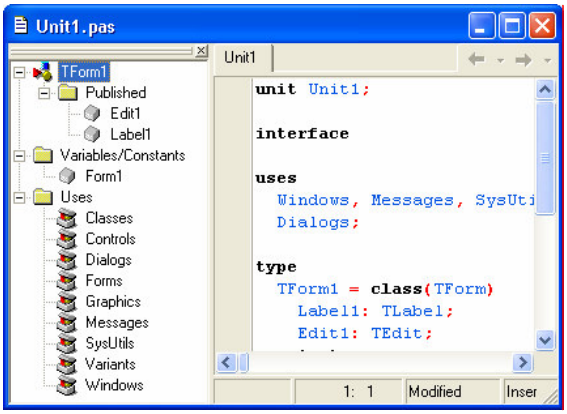
Dica: Um poderoso recurso disponível através do *Object Repository* é o *Virtual Form Inheritance* (Herança Visual de Formulário), visto em detalhes no módulo 3.

Você pode usar um repositório compartilhado configurando a opção *Shared Repository* em *Tools|Environment Options>Preferences*

Figura. Repositório compartilhado

Code Explorer

Permite acesso rápido (navegação) às várias seções de código de uma unit, como classes, variáveis, funções, procedimentos, métodos, cláusula uses, etc. Representa as seções organizadas de forma hierárquica. Funciona sincronizado ao *Code Editor*.



[g16] Comentário: Falar que essas seções (unit, interface, etc) são vistas mais adiante

Figura. Code Explorer

Você pode personalizar o *Code Explorer* acessando o menu *Tools|Environment Options>Explorer*. A tecla de atalho para exibir/ocultar o *Code Explorer* é Shift+Ctrl+E.

Code Browse

Segure a tecla *Control* e clique em um identificador para acessar sua definição no código fonte.

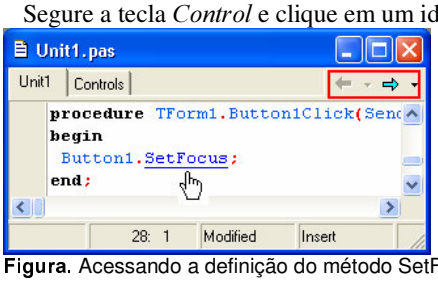
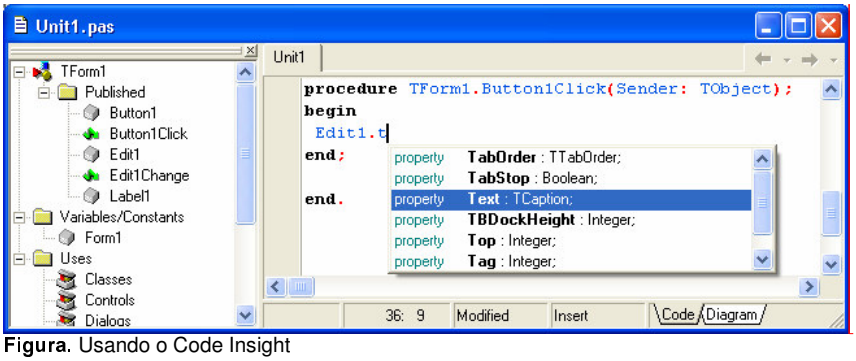


Figura. Acessando a definição do método SetFocus

Atenção: A localização das units de código fonte depende das configurações definidas em *Tools|Environment Options>Library>Browse Path* ou ainda nas opções do projeto. Essa opção indica em que diretórios os fontes podem ser localizados.

Code Insight

Permite inserir código rapidamente a partir do editor.



[g17] Comentário: Falar que se o code insight não funciona é pq tem erro no código

Figura. Usando o Code Insight

Teclas de atalho
Ctrl + Space - *Code Completation*
Ctrl + J – *Code Template* (veremos a seguir como configurar essa opção)
Ctrl + Shift + Space – *Code Parameters*

Dica: O Delphi 7 permite abrir a definição de um identificador a partir do próprio *Code Completation*. Para isso, segure a tecla *Control* enquanto o *Code Completation* é exibido, e clique sobre o identificador.

Você pode configurar o *Code Insight* acessando o menu *Tools|Editor Options>Code Insight.*

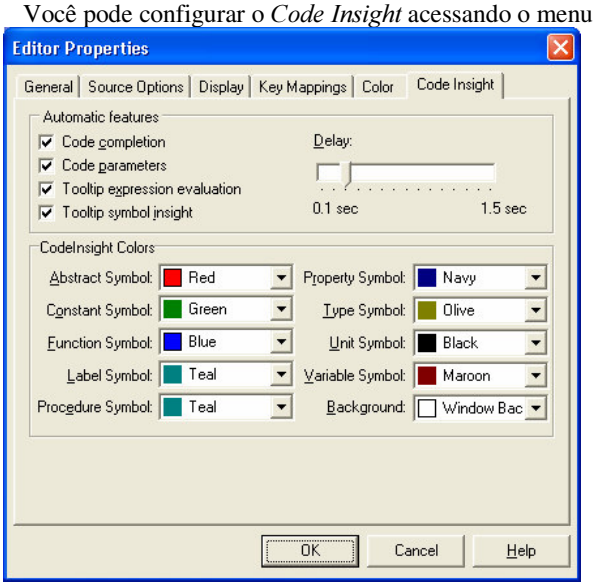


Figura. Configurando o Code Insight

Dica: Existe um "insight" bastante útil chamado *Class Completation*, que auxilia na criação de classes e geração de cabeçalhos de métodos e propriedades. Esse recurso será visto no módulo 3. A opção *Tooltip symbol insight* exibe detalhes sobre uma variável em forma de *Hint*, e será vista no módulo 3.

Source Options

Permite configurações opções para o código fonte que é exibido no editor. Podemos editar arquivos externos como HTML,XML, JS, SQL,etc., com *Sintax Hightlight.*

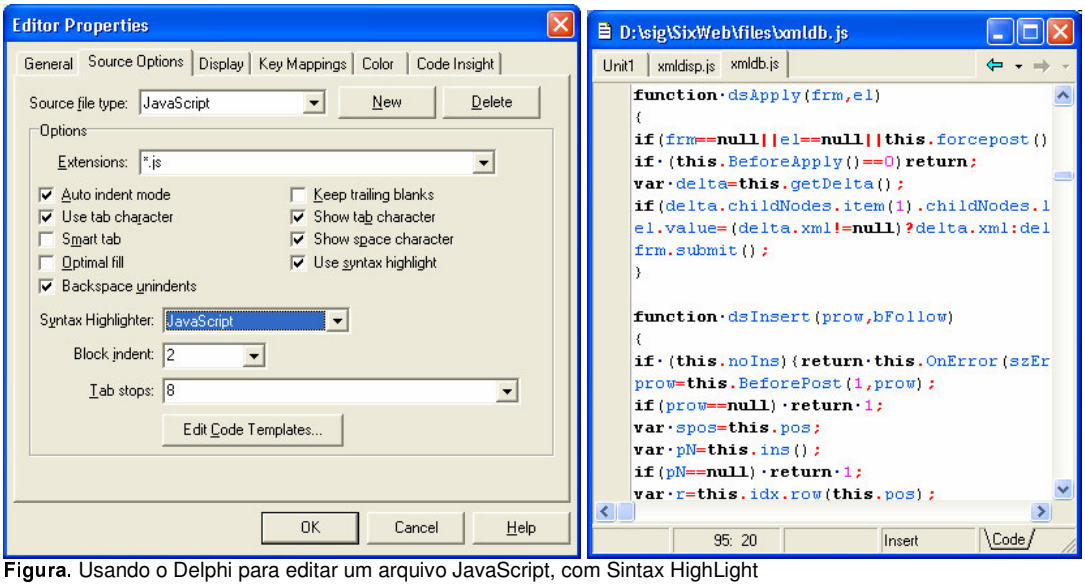


Figura. Usando o Delphi para editar um arquivo JavaScript, com Sintax HighLight

Code Templates

Permite definir modelos de códigos (*templates*), que podem ser usados no editor (**Ctrl+J**).

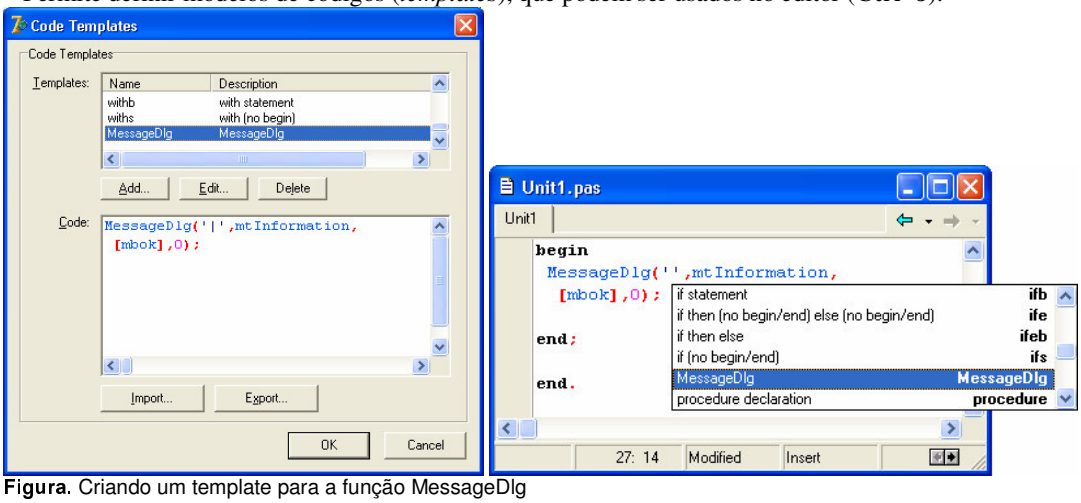


Figura. Criando um template para a função MessageDlg

Dica: O Delphi possui uma série de *tamplates* prontos, para classes, *procedures*, blocos *if*, *try*, etc.

[g18] Comentário: Mostrar como usar, dar um exemplo

Web Documents

Permite a criação de documentos HTML, WML e XHTML com *Sintax Highlight* e *Code Insight*.

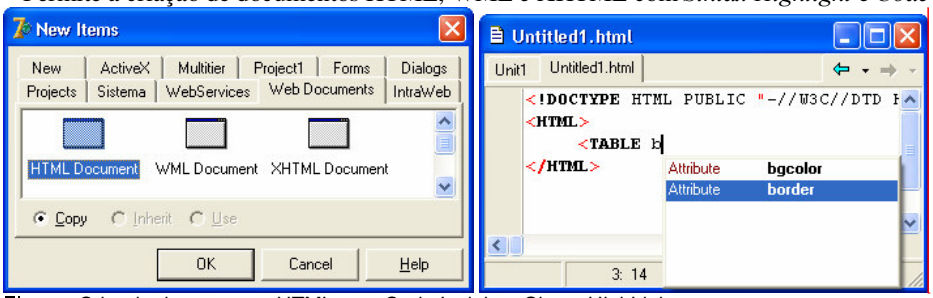


Figura. Criando documentos HTML com Code Insight e Sintax HighLight

[g19] Comentário: Falar que, obviamente, o Delphi NÃO é um editor HTML

Object Tree View e Diagram Page

Permite a criação de diagramas para documentação da relação entre componentes em formulários e *DataModules*. O diagrama fica armazenado em um arquivo com o mesmo nome da unit (extensão .ddp).

Figura. Relação entre uma Label e um Edit representada no Diagram Editor

[g20] Comentário: DataMo dule? Falar que isso Será visto mais adiante

[g21] Comentário: Falar que isso não é um diagrama de classes nem de tabelas, é só pra documentação mesmo, dar exemplo que é muito usado em artigo

2. Projetos, Units e Forms

Projeto (.dpr)

Arquivo do projeto (menu *Project|View Source*).

Figura. Arquivo do projeto

[g22] Comentário: Falar que o desenvolvedor raramente altera esse arquivo. Falar que no SIE há um código especial para validação nesse arquivo

[g23] Comentário: {$R *.res} ????
Falar que não é comentário, e explicar brevemente o que é um recurso

Unit (.pas)

Onde são definidos tipos, variáveis, classes, manipuladores de eventos, etc.

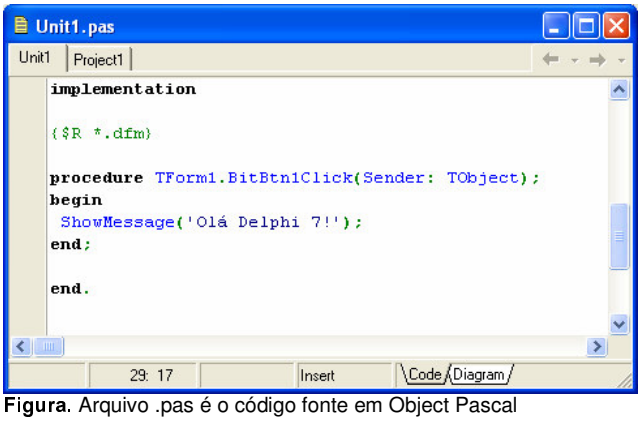


Figura. Arquivo .pas é o código fonte em Object Pascal

Descrição Textual do Formulário (.dfm)

Selecione o formulário e pressione ALT+F12, ou clique de direita e escolha *View as Text*.

Figura. Arquivo DFM é a descrição textual do formulário

[g24] Comentário: Explicar com o Delphi compila o .pas e vincula o .dfm

Text DFM x Binary DFM

Especifica como deve ser salvo o arquivo .dfm, sem em formato texto ou binário.

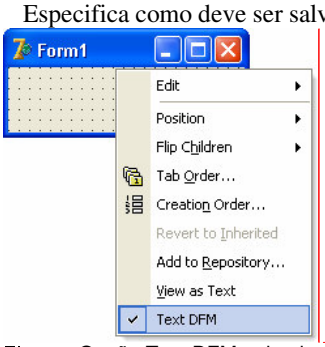


Figura. Opção Text DFM, ativada por padrão a partir do Delphi 5

[g25] Comentário: Explicar o porquê de a Borland ter feito isso

Mais Extensões

Além das extensões apresentadas (.dpr, .pas e .dfm), temos ainda:

Extensão	Descrição	É fonte?
cfg	Configurações do projeto (diretivas de compilação)	Sim
dof	Contém todas as opções de projeto definidas na janela *Project|Options*	Sim

res	Arquivo de Recursos, guarda o ícone da aplicação e outros recursos que você queira definir	Sim
exe	Projeto compilado, Win32	Não
dcu	Unit Compilada	Não
ddp	*Delphi Diagram Portfolio* – Diagrama do Editor.	
bpl	Pacote compilado	Não
dpk	Fonte de pacote	Sim
dcr	Recursos para componentes, como ícones da paleta	Sim
~dfm ~pas ~ddp	Backups – Você pode desativar a opção *Tools|Editor Options>Display>Create backup files*	

Tabela. Extensão de arquivos usados no Delphi

Project Manager

Utilize o *Project Manager* para gerenciar projetos, units, formulários, etc. Pode ser acessado a partir do menu *View|Project Manager* (Ctrl+Alt+F11)

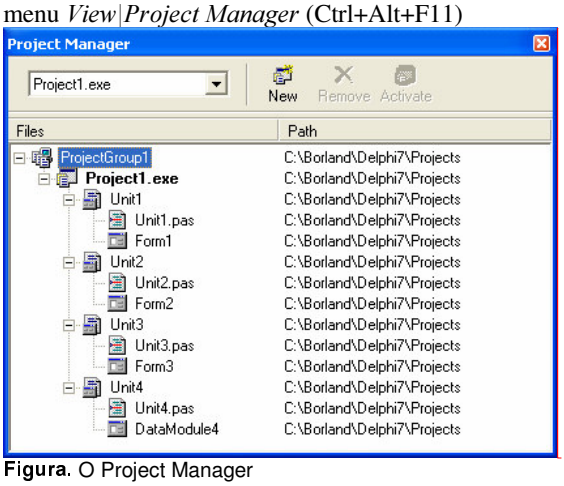


Figura. O Project Manager

[g26] Comentário: Explicar como adicionar / remover units a partir do Project manager. Importante – mostrar como gerenciar mais de um projeto

View Forms / View Units

Forms -> Menu *View|Forms* (Shift+F12)
Units -> Menu *View|Units* (Ctrl+F12)

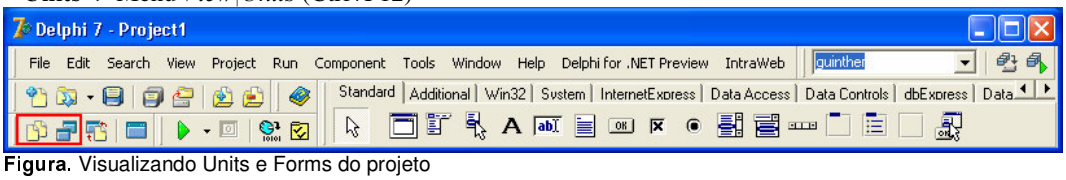


Figura. Visualizando Units e Forms do projeto

Adicionando e removendo units do projeto

Você pode usar o *Project Manager* ou ainda o menu *Project|Add to Project* e *Project|Remove from Project* para adicionar e remover units de um projeto. A remoção de uma unit do projeto não apaga o arquivo do disco. Após uma unit ser removida do projeto, se alguma unit ainda possuir uma referência a essa unit na cláusula uses, a unit ainda será compilada com o projeto.

Opções do Projeto – Forms e Application

Menu *Project*|*Options* (Shift+Ctrl+F11)

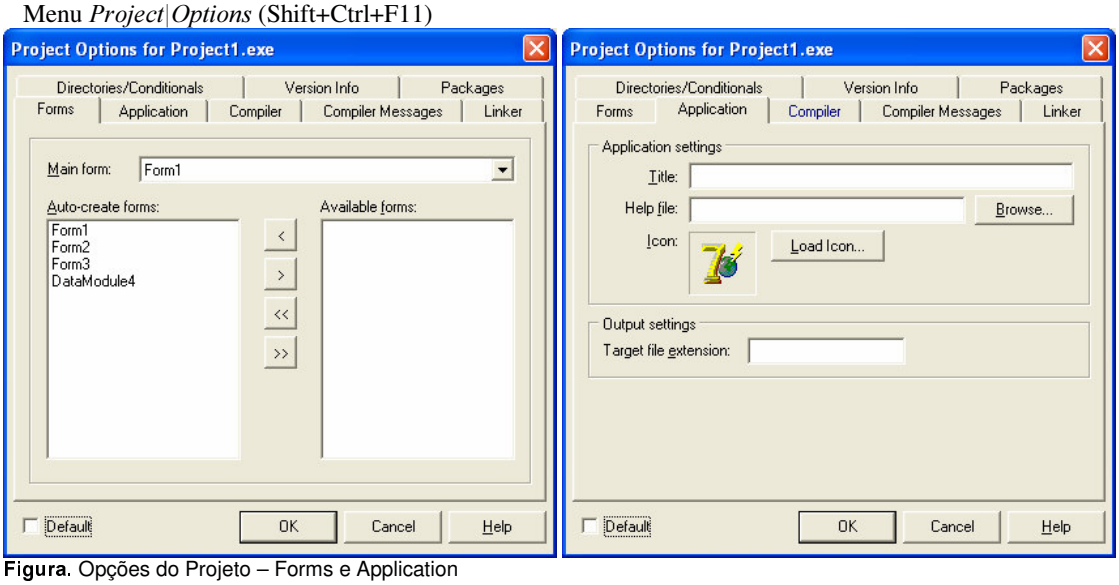


Figura. Opções do Projeto – Forms e Application

Aqui podemos definir a ordem de criação dos formulários, qual será o formulário principal da aplicação e quais os formulários que devem ser criados automaticamente. Alterações nas configurações da aba *Forms* refletem diretamente no arquivo .dpr.

Opções do Projeto – Compiler e Directories Conditionals

Compiler – opções do compilador. Essas configurações são refletidas em diretivas de compilação definidas no arquivo .cfg

Directories/Conditionals – diretórios onde serão gerados os arquivos compilados e onde devem ser localizadas outras units no momento da compilação. Permite ainda definir diretivas condicionais.

[g27] Comentário: Falar brevemente o que é uma diretiva de compilação ($X+)

[g28] Comentário: Criar dois diretórios e mostrar o .exe sendo gerado em um e as .dcu no outro

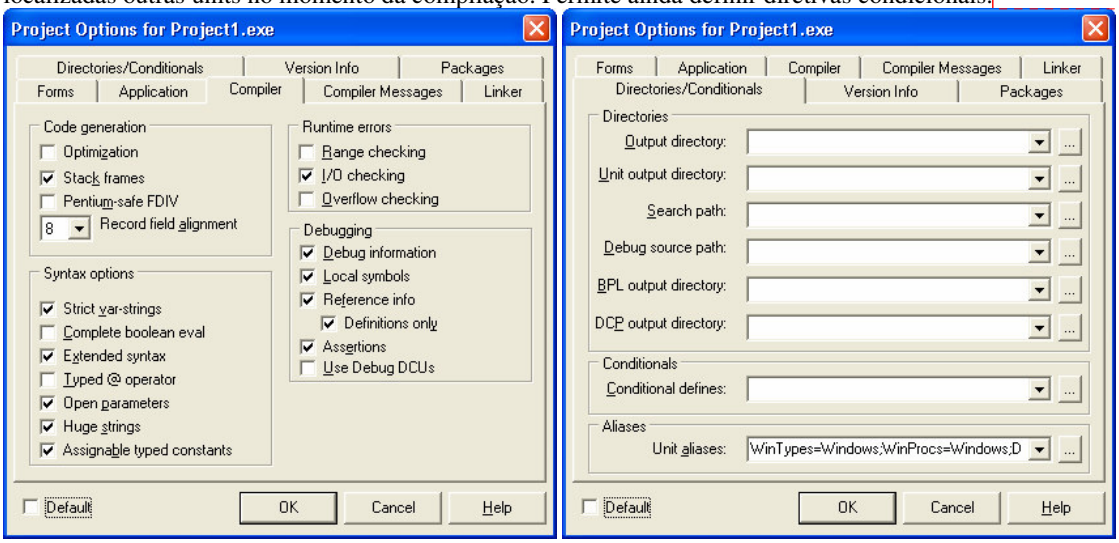


Figura. Opções do compilador e Directories/Conditionals

3. Form Designer

Configurando Propriedades

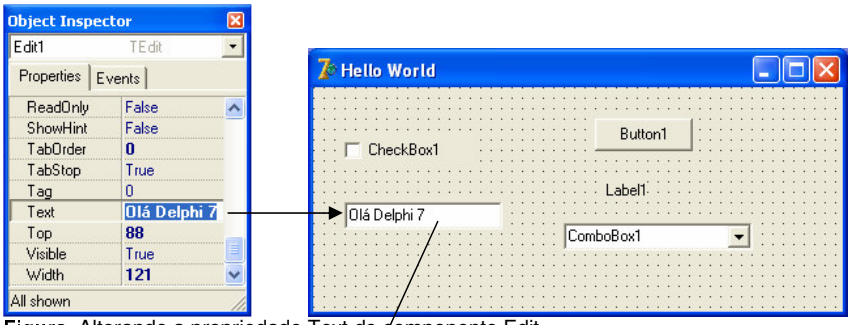


Figura. Alterando a propriedade Text do componente Edit

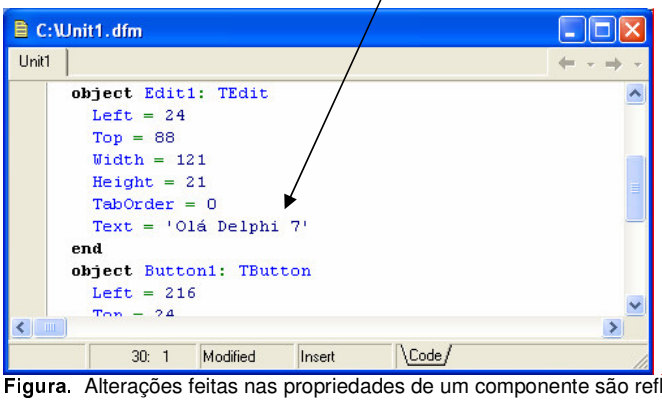


Figura. Alterações feitas nas propriedades de um componente são refletidas no arquivo DFM

[g29] Comentário: Copiar um objeto para o Word ou NotePad, alterar uma propriedade e colar de volta no formulário

Nomeando Componentes

Use a propriedade *Name* para definir o **NOME** do componente.
Não use identificadores inválidos, com acentos, espaços em branco, etc.
Abrevie o tipo de componente deixando somente as consoantes e alguma vogal.
Indique a função do componente no final do seu nome.
Ex.:
Um *TButton* (botão) que irá fechar o programa poderia se chamar *BtnFecharPrograma*;
Um *TForm* (formulário) de cadastro de clientes poderia se chamar *FrmClientes*;
Um *TEdit* (caixa de texto) que irá receber o nome do cliente poderia se chamar *EdtNomeCliente*.

Atenção: *Name* NÃO é o mesmo que *Caption* ou *Text*. Jamais limpe a propriedade *Name* de um componente.

[g30] Comentário: Mostrar o que acontece ao limpar o Name de um componente

Componentes visuais vs. não-visuais

Visuais – *Edit*, *Button, Label,* etc. Descendem de *TControl.*
Não-visuais – *Timer, ClientDataSet*, etc. Descendem de *TComponent*.

Movendo Componentes

Use *Ctrl*+Setas

Redimensionando Controles

Use *Shift*+Setas ou *Shit*+Ctrl+Setas

Alinhando Componentes

Menu *View*|*Alignment Palette*

Figura. Alignment Palette alinha os componentes no Form Designer

Opções do menu de contexto

Posição, ordem, alinhamento, etc.

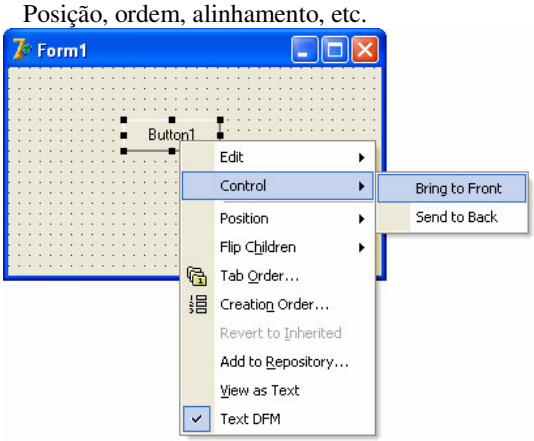


Figura. Opções do Menu de contexto do Form Designer

Lock

Use o menu *Edit|Lock Controls* para "travar" os controles do formulário.

Figura. Menu Lock Controls "trava" os controles no Form

Dica: Você também pode fazer um "lock" no editor de código, escolhendo a opção *Read-Only* no menu de contexto.

4. A linguagem Pascal (Delphi Language)

[g31] Comentário: Pascal não é Case-sensitive (falar execeção=Register)

[g32] Comentário: Falar do padrão de escrita Object Pascal

Comentários

```
(* Isso é um comentário *)
{Isso é um comentário}
// Isso é um comentário até o fim da linha
{$R *.RES} ← isso não é um comentário, é diretiva de compilação
```

Instruções

```
X:=10;
Saudacao:='Olá Mundo';
CalcularValorTotal;
CalcularValorTotal(10,20);
```

[g33] Comentário: Falar que Pascal é ***aspas simples***, e não dupla

Bloco condicional – If then else e Case

```
if X=10 then
  Instrução;
```

```
if X=20 then
begin
  Instrução;
  Instrução; ← o ponto e vírgula é opcional antes do end
```

```
end;
```

```
if X=10 then
  Instrução  ← não vai ponto e vírgula antes do else
else
  Instrução;
```

[g34] Comentário: Falar das duas exceções

```
case X of
  10 : Instrução;
  20 : Instrução;
  30 : Instrução; ← aqui pode
else
  Instrução;
end;
```

[g35] Comentário: Nem tudo pode ser colocado no case, só tipos enumerados. Mostrar code template para case

Atribuição vs. Igualdade

```
X:=10; ← significa "coloque o valor 10 na variável X"
```

```
if X=10 then … ← significa "se o valor de X for igual a 10 então..."
```

Estruturas de repetição – repeat until, while e for

```
X:=0;
repeat // repete até condição ser verdadeira
  X:=X+1;
until X>10;
```

```
X:=0;
while X<=10 then // repete até condição ser falsa
  X:=X+1;
```

[g36] Comentário: Falar que condição do while é geralmente o contrário da condição do repeat

```
for X:=1 to 10 do ← de 1 a 10
  Instrução;
```

```
for X:=10 downto 1 do ← de 10 a 1
  Instrução;
```

```
for X:=30 to 50 do
  Instrução;
```

Atenção: A variável usada no loop *for* deve ser declarada sempre **local**.

Break e Continue

- **Break** interrompe um laço *for*, *while* ou *repeat*;
- **Continue** pula para a próxima iteração do laço.

```
for X:=1 to 40 do
begin
  Instrução;
  if Condição then
    Break;
  Iinstrução;
end;
← break pula pra cá
```

Tipos de Dados Numéricos

- Inteiros: *Shortint, Smallint, Longint, Int64, Byte, Word, Longword*
- Real: *Real48, Single, Double, Extended, Comp, Currency*

[g37] Comentário: Quando usar cada um?

Char

Representa um *caracter*, como 'A', 'B', ou #13.

Strings

```
var
  ch : char;
```

```
  st : string; // isso é uma AnsiString
  ss1 : shortstring;
  ss2 : string[40];
begin
   ch:='A';
   st:='Olá Delphi!';
   ss1:='Olá Borland!';
   ss2:='Eu posso ter até 40 caracters';
   ch:=st[1];
end;
```

$H – define o comportamento do tipo **String**. Se ativado (+) uma string é uma *AnsiString*, caso contrário é uma *ShortString*.

[g38] **Comentário:** Fazer um comparativo com o pascal, falar do byte zero da string, falar + sobre AnsiString

Boolean

True ou *False*.

```
var
  b : boolean;
begin
  b:=(X>Y);
  if b then …
```

Variants

Usado também em Automação OLE e programação MIDAS/DataSnap. Pode receber qualquer tipo de dados, com algumas exceções.

Unit: **variants.pas** (Delphi6-7, Kylix)

[g39] **Comentário:** Exemplo com OLE (Word)

```
var
  msword : variant;
begin
 msword:=
  CreateOleObject
    ('Word.Basic');
  msword.AppShow;
  msword.FileNew;
  msword.Insert('OLA');
```

```
var
  x: variant;
begin
 x:=10;
 x:='Olá';
 x:=Button1; //erro
end;
```

Atenção: Variants não podem armazenar *records*, *sets*, *arrays* estáticos, files, classes, referência de classe e ponteiros.

[g40] **Comentário:** Falar da incompatibilidade com Delphi 5 e dar exemplo do SIE

OleVariants

Enquanto um *Variant* contém dados que só a aplicação corrente sabe o que fazer, *OleVariant* contém dados que podem ser compartilhados entre diferentes aplicações e através de uma rede. Ex.:

```
var
  pack : OleVariant;
begin
  pack:=CreatePack(Edit1.Text);
```

Arrays

Conjunto de valores (vetor, matriz).

```
var
  vetor : array [1..10] of integer;
begin
  vetor[1]:=10;
end;
```

Array dinâmico

Não tem tamanho definido, pode crescer em tempo execução (semelhante a uma lista encadeada).

```
var
  vetor : array of integer;
begin
  setlength(vetor,10);
  vetor[1]:=10;
end;
```

[g41] **Comentário:** Exemplo simples com tipos enumerados e conjuntos

5. Estrutura de Programa

Programas

```
program CadastroLocal;

uses
  Forms,
  Cadastro in 'Cadastro.pas' {FrmCadastro},
  DataMod in 'DataMod.pas' {DM: TDataModule},
  Cursos in 'Cursos.pas' {FrmCursos};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TDM, DM);
  Application.CreateForm(TFrmCadastro, FrmCadastro);
  Application.CreateForm(TFrmCursos, FrmCursos);
  Application.Run;
end.
```

[g42] Comentário: Falar novamente que o desenvolvedor não altera com freqüência esse arquivo

[g43] Comentário: Falar que essas são as units vistas no project manager

[g44] Comentário: Application? CreateForm? Initialize? Run? Explicar

Units

```
unit MinhaUnit;

interface

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

type
  TForm1 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    Label1: TLabel;
    procedure BitBtn1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  ShowMessage('Olá Delphi 7!');
end;

end.
```

[g45] Comentário: Ensinar como descobrir que unit deve ser declarada quando algo não for identificado

[g46] Comentário: TForm1?

[g47] Comentário: Falar que os componentes que são colocados no form são declarados aqui, além do DFM

[g48] Comentário: Form1 : TForm1?

Procedures

```
procedure AumentarSalario (var ASalario : single; const APerc : single);
begin
  ASalario:=ASalario * (1+APerc/100)
end;
```

[g49] Comentário: Const, var?

Funções

```
function Quadrado (ANumero : integer) : integer;
begin
  result:= ANumero * ANumero;
end;
```

[g50] Comentário: Result não precisa ser declarado?

6. Formulários e Frames

Propriedades

BorderStyle, BorderIcons, FormStyle, Caption, Icon, Position, WindowState,…

D6 e D7: *AlphaBlend, AlphaBlendValue, TransparentColor* (p/ Windows >= 2000)

[g51] **Comentário:** Mostrar exemplo

Mostrando

```
MeuForm.Show; ou
MeuForm.Visible:=true;
```

Mostrando Modal

Uma janela modal fica sobre as demais até que seja fechada.

```
MeuForm.ShowModal;
```

Atenção: Para exibir um formulário é necessário que ele esteja criado, caso contrário você receberá uma exceção do tipo *Access Violation*.

Fechando

[g52] **Comentário:** Falar da diferença de fechar e destruir

```
MeuForm.Close;
```

Ocultando

```
MeuForm.Visible:=false; ou
MeuForm.hide;
```

Criando

```
Application.CreateForm(TMeuForm,MeuForm); ou
MeuForm:=TMeuForm.create(Application);
```

Destruíndo

```
MeuForm.release; ou
MeuForm.free;
```

Trabalhando com múltiplos Formulários

Exemplo usando vários formulários.

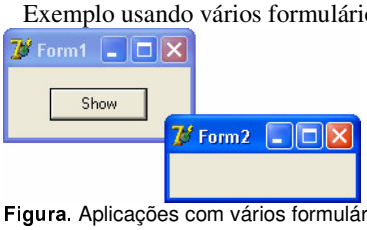


Figura. Aplicações com vários formulários

Formulários MDI

Formulário "container" -> *FormStyle = fsMDIForm*
Formulários "childs" (documentos) -> *FormStyle = fsMDIChild*

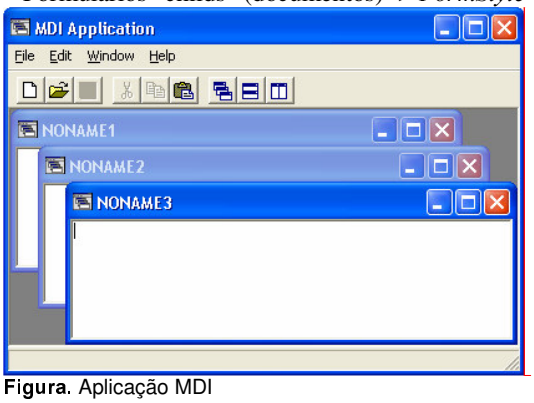


Figura. Aplicação MDI

[g53] **Comentário:** O uso de aplicações MDI tem sido desaconselhado pela Microsoft

Usando Frames

Crie um frame usando o menu *File|New|Frame*.

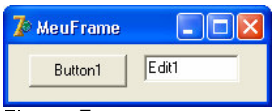


Figura. Frame

Adicione um *Frame* a um formulário usando o primeiro o botão da paleta *Standard* (veja a figura a seguir).

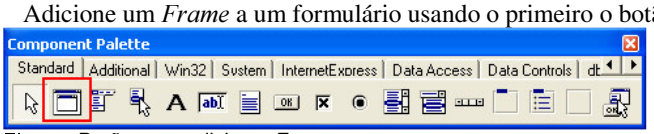


Figura. Botão para adicionar Frames

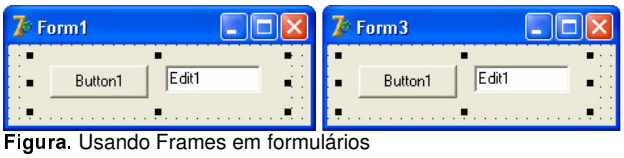


Figura. Usando Frames em formulários

7. Programando com Componentes - VCL e CLX

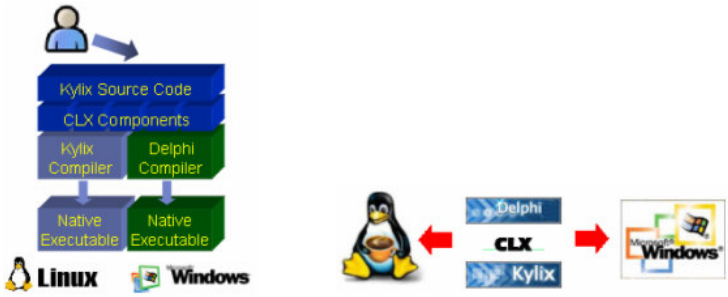


Figura. Código CLX é compatível com o Delphi e Kylix

VCL e CLX no Delphi 7

VCL → biblioteca de componentes do Delphi
CLX → Biblioteca de Componentes *Cross-Platform* do Delphi e Kylix

Component Palette (VCL)

A VCL é composta por componentes visuais e componentes não-visuais.

[g54] Comentário: Explicar que a Borland agora diz que CLX é base para VCL e VisualCLX

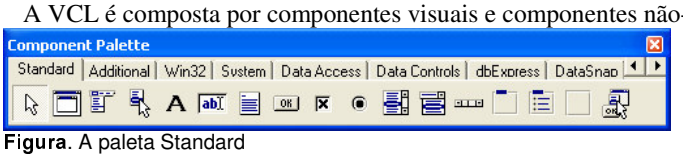


Figura. A paleta Standard

Figura. A paleta Additional

Figura. A paleta Win32

Configurando o Install Packages

Pacotes Instalados - menu *Tools|Install Packages*

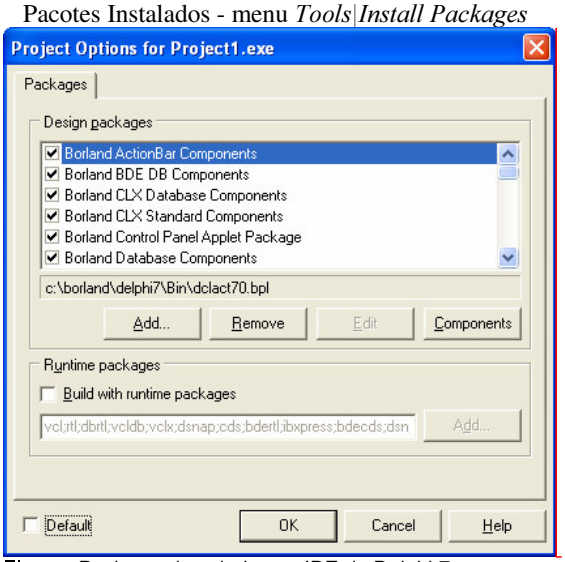


Figura. Packages instalados na IDE do Delphi 7

[g55] Comentário: Mostrar como adicionar / remover, por exemplo o BDE

[g56] Comentário: Falar que a remoção de um pode ocasionara remoção de outros. Falar que pacotes são vistos em detalhes no módulo 3

Conhecendo os componentes da VCL

Menus/PopupMenus – Usando o *Menu Designer*.

Figura. O Menu Designer do Delphi 7

[g57] Comentário: Mostrar os menus prontos (templates)

Standard - *Label, Edit, Button, CheckBox, RadioButton, RadioGroup, Memo, ListBox, ComboBox, GroupBox, Panel.*

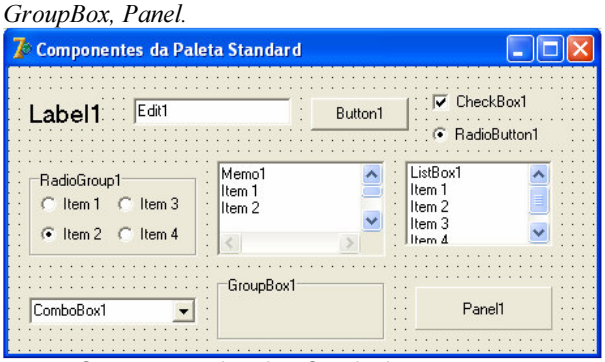


Figura. Componentes da paleta Standard

Additional - *BitBtn, SpeedButton, MaskEdit, Image, Shape, Bevel, ScrollBox, LabeledEdit, CheckListBox, ValueListEditor.*

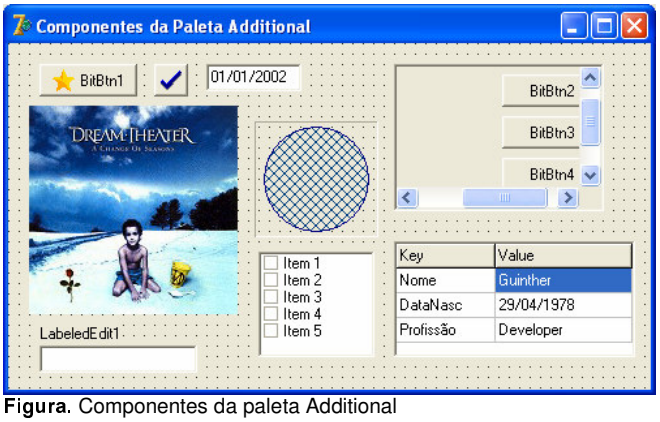


Figura. Componentes da paleta Additional

Win 32 - *ImageList, ToolBar, TabControl, PageControl, RichEdit, MonthCalendar, ProgressBar, Animate, TreeView, DateTimePicker, StatusBar*.

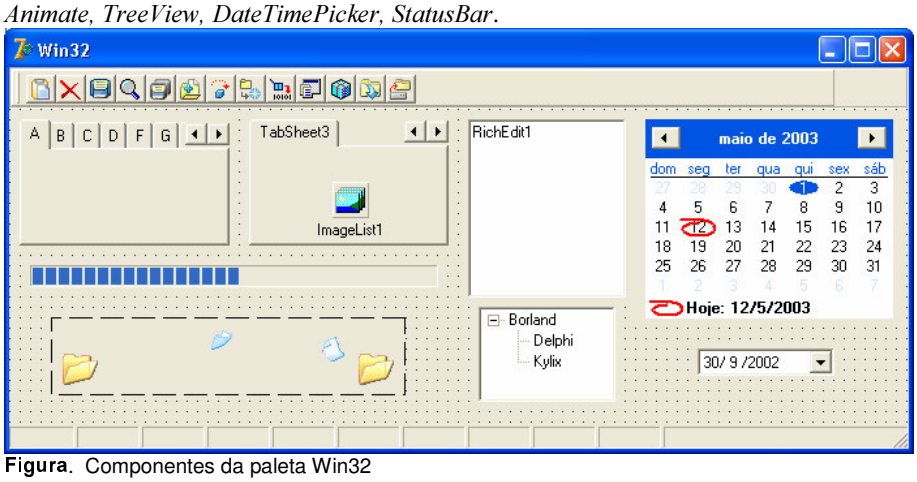


Figura. Componentes da paleta Win32

Usando Actions

Para entender o uso de *Actions* considere o seguinte exemplo: A IDE do Delphi permite a gravação de uma unit através do menu *File|Save*. A mesma opção está disponível em um botão na barra de ferramentas. Isso significa que existe uma *Action* chamada *Save* da categoria *File* e ambos os controles estão associados a essa mesma *Action*.

O componente *ActionList* representa um conjunto de *Actions*, e centraliza o tratamento de eventos em um formulário. O *Delphi* já fornece várias *Actions* pré-definidas.

Coloque o código de execução da *Action* no seu evento *OnExecute*.

Figura. Usando Actions

Quando você associa um controle a uma *Action* todas as propriedades da *Action* (como *Caption, Hint, ImageIndex*, etc.) são repassadas ao componente. Se você trocar o *Caption* da *Action* todos os controles associados a *Action* também terão seus *Captions* alterados.

[g58] Comentário: Lembrar que um controle que aponta para uma Action sem manipulador de evento fica desabilitada

Dica: Utilize o evento *OnUpdate* de uma *Action* para habilitar/desabilitar uma *Action*. Dessa forma, você não precisará ficar testando em vários pontos (eventos) do formulário quando um determinado botão ou item de menu deve estar habilitado. Por exemplo, poderíamos habilitar uma ação *Gravar* somente quando o *DataSet* estivesse em modo de inserção ou alteração, como mostra o código a seguir:

```
procedure TForm1.ActnGravarUpdate(Sender: TObject);
begin
 ActnGravar.Enabled:=DataSource1.State in [dsinsert,dsedit];
end;
```

Atenção: Use *OnUpdate* com cuidado, não coloque código "pesado" nesse evento, pois ele é controlado pela VCL e é disparado constantemente.

Dica: O componente *ActionManager* do Delphi 6/7 estende o *ActionList* e permite criar menus e barras de ferramentas estilo Windows / Office XP.

[g59] Comentário: Mostrar exemplo da revista

Exercícios – Propriedades e Eventos dos componentes da VCL

1. Colocar um *Edit* no formulário e fazer o seu texto aparecer na barra de título da janela quando o usuário digitar algo.
2. Faça o mesmo para uma *Label* (chame-a de *LblTeste*)
3. Depois coloque um *CheckBox* de modo que o usuário não possa digitar se esse *CheckBox* estiver desmarcado (ou seja, desabilitar o *Edit* quando estiver desmarcado e habilitar quando marcado)
4. Colocar um botão que feche o programa quando clicado. Porém ele será habilitado quando o usuário digitar a palavra "magic" no *Edit*. Depois faça o mesmo usando uma *Action*.
5. Colocar um botão e um *Memo* no formulário, e quando o usuário apertar este botão, o que está escrito no *Edit* é adicionado ao *Memo*.
6. Colocar outro botão no formulário, de modo que quando o usuário clique neste botão, os números ímpares de 1 a 100 são colocados no *Memo*.
7. Colocar um *SpinEdit* no formulário, de modo que a *LblTeste* receba como tamanho da fonte o valor deste *SpinEdit.*
8. Colocar um *ComboBox* no formulário. Coloque nome de animais nos *Items* desse *ComboBox*. Quando o usuário escolher um animal, *LblTeste* deve receber o texto escolhido.
10. Faça o mesmo só que agora usando um *RadioGroup*.
10. Faça um relógio.
11. Fazer um *TrackBar* ajustar a posição de um *ProgressBar* e de um *Gauge*.
12. Fazer um opção de *Load* e *Save* no *Memo,* usando dois botões, um *OpenDialog* e um *SaveDialog*.
13. Colocar dois *DateTimePicker* e um botão. Quando o usuário clicar no botão, aparecerá em *LblTeste* a diferença em dias entre as duas datas escolhidas.
14. Coloque um *TabControl* com as letras de A..Z na propriedade *Tabs*. No meio deste *TabControl* coloque um *Label*. Quando o usuário escolher uma letra na aba, a *Label* do centro deve receber esta letra como texto.

[g60] Comentário: TrackBar.Max=100 Frequency=3

[g61] Comentário: Usar DaysBetween de DateUtils

8. Introdução a Banco de Dados e ClientDataSet

Atenção: Neste capítulo de introdução a banco de dados e *ClientDataSet* não usaremos um servidor de dados SQL, de forma que salvaremos os dados localmente em disco. No entanto, todas as práticas que serão vistas aqui na manipulação do *ClientDataSet* podem ser utilizadas no desenvolvimento Client/Server com o *ClientDataSet* e dbExpress.

Engines de Acesso a Dados da Borland

No Delphi 7 temos quatro engines de acesso a servidores SQL e um engine de acesso local.
BDE – *Borland Database Engine* – conjunto de bibliotecas e drivers de acesso. Foi descontinuado.
dbGo – Acesso via ADO da *Microsoft* – *Active Data Objects* – OLEDB e ODBC
IBX – *Interbase Express* – Acesso ao Interbase
dbExpress – substitui o BDE – padrão recomendado pela Borland

MyBase – substitui o acesso local a *Paradox/DBase*

[g62] **Comentário:** Qual usar? Falar da história do BDE, brevemente. Paradox, etc.

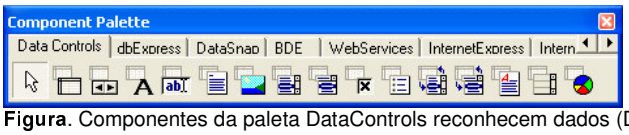
Nota: O dbExpress é estudado no módulo 2

DataSets e DataSource

[g63] **Comentário:** Falar que DataSource abstrai, mostrar exemplo com Table e ClientDataSet

TDataSet é um classe abstrata que serve de base para muitos outros componentes *DataSets*, como *TQuery, TTable, TSQLQuery, TSQLDataSet,* etc. Em um sistema, por exemplo, para a tabela de CLIENTES no banco de dados você terá um *DataSet* de CLIENTES no Delphi. Normalmente um *DataSet* também pode representar os dados de várias tabelas relacionadas. Você usará um *DataSource*, que ligará os controles consciente de dados ao *DataSet.* Esses controles conscientes de dados são os componentes que o usuário final utilizará para lançar as informações na tabela do banco de dados.

DataControls



[g64] **Comentário:** DataSource / DataField.

Figura. Componentes da paleta DataControls reconhecem dados (Data-Aware)

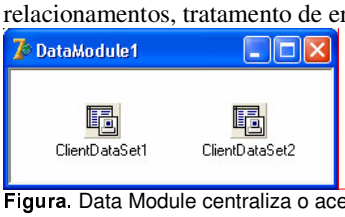
Utilizamos os componentes da guia *DataControls* para construir a interface de entrada de dados para uma Tabela (*DataSet*). Estes controles são também chamados *Data-Aware* (consciente de dados). Todos os componentes *Data-Aware* devem apontar para um *DataSource***,** e quase todos apontam para um *DataField.*

Estrutura Simples de como funciona o Acesso a Dados

Figura. Estrutura de relação entre componentes DataControls e DataSource

DataModules

Utilizamos um *DataModule* como container para componentes não-visuais. Geralmente esses componentes serão os objetos de Acesso a Dados (guia *Data Access*, *BDE, ADO, dbExpress,* etc). Utilize o *DataModule* para codificar métodos que não tratem de interface, como de regras de acesso a dados e validação, relacionamentos, tratamento de erros de dados, etc.



[g65] **Comentário:** Em um DataModule não se pode colocar controles visuais

Figura. Data Module centraliza o acesso ao banco de dados

Banco Local com ClientDataSet (MyBase)

- *MyBase* é simplesmente um formato de arquivo de dados. O *MyBase* não é um Banco de Dados do tipo SGBDR (Sistema Gerenciador de Banco de Dados Relacional), por isso não é capaz de processar solicitações SQL para servir dados;
- O *MyBase* não precisa ser baixado, instalado, configurado, etc. Utilizando o componente *ClientDataSet* já podemos fazer uso do *MyBase*;
- Usar *MyBase* não requer BDE, não requer a criação de um ALIAS ou o envio de qualquer biblioteca adicional. Aplicações que usam *MyBase* cabem em um disquete.
- Como não existe um servidor SQL não há como utilizar instruções SQL. Quem fazia isso para o *Paradox* era o BDE;
- O *MyBase* é atualmente a única opção Desktop para o Kylix, e no Delphi pode ser usado em substituição ao *Paradox* / *DBase*
- O *MyBase* existe desde o Delphi 3, e era chamado de *BriefCase Model* – Modelo Pasta de Arquivos.

- O componente *ClientDataSet* só estava disponível nas versões *Client/Server* e *Enterprise* do Delphi, até a versão 5. No Delphi 6 e 7 (e Kylix) o *ClientDataSet* se tornou o padrão para cache e acesso a dados, e está disponível na versão *Professional* do produto.
- Sistemas que usam os dados armazenados em arquivos locais também são conhecidos como Sistemas *Flat-File;*
- O *MyBase* é baseado no *ClientDataSet.* Assim, podemos utilizar inúmeros recursos que não existem na *Table* ou *Query* usando *Paradox*, como: cache em memória, índices em memória (evitando mensagens de índice corrompido), pesquisas rápidas, campos agregados, campos calculados internos, suporte a inúmeros tipos de dados, suporte a XML, e vários outros recursos;
- O *MyBase* é extremante rápido, leve, portável, de configuração zero;
- Os arquivos de dados do *MyBase* podem estar em formato XML ou binário. A extensão do formato binário padrão é CDS. Obviamente, arquivos binários são menores que XML. Use o formato binário caso não precise usar os recursos do XML;

Veja um fragmento de um arquivo de dados *MyBase* em formato *XML DataPacket.*

```
<?xml version="1.0" standalone="yes" ?>
  <DATAPACKET Version="2.0">
    <METADATA>
      <FIELDS>
        <FIELD attrname="CustNo" fieldtype="r8" />
        <FIELD attrname="Company" fieldtype="string" WIDTH="30" />
        <FIELD attrname="Addr1" fieldtype="string" WIDTH="30" />
        ...
      </FIELDS>
      <PARAMS DEFAULT_ORDER="1" PRIMARY_KEY="1" LCID="1033" />
    </METADATA>
    <ROWDATA>
      <ROW CustNo="1221" Company="Kauai Dive Shoppe" Addr1="4-976 Sugarloaf Hwy" />
      <ROW CustNo="1222" Company="SIG" Addr1="Alberto Pasqualini X" />
      ...
    </ROWDATA>
  </DATAPACKET>
```

[g66] Comentário: XML pode facilmente ser transportado pela Web

Quando usar ClientDataSet local

- Construção de protótipos;
- Aplicações Desktop, mono-usuário, quando há poucos recursos de hardware, quando o sistema possui poucas tabelas e não haverá um grande volume de dados;
- Versões de demonstração de sistemas cliente / servidor e multicamadas;
- Softwares que precisam rodar diretamente de um disquete ou CD, sem necessidade de instalação ou configuração;
- Quando o usuário precisa trabalhar com uma versão do software em sua casa, desconectado da base de dados principal, ou em um laptop. Os dados são então cadastrados off-line, e depois trazidos em um disquete / internet para a empresa e lançados no servidor SQL;
- Não use *MyBase* quando precisar trabalhar com solicitações SQL (cliente / servidor), ou um sistema com muitas tabelas, muitos relacionamentos e integridades, grande volume de transações e relatórios complexos. Use para isso *Interbase*, por exemplo.

Dica: O componente *ClientDataSet* pode ser usado localmente, no *dbExpress* em aplicações Cliente/Servidor, na Web e em aplicações distribuídas (*DataSnap*). O *ClientDataSet* está presente em aplicativos de 1 camada (Desktop), 2 camadas (Two-Tier) e 3 ou mais camadas (multicamadas) . Isto significa que você pode construir diferentes versões de seu software para tecnologias diferentes, usando sempre os mesmos códigos sobre o componente *ClientDataSet.* Você pode iniciar o desenvolvimento de seu projeto com *MyBase,* fazendo um protótipo, e conforme ele vai crescendo, migre facilmente para *dbExpress, WebSnap*, ou ainda *Midas/DataSnap.* A arquitetura *DataSnap* é vista em detalhes no módulo 4.

Arquitetura de um aplicativo usando ClientDataSet local

A arquitetura de uma aplicação *MyBase* é bastante simples. Você terá um formulário com controles consciente de dados, conectados a um *DataSource* que se liga a um *ClientDataSet*, que por fim busca dados de um arquivo XML ou binário.

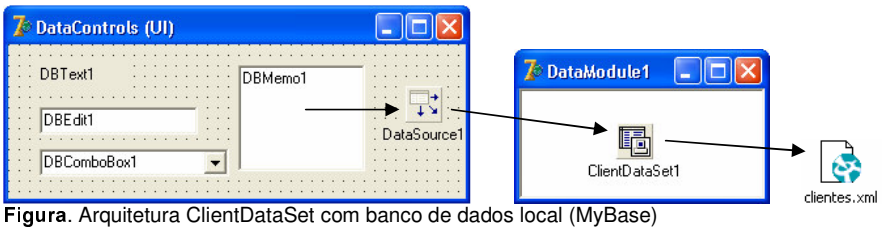


Figura. Arquitetura ClientDataSet com banco de dados local (MyBase)

Nota: Observe que uma arquitetura 2-tier com *dbExpress* é basicamente a mesma, de forma que o *ClientDataSet* se ligará a um *DataSetProvider* local. Em ambiente multicamadas, o *ClientDataSet* se ligar a um *DataSetProvider* remoto. *DataSetProviders* são vistos nos módulos 2 e 4.

Exercício – Cadastro simples

Crie uma nova aplicação Delphi. Clique em *File|Save All* e salve o formulário como "uFrmAlunos.pas", e o projeto como "CadastroCDS.dpr". Dê o nome de "FrmAlunos" ao formulário. Clique em *File|New|DataModule*. Salve o *DataModule* como "uDM" e configure sua propriedade *Name* para "DM".

Coloque no *DM* três *ClientDataSets* (da guia *Data Access*). Configure seus nomes para "cdsAlunos", "cdsCursos" e "cdsDependentes" respectivamente. Seu DM deve estar como mostra a figura a seguir.

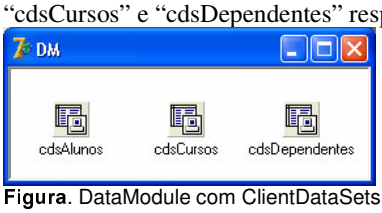


Figura. DataModule com ClientDataSets

Dê um duplo clique em *cdsAlunos*. Será mostrado o editor de campos (*Field Editor*). Dê um clique de direita no editor de campos e escolha *New Field*. Adicione os campos como mostrado na tabela a seguir:

Name	Type	Size
MATR_ALUNO	AutoInc	
NOME_ALUNO	String	40
SEXO	String	1
ENDERECO	String	60
CEP	String	8
CIDADE	String	60
UF	String	2
EST_CIVIL	String	20
MENSALIDADE	Currency	
DESCONTOS	Currency	
ATIVO	Boolean	
COD_CURSO	Integer	
OBS	Blob	
FOTO	Blob	

Tabela. Cadastro de Alunos

Atenção: Para simplificar esse exemplo levamos em conta que um aluno faz apenas um curso. Em uma situação real um aluno pode ter mais de um curso, surgindo então uma relação n para n e uma taerceira tabela (AlunoCurso). O mesmo vale para as mensalidades.

Selecione todos os campos *TField* e arraste-os para o formulário. Serão criados os controles *Data-Aware*, e um *DataSource*. Dê o nome de "dsAlunos" para o *DataSource*. Coloque também um *DBNavigator* e configure sua propriedade *DataSource*.

Feche o *Field Editor* e dê um clique de direita sobre o *ClientDataSet*, escolhendo a opção *Create DataSet.* Isto cria uma estrutura em memória, como um vetor.

Na propriedade *FileName* do *ClientDataSet* escreva "Alunos.cds". Caso não queira usar a propriedade *FileName*, você também pode usar os comandos de carga e gravação de dados em runtime, como mostrado a seguir:

```
ClientDataSet1.SaveToFile('Tabela.cds'); // salva para binário
ClientDataSet1.LoadFromFile('Tabela.cds'); // abre um arquivo binário
ClientDataSet1.SaveToFile('Tabela.xml',dfXML); // salva para formato XML
```

Você pode especificar o formato dos dados, como mostrado na última linha, onde foi usado XML. Os possíveis valores para este parâmetro estão na enumeração abaixo:

```
TDataPacketFormat = (dfBinary, dfXML, dfXMLUTF8);
```

Adicione a unit *MidasLib* a cláusula uses e execute a aplicação. Você pode agora cadastrar informações normalmente, e os dados serão gravados no arquivo binário. Rode a aplicação. Observe que trocamos alguns controles *DBEdit* por *DBComboBox* e *DBRadioGroup*. As configurações dos campos *TField* são vistas a seguir.

[g67] Comentário: Por que? Falar brevemente da MIDAS.DLL

Figura. Cadastro simples usando ClientDataSet com banco de dados local (MyBase)

Atenção: Quando usar cadastro local com o *ClientDataSet* (*MyBase*) desative o log de alterações - *ClientDataSet1.LogChanges:=false;* Não é necessário gravar logs quando se trabalha em aplicações locais

Você pode facilmente distribuir essa aplicação *MyBase* simplesmente copiando o executável (que cabe em um disquete), sendo que seu tamanho não passa de 1 MB. A tabela de dados não precisa ser distribuída porque é automaticamente criada assim que os dados são salvos pela primeira vez. Não é necessário BDE, nem criar ALIAS, nem distribuir DLLs, nem instalar cliente de banco. A configuração é zero.

Métodos do DataSet

[g68] Comentário: Quem é DataSet? É o ClientDataSet?

Normalmente você não vai querer usar o componente *DBNavigator* e sim construir sua própria barra de navegação. Por exemplo, para fazer um botão de inserção, coloque um *TButton* no formulário e no seu evento *OnClick* digite:

```
DM.cdsAlunos.Append;
```

[g69] Comentário: Depois fazer o mesmo usando DataSource.DataSet.Append;

Veja a seguir a lista dos principais métodos de manipulação de dados de um *TDataSet*:

Método	Função
Insert	Inserir
Append	Inserir no final
Delete	Excluir
Post	Gravar
First	Primeiro registro
Prior	Registro anterior
Next	Próximo registro
Last	Último Registro
Refresh	Atualizar dados
Cancel	Cancela alterações
Close	Fecha o DataSet
Open	Abre o DataSet
Edit	Altera (Edição)

Tabela. Métodos de TDataSet

TFields

[g70] Comentário: Falar que o desenvolvedor não pode capturar o valor de um campo pelo controle data-aware

Um *TField* é um objeto persistente (fica salvo no arquivo .dfm) que define o comportamento de um determinado campo de um *DataSet*. Podemos usar um *TField* para configurar máscaras de dados, regras de dados, valores requeridos, tamanhos, alinhamentos , etc.

As principais propriedades de um *TField* são:

Required: se campo é requerido ou não;

Visible: se o campo será visível em um *DBGrid;*

Read-Only: se o campo é somente-leitura;

EditMask : máscara de entrada de dados;

Ex.: CPF = 999999999.99 CEP = 99.999-999 DATA = !99/99/0000;1;_

Dica: A opção *Save Literal Characters* indica se os caracteres especiais devem ser salvos no banco de dados. A exceção é data e hora, onde essa opção deve estar sempre marcada.

DisplayFormat – como o campo deve aparecer nos controles consciente de dados;

Ex.: para 1 se tornar 000001 use "000000".

DisplayLabel – título de apresentação do campo (rótulo);

DefaultExpression – valor padrão;

EditFormat – formato de entrada de dados (edição). Ex. #,###,##0.00 -> valores numéricos;

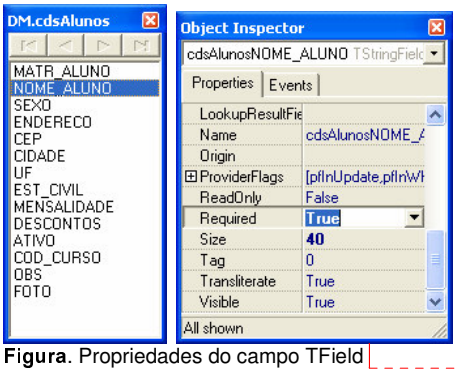


Figura. Propriedades do campo TField

[g71] Comentário: Fazer um exemplo do OnGetText e dizer que não está na apostila

Validando um Campo (OnValidate)

Para validar o valor de um campo utilize o evento *OnValidate* do objeto *TField.*

O código abaixo não permite que o desconto dado a um aluno seja maior que a mensalidade:

```
procedure TDM.cdsAlunosMENSALIDADEValidate(Sender: TField);
begin
  if cdsAlunosDESCONTOS.AsCurrency>cdsAlunosMENSALIDADE.AsCurrency then
   raise Exception.create('Desconto ñ pode ser maior que a mensalidade');
end;
```

Figura. Usando TField.OnValidate

Atenção: Você deve usar *raise* no *OnValidate* para cancelar a gravação do dado no buffer. *Raise* é visto em detalhes no módulo 3.

Índices

Um *ClientDataSet* usa índices em memória (diferente de uma *Table* / *Paradox*). Para criá-lo, basta fornecermos o nome do campo na propriedade *IndexFieldNames*, como por exemplo, o campo NOME_ALUNO. O código a seguir define um índice em tempo de execução:

```
ClientDataSet1.IndexFieldNames:='NOME_ALUNO';
```

Filtrando

[g72] **Comentário:** Filtro x SQL

Usamos filtros para selecionar um grupo de registros que obedeçam a uma determinada condição. Por exemplo, podemos mostrar somente os ALUNOS que estejam cursando a disciplina "BANCO DE DADOS", ou ainda, as CONTAS A PAGAR que vencem entre "01/03/2002" e "31/03/2002".

- O primeiro passo é definir a propriedade *Filtered* como *True;*
- Para filtrar um *DataSet*, utilizamos sua propriedade *Filter;*
- Uma outra opção é utilizar o evento *OnFilterRecord.*

Atenção: Utilize filtros na programação desktop, em desenvolvimento Cliente/Servidor use SQL.

Atenção: O filtro afeta sempre o ResultSet que está na memória. ResultSet é o conjunto de dados retornado de um servidor de banco de dados.

Filtrando Inteiros:

```
cdsAlunos.filter:='MATR_ALUNO>1000';
cdsAlunos.filter:=' MATR_ALUNO >=10 and MATR_ALUNO <=100';
```

Filtrando strings

```
cdsAlunos.filter:='NOME_ALUNO='+QuotedStr('GUINTHER');
cdsAlunos.filter:='NOME_ ALUNO'+QuotedStr('GUINTHER*'); ← Filtrar parte de um nome (use "*")
```

Filtrando Datas

```
cdsAlunos.filter:='DATA_NASC>'+QuotedStr('01/01/1980');
```

Procura com Locate

Locate -> localiza um valor em um campo

```
cdsAlunos.locate('CEP',['97.700.000'],[]); ← procura pela pessoa que tem este CEP
cdsAlunos.locate('NOME_ALUNO',['a'],[lopartialkey]); ← procura pela pessoa que começa com 'a'
(* procura pela pessoa que começa com 'a' ou 'A' *)
cdsAlunos.locate('NOME_ALUNO',['a'],[lopartialkey,locaseinsensitive]);
```

Dica: *Locate* pode usar procura aproximada ou não, segundo a opção *lopartialkey*; Se o campo de procurar do *Locate* for indexado, então *Locate* usará o índice; *Locate* pode pesquisar em mais de um campo.

Uma alternativa ao uso do *Locate* é *FindKey* e *FindNearest*.

Atenção: *Locate* usa o ResultSet que está na memória, ou seja, só localiza na cache. Se o resultset estiver filtrado, locate só procurar nos dados filtrados.

Dica: O método *Lookup* de *TDataSet* permite buscar um valor sem perder a posição do cursor.

Relacionamento LookUp

Utilize o relacionamento *Lookup* quando precisar buscar um campo de um segundo DataSet, baseada em um relacionamento entre campos.

Ex.: Nossa tabela ALUNOS possui um campo chamado COD_CURSO, que guarda o código do curso em que o aluno está matriculado. Para mostrar o nome do curso você pode criar um campo LOOKUP na tabela ALUNOS para buscar o campo NOME_CURSO, baseado numa ligação entre COD_CURSO da tabela alunos e COD_CURSO da tabela CURSOS.

No DM, dê um duplo clique em *cdsCursos* e crie a seguinte estrutura:

Name	Type	Size
COD_CURSO	AutoInc	
NOME_CURSO	String	40

Tabela. Cadastro de Cursos

Defina sua propriedade *FileName* como "Cursos.cds". Faça uma tela de cadastros para curso, colocando *DBEdits* e um *DataSource*. Dê o nome de "dsCursos" ao *DataSource*. Aperte ALT+F11 para usar a unit do *DataModule*.

Figura. Cadastro de Cursos

Configure o campo *Lookup* como mostrado a seguir.

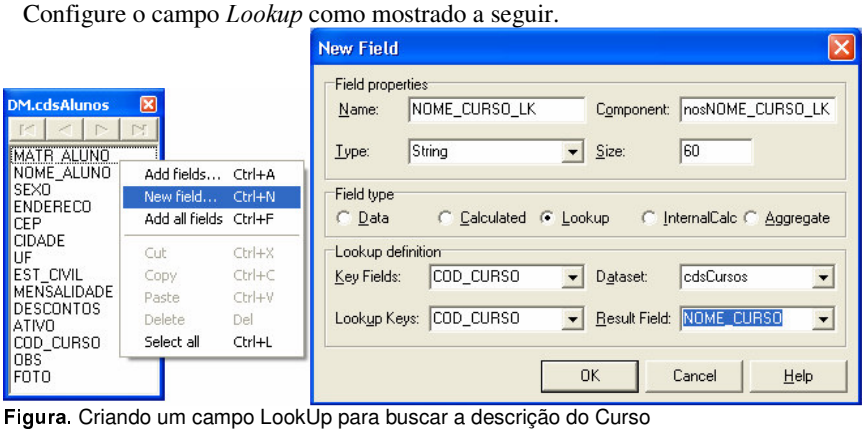


Figura. Criando um campo LookUp para buscar a descrição do Curso

KeyField: o campo-chave da tabela principal;
DataSet: o *dataset* que contém as descrições, no caso o campo que se quer buscar;

Lookup Keys: com qual campo do segundo *dataset* deve ser comparado o campo indicado em *KeyField;*
ResultField: campo que será mostrado na tela (campo de retorno);
Arraste o campo *TField* para o formulário. Será criado um *DBLookupComboBox;*

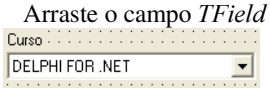


Figura. DBLookupComboBox

Coloque no formulário principal um botão que permita o acesso ao cadastro de cursos.

Relacionamento Master Detail

Utilize relacionamento *MasterDetail* quando for necessário exibir dados de dois *DataSets* relacionados (1-n). Por exemplo, um Aluno pode ter vários Dependentes. Assim, é necessária a criação de uma outra tabela (Dependentes), sendo que cada registro dessa tabela deve fazer referência ao registro principal (Aluno).

No DM, dê um duplo clique em *cdsDependentes* e crie a seguinte estrutura:

Name	Type	Size
MATR_ALUNO	Integer	
NOME_DEPENDENTE	String	40

Tabela de Dependentes

Defina a propriedade *FileName* desse *ClientDataSet* como "Dependentes.cds". No formulário de alunos coloque um *DataSource* e aponte-o para esse *ClientDataSet*. Coloque um *DBGrid* (paleta *DataControls*) na parte inferior do formulário de cadastro do aluno.

Vá até o DM. Aperte Alt+F11 e selecione a unit do formulário de alunos. Aponte a propriedade *MasterSource* de *cdsDependentes* para *FrmAlunos.dsAlunos* e na propriedade *MasterFields* configure o relacionamento como mostrado a seguir.

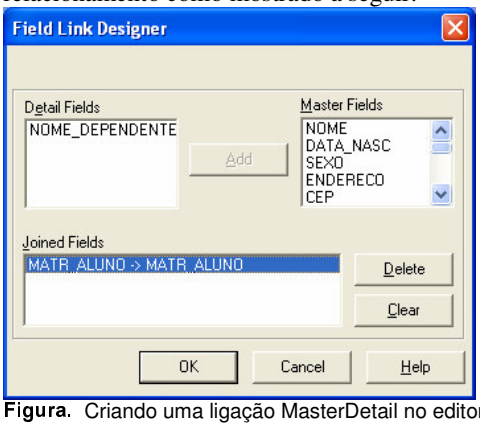


Figura. Criando uma ligação MasterDetail no editor da propriedade MasterFields

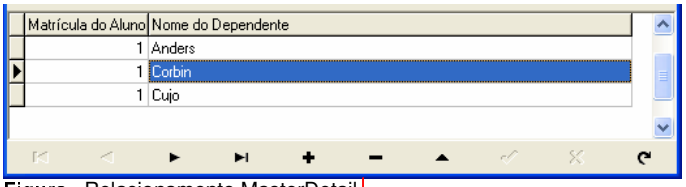


Figura. Relacionamento MasterDetail

[g73] Comentário: Comentário sobre DataSetFields que será visto no módulo 4

[g74] Comentário: Mostrar opções da Grid, colunas, títulos, etc.

Campos Calculados

Um campo calculado se comporta como um campo qualquer, porém seu valor não é armazenado no banco de dados. Seu valor geralmente é baseado em um cálculo sobre um ou mais campos do mesmo registro do *DataSet*.

Dê um clique de direita em *cdsAlunos* e escolha *New Field*. Preencha as opções como mostrado a seguir.

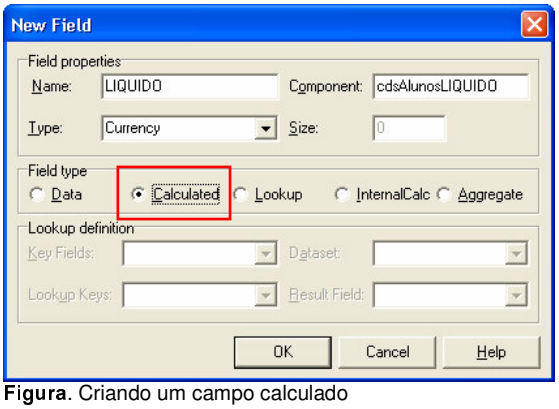


Figura. Criando um campo calculado

Utilize o evento *OnCalcFields* do *DataSet* para fornecer o valor do campo calculado. Em nosso exemplo, o valor total da mensalidade (líquido) é o valor da mensalidade menos o valor do desconto. Digite então no evento *OnCalcFields* de *cdsAlunos.*

```
procedure TDM.cdsAlunosCalcFields(DataSet: TDataSet);
begin
 cdsAlunosLIQUIDO.AsCurrency:=
   cdsAlunosMENSALIDADE.AsCurrency-cdsAlunosDESCONTOS.AsCurrency;
end;
```

[g75] Comentário: Quem é esse DataSet? Analogia com Sender

Figura. Campo Calculado

[g76] Comentário: Falar que campo calculado só pode ser alterado dentro desse evento

[g77] Comentário: Falar que R$, assim como configurações de Data e Hora, são obtidas a partir do Painel de Controle

Campos padrão (default)

Utilize o evento *OnNewRecord* para fornecer valores inicializados para um campo. Por exemplo, o valor do desconto da mensalidade do aluno pode sempre iniciar como valor zero:

```
procedure TDM.cdsAlunosNewRecord(DataSet: TDataSet);
begin
 cdsAlunosDESCONTOS.AsFloat:=0;
end;
```

Estados do DataSet e DataSource

Representa o estado atual de um conjunto de dados. Por exemplo, um *DataSet/DataSource* pode estar em *navegação, Inserção, Edição, ...* Os possíveis valores são:

```
  TDataSetState = (dsInactive, dsBrowse, dsEdit, dsInsert, dsSetKey, dsCalcFields, dsFilter,
dsNewValue, dsOldValue, dsCurValue, dsBlockRead,  dsInternalCalc, dsOpening);
```

O evento *OnStateChange* do *DataSource* indica quando um *DataSet* trocou de estado.

Como exemplo, coloque uma *StatusBar* no formulário, com dois *Panels*. No evento *OnStateChange* de *dsAlunos* digite:

```
procedure TFrmAlunos.dsAlunosStateChange(Sender: TObject);
begin
  with StatusBar1 do
    case DM.cdsAlunos.State of
      dsInsert : SimpleText:='Inserindo';
      dsEdit : SimpleText:='Editando';
      dsBrowse : SimpleText:='Navegando';
    end;
 {alternativa, usando RTTI}
 // declare TypInfo no uses
 { StatusBar1.Panels[0].Text:=Format('Estado do DataSet: %s',
          [GetEnumName(TypeInfo(TDataSetState),Integer(dsAlunos.state))]);}
end;
```

[g78] Comentário: Introduç ão à função Format

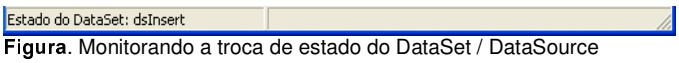


Figura. Monitorando a troca de estado do DataSet / DataSource

O código anterior usar RTTI para extrair o valor string de um tipo enumerado e mostrar na *StatusBar*.

[g79] **Comentário:** Breve introdução à RTTI

Campos Agregados

[g80] **Comentário:** Explicar o problema, de se fazer um SQL, ou uma varredura, etc.

Campos agregados são como campos calculados, só que podem atuar sobre uma coluna inteira de uma tabela, e não sobre um registro, como fazem campos calculados. Use o *FieldEditor* para criar campos agregados. Observe nas telas a seguir as configurações necessárias. Configure também a propriedade *AggregatesActive* do *ClientDataSet* para *True.*

Utilize campos agregados para fazer somas, médias, contadores, extrair o maior valor, menor, etc. Algumas funções são: SUM, COUNT, MAX, MIN, AVG.

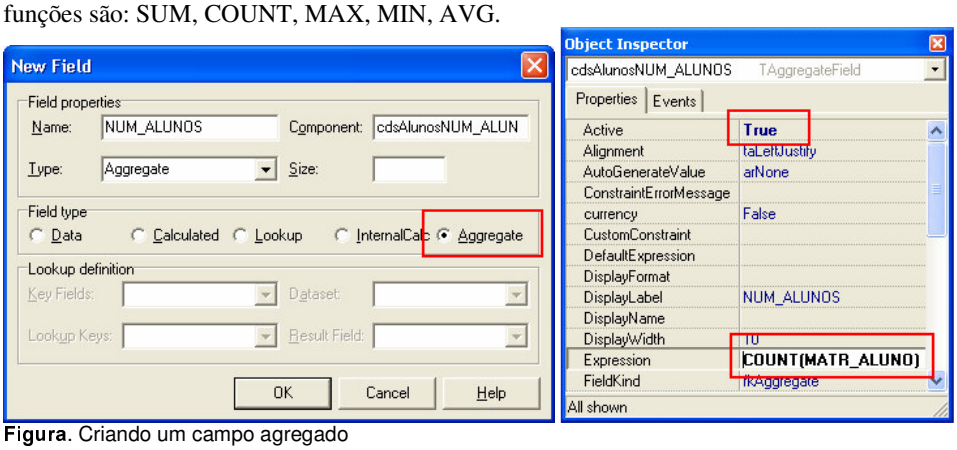


Figura. Criando um campo agregado

Coloque um *DBText* (paleta *DataControls*) no formulário e configure-o para exibir o valor do campo agregado.

[g81] **Comentário:** Falar que tem que digitar pois esse tipo de campo não aparece no FieldList

Antes de surgirem os campos agregados a solução mais comum era criar uma rotina auxiliar que fizesse uma varredura na tabela (*while not eof*) fazendo um operação (por exemplo soma) sobre os valores de uma coluna. Ou ainda usar uma consulta SQL auxiliar (por exemplo um *select sum*).

Constraints

Constraints são regras que são impostas sobre um determinado campo, e que são verificadas antes que os dados sejam gravados na memória (*buffer*). Por exemplo, poderíamos estabelecer uma regra para que somente pessoas de uma determinada cidade fossem aceitas no cadastro, ou somente pessoas que não sejam menores de idade. Você pode utilizar as propriedades *CustomConstraint* para criar a regra e *ContraintErrorMessage* para definir uma mensagem de erro se essa regra for violada.

Figura. Criando uma constraint para validar um campo TField

[g82] **Comentário:** Breve introdução sobre separar a regra de negócio da interface

Varreduras em DataSets

Muitas vezes precisamos fazer uma varredura em um *DataSet*, do primeiro ao último registro, alterando ou procurando um valor, ou limpando o conteúdo de um campo, etc. O exemplo abaixo mostra como varrer o *DataSet* aumentando a mensalidade de todos os alunos 10 %.

```
TDM = class(TDataModule)
 (...)
  public
    procedure AumentarMensalidadeAlunos (APercentual : single);
```

[g83] Comentário: Single ?

```
procedure TDM.AumentarMensalidadeAlunos(APercentual: single);
begin
  with cdsAlunos do
  begin
    First;
    while not eof  do
    begin
       Edit;
       if not FieldByName('MENSALIDADE').isNull then
         FieldByName('MENSALIDADE').AsCurrency:=
            FieldByName('MENSALIDADE').AsCurrency*(1+APercentual/100);
       Post; //opcional
       Next; //cuidado, se esquecer de chamar esse método causa loop infinito
    end;
  end;
end;
```

[g84] Comentário: Falar que post dispara o BeforePost

[g85] Comentário: Alertar sobre loop infinito

Dica: Observe que o procedimento anterior é uma regra de negócio, logo deve ficar no *DataModule*.

Atenção: Não há como fazer transação local com *ClientDataSet*. Nesse caso, se uma exceção fosse levantada durante o processamento, alguns alunos ficariam sem receber o reajuste. Veremos transações no 2º módulo.

Atenção: Em ambiente cliente/servidor você deverá usar SQL para realizar o processamento acima.

BookMarks e Disable/Enable Controls

Utilize *BookMarks* para "sinalizar" um determinado registro, navegar pelo *dataset*, e depois retornar ao registro marcado (semelhante a um marcador de livro). No exemplo anterior, após a varredura, o cursor fica posicionado no último registro, e não naquele que estava antes de iniciar a operação.

Em uma varredura, para que os controles *Data-Aware* não fiquem exibindo o conteúdo do registro durante a varredura (o que causaria uma perda de desempenho), usamos o método *DisableControls* de *TDataSet*.

```
procedure TDM.AumentarMensalidadeAlunos(APercentual: single);
var
  BM : TBookMark;
begin
  with cdsAlunos do
  begin
    DisableControls;
    BM:=GetBookmark;
    First;
    while not Eof  do
    begin
      Edit;
      if not FieldByName('MENSALIDADE').IsNull then
        FieldByName('MENSALIDADE').AsCurrency:=
          FieldByName('MENSALIDADE').AsCurrency*(1+APercentual/100);
      Post; //opcional
      Next; //cuidado, se esquecer causa loop infinito
    end;
    GotoBookmark(BM);
    FreeBookmark(BM);
    EnableControls;
  end;
end;
```

Atenção: Para simplificar o exemplo, na varredura anterior não usamos os blocos ***try finally*** para assegurar a liberação do *BookMark* e assegurar que *EnableControls* sempre seja chamado. Em uma situação real você deve sempre usar ***try finally.*** Veremos o uso dessa técnica no módulo 3.

Endereço para download dos exemplos
http://codecentral.borland.com/codecentral/ccweb.exe/author?authorid=222668