

# ClubeDelphi



Ano 7 · Edição 90 · R\$9,90

ISSN 1517990-7



**Agora com + páginas e com duas grandes novidades!**

- Artigos de PHP
- e muito mais artigos para iniciantes em Delphi

**Construa suas primeiras aplicações Web com**

# DELPHI & ASP.NET

**Na Web** >>>

Confira no portal ClubeDelphi duas vídeo-aulas sobre DataSnap e SOAP!

#### **RAD + POO?**

Saiba como implementar uma solução robusta baseada em MVC, POO e multicamadas em suas aplicações client/Server

#### **PHP**

Mini-curso de Delphi for PHP – Crie uma aplicação Web completa – Parte 3

#### **Mão na Massa**

Implemente leitura de código de barras em suas aplicações

#### **PHP**

Mini-Curso - Introdução à linguagem PHP – Parte 2

#### **Mobilidade**

Desenvolva aplicativos móveis com Delphi e Pocket PCs – Parte 2

#### **Mini-curso**

Construa um sistema completo de Contas a Pagar e Cobrança com dbExpress Firebird 2.0 – Parte 4

#### **Easy Delphi**

Data Modules em aplicações client/server

**Coluna Ask the Expert**

# Quantas cópias de seu software existem no mercado?

cópia

cópia

cópia

cópia

original

cópia

cópia

cópia

Esta é a chave mais segura do mundo  
contra pirataria de software.



Comprove você mesmo!



Alameda Tocantins, 280 - Barueri-SP

Tel: +55 11 4208-7700

<http://br.safenet-inc.com> - Contato: [infobrasil@safenet-inc.com](mailto:infobrasil@safenet-inc.com)

# ClubeDelphi

Ano 7 - 90ª Edição - 2007 - ISSN 1517990-7

Impresso no Brasil

## Corpo Editorial

### Editor Geral

Guinther Pauli  
guinther@devmedia.com.br

### Editor Técnico

Adriano Santos  
adrianosantos@devmedia.com.br

### Equipe Editorial

Adriano Santos, Bruno Lichot, Everson Volaco, Fabrício Desbessel, Gustavo Chaurais e Michael Benford

### Editor de Arte

Vinicius O. Andrade  
viniciusoandrade@gmail.com

### Capa

Antonio Xavier  
antonioxavier@devmedia.com.br

### Revisão

Luis Felipe de Oliveira  
luisfelipe@clubedelphi.net

Gregory Monteiro  
gregory@clubedelphi.net

### Distribuição

Fernando Chinaglia Dist. S/A  
Rua Teodoro da Silva, 907  
Grajaú - RJ - 206563-900

## Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

**Carmelita Mullin**  
www.devmedia.com.br/central/default.asp  
(21) 2215-0033

**Kaline Dolabella**  
Gerente de Marketing e Atendimento  
kalined@terra.com.br  
(21) 2215-0033

## Publicidade

Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:

**Kaline Dolabella**  
publicidade@devmedia.com.br

## EDITORIAL

Comecei a desenvolver com ASP.NET ainda quando o Delphi for .NET era apenas um compilador. E desde lá fiquei encantado pela riqueza de recursos que o framework Web oferece. Temos controles servidores, temos muitos recursos para performance (como cache), um mecanismo robusto para acesso a dados (ADO.NET), a facilidade do desenvolvimento drag & drop e baseado em eventos. Não tenho dúvidas que o ASP.NET é hoje a melhor solução para o desenvolvimento Web com Delphi. E digo mais, a nova versão do Delphi, a RAD Studio 2007, vem com suporte ao ASP.NET 2.0, e com ele uma vasta variedade de novos recursos antes só disponíveis para quem trabalhava com Visual Studio 2005. Temos Master Pages, DataSources, Web Parts, Temas e Skins, serviços de Membership e Login (tudo pronto), Profiles, Navegação, novos controles (mais de 50) e muito mais. Acredite, você vai desenvolver para Web com o mesmo poder RAD que tem hoje no Delphi Win32. Amigo leitor, se ainda não começou a desenvolver com essa poderosa e robusta tecnologia, essa edição é para você.

Ainda nesta edição, o Manoel traz a segunda parte do artigo que mostra como desenvolver aplicativos móveis para Pocket PC com o Delphi, criando agora a aplicação Pocket propriamente dita, que acessa o Web Service criado na edição anterior. O Adriano aproveita o seu mini-curso para mostrar um interessantíssimo recurso, como trabalhar com arquivos de remessa e retorno para realizar a cobrança junto a instituições bancárias, segundo as normas da FEBRABAN. Em sua outra colaboração, Adriano traz mais um artigo sobre automação comercial mostrando agora como fazer com que sua aplicação interaja com leitores de códigos de barra.

O uso do RAD e POO em uma aplicação é algo inversamente proporcional. Quanto mais RAD você desenvolve, menos faz uso da POO. Quanto mais POO você utiliza, mais perde os recursos do RAD. O Paulo Quicoli apresenta um poderoso padrão de projeto largamente utilizado em médios e grandes projetos, chamado Model-View-Controller (MVC), que permite que você separe em camadas as diferentes partes da sua aplicação. E o melhor de tudo, sem perder o poder RAD do Delphi. DataModules é o tema do artigo do Pablo e Fernando Prass, na coluna Easy Delphi. Aprenda a utilizar corretamente DataModules em suas aplicações client/Server.

E temos muito Delphi for PHP: o Fabrício continua o seu mini-curso que está mostrando como construir uma aplicação completa com Delphi for PHP, e temos a continuação do artigo introdutório sobre a linguagem (2ª parte) agora mostrando os tipos de dados utilizados no PHP.

Sucesso a todos, bons códigos, muito Delphi na cabeça e boa leitura!



A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS. Para mais informações sobre o pacote PLUS, acesse:  
<http://www.devmedia.com.br/clubedelphi/portal.asp>

## SEÇÃO DELPHI: Artigos intermediários sobre Delphi Win32 e Delphi .net.

### 06 - PocketPC com Delphi - Parte 2

Manoel Campos da Silva Filho

### 14 - Contas a Pagar e Cobrança - Parte 4

Adriano Santos

### 22 - Leitores de Código de Barras

Adriano Santos

### 26 - MVC (Model-View-Controller)

Paulo Roberto Quicoli

### 36 - Gráficos

Maikel Marcelo Scheid

## SEÇÃO EASY DELPHI: Artigos para iniciantes em Delphi

### 42 - Aplicações Cliente/Servidor

Pablo Tøndolo de Vargas e Fernando Sarturi Prass

### 48 - Treinamento em ASP.NET - Parte 1

Renato Haddad

## SEÇÃO PHP: Artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

### 52 - Delphi for PHP

Fabrício Desbessel

### 58 - Introdução ao PHP - Parte 2

Ewald Geschwinde e Hans-Juergen Schoenig



## Guinther Pauli

guinther@devmedia.com.br  
Microsoft Certified: MCP, MCAD, MCS.D.NET  
Borland Certified: Delphi 6, 7, 2005, 2006, Web, Kylix

## Fale com o Editor

É muito importante para a equipe saber entre em contato com os editores, informando o que você está achando da revista: que do o título e mini-resumo do tema que você tipo de artigo você gostaria de ler, que gostaria de publicar: artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!

**Guinther Pauli - Editor da Revista**  
guinther@devmedia.com.br

Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

**Luciano Pimenta - Editor do Site**  
lucianopimenta@devmedia.com.br

## Portal do Assinante

A ClubeDelphi tem uma novidade para você que comprou este exemplar na banca de jornal: você pode acessar GRATUITAMENTE, o Portal do Assinante ClubeDelphi!

### Confira o que você encontra no Portal do Assinante:

- Mais de 460 Vídeos Aulas!
- 6 cursos online!
- 1 Livro Eletrônico sobre ADO.NET e BDP!
- Mais de 140 Artigos Exclusivos!

Para Utilizar o Portal do Assinante, acesse [www.devmedia.com.br/clubedelphi/portal.asp](http://www.devmedia.com.br/clubedelphi/portal.asp) e utilize as informações abaixo: **LogIn: DVM.PL e Senha: CRAA7**

O acesso é válido por 30 dias a partir da data de lançamento da revista. Todos os meses a ClubeDelphi lhe dará uma senha válida para acessar o portal. Comprando a revista regularmente em bancas, você terá acesso ininterrupto a ele!

NÃO PERCA



# Ask The Expert

## Perguntas e Respostas

Dúvidas respondidas por **Luciano Pimenta**  
(envie as suas para [lucianopimenta@devmedia.com.br](mailto:lucianopimenta@devmedia.com.br))

### Como detectar internet na máquina

Preciso saber qual via programação se tem internet disponível no micro do meu cliente. Como posso fazer isso? Existe alguma API do Windows?

*Anderson Marques*

Olá Anderson, respondendo a sua questão: sim, existe uma API do Windows que retorna se existe internet disponível no micro e é muito simples de implementar. Basta fazer uma chamada a função *InternetGetConnectedState*. Veja a na **Listagem 1**.

Veja na **Figura 1** como ficou o exemplo ao detectar a internet.

**Listagem 1.** Chamada a função *InternetGetConnectedState*

```
uses
  WinInet
...
procedure TForm1.Button1Click(Sender: TObject);
var
  Flags : dword;
begin
  if InternetGetConnectedState(@Flags, 0) then
    Label1.Caption := '[Internet Disponível]';
  else
    Label1.Caption := '[Internet indisponível]';
end;
```



**Figura 1.** Detectando internet



“ O Maior Portal para Desenvolvedores da América Latina.”



Acesse:

[www.devmedia.com.br](http://www.devmedia.com.br)



Assinatura

**ClubeDelphi PLUS**

Mais conteúdo .NET por menos!

**+460 vídeo aulas | 6 cursos online**

[www.clubedelphi.net/portal](http://www.clubedelphi.net/portal)

Caro Leitor,

O portal ClubeDelphi PLUS é a continuação, na Web, da revista ClubeDelphi. O portal recebe um conteúdo novo todo dia e hoje conta com: i) mais de 460 vídeo aulas; ii) 6 cursos online; iii) 1 livro eletrônico gratuito, de Guinther Pauli, sobre ADO.NET e BDP; iv) mais de 150 artigos exclusivos (que não foram publicados na revista!);

Acesse o portal ClubeDelphi PLUS e receba muito mais conteúdo sobre Delphi! E o que é melhor: de graça! Todo leitor da revista ClubeDelphi, seja ele assinante ou comprador da revista em bancas, tem acesso ao portal (para quem compra em bancas, o acesso é válido por 30 dias).

Se você é assinante, utilize o seu login e senha pessoais para acessar o portal. Se você comprou em bancas, utilize o login e senha publicados na página do editorial desta edição.

Confira a seguir as últimas novidades do portal!

Boa leitura e sucesso!  
Equipe DevMedia

**:: DESTAQUE ::**

**Confira no portal ClubeDelphi PLUS 2 vídeo aulas sobre DataSnap e SOAP**

**DataSnap e SOAP**

<http://www.devmedia.com.br/articles/viewcomp.asp?comp=6771>  
<http://www.devmedia.com.br/articles/viewcomp.asp?comp=6733>

**Curso Online - PLUS**

**O portal ClubeDelphi PLUS conta com 6 cursos online, confira!**

Sistema completo com Delphi 7, dbExpress e Firebird 2.0 [Luciano Pimenta]  
Aplicações WEB com IW e Delphi 7 (Delphi Win32)! [Guinther Pauli]  
Aplicações client/server no Delphi 2006 [Luciano Pimenta]  
Criando uma Aplicação multi-camadas Completa com Delphi [Guinther Pauli]  
Criando uma aplicação completa: sistema SysCash [Everson Volaco]  
Aplicações Client/Server com dbExpress e Firebird [Luciano Pimenta]

## Últimas Vídeo-Aulas (Somente para assinantes)

### **Aprenda a desenvolver sistemas para o sistema operacional PalmOS**

Acompanhe as aulas de Ricardo Boaro que falam unicamente do desenvolvimento de aplicações para PalmOS utilizando a IDE PocketStudio que é bastante semelhante ao Delphi inclusive utilizando-se de linguagem Pascal.

### **Migrando de 2 para 3 camadas – Parte I e II**

Veja nesta série de duas aulas como construir aplicações 2-camadas que podem ser facilmente migradas para multicamadas com DataSnap, por Guinther Pauli.

### **Imprimindo etiquetas no QuickReport - Parte I a III**

Veja nessa vídeo aula de Paulo Quicoli como criar etiquetas personalizadas usando o editor de relatórios nativo do Delphi, QuickReport.

### **Crie um loja virtual com Delphi for PHP – Parte VI**

Veja nessa vídeo aula de Fabrício Desbessel, a continuação de mais um curso on-line do Portal ClubeDelphi. Aprenda a criar uma loja virtual com Delphi for PHP.

### **Rave Reports – Parte I e II**

Veja nessa vídeo aula de Jefferson Junglaus, como trabalhar com Rave Reports no Delphi.

## Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET

# PocketPC com Delphi

## Desenvolvendo aplicativos móveis – Parte 2



### Manoel Campos da Silva Filho

([manoelcampos@gmail.com](mailto:manoelcampos@gmail.com))

é Analista de Sistemas, Tecnólogo em Processamento de Dados pela Universidade do Tocantins, especializando em Gestão e Consultoria em Telecomunicações, Professor da Escola Técnica Federal de Palmas-TO. Trabalha com desenvolvimento em Delphi, PHP e ASP.NET, programação para Linux em C. Desenvolve em Delphi desde a versão 3.0

Continuando o artigo da edição anterior, veremos agora como criar a aplicação cliente no Pocket PC para consumir o Web Service criado na última edição.

### Criando a aplicação

O CF Builder inclui algumas opções no *Repository* do BDS para criação de aplicações para dispositivos móveis. Acesse o menu *File>New>Other*, na opção *Smart Device* escolha *Smart Device Application* para criar um novo projeto (Figura 1). Salve a unit principal com o nome "MainForm.pas" e o projeto como "PesquisaOpinioao.dpr".

Para testarmos a aplicação no emulador acesse o menu *Compact Framework>Run Smart Device Project* no BDS, ou utilize o *Play* existente na barra de ferramentas adicionado pelo CF Builder (Figura 2).

Na barra você pode selecionar o emulador que deseja rodar a aplicação. O executável é copiado automaticamente para a

pasta compartilhada que você informou no campo *Shared Folder* nas configurações do emulador.

Assim, após rodar o emulador clicando no *Play* da barra do CF Builder, você deve acessar o *Storage Card*, como mostrado anteriormente, para executar a aplicação.

Lembre-se que se você não fizer um *Soft Reset* no emulador o *Storage Card* não aparecerá. Após o emulador abrir pela primeira vez, você não precisa fechá-lo, assim, o *Storage Card* não desaparece.

Para atualizar a aplicação no emulador, basta clicar no *Play* para que o executável seja copiado novamente para o *Storage Card*.

### Acessando o Web Service na aplicação desktop

Para acessar o Web Service a partir da aplicação desktop, precisamos adicionar uma referência à URL do arquivo WSDL que descreve as funções disponíveis no Web Service.

Como vamos trabalhar com dois projetos ao mesmo tempo, e você já está com o projeto do Web Service aberto, para abrir o projeto da aplicação desktop clique com o botão direito em *Project Group 1*, no *Project Manager*, selecione *Add Existing Project* e localize o projeto *PesquisaOpiniaio*.

Clique em *Save All* na barra de ferramentas para salvar o grupo de projetos. Salve com o nome de "Projetos Compact Framework". Agora temos os dois projetos no *Project Manager*.

Dê um duplo clique no *PesquisaOpiniaio*, a aplicação desktop para Compact Framework. Clique com o botão direito e escolha a opção *Add Web Reference* para adicionar uma referência ao Web Service criado.

Na janela que é aberta você deve informar a URL do Web Service, que se você utilizou os nomes que sugeri, deve ser "http://localhost/PesquisaOpiniaioWS/PesquisaOpiniaio.asmx". Porém, como a aplicação cliente não estará rodando

no seu PC e sim no dispositivo móvel, por meio do emulador, você não deve informar *localhost* e sim o endereço IP do seu PC (onde está o servidor web com o WebService hospedado) ou o nome DNS dele.

No meu caso, utilizei o endereço "http://developer/PesquisaOpiniaioWS/PesquisaOpiniaio.asmx". Rode o projeto do Web Service que o navegador é aberto indicando a URL, porém, substitua *localhost* pelo IP ou nome DNS do seu PC.

Após informar a URL, pressione ENTER para abrir a página no navegador interno. Clique em *Service Description* para abrir o WSDL (o descritor do Web Service). O valor do campo *Web reference folder name* será usado como prefixo do namespace que será criado para acesso ao Web Service.

Altere o valor do campo para "PesquisaOpiniaioServer". Depois clique em *Add Reference* para adicionar a referência ao projeto. O BDS cria um arquivo PAS

mapeando as funções existentes no Web Service para serem usadas em nossa aplicação.

Agora inclua um *MainMenu* no formulário principal da aplicação desktop. Nele inclua um item "Arquivo" e dentro desse um item "Selecionar dados no Servidor". Para usarmos as funções do Web Service precisamos incluir a unit que foi criada após a referência ao Web Service ser adicionada ao projeto.

Assim, pressione ATL+F11 e selecione a unit *PesquisaOpiniaioServer.PesquisaOpiniaio*. Inclua outros componentes no formulário para que esse fique semelhante à **Figura 3**.

---

**Nota:** Caso tente testar a aplicação e receba uma mensagem do tipo: *File not found: 'MainForm.frmMain.resources'*, você deve fazer uma cópia do arquivo RESX colocando o mesmo nome da mensagem, sendo esse erro problema no BDS ou *plugin*.

---



---

**Nota:** Não crie um formulário muito grande, pois dependendo do equipamento, o mesmo pode ficar desconfigurado.

---

Para armazenar os dados na aplicação, utilizaremos o *DataSet*. Como as funções do Web Service, que retornam dados para a aplicação, devolvem *DataSets*,

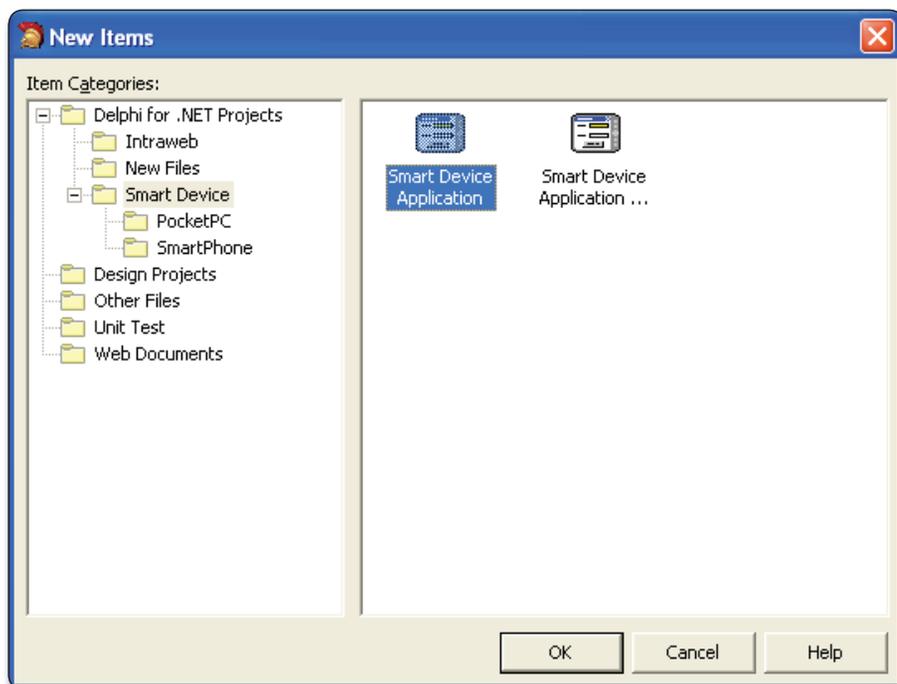


Figura 1. Opções Adicionadas pelo CF Builder no Repository

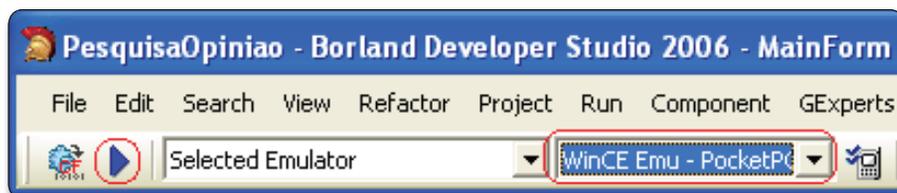


Figura 2. Barra de Ferramentas do CF Builder no BDS

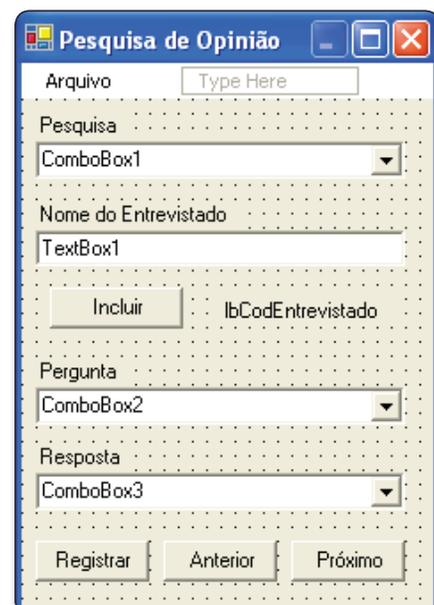


Figura 3. Design do Formulário

utilizaremos um *DataSet* para cada tabela que precisarmos armazenar dados localmente.

Assim, inclua cinco *DataSets* e altere seus nomes, respectivamente, para “ds-Pesquisa”, “dsPergunta”, “dsAlternativa”, “dsEntrevistado” e “dsResposta”. Na **Tabela 1** temos as propriedades que devem ser alteradas para alguns componentes.

Como precisaremos trabalhar com arquivos XML, criaremos uma unit que terá uma classe com funções estáticas (*class function*) que serão acessadas sem a necessidade de instanciar um objeto da classe.

Classes assim servem apenas para agrupar funções e constantes, organizando melhor o código. Crie a unit com o nome

de “ClubeDelphi.PesquisaOpinio.XML.pas” e adicione o código da **Listagem 1**.

**Nota:** O código possui comentários sobre as características de cada função.

A função *XMLCache.ApplicationDir* retorna o caminho onde a aplicação está. Foi usada uma variável de classe *FAppDir* para armazenar, quando a função for executada pela primeira vez, o caminho do executável, para não ter que ficar executando este código repetidamente. Veja o código na **Listagem 2**.

A função *XMLCache.CarregarXML* (**Listagem 3**) verifica se o arquivo informado no parâmetro *NomeArq* existe para poder carregá-lo no *DataSet* passado por referência. Como o arquivo XML estará na pasta da aplicação, e estaremos usando o .NET Compact Framework, necessita que informe o caminho completo do arquivo que se deseja manipular.

Na função *XMLCache.IncluirEntrevistado* (**Listagem 4**), assim como nas outras funções de inclusão, como o registro dos entrevistados é feito localmente, o valor da chave primária é gerado localmente também, de acordo com o valor do último registro. Se o *DataSet* está vazio, o código para o registro a ser adicionado no *DataSet* será 1.

A função *XMLCache.EnviarCacheXML* (**Listagem 5**) envia os dados dos entrevistados, armazenados localmente no dispositivo em arquivo XML, e suas respostas ao Servidor, repassando os dados desses arquivos para as funções do Web Service.

Abra o formulário principal e pressione ALT+F11 para usar a unit recém criada. No evento *Click* do último item de menu criado, vamos solicitar ao usuário que digite o título da pesquisa (ou parte dele) para que seja feito o acesso ao Web Service e retornados os dados da pesquisa.

Como a aplicação estará rodando em um dispositivo móvel, nem sempre teremos conectividade para acessar o Web Service, pois, como é um sistema de pesquisa, o usuário poderá estar em qualquer parte da cidade, sem acesso à internet para fazer a comunicação com o servidor.

Assim, precisaremos carregar os dados necessários para o dispositivo e guardar

**Listagem 1.** Funções para manipular os arquivos XML da aplicação

```
unit ClubeDelphi.PesquisaOpinio.XML;
interface
uses System.Data, System.Reflection, System.IO, System.Windows.Forms;

type
{ Classe para agrupar as funções para manipulação dos DataSets }
XMLCache = class
private
{ Variável de classe: possui um único valor para todas as instâncias da classe. Ela
é acessada sem a necessidade de instanciar um objeto da classe. Armazenará o caminho
da aplicação }
class var FAppDir: string;
public
{ Constantes que definem o nome dos XML usados }
const arqPesquisa = 'pesquisa.xml';
const arqPergunta = 'pergunta.xml';
const arqAlternativa = 'alternativa.xml';
const arqEntrevistado = 'entrevistado.xml';
const arqResposta = 'resposta.xml';
{ Descobre a pasta do executável }
class function ApplicationDir: string;
{ Inclui um registro de entrevistado no DataSet
passado por parâmetro }
class function IncluirEntrevistado(
var ds: DataSet; Nome: string;
CodPesquisa: Integer): Integer;
{ Registra, no DataSet passado por parâmetro, uma resposta de um entrevistado }
class function RegistrarResposta(
var ds: DataSet; CodEntrevistado, CodPergunta, CodAlternativa: Integer): Boolean;
{ Carrega um XML, verificando se o mesmo existe, no DataSet passado por parâmetro }
class function CarregarXML(var ds: DataSet; NomeArq: string): Boolean;
{ Grava os dados de um DataSet passado por parâmetro, em um arquivo XML }
class procedure GravarXML(ds: DataSet; NomeArq: string);
{ Envia os dados dos entrevistados e suas respostas ao servidor }
class procedure EnviarCacheXML(var dsEntrevistado, dsResposta: DataSet);
end;
```

Componente	Propriedade	Valor	Descrição
ComboBox	Name	cbxPesquisa	
	ValueMember	cod_pesquisa	Contém os dados a serem usados como valores para cada um dos itens do componente. A propriedade <i>DataSource</i> será atribuída via programação.
	DisplayMember	titulo	Contém os dados que serão exibidos no componente.
	DropDownStyle	DropDownList	Não permite que o usuário digite qualquer coisa no componente, só podendo escolher um dos itens existentes nele.
TextBox	Name	txtEntrevistado	
Label	Name	lbCodEntrevistado	
ComboBox	Name	cbxPergunta	
	ValueMember	cod_pergunta	
	DisplayMember	titulo	
	DropDownStyle	DropDownList	
ComboBox	Name	cbxResposta	
	ValueMember	cod_alternativa	
	DisplayMember	descricao	
	DropDownStyle	DropDownList	

**Tabela 1.** Configuração dos componentes do formulário

localmente em arquivos XML, antes do usuário ir à rua para fazer seu trabalho de pesquisa. Quando o usuário estiver na rua, os dados das entrevistas que fizer devem ser armazenados localmente em arquivos XML, para quando ele chegar a um local que tenha acesso à internet ou à rede local da empresa, transferir os dados ao servidor. Para isso que servirão os *DataSets* adicionados.

Para solicitar ao usuário o título da pesquisa, quando clicar no menu, poderíamos utilizar uma função como *InputBox* do Delphi Win32 (que abre uma janela solicitando um dado do usuário), porém, não temos uma função semelhante em Delphi for .NET.

Dessa forma, criaremos uma classe que implementará um *InputBox*. Crie uma nova unit e salve com o nome de "ClubeDelphi.InputBox.pas". O código dessa unit é mostrado no **Listagem 6**. Para usar o *InputBox* chame a função de classe *InputBox.Show* passando os devidos parâmetros (a função é chamada a partir da classe e não de uma instância).

A função do *InputBox.Show* é chamar o *InputBox* sem a necessidade de criar um objeto da classe. Para chamá-lo basta fazer *InputBox.Show* passando os parâmetros, sem criar objeto algum, com o código da **Listagem 7**.

Precisamos agora incluir o código para buscar os dados das perguntas e alternativas no servidor. No formulário principal, pressione ALT+F11 para usar a unit recém criada. No evento *Click* do item de menu adicione o código da **Listagem 8**.

Esse código chama as funções existentes no Web Service para retornar os dados que desejamos e em seguida estes dados são armazenados localmente em arquivos XML para a aplicação poder trabalhar *off-line*.

Cadastre algumas pesquisas, compile e execute a aplicação (usando os botões da barra de ferramentas do CF Builder) e clique no menu para ver o resultado. O emulador padrão é aberto.

Acesse o *Storage Card* para executar o programa. Lembre-se de usar o *Soft Reset* caso o emulador não estivesse aberto anteriormente.

## Utilizando o Device Emulator

A aplicação, acessando o Web Service

### Listagem 2. Retorna o caminho onde a aplicação se encontra

```
class function XMLCache.ApplicationDir: string;
var
  caminho: string;
begin
  if FAppDir = '' then
  begin
    { Pega o caminho completo do Assembly em execução (a aplicação) }
    caminho := Assembly.GetExecutingAssembly.GetName().CodeBase;
    { Extrai somente a pasta onde da aplicação, incluindo uma barra no final }
    Result := System.String.Format('{0}\', Path.GetDirectoryName(caminho), '\');
    { Se a função for chamada a partir de uma aplicação desktop que não seja para Compact
      Framework, o texto file:\ aparecerá no início da string. O if a seguir remove este texto }
    if Result.StartsWith('file:\') then
      Result := Result.Substring(6, result.Length-6);
    FAppDir := Result;
  end
  else Result := FAppDir;
end;
```

### Listagem 3. Carrega um arquivo XML passado como parâmetro

```
class function XMLCache.CarregarXML(var ds: DataSet; NomeArq: string): Boolean;
begin
  Result := &File.Exists(ApplicationDir + NomeArq);
  { Se o arquivo existe, carrega-o no DataSet passado por parâmetro }
  if result then
    ds.ReadXml(ApplicationDir + NomeArq);
end;
class procedure XMLCache.GravarXML(ds: DataSet; NomeArq: string);
begin
  { Grava os dados do DataSet no XML, na pasta da aplicação, com o nome passado por parâmetro }
  ds.WriteXml(ApplicationDir + NomeArq);
end;
```

### Listagem 4. Incluir um entrevistado

```
class function XMLCache.IncluirEntrevistado(
  var ds: DataSet; Nome: string; CodPesquisa: Integer): Integer;
var
  row: DataRow;
  total, cod: Integer;
begin
  total := ds.Tables[0].Rows.Count;
  if total = 0 then
    cod := 1
  else
    begin
      row := ds.Tables[0].Rows[total-1];
      { Gera um código local que será substituído no servidor quando os dados forem enviados }
      cod := Convert.ToInt32(row['cod_entrevistado']+1);
    end;
  { Cria uma nova linha (um novo registro) }
  row := ds.Tables[0].NewRow;
  row['cod_entrevistado'] := cod.ToString;
  row['nome'] := Nome;
  row['cod_pesquisa'] := CodPesquisa.ToString;
  { Adiciona a linha no DataSet }
  ds.Tables[0].Rows.Add(row);
  Result := cod;
end;
class function XMLCache.RegistrarResposta(
  var ds: DataSet; CodEntrevistado, CodPergunta, CodAlternativa: Integer): Boolean;
var
  row: DataRow;
  total, cod: Integer;
begin
  total := ds.Tables[0].Rows.Count;
  if total = 0 then
    cod := 1
  else
    begin
      row := ds.Tables[0].Rows[total-1];
      { Gera um código local que será substituído no servidor }
      cod := Convert.ToInt32(row['cod_resposta_entrevistado']+1);
    end;
  row := ds.Tables[0].NewRow;
  row['cod_resposta_entrevistado'] := cod.ToString;
  row['cod_entrevistado'] := CodEntrevistado.ToString;
  row['cod_pergunta'] := CodPergunta.ToString;
  row['cod_alternativa'] := CodAlternativa.ToString;
  ds.Tables[0].Rows.Add(row);
  Result := True;
end;
```



### Listagem 5. Salva os dados em Cache (arquivo XML)

```
uses PesquisaOpiniaoServer.PesquisaOpiniao;
...
class procedure XMLCache.EnviaCacheXML(
  var dsEntrevistado, dsResposta: DataSet);
var
  row: DataRow;
  row2: DataRowView;
  ws: TPesquisaOpiniaoWS;
  i, j, CodEntrevistado: Integer;
begin
  { Instancia a classe do Webservice para acessar suas funções }
  ws := TPesquisaOpiniaoWS.Create;
  try
    { Percorre as linhas do DataSet de entrevistado para enviar
      essas linhas ao servidor para serem incluídas no banco }
    for i := dsEntrevistado.Tables[0].Rows.Count - 1 downto 0 do
      begin
        { Pega a linha atual }
        row := dsEntrevistado.Tables[0].Rows[i];
        { Passa os dados da linha atual do DataSet para a função
          IncluirEntrevistado do Webservice, para incluir estes dados
          no banco, retornando o código gerado pelo Webservice para o
          registro do entrevistado }
        CodEntrevistado := ws.IncluirEntrevistado(row['nome'].
          ToString, Convert.ToInt32(row['cod_pesquisa']));
        { Se retornou maior que zero é porque o registro foi incluído }
        if CodEntrevistado > 0 then
          begin
            { Filtra o DataSet de respostas retornando somente as respostas
              do entrevistado do entrevistado que foi adicionado ao banco
              (utilizando o código existente na linha atual (row) do
              DataSet dsEntrevistado) }
            dsResposta.Tables[0].DefaultView.RowFilter :=
              'cod_entrevistado=' + row['cod_entrevistado'].ToString;
            { Remove o registro do entrevistado do DataSet, pois esse já
              foi enviado ao servidor }
            dsEntrevistado.Tables[0].Rows.RemoveAt(i);
            { Percorre o DataSet dsResposta pegando todas as respostas
              do entrevistado atual }
            for j := dsResposta.Tables[0].DefaultView.Count-1 downto 0 do
              begin
                { Pega a linha atual do DataSet dsResposta }
                row2 := dsResposta.Tables[0].DefaultView.Item[j];
                { Chama a função RegistrarResposta do Webservice, para incluir
                  no banco os dados do registro atual do DataSet dsResposta.
                  Se a função retornar True, indica que a resposta foi incluída
                  no banco(ou atualizada), assim, remove do DataSet, dsResposta o
                  registro na posição atual(j) }
                if ws.RegistrarResposta(CodEntrevistado, Convert.
                  ToInt32(row2['cod_pergunta']),
                  Convert.ToInt32(row2['cod_alternativa'])) then
                  dsResposta.Tables[0].DefaultView.Delete(j);
              end;
            end;
          end;
        end;
      end;
    finally
      { Grava as alterações nos DataSet's em arquivo }
      dsEntrevistado.WriteXml(arqEntrevistado);
      dsResposta.WriteXml(arqResposta);
    end;
  end;
end.
```

### Listagem 6. Unit ClubeDelphi.InputBox.pas

```
unit ClubeDelphi.InputBox;
interface
uses System.Drawing, System.Windows.Forms, System.ComponentModel;
type
  InputBox = class(System.Windows.Forms.Form)
  private
    TextBox1: System.Windows.Forms.TextBox;
    Label1: System.Windows.Forms.Label;
    btnOK: System.Windows.Forms.Button;
    btnCancel: System.Windows.Forms.Button;
  constructor Create();
  procedure TextBox1_KeyDown(sender: &Object;
    e: System.Windows.Forms.KeyEventArgs);
  procedure InitializeComponent();
  public
    class function Show(Title, Question: string): string;
  end;
implementation
constructor InputBox.Create();
begin
  inherited Create;
  InitializeComponent();
end;
procedure InputBox.TextBox1_KeyDown(sender: &Object;
  e: System.Windows.Forms.KeyEventArgs);
begin
  if e.KeyCode = Keys.Enter then
    Self.DialogResult := System.Windows.Forms.DialogResult.OK;
end;
procedure InputBox.InitializeComponent();
begin
  Self.Label1 := System.Windows.Forms.Label.Create;
  Self.TextBox1 := System.Windows.Forms.TextBox.Create;
  Self.btnOK := System.Windows.Forms.Button.Create;
  Self.btnCancel := System.Windows.Forms.Button.Create;
  { As propriedades e métodos que estão dentro da diretiva $IFDEF
    CF só serão compiladas se não estivermos compilando para o
    Compact Framework }
  {$IFDEF CF}
  Self.SuspendLayout();
  {$ENDIF}
  Self.TextBox1.Location := System.Drawing.Point.Create(16, 16);
  {$IFDEF CF}
  Self.TextBox1.Name := 'textBox1';
  Self.TextBox1.TabIndex := 0;
  {$ENDIF}
  Self.TextBox1.Size := System.Drawing.Size.Create(180, 20);
  Self.TextBox1.Text := '';
  Self.TextBox1.Top := 28;
  Self.TextBox1.Left := 10;
  Include(Self.TextBox1.KeyDown, Self.TextBox1_KeyDown);
  Self.Label1.Text := 'Digite o dado';
  {$IFDEF CF}
  Self.Label1.AutoSize := true;
  Self.Label1.TabIndex := 1;
  {$ELSE}
  Self.Label1.Size.Width := Self.TextBox1.Size.Width;
  {$ENDIF}
  Self.Label1.Top := 10;
  Self.Label1.Left := 10;
  Self.btnOK.Text := 'OK';
  Self.btnOK.DialogResult := System.Windows.Forms.DialogResult.OK;
  {$IFDEF CF}
  Self.btnOK.TabIndex := 2;
  Self.btnOK.NotifyDefault(true);
  {$ENDIF}
  Self.btnOK.Top := 55;
  Self.btnOK.Left := 10;
  Self.btnCancel.Text := 'Cancelar';
  {$IFDEF CF}
  Self.btnCancel.TabIndex := 3;
  {$ENDIF}
  Self.btnCancel.Top := 55;
  Self.btnCancel.Left := 100;
  Self.btnCancel.DialogResult :=
    System.Windows.Forms.DialogResult.Cancel;
  {$IFDEF CF}
  Self.AutoScaleBaseSize := System.Drawing.Size.Create(5, 13);
  {$ENDIF}
  Self.ClientSize := System.Drawing.Size.Create(220, 90);
  Self.ControlBox := false;
  Self.Controls.Add(Self.TextBox1);
  Self.Controls.Add(Self.Label1);
  Self.Controls.Add(Self.btnOK);
  Self.Controls.Add(Self.btnCancel);
  Self.FormBorderStyle :=
    System.Windows.Forms.FormBorderStyle.FixedDialog;
  {$IFDEF CF}
  Self.Name := 'InputBox';
  Self.ResumeLayout(false);
  {$ENDIF}
end;
```



a partir do WindowsCE Emulador não se comportou muito bem. Assim, você pode utilizar o Device Emulador para executar o programa. Como já mencionado, esse não pode ser configurado totalmente usando o CF Builder.

Assim, vamos usar os *links* que são criados na pasta *Microsoft Windows Mobile 5.0 Emulador* no menu *Iniciar* para executá-lo. Na pasta do menu *Iniciar*, indicada anteriormente, existem seis atalhos para imagens de dispositivos. Três terminam com a palavra *Coldboot* e outras três com a palavra *Savestate*.

Os *Coldboot* iniciarão o dispositivo dando um *boot* completo, o que vai demorar bastante. Os *Savestate* iniciam o dispositivo a partir de uma sessão salva anteriormente, o que será bem rápido.

Os atalhos *Savestate* só vão funcionar depois que cada uma das sessões dos atalhos *Coldboot* forem salvas. Ao salvar a sessão no emulador é criado um arquivo DESS, permitindo reiniciar a imagem a partir do mesmo ponto onde ela parou de rodar quando o emulador foi fechado. Para salvar a sessão no emulador, acesse o menu *File* e clique na opção *Save State and Exit*.

Para configurar o *Storage Card* para as imagens, você terá que configurar cada uma das *Coldboot*. Assim, clique com o botão direito em uma e escolha *Propriedades*. Na aba *Atalho*, no campo *Destino*, adicione no final a linha *"/sharedfolder PastaQueDesejarUsarComoStorageCard"*.

Se o caminho da pasta a ser utilizada tiver espaço, coloque todo o caminho "entre aspas". Após configurar o *Storage Card* nos três atalhos *Coldboot*, execute algum deles.

Para abrir o *Storage Card*, clique no menu *Iniciar* do emulador e escolha *File Explorer*. Na parte superior da janela existe um botão onde você pode selecionar *Storage Card* para abrir a pasta compartilhada no seu computador e acessar os arquivos lá.

## Configurando a rede no Device Emulador

Com as imagens de Windows Mobile (o Device Emulador só usa essas) o acesso à Internet ou a uma rede local é feito por *Bluetooth*, infravermelho ou via *ActiveSync*, esse último quando se conecta o dispositivo a um PC. Temos também como emular essa conexão do emulador

### Listagem 7. Método Show do InputBox

```
class function InputBox.Show(Title,
    Question: string): string;
var
    box: InputBox;
begin
    box := InputBox.Create;
    box.Text := Title;
    box.Label1.Text := Question;
    if box.ShowDialog = System.Windows.Forms.DialogResult.OK then
        Result := box.TextBox1.Text
    else Result:= '';
end;
```

### Listagem 8. Menu Selecionar dados no Servidor

```
var
    ws: TPesquisaOpiniaowS;
    titulo: string;
    codPesquisa: Integer;
begin
    { Chama o InputBox para solicitar o título da pesquisa ao usuário }
    titulo := InputBox.Show('Selecionar Pesquisa', 'Título da pesquisa');
    if titulo <> '' then
        begin
            { Instancia um objeto da classe do Webservice }
            ws := TPesquisaOpiniaowS.Create;
            { Busca os dados da pesquisa no Webservice }
            dsPesquisa := ws.GetPesquisa(titulo);
            if (dsPesquisa.Tables[0].Rows.Count > 0) then
                begin
                    { Liga a tabela do DataSet para que as pesquisas sejam exibidas no ComboBox }
                    cbxPesquisa.DataSource := dsPesquisa.Tables[0];
                    { Guarda o código da pesquisa retornada }
                    codPesquisa := Convert.ToInt32(dsPesquisa.Tables[0].Rows[0]['cod_pesquisa']);
                    { Armazena os dados da pesquisa localmente em um XML. XMLCache é a classe que criamos
                    para agrupar as funções para manipulação dos DataSets }
                    XMLCache.GravarXML(dsPesquisa, XMLCache.arqPesquisa);
                    dsPergunta := ws.GetPerguntas(codPesquisa);
                    XMLCache.GravarXML(dsPergunta, XMLCache.arqPergunta);
                    { Se existe alguma pergunta cadastrada para a pesquisa }
                    if dsPergunta.Tables[0].Rows.Count > 0 then
                        begin
                            dsAlternativa:= ws.GetAlternativas(codPesquisa);
                            XMLCache.GravarXML(dsAlternativa, XMLCache.arqAlternativa);
                            { Chama a função GetEntrevistado do Webservice informando um nome de pessoa que não
                            vai existir, para que seja retornado um DataSet vazio, apenas com a estrutura da
                            tabela para que se possa armazenar os dados localmente neste DataSet }
                            dsEntrevistado:= ws.GetEntrevistado('nenhum');
                            XMLCache.GravarXML(dsEntrevistado, XMLCache.arqEntrevistado);
                            { A mesma lógica anterior }
                            dsResposta := ws.GetRespostaEntrevistado(-1);
                            XMLCache.GravarXML(dsResposta, XMLCache.arqResposta);
                        end;
                    end
                else
                    MessageBox.Show('Nenhuma pesquisa retornada', 'Informação');
                end;
            end;
```

(o dispositivo que temos) ao PC.

Você precisará que o Device Emulador Manager (encontrado na mesma pasta do menu iniciar indicada anteriormente) e o ActiveSync estejam rodando. O ActiveSync disponibiliza o ícone  no *System Tray*.

Clique com o botão direito nele e escolha a opção *Configurações de conexão*. Na janela que abre, marque a opção *Permitir conexão com um dos seguintes itens* e no campo abaixo, escolha *DMA* para permitir conectar o computador virtualmente ao emulador, sem uma conexão física. Clique em *OK*.

Após rodar o emulador, no Microsoft Device Emulador deverá aparecer a referência para o dispositivo, se não, clique no *Refresh*. Com o botão direito sobre o link para dispositivo, dentro do Microsoft Device Emulador, escolha a opção *Cradle*, para

simular a conexão do dispositivo ao PC.

O ícone do ActiveSync deve ativar, ficando verde, indicando que está conectando ao dispositivo. Após ele concluir a conexão, você poderá acessar uma rede local ou à internet a partir do dispositivo, sem nenhuma configuração adicional.

Caso o dispositivo não seja conectado pelo ActiveSync automaticamente, clique com o botão direito no ícone do ActiveSync no *System Tray* e escolha a opção *Configurações de conexão*. Na janela que abre, clique em *Conectar*.

Após o emulador estar conectado no ActiveSync, acesse o *Storage Card* e execute a aplicação. Para atualizar o sistema, basta copiar o executável para a pasta compartilhada e executar novamente no emulador, sem precisar fechá-lo.

### Listagem 9. SelectedValueChanged do cbxPergunta

```
var
  codPergunta: Integer;
begin
  { Se há alguma pergunta selecionada, mostra no ComboBox de alternativa (cbxResposta),
  somente as referentes à pergunta selecionada }
  if (cbxPergunta.SelectedIndex <> -1) and (cbxPergunta.SelectedValue <> nil) then
  begin
    { Pega o código da pergunta selecionada }
    codPergunta := Convert.ToInt32(cbxPergunta.SelectedValue);
    { Filtra o DataSet de alternativas para exibir apenas as alternativas da resposta selecionada }
    dsAlternativa.Tables[0].DefaultView.RowFilter := 'cod_pergunta=' + codPergunta.ToString;
    { Vincula os registros filtrados ao ComboBox das Alternativas }
    cbxResposta.DataSource := dsAlternativa.Tables[0].DefaultView;
    { Não deixa nenhum item marcado no ComboBox }
    cbxResposta.SelectedIndex:= -1;
  end;
end;
```

### Listagem 10. Incluir Entrevistado

```
public
  CodEntrevistado: integer;
...
begin
  { Inclui o entrevistado no DataSet local, retornando o código temporário gerado para o
  registro, pois este código é alterado quando o registro é enviado ao banco de dados }
  CodEntrevistado := XMLCache.IncluirEntrevistado(dsEntrevistado, txtEntrevistado.Text,
  Convert.ToInt32(cbxPesquisa.SelectedValue));
  { Exibe o código gerado }
  lblCodEntrevistado.Text := 'Código: ' + CodEntrevistado.ToString;
  { Desvincula o procedimento
  cbxPergunta_SelectedValueChanged do evento SelectedValueChanged do cbxPergunta enquanto os dados
  são vinculados ao cbxPergunta, para evitar que o evento seja disparado no momento errado }
  Exclude(cbxPergunta.SelectedValueChanged, cbxPergunta_SelectedValueChanged);
try
  { Vincula os dados do DataSet dsPergunta ao cbxPergunta para que as perguntas sejam
  exibidas no ComboBox }
  cbxPergunta.DataSource := dsPergunta.Tables[0];
finally
  { Vincula novamente o procedimento cbxPergunta_SelectedValueChanged ao evento
  SelectedValueChanged do cbxPergunta }
  Include(cbxPergunta.SelectedValueChanged, cbxPergunta_SelectedValueChanged);
end;
{ Desmarca a pergunta atualmente selecionada }
cbxPergunta.SelectedIndex:= -1;
{ Manda o cursor para o campo }
cbxPergunta.Focus;
```

### Listagem 11. Procedimentos para navegar entre as perguntas

```
procedure frmMain.PerguntaAnterior;
begin
  { Volta à pergunta anterior }
  if cbxPergunta.SelectedIndex > 0 then
    cbxPergunta.SelectedIndex := cbxPergunta.SelectedIndex - 1;
  cbxPergunta.Focus;
end;

procedure frmMain.ProximaPergunta;
begin
  { Avança à próxima pergunta }
  if cbxPergunta.SelectedIndex < cbxPergunta.Items.Count - 1 then
    cbxPergunta.SelectedIndex := cbxPergunta.SelectedIndex + 1;
  cbxPergunta.Focus;
end;
```

### Listagem 12. Registrar a resposta do entrevistado

```
{ Registra a resposta do entrevistado no DataSet local e passa para a próxima pergunta }
XMLCache.RegistrarResposta(dsResposta, CodEntrevistado, Convert.ToInt32(
  cbxPergunta.SelectedValue), Convert.ToInt32(cbxResposta.SelectedValue));
ProximaPergunta;
```

### Listagem 13. Salva os dados dos DataSets em arquivos XML

```
procedure frmMain.SalvarCacheLocal;
begin
  { Se existe alguma tabela no DataSet e existe algum registro nela, salva os dados em arquivo }
  if (dsEntrevistado.Tables.Count > 0) and
    (dsEntrevistado.Tables[0].Rows.Count > 0) then
    XMLCache.GravarXML(dsEntrevistado, XMLCache.arqEntrevistado);
  if (dsResposta.Tables.Count > 0) and (dsResposta.Tables[0].Rows.Count > 0) then
    XMLCache.GravarXML(dsResposta, XMLCache.arqResposta);
end;
```

## Continuando o desenvolvimento da aplicação cliente

Depois de ter testado o Device Emulador, vamos continuar no desenvolvimento da aplicação. Quando o usuário selecionar uma pergunta, devemos mostrar no *ComboBox* de alternativas, somente as referentes à pergunta selecionada. Assim, no evento *SelectedValueChanged* do *cbxPergunta* adicione o código da **Listagem 9**.

Agora inclua o código para o *Click* do *Incluir*, que fará a inclusão do entrevistado no *DataSet* local, conforme a **Listagem 10**.

Crie os procedimentos “PerguntaAnterior” e “ProximaPergunta” para permitir ao usuário navegar entre as perguntas disponíveis. Veja o código dos dois procedimentos na **Listagem 11**.

Agora chame os procedimentos nos seus respectivos botões. No evento *Click* do *Registrar* digite o código da **Listagem 12**, para registrar a resposta do entrevistado no *DataSet* local.

## Testando a aplicação

Compile a aplicação (usando a barra do CF Builder), copie o executável para a pasta compartilhada e execute-o a partir do *Storage Card* no Device Emulador. Você deverá acessar o menu *Selecionar dados no Servidor*, caso ainda não tenha feito isso, para guardar os dados da pesquisa, as perguntas e alternativas, localmente em arquivos XML no dispositivo móvel.

Digite o nome do entrevistado e clique em *Incluir* para registrá-lo. Em seguida, selecione a pergunta, depois a resposta e clique em *Registrar*. Repita esse último passo até responder todas as perguntas.

Para incluir um novo questionário de um entrevistado, digite o nome do mesmo e clique em *Incluir*, repetindo todo o processo.

## Salvando em disco

Vamos agora fazer com que os dados sejam salvos em disco quando o sistema for fechado. Poderíamos fazer isso em outro evento também, como a cada pergunta respondida. Assim, crie a função da **Listagem 13** e depois chame-a no evento *Closed* do *FrmMain*.

Agora que fizemos a rotina para chamar a função para salvar os *DataSets* localmente, precisamos criar o código para carregar esses arquivos quando a aplicação inicializar. Assim, digite o código da **Listagem**

**Listagem 14.** Evento Load do FrmMain para carregar os arquivos XML

```

( Se carregou o arquivo de pesquisas, carrega os outros em seguida )
if XMLCache.CarregarXML(dsPesquisa, XMLCache.arqPesquisa) then
begin
  ( Vincula os dados do DataSet dsPesquisa ao cbxPesquisa para exibir o título da pesquisa
  no ComboBox )
  cbxPesquisa.DataSource := dsPesquisa.Tables[0];
  ( Se carregou o arquivo de perguntas, então carrega o de alternativas )
  if XMLCache.CarregarXML(dsPergunta, XMLCache.arqPergunta) then
  XMLCache.CarregarXML(dsAlternativa, XMLCache.arqAlternativa);
  ( Se carregou o arquivo de entrevistados, então carrega o de respostas )
  if XMLCache.CarregarXML(dsEntrevistado, XMLCache.arqEntrevistado) then
  XMLCache.CarregarXML(dsResposta, XMLCache.arqResposta);
end;

```

**14** no evento *Load* do *FrmMain*.

Quando o usuário tiver conexão com a internet ou diretamente à rede local da empresa, ele deve enviar os dados ao servidor para serem cadastrados no banco de dados, usando o *EnviarCacheXML* criada anteriormente.

No *FrmMain* inclua um novo item no *MainMenu* com o título (propriedade *Text*) igual a "Enviar dados ao Servidor". No *Click* deste item inclua o código a seguir para chamar a função recém-criada, que enviará os dados armazenados localmente em XML ao servidor:

```
XMLCache.EnviarCacheXML(dsEntrevistado,
dsResposta);
```

Rode a aplicação e faça os testes necessários (**Figura 4**).

**Conclusão**

Pronto, agora é só fazer os testes. Alguns detalhes para melhoria da interface não foram incluídos, como a habilitação/desabilitação dos botões nos momentos adequados, para evitar problemas caso o usuário não siga a ordem correta, mas isso é bem simples de fazer e fica para vocês poderem melhorar a aplicação.

Um recurso muito interessante seria a inclusão de um *CheckBox* para informar se é para trabalhar off-line. Estando desmarcado, a aplicação já mandaria os dados diretamente para o servidor.

Isso também é fácil de implementar, basta chamar as devidas funções, que já estão implementadas. Outro detalhe, que é bastante importante, é quanto ao controle de transações que não foi aplicado, para garantir, por exemplo, que o registro

do entrevistado com os registros das respostas sejam todos incluídos ou, em caso de erro, nenhum ser incluído.

Também não me preocupei em verificar se o usuário informou valores para os campos, antes de inserir os dados. Assim, fica o homework para vocês fazerem. Neste artigo procurei mostrar a maioria dos detalhes quanto à montagem do ambiente de desenvolvimento, configuração e utilização das ferramentas assim como macetes para permitir que você ingresse no mundo do desenvolvimento de aplicações para dispositivos móveis.

Espero que tenham gostado e até o próximo. ●



**Figura 4.** Aplicação sendo testada no Pocket PC

# Mais Vendidos!



**Gerenciando Projetos de Desenvolvimento de Software com PMI, RUP e UML**  
4ª. Edição

Brasport

**R\$ 75,00**

**Dominando Netbeans**  
Construa aplicativos Java tanto em Desktop, como para Web

Ciência Moderna

**De: R\$ 76,00**

**Por: R\$ 72,20**



**Desenvolvendo Aplicações Web com NetBeans IDE 5.5**

Ciência Moderna

**R\$ 99,00**

**Programação Orientada a Objetos Usando Delphi - 4ª edição**

Visual Books

**De: R\$ 75,00**

**Por: R\$ 71,25**



**Frete Grátis**  
Para todo o Brasil!

**Até 6x sem juros no cartão.\***

\*Parcela mínima de R\$ 20,00

Outras facilidades de pagamento:



**Tel: (11) 4062-5152**

**livrosdeprogramacao.com.br**

ClubeDelphi PLUS

[www.devmedia.com.br/clubedelphi/portal.asp](http://www.devmedia.com.br/clubedelphi/portal.asp)

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma série de vídeo aulas de Jefferson Junlaus, sobre o Compact Framework no Delphi.

[www.devmedia.com.br/articles/listcomp.asp?txtsearch=Compact+Framework+no+Delphi](http://www.devmedia.com.br/articles/listcomp.asp?txtsearch=Compact+Framework+no+Delphi)



# Contas a Pagar e Cobrança

Crie um sistema completo com Delphi, Firebird 2.0 e dbExpress - Parte 4

Chegamos a quarta parte do nosso mini-curso Sistema de Contas a Pagar e Cobrança onde estamos vendo alguns dos principais recursos em aplicações dessa natureza. Uma aplicação de contas a pagar e cobrança não se resume apenas em entradas e saídas de caixa. Deve-se pensar também em fluxo de caixa e geração de boletos bancários para cobrança, um dos carros-chefe da aplicação.

Para iniciarmos nosso artigo, veremos rapidamente o que foi criado no exemplo anterior:

- Cadastro de contas a receber;
- Manutenção do Data Module principal para criação da tabela Contas\_Receber;
- Criação do recurso de duplicação de contas parceladas;

Daremos continuidade ao curso entendendo e desenvolvendo outro recurso bastante importante: Geração de texto para Cobrança Escritural. Nesse artigo veremos como fazer a geração do arquivo

texto para envio à instituição bancária que por sua vez se encarregará de gerar os boletos e envio dos mesmos efetuando desta forma a cobrança bancária.

## Alterando o banco de dados de exemplo

Diferentemente dos artigos anteriores não precisaremos criar tabelas. Dessa vez criaremos uma *Stored Procedure* selecionável que resultará os dados necessários para geração dos arquivos de remessa.

Uma *Stored Procedure* selecionável é basicamente uma função criada no banco de dados capaz de preencher um ou mais parâmetros de saída que posteriormente serão usados pela aplicação Delphi como se fosse uma tabela comum.

Nossa *Stored Procedure* fará um filtro na tabela de contas a receber usando como parâmetros de entrada duas datas. Essas datas serão usadas na cláusula *Where* da SP (“*Stored Procedure*”) onde faremos o filtro no campo *DT\_VECTO*, ou seja, filtraremos



### Adriano Santos

(artes@doiscliques.com)

é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É colunista e membro da Comissão Editorial da revista ClubeDelphi. É também Editor Técnico da Revista ClubeDelphi. Mantém o blog Delphi to Delphi ([www.delphitodelphi.blogspot.com](http://www.delphitodelphi.blogspot.com)) com dicas, informações e tudo sobre desenvolvimento Delphi.

todos os registros de contas a receber que estejam entre as duas datas recebidas.

Será levado em consideração também se o registro é diferente de "I" ("inativo") e "B" ("baixado"). Assim evitamos enviar um título inativo por meio de cancelamento/exclusão ou ainda que tenha sido liquidado ("baixado").

Para criar nossa SP selecionável abra o IBExpert e conecte-se a base de dados de exemplo SysPague. Em seguida clique com o botão direito em *Procedure* e selecione *New Procedure*. Repare que o IBExpert exibe, em seu editor, uma divisão onde encontramos os botões *Input Parameters*, *Output Parameters*, *Variables* e *Cursor*. E mais acima podemos ver uma área vazia com os títulos *Name*, *Type*, *Size*, *Scale*, *Default Source*, *Subtype*, *Charset* e *Description*. É nessa área que criaremos os parâmetros de entrada ("Input Parameters") e saída ("Output Parameters") que por sinal se assemelha bastante com a criação de campos em uma tabela.

Pois bem, clique em *Input Parameters* e com o botão direito do mouse na área vazia selecione a opção *Append parameter/variable* ou localize este botão logo acima dessa área e clique-o. Um novo parâmetro será criado, agora basta modificar seu *Name* para "DT\_INICIO" e seu *Type* para "Timestamp" deixando o restante dos itens com seus valores padrão. Repita o processo e crie um parâmetro chamado "DT\_FIM" também como "Timestamp" em seu *Type*.

Nesse momento acabamos de criar dois parâmetros de entrada que serão usados em nossa aplicação. Enviaremos para

Campo	Tipo/Tamanho
CNPJ	VarChar(18)
VALOR_REAL	Numeric(15,2)
DT_CADASTRO	TimeStamp
DT_VECTO	TimeStatmp
CODIGO	VarChar(20)
RAZAO	VarChar(150)
ENDERECO	VarChar(100)
BAIRRO	VarChar(50)
CIDADE	VarChar(50)
ESTADO	VarChar(2)
CEP	VarChar(9)

Tabela 1. Parâmetros de saída da SP

a *Stored Procedure* uma data de início e outra de fim, assim nossa SP selecionará automaticamente o que deverá ser enviado à instituição.

Agora vamos aos parâmetros de saída chamados de *Output Parameters*. São eles que nossa SP alimentará e que serão utilizados pelo nosso sistema como se fosse uma tabela comum do banco. Portanto clique no item *Output Parameters* e repita os passos anteriores incluindo os parâmetros da **Tabela 1**.

Modifique o nome da *Stored procedure* para "RetornarBoletos" na parte superior direita do IBExpert. Caso prefira, execute o *script* da **Listagem 1**.

### Cobrança Escritural - CNAB

Para que possamos desenvolver a cobrança escritural, também chamada de CNAB, devemos primeiramente entender exatamente do que se trata e qual sua finalidade.

CNAB significa "Comissão de Tecnologia e Automação Bancária" e é basicamente um conjunto de regras estipuladas para geração de dados em formato texto. Esses dados são posteriormente enviados à instituição bancária para que sejam processados. Esse conjunto de regras,

também chamado de padrão, contém texto em forma de colunas que seguem regras definidas pela FEBRABAN ("Federação Brasileira dos Bancos"). O conjunto de regras estipuladas pela FEBRABAN é chamado de layout.

O layout do arquivo pode variar de banco para banco devido às particularidades de cada um, porém não é permitido que fuja demais do padrão estipulado.

Normalmente esses layouts são disponibilizados publicamente no website da instituição para que desenvolvedores possam adquirir e desenvolver suas próprias soluções. Contudo em alguns casos o layout só pode ser adquirido por meio de um correntista que possui usuário e senha de acesso ao portal do banco, por isso muitas vezes é necessário que o próprio cliente adquira o layout e o retransmita ao desenvolvedor responsável.

Os layouts CNAB seguem dois padrões:

- CNAB 240: Possui 240 posições, também chamado de colunas, em cada linha. Sendo que para cada boleto a ser gerado pela instituição há duas linhas de 240 colunas cada, ou seja, necessitamos gerar um boleto bancário para cada cliente e supondo que temos dez clientes, o arquivo deverá possuir vinte linhas de registro;

Listagem 1. Criação da tabela e índices Contas\_Receber

```

SET TERM ^ ;

CREATE PROCEDURE RETORNARBOLETOS (
    DT_INICIO TIMESTAMP,
    DT_FIM TIMESTAMP)
RETURNS (
    CNPJ VARCHAR(18),
    VALOR_REAL DOUBLE PRECISION,
    DT_CADASTRO TIMESTAMP,
    DT_VECTO TIMESTAMP,
    CODIGO VARCHAR(20),
    RAZAO VARCHAR(150),
    ENDERECO VARCHAR(100),
    BAIRRO VARCHAR(50),
    CIDADE VARCHAR(50),
    ESTADO VARCHAR(2),
    CEP VARCHAR(9))
AS
begin
FOR SELECT
    CP.CODIGO, CP.CNPJ, CP.VLR_REAL,
    CP.DT_CADASTRO, CP.DT_VECTO, CL.RAZAO,
    CL.ENDERECO, CL.BAIRRO, CL.CIDADE,
    CL.ESTADO, CL.CEP
FROM
    CONTAS_RECEBER CP
    INNER JOIN CLIENTES CL
    ON (CP.CNPJ = CL.CNPJ)
WHERE
    (CP.DT_VECTO BETWEEN :DT_INICIO AND :DT_FIM) AND
    (CP.STATUS NOT IN ('I', 'B'))
INTO
    :CODIGO, :CNPJ, :VALOR_REAL, :DT_CADASTRO,
    :DT_VECTO, :RAZAO, :ENDERECO, :BAIRRO, :CIDADE,
    :ESTADO, :CEP
DO
    SUSPEND;
end^
    
```

• **CNAB 400:** São 400 colunas para cada linha do arquivo;

Atualmente o formato 240 está sendo gradativamente desativado pelas instituições por ter se tornado obsoleto, complexo e trabalhoso de lidar.

A finalidade do **CNAB** é poder efetuar a cobrança bancária de títulos, ou seja, após enviada ao banco a remessa é processada e boletos bancários são expedidos ao cliente. O arquivo de remessa possui entre as várias colunas dados como: valor, vencimento, agência e conta onde o valor pago será creditado, instruções de protesto e dados do sacado, entre outros.

Além do arquivo Remessa o banco disponibiliza o arquivo Retorno, que é justamente o contrário de remessa. Com o arquivo retorno podemos efetuar a baixa do título em nosso sistema informando valor pago, juros, mora, data de pagamento, agência e conta creditada, e etc.

De porte de todas essas informações fica fácil conferir o extrato bancário no início ou fim do dia através do sistema tendo como diferença pequenas taxas e tarifas, tais como: taxa de manutenção da conta corrente e CPMF.

---

**Nota:** Nesse artigo veremos somente a geração do arquivo Remessa, porém a operação inversa segue basicamente a mesma idéia.

---

---

**Nota:** Para uma melhor compreensão do artigo é recomendada a leitura do artigo Sistema EDI na edição 84 da revista ClubeDelphi.

---

Como dito anteriormente, cada banco possui certas particularidades em relação ao padrão utilizado, portanto o mais sensato é encontrar o layout **CNAB** do banco desejado e fazer a leitura completa de todo o arquivo de ajuda que normalmente encontra-se em formato PDF ou DOC.

Além do layout as instituições costumam manter um departamento de suporte ao desenvolvedor para sanar possíveis dúvidas com o layout e/ou procedimentos de envio e retorno dos arquivos a serem trocados entre instituição financeira e cliente.

A grande maioria dos bancos disponibiliza uma área de testes para o de-

seenvolvedor efetuar os primeiros testes de carregamento e geração dos boletos bancários.

A geração destes arquivos de remessa pode ser feito utilizando-se de componentes de terceiros, mas por questões de didática e entendimento do mecanismo de desenvolvimento optei por fazer manualmente, até mesmo porque o processo é bastante simples.

## Preparando o sistema

Antes de iniciarmos o desenvolvimento da caixa de diálogo e geração dos arquivos, faremos a preparação pra a leitura dos dados que será feita através da *SP* RetornarBoletos criada no início do artigo.

Abra o Delphi 7 e em seguida o projeto SysPague.dpr de nosso mini-curso. Após a abertura do projeto abra a Unit do Data Module principal e inclua um novo grupo de componentes para acesso a nossa *SP*.

Insira um componente do tipo *SQLQuery* (“*sqlBoletos*”) e mude sua propriedade *SQLConnection* para “*sqlConexao*”. Em seguida digite a instrução *SQL* a seguir na propriedade *SQL* do componente *SQLQuery*.

```
select * from RetornarBoletos(:DT_INICIO,
:DT_FIM) order by CNPJ
```

Note que criamos dois parâmetros na instrução: *DT\_INICIO* e *DT\_FIM*. Esses parâmetros são usados para selecionar quais os dados serão retornados pela *Stored Procedure* para que possamos então gerar o arquivo remessa. Em nossa *SP* todos os registros da tabela *Contas\_Receber* que têm data de vencimento entre as *DT\_INICIO* e *DT\_FIM* e que seu status não seja “*I*” (inativo) ou “*B*” (baixado) serão retornados.

Poderíamos requerer outros dados, como por exemplo o *CNPJ* do cliente, assim poderíamos gerar o arquivo de remessa por cliente pagador. Para isso seria necessária a criação de um novo parâmetro de entrada na *SP*.

Feito isso devemos agora configurar os parâmetros criados pelo componente *SQLQuery*, por isso clique duas vezes na propriedade *Params* do *sqlBoletos* e veja que ambos os parâmetros foram criados.

Por se tratar de parâmetros do mesmo tipo, faremos a configuração de ambos ao mesmo tempo, por isso selecione-os simultaneamente usando a tecla *CTRL* e clicando com o mouse em cada um.

Agora pressione *F11* para alternar para o *Object Inspector* e mude a propriedade *DataType* para “*ftTimeStamp*” e *ParamType* para “*ptInput*”.

Feito isso, insira todos os parâmetros de saída, que agora se tornarão campos, no *Fields Editor* do *sqlBoletos*. Para isso clique duas vezes no *sqlBoletos* e em seguida com o botão direito escolha a opção *Add all fields*.

Como pode notar não há segredos na criação de *SP*'s selecionáveis. Faremos a leitura de sei *Result Set* (“resultado da consulta”) como se fosse uma tabela comum do banco de dados.

Insira agora um componente do tipo *DataSetProvider* (“*dspBoletos*”) da paleta *Data Access* e configure sua propriedade *DataSet* para “*sqlBoletos*”.

Coloque um *ClientDataSet* (“*cdsBoletos*”) também da paleta *Data Access* e ligue-o ao *DataSetProvider* usando a propriedade *ProviderName*. Em seguida clique com o botão direito no componente e selecione *Fetch Params* para que o *cdsBoletos* capture os parâmetros criados no *sqlBoletos*.

Use a opção *Add all fields* do *Fields Editor* para adicionar todos os campos da *SP* selecionável assim como fizemos no *sqlBoletos*. Se preferir modifique a propriedade *DisplayName* de cada campo adicionado conforme desejado. Essa modificação refletirá nos componentes visuais incluídos em tela tais como *Labels* e *DBGrids*. Veja como ficou nosso *Data Module* na **Figura 1**.

## Criando a tela geração da remessa

Nossa tela de geração dos arquivos de remessa consiste apenas em receber o período desejado para geração do **CNAB** e a escolha de qual banco, agência e conta serão os mantenedores dos boletos.

Fixaremos a pasta de geração do arquivo usando a função *ExtractFilePath* passando como parâmetro o método *ExeName* do objeto *TApplication* mais o nome do arquivo de remessa, que nesse caso fixaremos como: “*Remessa.txt*”.

Fazendo isso garantimos que a pasta de geração seja sempre o local onde nossa

aplicação esteja rodando, ou seja, a pasta de instalação do executável. Mostraremos este caminho em um *Label* azul para que o usuário visualize e identifique rapidamente o caminho da remessa.

Vamos iniciar então a criação de um novo formulário e para isso selecione *File|New>Form*. Salve o formulário usando o menu *File|Save as* e atribua a ele o nome "uCobrancaEscritural.pas". Em seguida mude a propriedade *Name* do formulário para "frmCobrancaEscritural". Se desejar modifique sua largura *Width* e altura *Height* bem como as configurações de bordas: *BorderStyle* e *BorderIcons*. Uma prévia da tela pode ser vista na **Figura 2**.

Primeiramente adicione um componente *GroupBox* da paleta *Standard* e dentro dele insira dois *DateTimePicker* com os nomes "dtpDtInicio" e "dtpDtFim" respectivamente. Acima de cada um adicione dois *Labels* com os *Captions* de "Data de início" e "Data de fim" respectivamente. Inclua um novo *Label* entre *dtpDtInicio* e *dtpDtFim* com o *Caption* "a".

Após isso insira três novos *Labels* e mude o *Caption* de cada um para "Banco", "Agência" e "Conta". Eles servirão de nomes aos três *Edits* que serão adicionados agora, sendo que cada um será responsável por representar um campo da tabela CONTAS, ou seja, coloque estes três componentes e renomeie-os para "edtBanco", "edtAgencia" e "edtConta".

Após isso inclua um *SpeedButton* ("sbbSelegcionarBanco") e adicione uma imagem ao mesmo usando a propriedade *Glyph*. Se preferir copie o botão de alguma janela que já tenhamos adicionado uma instância desse componente, como "Cadastro de Clientes", por exemplo, e cole-o no formulário.

Ao centro da janela insira um *ProgressBar* ("pgbProgresso") da paleta Win32. Modifique sua propriedade *Step* para "1".

No rodapé da página adicione dois *Labels* sendo que o primeiro deverá ter o *Caption* alterado para "Salvar arquivo como:". Já no segundo *Label* ("lblCaminhoArquivo") cujo o *Caption* será "C:\Temp" configure as propriedades o e *AutoSize* para *False*. Redimensione a largura e altura do *lblCaminhoArquivo* de forma que ocupe a largura da janela e duas ou mais linhas de altura.

Adicione dois botões à tela configurando seus respectivos *Captions* para "Gerar" e "Cancelar". Modifique também a propriedade *Name* do botão "Gerar" para "btnGerar" e do botão "Cancelar" para "btnCancelar".

### Codificando o formulário

A primeira providência a tomarmos em relação a codificação, será alterar a propriedade *Caption* do *lblCaminhoArquivo* em dinamicamente para que o usuário possa visualizar onde o arquivo será salvo, por isso entre no evento *OnShow* do formulário e apenas digite o código a seguir:

```
lblCaminhoArquivo.Caption :=
ExtractFilePath(Application.
ExeName)+'Remessa.txt';
```

Como mencionado anteriormente estamos concatenando o diretório de instalação do executável com o nome de arquivo que é fixado em "Remessa.txt".



Figura 1. Data Module alterado

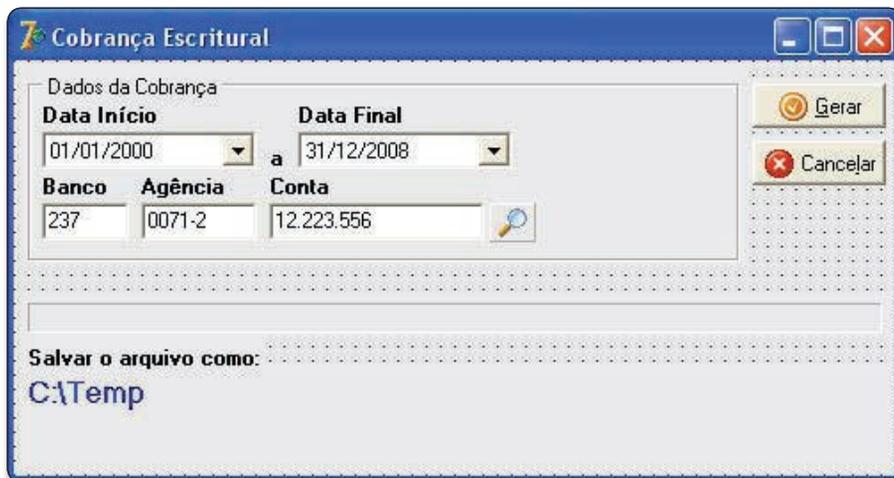


Figura 2. Tela de sugestão da Cobrança Escritural

#### Listagem 2. Botão de pesquisa de bancos

```
procedure TfrmCobrancaEscritural.
sbbSelegcionarBancoClick(Sender: TObject);
begin
frmPesquisaPadrao := TfrmPesquisaPadrao.Create(Self);
with frmPesquisaPadrao do
begin
Ftabela := 'Contas';
FDataSetAlvo := dmdPrincipal.cdsContas;
ShowModal;

if (ModalResult = mrOk) and not
(dstGenerico.DataSet.IsEmpty) then
begin
edtBanco.Text := dstGenerico.DataSet.FieldByName('BANCO').AsString;
edtAgencia.Text := dstGenerico.DataSet.FieldByName('AGENCIA').AsString;
edtConta.Text := dstGenerico.DataSet.FieldByName('CONTA').AsString;
end;
end;
FreeAndNil(frmPesquisaPadrao);
end;
```

Em seguida faremos a codificação do botão *sbbSelecionarBanco* que possui exatamente o mesmo método de pesquisa padrão encontrado nas janelas “Baixar contas a receber” e “Baixar contas a pagar”, ou seja, faz uso do formulário “Pesquisa padrão”.

Se preferir abra um dos formulários de baixa descritos anteriormente e copie o evento do botão de pesquisa para o *sbbSelecionarbanco*. O código deste botão está descrito na **Listagem 2**.

---

**Nota:** Não esqueça de que para funcionar a Pesquisa Padrão seu formulário deve ser adicionado ao Usos do formulário Cobrança Escritural e para isso será necessário pressionar ALT + F11 ou ir ao menu File|Use unit.

---

Nessa parte do artigo faremos uso também de algumas funções e procedimentos especiais para só então codificarmos

a geração da remessa propriamente dita. Essas funções e procedimentos especiais se fazem necessárias, pois devemos adequar nosso arquivo de remessa aos padrões de CNAB. Essas funções são:

- *LimparString*: Retira caracteres de uma *String* conforme o segundo parâmetro passado. Ideal para limpar máscaras em campos do tipo CPF, CEP e CNPJ;
- *PadL*: Alinha o texto à esquerda incluindo caracteres em branco à direita da *string*; Ideal para completar as posições à esquerda do texto que precisam ser preenchidas para se enquadrar com o número de caracteres solicitados pelo CNAB;
- *PadR*: Ao contrário de *PadL*, ou seja, alinha o texto à direita incluindo caracteres em branco à esquerda da *String*.
- *Replicar*: Repete um determinado caractere “n” vezes;
- *SoNumeros*: Retira máscaras de números deixando somente números inteiros; Para incluir as funções e procedimen-

tos acima, declare-as na seção *Public* do formulário conforme a seguir e pressione CTRL + SHIFT + C para que o Delphi crie para nós o cabeçalho das mesmas.

```
function LimparString(
  ATexto, ACaracteres: string): string;
function PadL(
  ATexto: string; ATamanho: Integer): string;
function PadR(
  ATexto: string; ATamanho: integer): string;
function Replicar
  (ATexto: string; AVEzes: Integer): string;
function SoNumeros(
  const ATexto: string): string;
```

Encontre na área *Implementation* do formulário o cabeçalho de cada função e codifique de acordo com a **Listagem 3** onde você pode ver o código de todos os procedimentos e funções.

Algumas considerações devem ser revistas antes de iniciarmos a codificação do layout. De acordo com o padrão CNAB:

- Todo campo de texto deve ser formatado com alinhamento à esquerda e preenchido com caracteres brancos à direita;
- Todo e qualquer campo numérico deve ser alinhado à direita precedido de zeros;
- Campos de valor, como juros, mora e valor do boleto, devem ser apresentados sem formatação monetária, ou seja, sem quaisquer separadores seja ele milhar ou decimal;
- Campos de data devem obedecer ao padrão estipulado no layout do banco podendo variar bastante de um banco para outro, ex: DDMMYYYY, DDMMYY, DMY e etc;

Muito bem, agora precisamos apenas codificar a parte mais importante de nosso exemplo que é o evento *OnClick* botão *btnGerar*. Por isso clique duas vezes no botão em questão e digite o código da **Listagem 4**.

## Entendendo o código

O código de geração da remessa não possui, na verdade, nenhuma magia ou efeito especial de cinema. Nas primeiras linhas do código estamos declarando duas variáveis que receberão o endereço de memória do arquivo e o número da linha gerada.

Primeiramente atribuímos o valor dos parâmetros DT\_INICIO e DT\_FIM do *cdsBoletos*, presente no Data Module, o mesmo valor dos campos *dtpDtInicio* e *dtpDtFim* que são as datas de início e fim do período de vencimento dos boletos.

### Listagem 3. Funções e procedimentos especiais

```
function TfrmCobrancaEscritural.
  LimparString(ATexto, ACaracteres: string): string;
var
  strAux      : string;
  I           : integer;
begin
  strAux := '';
  for I := 1 to Length(ATexto) do
    if Pos(Copy(ATexto, I, 1), ACaracteres) <= 0 then
      strAux := strAux + Copy(ATexto, I, 1);
  Result := strAux;
end;

function TfrmCobrancaEscritural.PadL(ATexto: string; ATamanho: Integer): string;
var
  I : integer;
begin
  Result := ATexto;
  for I := 1 to (ATamanho - Length(Result)) do
    Result := Result + ' ';
end;

function TfrmCobrancaEscritural.PadR(ATexto: string; ATamanho: integer): string;
var
  I : integer;
begin
  Result := ATexto;
  for I := 1 to (ATamanho - Length(Result)) do
    Result := ' ' + Result;
end;

function TfrmCobrancaEscritural.SoNumeros(const ATexto: string): string;
var
  I: Integer;
  S: string;
begin
  S := '';
  for I := 1 to Length(ATexto) do
    if (ATexto[I] in ['0'..'9']) then
      S := S + Copy(ATexto, I, 1);
  Result := S;
end;

function TfrmCobrancaEscritural.Replicar(ATexto: string; AVEzes: Integer): string;
var
  I: Integer;
begin
  Result := '';
  for I := 1 to AVEzes do
    Result := Result + ATexto;
end;
```

Em seguida abrimos o *ClientDataSet* que por sua vez executa a *Stored Procedure RetornarBoletos*. Caso o método *IsEmpty* do *cdsBoletos* retorne diferente de *True*, ou seja, se houverem dados no *Resultset* do *ClientDataSet* significa que existem boletos a serem gerados.

Verificamos então a existência do arquivo *Remessa.txt* no diretório da aplicação e o apagamos caso exista, assim criamos um arquivo novo no diretório.

**Nota:** Poderíamos criar uma função para retornar nomes aleatórios para que pudéssemos guardar o arquivo anterior, porém não há necessidade de se guardar os arquivos de remessa já enviados ao banco.

Os métodos *AssignFile* e *ReWrite*, criam e inicializam o arquivo *Remessa.txt* criando em memória uma instância deste arquivo para ser manipulado pelo sistema.

Um arquivo *CNAB* é composto por basicamente três seções:

- *Header*: Cabeçalho do arquivo contendo informações sobre o sacador;
- *Detail*: São os boletos em si. No padrão *CNAB 240* são duas linhas de *Detail* (“detalhe”) para cada boleto gerado;
- *Trailer*: Rodapé do arquivo, normalmente consta somente o número de registros gerados e a data;

A primeira parte do arquivo consiste em escrever o *Header* da remessa, ou seja, o cabeçalho do *CNAB*. Note que a última linha antes do fim do *Header*

usamos a função *WriteLn*. O *WriteLn* escreve o texto e quebra a linha enviando o cursor para a linha posterior ao passo que *Write* apenas escreve o texto sem mover o cursor.

Ao entrar no laço *while..do* estamos escrevendo o *Detail* do arquivo que é justamente o detalhe, ou detalhamento de cada boleto a ser gerado pela instituição financeira. Cada linha gerada é um boleto bancário que o banco se encarregará de gerar e enviar ao sacado. Após o término do laço escrevemos o *Trailer*, ou rodapé do arquivo remessa.

Esses são todos os passos efetuados no código visto agora a pouco. Como vimos não há segredos. Nas **Figuras 3 e 4** podemos ver o sistema em execução e uma prévia do arquivo carregado.

**Listagem 4.** Código do botão Gerar

```

procedure TfrmCobrancaEscritural.btnGerarClick(Sender: TObject);
var
  Arquivo: TextFile;
  Contador: Integer;
begin
  with dmdPrincipal, cdsBoletos do
  begin
    { Para uso da função de conversão
      DateTimeToSQLTimeStamp declare SQLTimSt }
    cdsBoletos.Params.ParamByName('DT_INICIO').
      AsSQLTimeStamp := DateTimeToSQLTimeStamp
      (dbeDtInicio.DateTime);
    cdsBoletos.Params.ParamByName('DT_FIM').
      AsSQLTimeStamp := DateTimeToSQLTimeStamp
      (dbeDtFim.DateTime);
    cdsBoletos.Open;
    if not cdsBoletos.IsEmpty then
    begin
      { Gera os arquivos para a cobrança escritural }
      if FileExists(ExtractFilePath(Application.
        ExeName)+'Remessa.txt') then
        DeleteFile(ExtractFilePath(Application.
          ExeName)+'Remessa.txt');
      { Cabeçalho / Header do Arquivo }
      Contador := 1;
      AssignFile(Arquivo, ExtractFilePath(
        Application.ExeName)+'Remessa.txt');
      Rewrite(Arquivo);
      Write (Arquivo, '01');
      Write (Arquivo, 'REMESSA');
      Write (Arquivo, '01');
      Write (Arquivo, 'COBRANCA');
      Write (Arquivo, edtAgencia.Text);
      Write (Arquivo, '00');
      Write (Arquivo, edtConta.Text);
      Write (Arquivo, Replicar(' ', 6));
      Write (Arquivo, 'CLUBEDELPHI LTDA. ');
      Write (Arquivo, '341');
      Write (Arquivo, 'BANCO ITAU SA ');
      Write (Arquivo, FormatDateTime('DDMMYYYY', Now));
      Write (Arquivo, Replicar(' ', 294));
      WriteLn(Arquivo, FormatFloat('000000', Contador));
      { Fim do Header do Arquivo }
      pgbProgresso.Position;
      pgbProgresso.Max := RecordCount;
      while not EOF do
      begin
        Inc(Contador);
        { Salva alguns dos principais dados no arquivo de remessa }
        Write (Arquivo, '1');
        Write (Arquivo, '02');
        Write (Arquivo, LimparString(FieldByName('CNPJ').AsString, '-'));
        Write (Arquivo, PadL(edtBanco.Text, 4));
        Write (Arquivo, '00');
        Write (Arquivo, PadL(LimparString(edtAgencia.Text, '-'), 6));
        Write (Arquivo, Replicar(' ', 4));
        Write (Arquivo, Replicar(' ', 4));
        Write (Arquivo, PadL('USO DA EMPRESA', 25));
        Write (Arquivo, PadL(Copy(FieldByName('CODIGO').AsString, 1, 8), 8));
        Write (Arquivo, Replicar('0', 12));
        Write (Arquivo, '109');
        Write (Arquivo, '00');
        Write (Arquivo, PadL(Copy(FieldByName('CODIGO').AsString, 1, 10), 10));
        Write (Arquivo, FormatDateTime('DDMMYY',
          FieldByName('DT_VECTO').AsDateTime));
        Write (Arquivo, FormatFloat('#####.##',
          FieldByName('VALOR_REAL').AsFloat));
        Write (Arquivo, '341');
        Write (Arquivo, PadL(LimparString(edtAgencia.Text, '-'), 6));
        Write (Arquivo, '01');
        Write (Arquivo, 'A');
        Write (Arquivo, FormatDateTime('DDMMYY',
          FieldByName('DT_CADASTRO').AsDateTime));
        Write (Arquivo, '02');
        Write (Arquivo, '09');
        Write (Arquivo, FormatFloat('#####.##', 0));
        Write (Arquivo, FormatDateTime('DDMMYY',
          FieldByName('DT_VECTO').AsDateTime));
        Write (Arquivo, FormatFloat('#####.##', 0));
        Write (Arquivo, FormatFloat('#####.##', 0));
        Write (Arquivo, '02');
        Write (Arquivo, PadL(LimparString(edtAgencia.Text, '-'), 6));
        Write (Arquivo, PadL(Copy(FieldByName('RAZAO')
          .AsString, 1, 30), 30));
        Write (Arquivo, Replicar(' ', 10));
        Write (Arquivo, PadL(Copy(FieldByName('ENDERECO').AsString, 1, 40), 40));
        Write (Arquivo, PadL(Copy(FieldByName('BAIRRO').AsString, 1, 12), 12));
        Write (Arquivo, LimparString(FieldByName('CEP').AsString, '-'), 8);
        Write (Arquivo, PadL(Copy(FieldByName('CIDADE').AsString, 1, 15), 15));
        Write (Arquivo, FieldByName('ESTADO').AsString);
        Write (Arquivo, Replicar(' ', 42));
        WriteLn(Arquivo, FormatFloat('000000', Contador));
        Next;
      pgbProgresso.StepIt;
    end;
    { Rodapé / Trailer do Arquivo }
    Inc(Contador);
    Write (Arquivo, '9');
    Write (Arquivo, Replicar(' ', 393));
    WriteLn(Arquivo, FormatFloat('000000', Contador));
    { Fim do Header do Arquivo }
    CloseFile(Arquivo);
    MessageDlg('Arquivo gerado com sucesso!', mtInformation, [mbOk], 0);
  end
  else
  begin
    MessageDlg('Nada selecionado!', mtWarning, [mbOk], 0);
    dbeDtInicio.SetFocus;
  end;
end;
end;
end;

```

## Últimas considerações

Além de todas essas informações que vimos há ainda que se pensar e lembrar que existem diversas carteiras de cobrança a serem utilizadas. As carteiras de cobranças são as regras básicas de como o banco fará a execução da cobrança junto ao cliente de seu usuário.

Também é importante salientar as formas de cobrança que são com e sem registro. Sem registro e com registro são na verdade algumas das modalidades de cobrança. Em nosso caso estamos utilizando cobrança com registro o que significa que estamos registrando previamente os dados de cobrança no banco de dados da instituição através do envio de arquivos texto.

Para mais detalhes consulte o gerente de sua conta corrente para que ele possa lhe passar todas as informações necessárias a respeito do assunto.

## Conclusão

Finalizamos a quarta parte deste minicurso gerando a cobrança escritural, CNAB, onde tivemos a oportunidade de ver como fazer na prática a geração de um arquivo remessa. Esses são apenas os passos mais importantes da geração, porém o desenvolvimento como um todo requer um grau de atenção ainda maior para que todas as regras impostas no layout sejam cumpridas.

Sempre que for preciso consulte o suporte técnico do banco para qual esteja desenvolvendo o CNAB para que não fiquem dúvidas durante a programação do processo.

Um forte abraço e até a próxima. ●



Figura 3. Arquivo gerado com sucesso

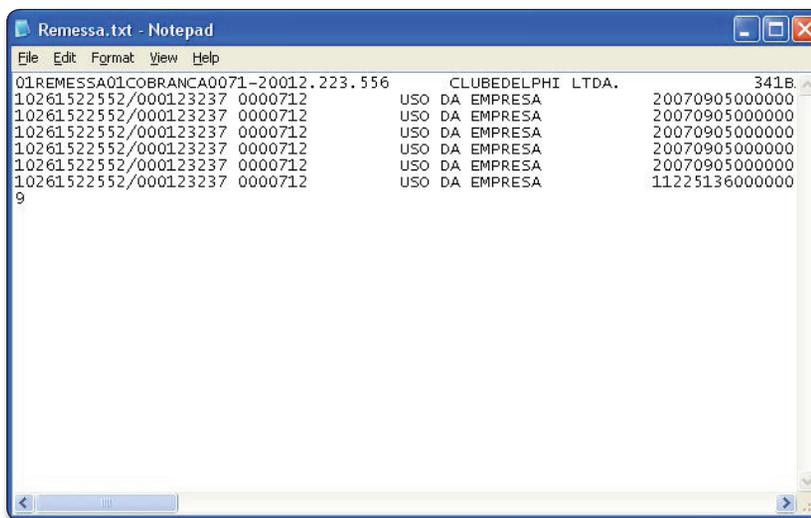


Figura 4. Remessa gerada com sucesso



NOVOS PLANOS

# Descubra porque Hospedix é a hospedagem preferida dos profissionais.

Plano **Profissional 1**



Windows Server 2003

1 Registro de domínio grátis\*  
3 Domínios por conta  
50 GB de transferência  
30 GB para e-mails com SSL  
1 GB de espaço  
ASP, ASP.NET 2.0 e PHP 5  
Access e MySQL 5  
SQL Server 2005 Express  
Painel de controle e webmail  
Editor de páginas on-line  
Anti-spam e Anti-vírus

R\$  
**25,90**  
por mês

## 30 DIAS GRÁTIS PARA EXPERIMENTAR

Ao assinar, digite o código de desconto abaixo exclusivo para leitores da .NET Magazine e ganhe o primeiro mês inteiramente grátis.

**RVSTNET**

## DOMÍNIO GRÁTIS ENQUANTO FOR CLIENTE

Só na Hospedix você tem registro de domínio (.com .net .org ou .info) grátis e isento de taxas anuais enquanto for cliente.

## PAINEL DE CONTROLE PLESK 8.1 PROFESSIONAL

Painel de controle completo onde você mesmo adiciona domínios, sub-domínios, cria contas de e-mail, e muito mais.

Transfira seu site hoje mesmo para a Hospedix e descubra porque somos a empresa de hospedagem preferida dos profissionais.

[www.hospedix.com.br](http://www.hospedix.com.br)



**HOSPEDIX**  
HOSPEDAGEM PROFESSIONAL



# Leitores de Código de Barras

Veja como sua aplicação pode trabalhar com esse padrão

O uso intenso de código de barras em documentos e produtos virou rotina nos mais diversos meios. Com isso diversas empresas desenvolveram leitores de código de barras atendendo a demanda de mercado. Há leitores para vários padrões. Nesse artigo veremos algumas características desses leitores, como se comportam, quais as diferenças e um pouco de conceito e história sobre Código de Barras.

Vamos montar duas pequenas aplicações que podem fazer uso destes equipamentos para agilizar determinados processos.

---

**Nota:** Para esse artigo é necessário ter instalado um leitor de documentos e um leitor de código de barras. Sugestão para compra: DR-20 e BR-210 respectivamente, ambos da empresa Bematech.

---

## Entendendo o projeto

Para entendermos como funcionam os leitores de código de barras e de documentos, criaremos duas aplicações simples que farão uso de técnicas, também simples, para leitura e localização das informações no banco de dados.

Optei por utilizar os equipamentos DR-20 (“Leitor de Documentos”) e BR-210 (“Leitor de código de barras”) da empresa Bematech, ambos de baixo custo.

Ambas aplicações receberão o valor lido do código de barras e então farão a procura da informação no banco de dados de exemplo. Faremos uma aplicação para entrada de produtos em estoque e outra para leitura de boletos bancários.

## Conceitos e diferenças entre os equipamentos

Uma das primeiras coisas que devemos entender antes de desenvolver qualquer tipo de aplicação que faça uso de código de barras é: como funcionam?



### Adriano Santos

(artes@doiscliques.com)

é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É colunista e membro da Comissão Editorial da revista ClubeDelphi. É também Editor Técnico da Revista ClubeDelphi. Mantém o blog Delphi to Delphi ([www.delphitodelphi.blogspot.com](http://www.delphitodelphi.blogspot.com)) com dicas, informações e tudo sobre desenvolvimento Delphi.

Segundo o verbete “Código de Barras” no site da *wikipedia.org*:

“O código de barras é uma representação gráfica de dados que podem ser numéricos ou alfanuméricos dependendo do tipo padrão empregado.

As linhas paralelas e verticais escuras e os espaços entre elas têm diferentes larguras em função das várias técnicas de codificação de dados empregada. A decodificação dos dados representados é realizada por um equipamento chamado “scanner”, dotado de uma fonte luminosa vermelha, que por contraste das barras e seus espaços converte a representação gráfica em “bits” (seqüências de 0 ou 1), compreendidos pelo computador, que por sua vez converte-os em letras ou números legíveis para nós.”.

Entendemos então, que a leitura do código somente pode ser efetuada através de um equipamento que segue as normas internacionais de decodificação. Esses equipamentos “escaneiam” as diversas barras e traduzem para uma leitura legível pelo ser humano através de caracteres numéricos e/ou alfanuméricos. E por falar em normas internacionais, um dos códigos de barras mais conhecido é o EAN/UPC, usado para identificar produtos no comércio. Esses códigos são impressos nas embalagens de todo tipo de produto.

Há diversos EAN’s disponíveis para os mais variados produtos, como por exemplo EAN8 e EAN13. Veja a seguir alguns tipos de código de barras.

Em nosso exemplo usaremos o código de barras numérico para identificação de boletos bancários e outro alfa numérico para entrada de dados no estoque.

## Entrada no estoque

Faremos uma aplicação simples, então mãos a obra. Entre no Delphi 7 e crie uma nova aplicação usando *File\New>Application*. Salve o formulário principal como “*uPrincipal.pas*” e mude seu *Name* para “*frmPrincipal*”. Modifique as propriedades *Caption*, *BorderIcons* e *BorderStyle* a seu gosto. Para este exemplo fiz as seguintes alterações: *Caption* igual a “*Entrada de Estoque*”, *biMaximize* igual a *False* e *bsSingle* respectivamente. Aproveite e salve também o projeto como “*Estoque.dpr*”. Desenhe uma janela como mostrado na **Figura 1**.

Criaremos uma tabela de estoque com os campos vistos na **Figura 1**, portanto insira um componente *ClientDataSet* (“*cdsEstoque*”) da paleta *Data Access* e dê um duplo clique de direita nele. Usando a opção *New Field* no menu de contexto do *Fields Editor*, crie cinco campos no *cdsEstoque* conforme a **Tabela 1**.

Como pode notar, o campo CODIGO, principal campo da tabela, recebe o código de barras do produto, é do tipo *String*. Isso é necessário, já que estamos usando o padrão alfanumérico para identificação do produto.

O correto em caso da identificação de produtos é fazer uso dos padrões EAN8 e EAN13 que são código de barras numéricos, porém é perfeitamente possível fazer uso de qualquer outro tipo padrão. Nesse caso estamos levando em consideração que nosso cliente deseja manipular seus produtos no estoque usando caracteres alfa, ou seja, letras e números.

Alguns editores de relatórios, como o

Report Builder e QuickReport por exemplo, são capazes de gerar etiquetas de código de barras automaticamente com base em um campo do banco de dados, desta forma ficaria fácil a criação e leitura de barras em um estoque.

Após a inclusão dos campos no *Fields Editor* do *cdsEstoque*, crie o *DataSet* em memória usando a opção *Create DataSet* e em seguida gere um arquivo XML com o nome de “*Estoque.xml*”. Para criação do arquivo XML use a opção *Save MyBase to Xml table* com o botão direito no *cdsEstoque*.

Insira um *DataSource* da paleta *Data Access* e faça as devidas ligações entre *DataSource*, *ClientDataSet* e *DbGrid*.

**Nota:** Os nomes dos componentes envolvidos na programação são: Edit (“*edtCodigoBarras*”) e SpinEdit (“*speQtde*”), Button “*Confirma*” (“*btnConfirma*”), Button “*Salvar*” (“*btnSalvar*”).

## Codificando

Você vai notar que a codificação desse exemplo não contém nenhuma programação em especial. Isso porque, diferentemente dos dois artigos anteriores, ECF e Impressão de Cheques, não há interatividade direta com o equipamento nem com arquivos DLL, já que o equipamento é apenas um periférico de entrada, ou seja, basta colocar o cursor do mouse no campo onde se deseja receber o código de

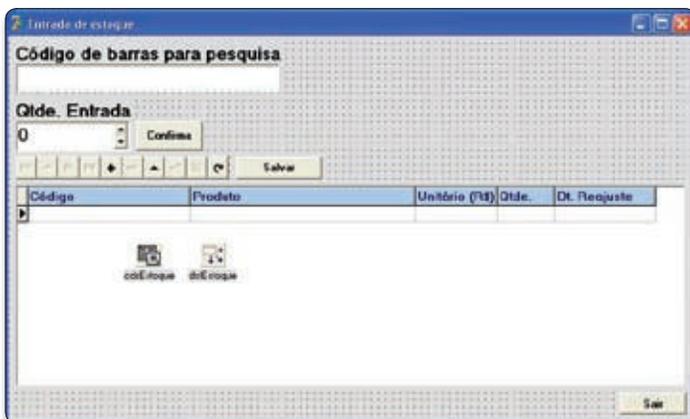


Figura 1. Exemplo de janela de estoque



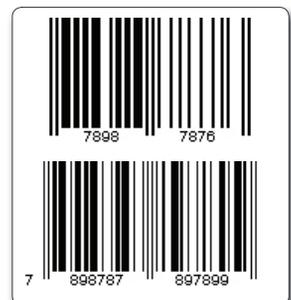
Código de Barras alfanumérico utilizado em diversas situações



Código de Barras número, utilizado para diversas finalidades como boletos bancários por exemplo.



ISBN, usado para identificação de obras literárias como livros, revistas etc.



EAN8 e EAN13 respectivamente são utilizados em identificações de produtos em todo o mundo.

barras, apontar o leitor para a etiqueta e aguardar a captura dos dados.

O leitor faz o "escaneamento" das barras decodificando os dados e enviando-os ao componente que espera o código. Em seguida um "enter" é simulado pelo próprio dispositivo. Desta forma, basta codificarmos o evento *OnExit* do *Edit* de maneira que ele busque no banco de dados o código de barras recebido.

Para testarmos inclua o código da **Listagem 1** ao evento em questão. A expli-

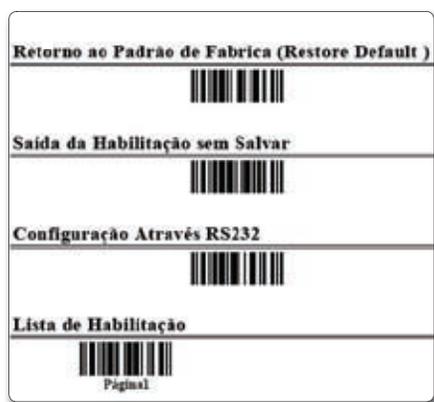


Figura 2. Exemplo de cartão de calibração

cação é simples: apenas fazemos uso do método *Locate* do *cdsEstoque* para então localizar o produto no campo CODIGO da tabela *Estoque*.

Caso encontre, a tabela entra em modo de edição e o foco é colocado no *SpinEdit* afim de aguardar a digitação da quantidade de entrada. E em caso negativo o sistema pergunta ao usuário se ele deseja fazer a inclusão do produto usando como código o próprio conteúdo do *Edit*.

Na **Listagem 2** encontramos o código dos botões *btnConfirmar* e *btnSalvar*. O botão Confirmar incrementa a quantidade do produto em estoque e o botão Salvar persiste os dados no *Estoque.xml*.

E por fim no evento *OnShow* do formulário apenas chama o método *LoadFromFile* do *cdsClientes* para abrir o arquivo XML e disponibilizar para uso.

```
cdsEstoque.LoadFromFile(ExtractFilePath
(Application.ExeName)+'Estoque.xml');
cdsEstoque.Open;
```

A leitura dos dados da etiqueta é feita pelo equipamento BR-210 da Bematech mencionado no início do artigo. Segun-

do o manual do dispositivo bem como o suporte técnico da empresa, um cartão com código de barras para calibração e programação do equipamento é enviado com a ele na compra.

Isso significa que para utilização do leitor é necessário apontá-lo para um dos código de barras impresso em seu cartão de calibração para só então utilizá-lo. Normalmente essa configuração é feita apenas uma vez. Um exemplo desse cartão pode ser visto na **Figura 2**. Veja o sistema em funcionamento na **Figura 3**.

## Testando

Da mesma forma que no exemplo anterior, veremos o que fazer para desenvolver uma aplicação capaz de ler o código de barras e localizar um documento na tabela de contas a pagar. Nesse caso temos um agravante: o campo não pode ser do tipo *Integer* em um *ClientDataSet* usando como tabela um XML, pois campos *Integer* guardam valores entre -2147483648..2147483647 e o não comportaria o código de barras de um documento que possui 34 números. Por esse motivo criaremos um campo BOLETO do tipo *String* no *ClientDataSet*.

Partiremos do princípio que nosso usuário quer ter a possibilidade de localizar em seu banco de dados boletos bancários previamente inseridos e necessita fazer a baixa dos mesmos. Ou ainda quer efetuar a baixa de boletos e duplicatas disparadas ao seus clientes por meio do contas a receber. Desta forma teria mais agilidade na hora de localizar e baixar o registro evitando ainda a baixa de

### Listagem 1. Evento OnExit

```
procedure TfrmPrincipal.edtCodigoBarrasExit(Sender: TObject);
begin
  if cdsEstoque.Locate('CODIGO', edtCodigoBarras.Text, []) then
  begin
    cdsEstoque.Edit;
    speQtde.SetFocus;
  end
  else if not (ActiveControl = btnSair) then
  begin
    if MessageDlg('Produto não encontrado no estoque!' + #13 +
      'Deseja incluir agora?', mtConfirmation, [mbYes, mbNo], 0) = mrYes then
    begin
      cdsEstoque.Append;
      cdsEstoque.FieldName('CODIGO').AsString := edtCodigoBarras.Text;
      speQtde.SetFocus;
    end;
  end;
end;
```

### Listagem 2. Código dos botões btnConfirmar e btnSalvar

```
procedure TfrmPrincipal.btnConfirmaClick(Sender: TObject);
begin
  if cdsEstoque.State in [dsEdit] then
  begin
    cdsEstoque.FieldName('QTDE').AsInteger :=
      cdsEstoque.FieldName('QTDE').AsInteger + speQtde.Value;
    cdsEstoque.Post;
  end;
end;
procedure TfrmPrincipal.btnSalvarClick(Sender: TObject);
begin
  cdsEstoque.SaveToFile(ExtractFilePath(Application.ExeName)+'Estoque.xml');
end;
```

Campo	Tipo/Tamanho
CODIGO	String (8)
PRODUTO	String(120)
UNITARIO	Float
QTDE	Integer
DATA_REAJUSTE	DateTime

Tabela 1. Tabela de estoque do sistema

Campo	Tipo/Tamanho
BOLETO	String(34)
DESCRICA0	String (50)
VALOR	Float
DT_VECTO	DateTime
BAIXA	Boolean

Tabela 2. Tabela de contas a pagar

ClubeDelphi PLUS

[www.devmedia.com.br/clubedelphi/portal.asp](http://www.devmedia.com.br/clubedelphi/portal.asp)

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Luciano Pimenta que mostra como criar relatórios com código de barras no QuickReport.

[www.devmedia.com.br/articles/viewcomp.asp?comp=3017&hl=c%F3digo%20de%20barras](http://www.devmedia.com.br/articles/viewcomp.asp?comp=3017&hl=c%F3digo%20de%20barras)

boletos equivocadamente.

Crie uma nova aplicação, salve-a como "Leitores.dpr" e desenhe-a semelhante a **Figura 4**. Salve o formulário principal como "uPrincipal.pas". Insira um *ClientDataSet* ("cdsContasPagar") e crie a estrutura de campos conforme a **Tabela 2**.

Crie a estrutura da tabela em memória usando *Create DataSet* e em seguida salve o arquivo XML usando a opção *Save to MyBase Xml table*. Crie o XML com o nome "ContasPagar.xml".

No evento *OnShow* do formulário inclua uma chamada ao método *LoadFromFile* do *cdsContasPagar* conforme a seguir:

```
cdsContasPagar.LoadFromFile
(ExtractFilePath(Application.ExeName)
+'ContasPagar.xml');
cdsContasPagar.Open;
```

Em seguida codifique o evento *OnExit* do *edtBoleto* de maneira que o conteúdo do *Edit* seja procurado no campo BOLETO da tabela *ContasPagar.xml*. **Listagem 3**.

Novamente note que não há segredos. Apenas utilizamos o método *Locate* do *DataSet* *cdsContasPagar* e então perguntando se o usuário deseja efetuar a baixa.

Em contato com a Bematech obtive a informação de que os leitores de documentos não lêem código de barras do tipo EAN, ou seja, somente código de barras de documentos como boletos de lojas e concessionárias de água, luz, gás, telefone e outras, tais como TIM, Telefônica, Sabesp e etc. Teste a aplicação e veja o resultado. **Figura 5**

## Conclusão

Com esse artigo finalizamos a série demonstrando como trabalhar com equipamentos de impressão tais como ECF, Impressoras de Cheques, Leitores de Código de Barras e Documentos. Vimos em todos os artigos como é fácil trabalhar com esses tipos de dispositivos e que não há segredos para se desenvolver um bom aplicativo.

Um forte abraço e até a próxima. ●



**Listagem 3.** Código do evento *OnExit* do Exemplo 2

```
procedure TForm1.edtBoletoExit(Sender: TObject);
begin
if cdsContasPagar.Locate('BOLETO', edtBoleto.Text, []) then
begin
if MessageDlg('Confirma baixa do boleto bancário?', mtConfirmation, [mbYes, mbNo], 0) = mrYes then
begin
cdsContasPagar.Edit;
cdsContasPagar.FieldByName('BAIXA').AsBoolean := True;
cdsContasPagar.Post;
cdsContasPagar.SaveToFile(ExtractFilePath(Application.ExeName)+'ContasPagar.xml');
end;
end;
end;
```



**Figura 3.** Sistema de estoque em funcionamento



**Figura 4.** Exemplo de Contas a Pagar



**Figura 5.** Contas a pagar em execução



# MVC (Model-View-Controller)

Aplice esse poderoso padrão em suas aplicações cliente/server

A grande maioria dos programadores Delphi desenvolve seus sistemas utilizando o modelo Cliente/Servidor, dividindo assim teoricamente seu sistema em duas camadas: interface e banco de dados.

É muito comum vermos no código de um formulário a construção de sentenças SQL e validação de regras de negócio, tudo muito RAD, já que é neste ponto que o Delphi se destaca. É possível construir rapidamente um cadastro e sua manutenção utilizando Delphi, porém um sistema completo não é composto apenas por cadastros. Muita lógica está envolvida.

Além disso existe outro fator que muitos não atentam: a vida do sistema. Por quanto tempo ele vai rodar. Se você quer desenvolver um sistema que tenha uma vida longa tem que pensar no tempo e custo das manutenções que ocorrerão.

Nessa hora que surgirão os problemas em se utilizar o RAD sem pensar na

arquitetura do sistema. É muito fácil construir uma aplicação arrastando componentes, gerar os eventos no formulário e dentro deles mesmo acessar banco de dados, SQL e etc. O difícil será reutilizar rotinas, regras de negócio e aplicação de testes, que juntos garantem uma facilidade na adequação dos sistemas para as necessidades dos usuários.

Agora você pode estar se perguntando, por onde começo? A primeira coisa a se fazer é pensar que seu software é composto por camadas e não apenas por formulários, relatórios e um banco de dados. Vamos explorar um pouco as camadas que um sistema pode ter.

## Camadas

Em um sistema com boa arquitetura encontramos algumas divisões lógicas que visam facilitar a manutenção e adição de novos recursos. Essas camadas podem ser vistas na **Figura 1**.

Para tornar a explicação sobre as ca-



**Paulo Roberto Quicoli**

([pauloquicoli@gmail.com](mailto:pauloquicoli@gmail.com))

é analista e programador da Control-M Informática. Trabalha com Delphi, desde sua primeira versão, e Firebird desenvolvendo aplicações cliente/servidor. Formado em Tecnologia de Processamento de dados pela FATEC, na cidade de Taquaritinga/SP

mas mais intuitiva quero que imagine seu sistema. Com toda certeza nele você possui um formulário de cadastro. O seu formulário representa a camada de apresentação do seu sistema, isto porque é nela que os dados são apresentados. Se seu sistema é cliente/servidor, a sua camada de domínio, de base dados e até mesmo serviços podem ser representadas pelos Data Modules existentes e seus componentes de acesso ao banco de dados. Por último temos a camada global que pode ser representada pelo seu conjunto de funções e procedimentos públicos.

Continuando com o paralelo a um sistema comum, desenvolvido por um programador Delphi, nos formulários nós fazemos acesso direto aos Data Modules, aos *ClientDataSets*, geramos SQL, tudo sem preocupação alguma, ou seja, nós misturamos o que seriam nossas camadas sendo que o correto é **separar** as camadas.

Quando essas camadas se misturam é dito que existe um acoplamento entre elas. No momento que você insere código SQL em evento *OnClick* de um botão o acoplamento foi realizado. Seu formulário que é sua camada de apresentação, está fazendo acesso direto aos recursos do que seria sua camada de dados. Isso impede a reutilização da lógica contida no formulário. Tudo isso dificulta a manutenção do seu sistema porque você terá SQL espalhado por todo código e não apenas centralizado em uma área.

Outra prática que é muito comum é de se criar os eventos e dentro deles escrever um código tão longo que para entender o que está sendo feito é quase como ler um livro. Consequentemente a rotina ali dentro poderia ser quebrada em classes ou funções.

Quando uma rotina faz mais do que foi projetada a fazer é dito que existe uma baixa coesão nela. A baixa coesão é outro problema para a boa manutenção do seu software. Então, como podemos separar as camadas sem, contudo uni-las? Pensando nessa solução alguns arquitetos de software criaram os **padrões de projeto**. Os padrões são de uma forma simplificada, como uma receita de bolo a ser seguida. Eles expõem o problema e mostram como solucioná-lo.



Figura 1. Camadas lógicas de um software

O problema apresentado neste artigo é como separar a interface do sistema do restante da aplicação. Como torná-la independente. A solução já existe, foi desenvolvida na forma de um padrão de projeto chamado **MVC – Model View Controller**, este é o foco do nosso artigo.

### Utilizando o padrão em um exemplo

Nosso primeiro aplicativo de exemplo será simples, porém irá mostrar como iniciar o desenvolvimento utilizando o padrão MVC e fazer uso de interfaces.

Constará apenas de uma tela principal que irá chamar um formulário que permite ao usuário realizar multiplicações. Aqui utilizo o Delphi 2007, contudo, o exemplo pode ser reproduzido também no Delphi 5, 6, 7, 2005 e 2006 sem problema algum e para iniciarmos, inicie um novo projeto Win32.

Para separar a camada de apresentação o MVC cria três partes distintas, o Model, a View e o Controller. Podemos defini-los da seguinte forma:

- Model – É a representação da informação que a aplicação utiliza, que no nosso exemplo é o cálculo de multiplicação e os números que serão calculados.
- View – É a representação gráfica do

Model. Por exemplo, um formulário expondo as informações de um cliente. Uma View está sempre ligada a um model e sabe como exibi-lo ao usuário. E para um determinado Model podem existir várias views. Além disso também pode ser composta por várias Views. Aplicando isso ao exemplo, a view é nosso formulário que irá permitir ao usuário digitar o valores e efetuar sua multiplicação.

Controller – É responsável em mapear as ações do usuário em ações do sistema, por exemplo se um botão ou item de menu é clicado é ele que deve definir como a aplicação responderá. No nosso exemplo é o controller que irá fazer com que ao clicar no botão “calcular” o cálculo seja feito.

### Colocando a mão na massa

Vamos iniciar criando as partes básicas definidas pelo padrão, o Model, a View e o Controller. Adicione uma unit ao projeto renomeando-a para “MVCInterfaces.pas”. Nela adicione as seguintes interfaces, conforme listagem **Listagem 1**.

Como já foi dito, o Model é nossa regra de negócio, ou seja, nosso cálculo de multiplicação. Existem algumas formas de se implementar o MVC, variando de acordo com o entendimento do desen-

### Listagem 1. MVC

```
type
  TModelChangedEvent = procedure of object;

  IModel = interface ['{FDF18F94-0430-4C48-BE64-F4516C4C1011}']
    procedure Initialize;
    function GetOnModelChanged: TModelChangedEvent;
    procedure SetOnModelChanged(value: TModelChangedEvent);
    property OnModelChanged: TModelChangedEvent read GetOnModelChanged write SetOnModelChanged;
  end;

  IView = interface ['{A1F411E9-294D-444D-9D5B-1D6AC9988819}']
    procedure Initialize;
  end;

  IController = interface ['{E4BD8853-F6C8-4BD4-B19D-D70B156BD712}']
    procedure Initialize;
  end;
```

### Listagem 2. Model

```
type
  TModelMultiply = class(TInterfacedObject, IModel)

  private
    FValue1: integer;
    FValue2: integer;
    FOnModelChanged: TModelChangedEvent;
    FSolution: integer;
    procedure SetValue1(const Value: integer);
    procedure SetValue2(const Value: integer);
    function GetOnModelChanged: TModelChangedEvent;
    procedure SetOnModelChanged(value: TModelChangedEvent);
    procedure Initialize;
    procedure SetSolution(const Value: integer);

  public
    constructor Create; reintroduce;
    function Multiply: integer;
    property Value1: integer read FValue1 write SetValue1;
    property Value2: integer read FValue2 write SetValue2;
    property Solution: integer read FSolution write SetSolution;
    property OnModelChanged: TModelChangedEvent read GetOnModelChanged write SetOnModelChanged;
  end;

implementation

{ TModelMultiply }

constructor TModelMultiply.Create;
begin
  Initialize;
end;

function TModelMultiply.GetOnModelChanged: TModelChangedEvent;
begin
  result := FOnModelChanged;
end;

procedure TModelMultiply.Initialize;
begin
  Value1 := 0;
  Value2 := 0;
  Solution := 0;
end;

function TModelMultiply.Multiply: integer;
begin
  result := Value1 * Value2;
  Solution := result;
  if Assigned(OnModelChanged) then OnModelChanged;
end;

procedure TModelMultiply.SetOnModelChanged(value: TModelChangedEvent);
begin
  FOnModelChanged := value;
end;

procedure TModelMultiply.SetSolution(const Value: integer);
begin
  FSolution := Value;
end;

procedure TModelMultiply.SetValue1(const Value: integer);
begin
  FValue1 := Value;
end;

procedure TModelMultiply.SetValue2(const Value: integer);
begin
  FValue2 := Value;
end;
```

volvedor. Em sua forma conceitual um Model possui uma referência à View. Neste exemplo não vejo a necessidade disso, já que o Model aqui é a regra de negócio em si.

Em sistemas mais complexos isso pode até fazer sentido por que o Model poderá encapsular várias regras e será um encapsulamento para elas. O que quero dizer é que a parte chamada Model do MVC nem sempre é diretamente uma classe da nossa camada de domínio. Isso significa que a classe *TAluno* pode conter rotinas e/ou outras classes que têm relação com *TAluno*, como por exemplo, *TCurso* e métodos para matrícula.

Adicione uma nova unit ao projeto, salvando-a como "Multiply.pas" e adicione ao uses da interface a *Unit MVCInterfaces*. Crie uma nova classe conforme **Listagem 2**.

Veja que a classe *TModelMultiply* contém duas propriedades que armazenarão os números digitados pelo usuário e disponibiliza o método *Multiply* que realiza a multiplicação. Vamos agora criar nossa View correspondente.

A finalidade da View é saber como apresentar ao usuário os dados contidos no Model. Adicione um novo formulário ao projeto e dê a ele o nome de "View1". Salve sua unit como "ViewForm1.pas" e adicione ao uses da seção interface as *Units MVCInterfaces* e *Multiply*. Construa o formulário conforme **Figura 2**.

Nesse formulário vamos criar um evento que será utilizado pelo Controller. Esse evento representa o clique do botão calcular. Lembre-se que é o controller que transfere as ações do usuário para o Model, portanto não é correto fazer o formulário chamar diretamente nosso cálculo, que é o Model. Vamos apenas adicionar uma referência ao Model, veja a **Listagem 3**.

Observe na **Listagem 4** a implementação dos métodos *Initialize* e *Model*

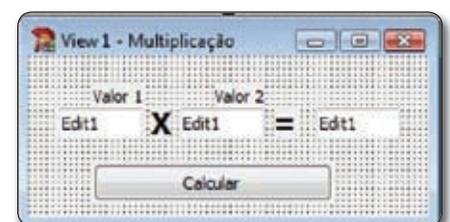


Figura 2. View

*Changed*. O *Initialize* é responsável por inicializar a *View*, trazendo para ela o estado inicial do *Model*. O *ModelChanged* por sua vez responde ao disparo do evento *OnModelChanged* realizado pelo *Model* e lê suas propriedades para atualizar sua representação. Veja também que no *OnClick* do botão disparamos o evento *DoCalc*.

O evento *DoCalc* será implementado pelo *Controller*. Adicione uma nova *Unit* e a salve-a como "FormController.pas", inclua no *uses* as *units* *Multiply*, *MVCInterfaces* e *ViewForm1*. Implemente o controle como na **Listagem 5**.

Vamos entender o que o *controller* está fazendo. De acordo com a forma conceitual o *Controller* precisa ter uma referência à *View* e ao *Model*. Temos isso representado pela variáveis *FView* e *FModel*. Temos também uma *procedure* *Calc* que executa o método *Multiply* do *Model*. Essa *procedure* é que irá responder ao evento *DoCalc* da *View*, assim ao clicar no botão *Calcular* da *View*, o *controller* responderá, criando assim o comportamento da *View*. Tudo isso é implementado no método *Initialize*. É nele que estamos atando cada parte que compõe a tríade MVC.

## Executando

No formulário principal da aplicação adicione uma referência à *unit* *FormController*. Adicione também um botão, e no seu evento *OnClick* vamos chamar o *controller*. Veja **Listagem 6** o código e na **Figura 3** o resultado.

## Complicou?

A primeira vista pode parecer complicada, mas não é. O segredo é ter em mente o que cada parte do MVC deve fazer e como se dá a comunicação entre elas. Na **Figura 4** temos uma ilustração dessa comunicação.

Basicamente o *Model* aponta para a *View* o que permite enviar a ela notificações de mudança, no nosso exemplo isso está implementado pela definição do evento *OnModelChanged*. O *Model* não deve saber nada sobre a *View*, portanto esta referência é feita através de uma superclasse da *View* que expõe meios de ser notificada. Veja que nossa *View* (formulário) implementa o evento anteriormente definido pelo *Model* e o utiliza para receber a notificação de

### Listagem 3. Definição da View

```
type
  TCalcEvent = procedure of object;
  TView1 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Label2: TLabel;
    Edit2: TEdit;
    Label3: TLabel;
    Label4: TLabel;
    Edit3: TEdit;
    btnCalc: TButton;
    procedure btnCalcClick(Sender: TObject);
  private
    FModel: TModelMultiply;
    FDoCalc: TCalcEvent;
    procedure SetModel(const Value: TModelMultiply);
    { Private declarations }
    procedure SetDoCalc(const Value: TCalcEvent);
  public
    { Public declarations }
    procedure Initialize;
    procedure ModelChanged;
    property Model: TModelMultiply read FModel write SetModel;
    property DoCalc: TCalcEvent read FDoCalc write SetDoCalc;
  end;
```

### Listagem 4. Atualizando a View

```
procedure TView1.btnCalcClick(Sender: TObject);
begin
  Model.Value1 := StrToInt(Edit1.Text);
  Model.Value2 := StrToInt(Edit2.Text);
  if Assigned(DoCalc) then DoCalc;
end;
procedure TView1.Initialize;
begin
  Edit1.Text := IntToStr(Model.Value1);
  Edit2.Text := IntToStr(Model.Value2);
  Edit3.Text := IntToStr(Model.Solution);
end;
procedure TView1.ModelChanged;
begin
  Initialize;
end;
procedure TView1.SetDoCalc(const Value: TCalcEvent);
begin
  FDoCalc := Value;
end;
procedure TView1.SetModel(const Value: TModelMultiply);
begin
  FModel := Value;
end;
```

### Listagem 5. Controller

```
type
  TFormController = class(TInterfacedObject, IController)
  private
    FView: TView1;
    FModel: TModelMultiply;
  public
    constructor Create; reintroduce;
    destructor Destroy; override;
    procedure Initialize;
    procedure Calc;
  end;
implementation
  { TFormController }
  procedure TFormController.Calc;
  begin
    FModel.Multiply;
  end;
  constructor TFormController.Create;
  begin
    FModel := TModelMultiply.Create;
    FView := TView1.Create(nil);
  end;
  destructor TFormController.Destroy;
  begin
    FView.Free;
    inherited;
  end;
  procedure TFormController.Initialize;
  begin
    FView.Model := FModel;
    FModel.OnModelChanged := FView.ModelChanged;
    FView.DoCalc := Calc;
    FView.Initialize;
    FView.ShowModal;
  end;
```

atualização do mesmo.

Por outro lado, a View está diretamente ligada a um Model e sabe exatamente os detalhes do Model, de tal forma que seus atributos possam ser acessados, isso pode ser visto no acesso que a View faz às propriedades *Value1* e *Value2* do Model.

Além disso possui também uma referência ao Controller que deve estar ligada a uma classe base do Controller. Isso, para permitir que um controller possa ser trocado. Este por sua vez possui referências diretas ao Model e à View porque é ele que define o comportamento

geral. Observe a existência de dois campos privados, respectivamente dos tipos View e Model.

### Um exemplo mais real

Ainda ficou complicado não é? Programador Delphi não se dá por satisfeito até ver sua comum pergunta respondida: "Mas o que fazer com minhas queries e tabelas?". Para provar que não é tão complicado retirar do seu formulário todo SQL que está lá vamos transformar um aplicativo Cliente/Servidor comum, em um aplicativo Cliente/Servidor com a

interface separada das regras de negócio e do banco de dados.

Neste exemplo não seremos puristas quanto ao uso da orientação a objetos, mas vamos utilizar o que há de melhor no Delphi, RAD e a orientação a objetos, para que você, programador, possa ir se acostumando e ver que vale a pena aplicar tudo isso em seus projetos.

### O tradicional

Vamos criar uma agenda de telefone. Primeiramente vamos criá-la de forma tradicional e posteriormente vamos aproveitar o que já tem e adaptar um MVC. Crie um banco de dados Firebird contendo a seguinte tabela, vista na **Figura 5**.

Inicie agora uma nova aplicação Win32, adicione um Data Module e crie uma conexão DBX com o banco de dados criado. Adicione ao Data Module um *TSQLQuery* que acesse a tabela CONTATO e ligue a esse *TSQLQuery* um *ClientDataSet*. Vamos utilizar esse *ClientDataSet* em nossa aplicação. Configure o formulário principal conforme **Figura 6**. Adicione um segundo formulário e o deixe como na **Figura 7**.

Não vou entrar em detalhes de como ligar os controles *Dataware* ao *ClientDataset*, porque como disse, esta primeira parte do exemplo você usa no seu dia-a-dia. Observe a **Listagem 7**, ela contém a lógica do sistema, SQL e validação. Não existiria nada de errado com elas desde que não estivessem dentro dos formulários. Essas listagens mostram o que a grande maioria dos desenvolvedores Delphi faz de forma natural.

### Aplicando o MVC sem perder o RAD

Como já mencionei anteriormente não serei um "purista", quero aproveitar o que o Delphi oferece, contudo, aplicando boas práticas OO. Para relembrar, o MVC é "Model-View-Controller". Aproveitando a estrutura existente a View são os nossos formulários, o Model são as rotinas de Inclusão, Edição, Exclusão e Localização que podem ser encapsuladas em uma classe que as represente, por exemplo *ManterContato*. Os dados dos contatos continuarão sendo acessados pela estrutura *SQLQuery*, *DataSetProvider* e *ClientDataSet*. O Controller se encarregará de chamar os métodos do Model de acordo com as ações do usuário em cima da View.

Listagem 6. Executando o controller

```
procedure TMainF.Button1Click(Sender: TObject);
var
  Controller: TFormController;
begin
  Controller := TFormController.Create;
  Controller.Initialize;
end;
```

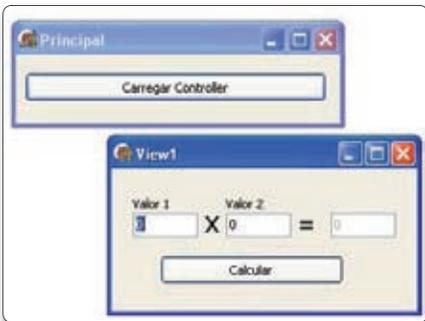


Figura 3. MVC em ação

CONTATO	
ID_CONTATO	INTEGER
NOME	VARCHAR(100)
TELEFONE	VARCHAR(15)

Figura 5. Tabela CONTATO

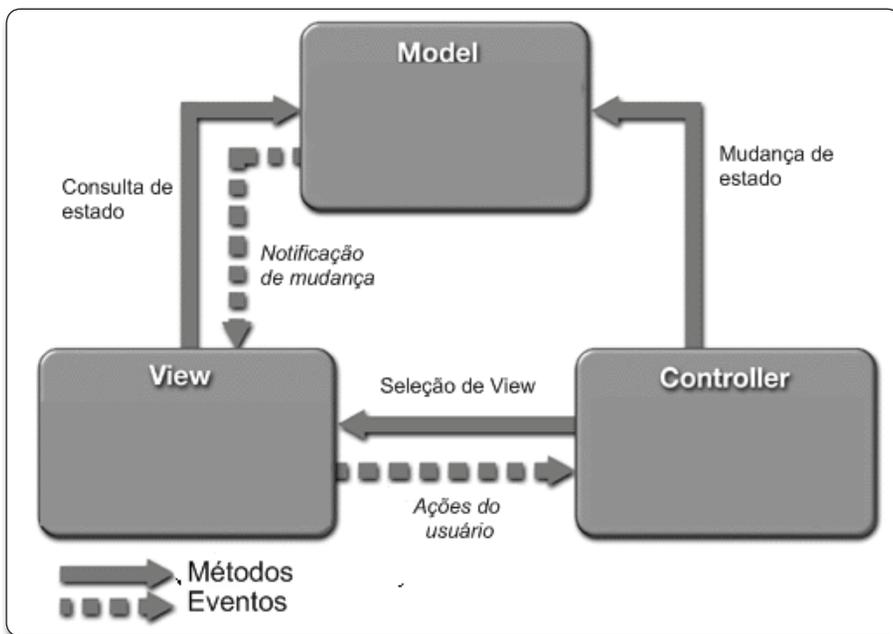


Figura 4. A comunicação dentro do MVC

Semelhantemente ao primeiro exemplo construído vamos adicionar uma *Unit* ao projeto e chamá-la de "MVCInterfaces.pas". Adicione a ela o código da **Listagem 8**.

Agora vamos utilizar essas interfaces para implementar a lógica do nosso sistema de Agenda. Para começar vamos criar o Model. Adicione uma nova *Unit*, salve-a como "ModelAgenda.pas". A ela adicione uma referência na seção *uses* à *unit MVCInterface* e ao *Data Module* da aplicação, lembre-se que vamos utilizar a estrutura já criada. Crie a classe *TManterContato* conforme visto na **Listagem 9**.

Veja que nossa classe *TManterContato*, que é nosso Model, centraliza em si a manipulação de SQL, o CRUD e sua validação. Também criamos duas exceções que são utilizadas caso algo de errado aconteça. Então você pode se perguntar: Porque não exibimos uma mensagem ao usuário ao invés de levantar a exceção? A razão é bem simples, a função de interação com o usuário é da *View* e não do Model. O Model nem sabe como seus dados serão exibidos, por isso ele apenas notifica que algo está errado. Isso garante a reutilização desse Model.

Continuando, temos o Controller e é ele que irá chamar os métodos do Model. Na **Listagem 10** temos a implementação completa do Controller, adicione um nova *Unit* e adicione o código.

Vamos agora ajustar nosso formulário principal para agir como uma *View*. Adicione na seção *Uses* da seção *Interface* as *units MVCInterfaces, ControllerAgenda, e ModelAgenda*. Na declaração do formulário indique que ele implementa *IView*, conforme abaixo:

```
TPrincF = class(TForm, IView)
```

E por implementar *IView*, ele precisa ter referências ao Model, ao Controller e deve implementar o método *Initialize* de *IView*. Veja isso na **Listagem 11**.

Com isso temos nossa *View* pronta. Agora vamos remover as chamadas aos componentes de acesso e adicionar as chamadas ao controller. A primeira a ser removida é a que consta no evento *OnShow* do formulário principal. Substitua o código a seguir:

```
Self.Initialize;
```

Veja que o *Initialize* da *View* é chamado

e este por sua vez realiza a ligação entre os componentes do MVC e inicia o Controller. Depois, vamos substituir o evento *OnKeyUp* do *Edit* de busca pelo código da **Listagem 12**.

Observe o código. Veja que a chamada ao controller está dentro de um bloco *try..except*. Lembre-se que o Controller chama o método *LocalizarPorNome* de *TManterContato*, que é nosso Model, portanto a regra para validar se a consulta pode ser feita ou não está contida no Model e não no formulário. Vamos substituir agora o código do botão excluir para o código da **Listagem 13**.

Para utilizar agora o Controller para fazer a inserção e edição de um contato temos que fazer uma alteração também no formulário de edição (**Figura 7**). A primeira coisa a se fazer é adicionar no *uses* da seção *Interface* uma referência às

*Units ControllerAgenda e ModelAgenda*, depois vamos criar uma propriedade publica do tipo *IAgendaController*, conforme a seguir:

```
...
public
  { Public declarations }
  property Controller: IAgendaController
    read FController write SetController;
...
```

Aí então podemos substituir os códigos do botão *Salvar* e *Cancelar* pelos da **Listagem 14**.

Com o nosso formulário de edição pronto, podemos agora terminar a alteração no formulário principal. Vamos remover os códigos dos botões *Inserir* e *Editar* pelo código da **Listagem 15**.

Antes de executar a aplicação, dê uma olhada no código do formulário principal por completo. Veja que não temos no código nenhuma chamada a *ClientDataSet* ou

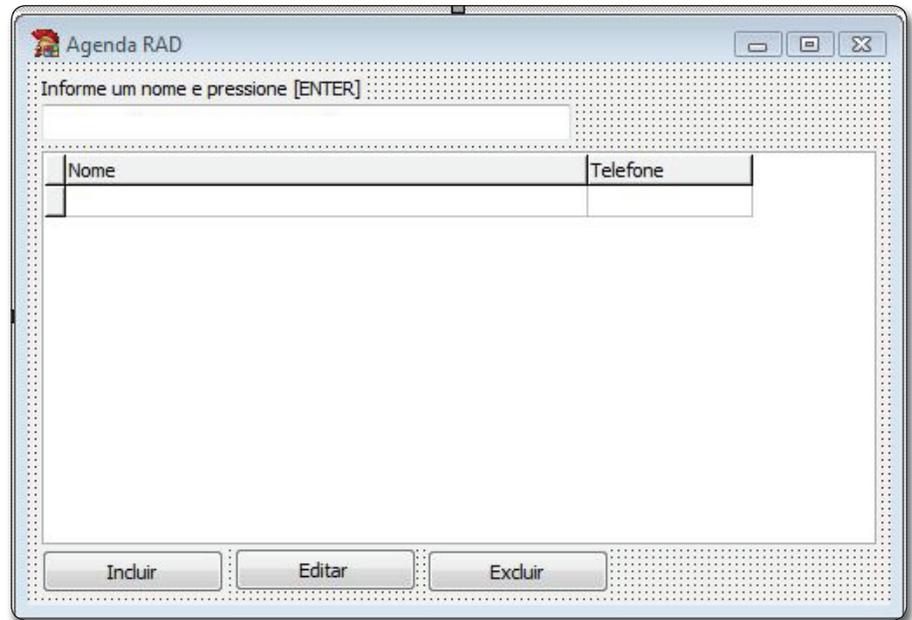


Figura 6. Formulário principal

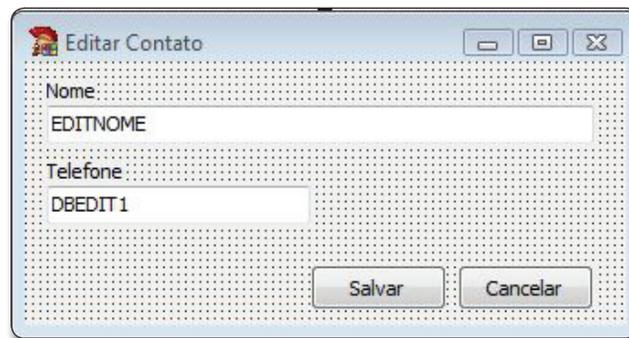


Figura 7. Formulário de edição



### Listagem 7. Evento onKeyUp do Edit de busca

```
procedure TPrincF.editBuscaKeyUp(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if (Key = VK_RETURN) then
  begin
    begin
      if (length(Trim(EditBusca.Text)) >= 3) then
      begin
        dm.cdsContatos.Close;
        dm.qryContatos.SQL.Clear;
        dm.qryContatos.SQL.Add('SELECT * FROM CONTATO');
        dm.qryContatos.SQL.Add('WHERE NOME STARTING WITH ' + QuotedStr(editBusca.Text));
        dm.qryContatos.SQL.Add('ORDER BY NOME');
        dm.cdsContatos.Open;
      end
    end
  else
  begin
    if Trim(EditBusca.Text) = '' then
    begin
      dm.cdsContatos.Close;
      dm.qryContatos.SQL.Clear;
      dm.qryContatos.SQL.Add('SELECT * FROM CONTATO');
      dm.qryContatos.SQL.Add('ORDER BY NOME');
      dm.cdsContatos.Open;
    end
  else
  begin
    ShowMessage('Para realizar a busca você deve informar no mínimo 3 caracteres');
    editBusca.SetFocus;
  end;
end;
end;
end;

procedure TPrincF.btIncluirClick(Sender: TObject);
begin
  dm.cdsContatos.Insert;
  EditarContatoF := TEditarContatoF.Create(nil);
  EditarContatoF.ShowModal;
  FreeAndNil(EditarContatoF);
end;

procedure TPrincF.btEditarClick(Sender: TObject);
begin
  dm.cdsContatos.Edit;
  EditarContatoF := TEditarContatoF.Create(nil);
  EditarContatoF.ShowModal;
  FreeAndNil(EditarContatoF);
end;

procedure TPrincF.btExcluirClick(Sender: TObject);
begin
  if Application.MessageBox('Confirma exclusão de contato?', 'Atenção', MB_YESNO + MB_ICONQUESTION +
  MB_DEFBUTTON2)=ID_YES then
  begin
    dm.cdsContatos.Delete;
    dm.cdsContatos.ApplyUpdates(-1);
  end;
end;

procedure TEditarContatoF.btSalvarClick(Sender: TObject);
begin
  if trim(DBEdit1.Text) = '' then
  begin
    ShowMessage('Nome de contato inválido');
    exit;
  end;
  dm.cdsContatos.Post;
  dm.cdsContatos.ApplyUpdates(-1);
  Close;
end;

procedure TEditarContatoF.btCancelarClick(Sender:
TObject);
begin
  dm.cdsContatos.CancelUpdates;
  Close;
end;
end;
```

### Listagem 8. Interfaces básicas do MVC

```
unit MVCInterfaces;
interface
type
  IModel = interface ['{FDF18F94-0430-4C48-BE64-F4516C4C1011}']
    procedure Initialize;
  end;
  IView = interface ['{A1F411E9-294D-444D-9D5B-1D6AC9988819}']
    procedure Initialize;
  end;
  IController = interface ['{E4BD8853-F6C8-4BD4-B19D-D70B156BD712}']
    procedure Initialize;
  end;
implementation
end.
```

instrução SQL. Execute agora e você verá tudo funcionando. Insira um *breakpoint* por exemplo na chamada ao *Controller*. *LocalizarPorNome* e vá debugando. Você irá perceber como se dá a passagem de chamadas de um item a outro do MVC. Na **Figura 8** temos uma simplificação desse processo.

## Conclusão

Neste artigo vimos que podemos utilizar o RAD e utilizar ao mesmo tempo boas práticas, que neste artigo é a separação da interface (formulários) com o restante do sistema. Utilizamos uma adaptação do modelo MVC para a realidade dos programadores Delphi, conseguimos retirar dos formulários as chamadas SQL e chamadas diretas a *ClientDataSet* passando isso ao Controller que por sua vez passa ao Model.

Dessa forma obtivemos um formulário que não possui regra de negócio e com isso, pode ser substituído por outro formulário qualquer, desde que implemente a interface *IView*. As regras do negócio ficaram encapsuladas em classes e podem ser reutilizadas em outras variações da agenda, por exemplo uma agenda pra web.

Ao utilizar os *ClientDataSets* como chamada de dados ganhamos em agilidade na montagem do formulário e no acesso ao banco de dados, isto porque nos prendemos em melhorar um sistema já pronto. Com a separação do formulário podemos agora aplicar testes unitários ao controller e model, coisa que é impossível de se fazer programando do "jeitão" comum.

Sem muito esforço acrescentamos qualidade ao nosso software e ganhamos conhecimento, lembrando que aplicar essa separação de formulários é apenas um passo para se conseguir uma separação ideal das camadas de um sistema, porém já temos um começo utilizando o RAD e OO. ●



**Listagem 9.** Manter contatos

```

uses MVCInterfaces, dmU, SysUtils;

type
  IManterContato = interface(IModel)
    ['{01AFE56E-1787-4B79-9B48-A018B2859B0C}']
    procedure LocalizarPorNome(Nome: string);
    procedure Inserir;
    procedure Excluir;
    procedure Editar;
    procedure Salvar;
    procedure Cancelar;
  end;

  TManterContato = class(TInterfacedObject, IManterContato)
  public
    procedure LocalizarPorNome(Nome: string);
    procedure Inserir;
    procedure Excluir;
    procedure Editar;
    procedure Salvar;
    procedure Cancelar;
    procedure Initialize;
  end;

  TQuantidadeCaracteresInvalidaEx = class(Exception);
  TDadosInvalidosEx = class(Exception);

implementation
  { TManterContato }

  procedure TManterContato.Cancelar;
  begin
    dm.cdsContatos.CancelUpdates;
  end;

  procedure TManterContato.Editar;
  begin
    dm.cdsContatos.Edit;
  end;

  procedure TManterContato.Excluir;
  begin
    dm.cdsContatos.Delete;
    dm.cdsContatos.ApplyUpdates(-1);
  end;

  procedure TManterContato.Initialize;
  begin
    dm.cdsContatos.Close;
    dm.qryContatos.SQL.Clear;
    dm.qryContatos.SQL.Add('SELECT * FROM CONTATO ORDER BY NOME');
    dm.cdsContatos.Open;
  end;

  procedure TManterContato.Inserir;
  begin
    dm.cdsContatos.Insert;
  end;

  procedure TManterContato.LocalizarPorNome(Nome: string);
  begin
    if (length(Trim(Nome)) >= 3) then
    begin
      dm.cdsContatos.Close;
      dm.qryContatos.SQL.Clear;
      dm.qryContatos.SQL.Add('SELECT * FROM CONTATO');
      dm.qryContatos.SQL.Add('WHERE NOME STARTING WITH ' +
        QuotedStr(Nome));
      dm.qryContatos.SQL.Add('ORDER BY NOME');
      dm.cdsContatos.Open;
    end
    else
    begin
      if Trim(Nome) = '' then
      begin
        Self.Initialize;
      end
      else
      begin
        raise TQuantidadeCaracteresInvalidaEx.Create(
          'Para realizar a busca você deve informar no mínimo 3
          caracteres');
      end;
    end;
  end;

  procedure TManterContato.Salvar;
  begin
    if Trim(dm.cdsContatos.NOME.AsString) = '' then
    begin
      raise TDadosInvalidosEx.Create('Nome de contato inválido');
    end;
    dm.cdsContatos.Post;
    dm.cdsContatos.ApplyUpdates(-1);
  end;

```

**Listagem 10.** Controller

```

interface
uses MVCInterfaces, ModelAgenda, SysUtils;
type
  IAgendaController = interface(IController)
    ['{E52C60DD-1B1B-4272-A3DF-72610E9BA2F7}']
    procedure LocalizarPorNome(Nome: string);
    procedure Inserir;
    procedure Excluir;
    procedure Editar;
    procedure Salvar;
    procedure Cancelar;
    procedure Initialize;
    procedure SetModel(const Value: IManterContato);
    procedure SetView(const Value: IView);
    function GetView: IView;
    function GetModel: IManterContato;
    property Model: IManterContato read GetModel write SetModel;
    property View: IView read GetView write SetView;
  end;

  TAgendaController = class(TInterfacedObject, IAgendaController)
  private
    FModel: IManterContato;
    FView: IView;
    procedure SetModel(const Value: IManterContato);
    procedure SetView(const Value: IView);
    function GetView: IView;
    function GetModel: IManterContato;
  public
    procedure LocalizarPorNome(Nome: string);
    procedure Inserir;
    procedure Excluir;
    procedure Editar;
    procedure Salvar;
    procedure Cancelar;
    procedure Initialize;
    property Model: IManterContato read GetModel write SetModel;
    property View: IView read GetView write SetView;
  end;

implementation
  { TAgendaController }
  procedure TAgendaController.Cancelar;
  begin
    FModel.Cancelar;
  end;
  procedure TAgendaController.Editar;
  begin
    FModel.Editar;
  end;
  procedure TAgendaController.Excluir;
  begin
    FModel.Excluir;
  end;
  function TAgendaController.GetModel: IManterContato;
  begin
    result:= FModel;
  end;
  function TAgendaController.GetView: IView;
  begin
    result := FView;
  end;
  procedure TAgendaController.Initialize;
  begin
    FModel.Initialize;
  end;
  procedure TAgendaController.Inserir;
  begin
    FModel.Inserir;
  end;
  procedure TAgendaController.LocalizarPorNome(Nome: string);
  begin
    FModel.LocalizarPorNome(Nome);
  end;
  procedure TAgendaController.Salvar;
  begin
    FModel.Salvar;
  end;
  procedure TAgendaController.SetModel(const Value: IManterContato);
  begin
    FModel := Value;
  end;
  procedure TAgendaController.SetView(const Value: IView);
  begin
    FView := Value;
  end;

```

### Listagem 11. Implementando a View

```
type
  TPrincF = class(TForm, IView)

  private
    FController: IAgenController;
    { Private declarations }
    procedure Initialize;
    procedure SetController(const Value: IAgenController);

  public
    { Public declarations }
    property Controller: IAgenController read FController;

write SetController;
end;

procedure TPrincF.Initialize;
var
  TModel: TManterContato;
begin
  Controller := TAgendaController.Create;
  TModel := TManterContato.Create;
  Controller.Model := TModel;
  Controller.View := Self;
  Controller.Initialize;
end;
```

### Listagem 12. Utilizando o controller para realizar a busca

```
procedure TPrincF.editBuscaKeyUp(Sender: TObject; var Key: Word;
Shift: TShiftState);
begin
  if (Key = VK_RETURN) then
  begin
    try
      Controller.LocalizarPorNome(editBusca.Text);
    except
      on e: TQuantidadeCaracteresInvalidaEx do
        begin
          ShowMessage(e.Message);
          editBusca.SetFocus;
        end;
      end;
  end;
end;
```

### Listagem 13. Utilizando o controller para realizar uma exclusão

```
procedure TPrincF.btExcluirClick(Sender: TObject);
begin
  if Application.MessageBox(
    'Confirma exclusão de contato?', 'Atenção',
    MB_YESNO + MB_ICONQUESTION + MB_DEFBUTTON2) = ID_YES then
  begin
    Controller.Excluir;
  end;
end;
```

### Listagem 14. Utilizando Controller para Salvar e Cancelar

```
procedure TEditarContatoF.btCancelarClick(Sender: TObject);
begin
  Controller.Cancelar;
  Close;
end;
procedure TEditarContatoF.btSalvarClick(Sender: TObject);
begin
  try
    Controller.Salvar;
    Close;
  except
    on e: TDadosInvalidosEx do
      begin
        ShowMessage('Nome de contato inválido');
        exit;
      end;
  end;
end;
```

### Listagem 15. Passando o controller para o formulário de edição

```
procedure TPrincF.btIncluirClick(Sender: TObject);
begin
  Controller.Inserir;
  EditarContatoF := TEditarContatoF.Create(nil);
  EditarContatoF.Controller := Self.Controller;
  EditarContatoF.ShowModal;
  FreeAndNil(EditarContatoF);
end;
procedure TPrincF.btEditarClick(Sender: TObject);
begin
  Controller.Editar;
  EditarContatoF := TEditarContatoF.Create(nil);
  EditarContatoF.Controller := Self.Controller;
  EditarContatoF.ShowModal;
  FreeAndNil(EditarContatoF);
end;
```

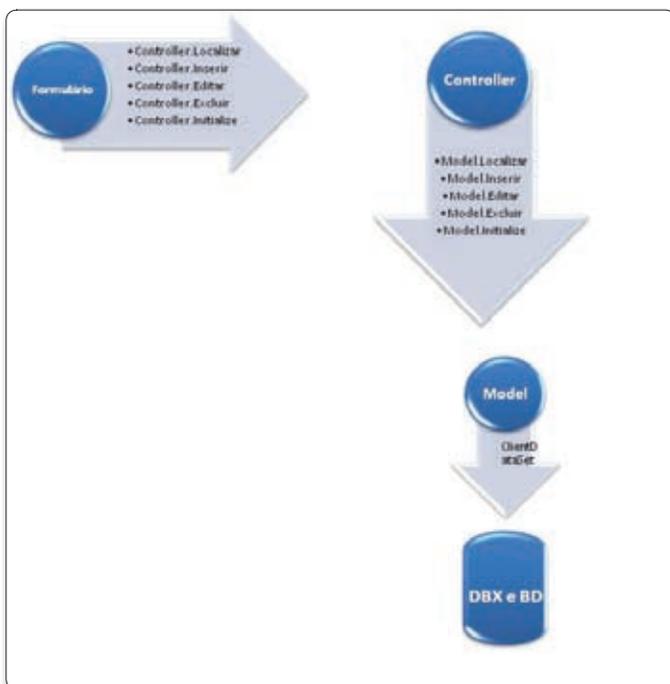
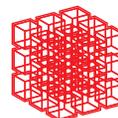


Figura 8. Comunicação do MVC

Unimos a inovação e excelência da **Alog** com  
o pioneirismo da **.comDomínio** para criar a  
**Alog Data Centers do Brasil:**  
**LÍDER EM HOSTING GERENCIADO®**



**.comDomínio**  
DATA CENTERS



## CONHEÇA OS NÚMEROS DESTA LIDERANÇA

2 Data Centers de Classe Mundial  
200 Profissionais Dedicados  
800 Clientes Corporativos  
3.000 Servidores Hospedados  
8.000 m<sup>2</sup> de Área Construída

[www.alog.com.br](http://www.alog.com.br) | 0800 282 3330



Hosting Gerenciado® | Colocation | Contingência | Email Corporativo | Conectividade | Serviços Profissionais

## Short-term interest rates

3-month rates, in % p.a.



## Gráficos

### Dicas práticas para a criação de gráficos em aplicações

Criado por Descartes para desenhar um raciocínio, a utilização de representação por Gráficos (gráficos), possibilita transmitir um significado de planilhas ou tabelas complexas de uma forma mais eficiente e simples. Em nossos sistemas, com informações armazenadas em banco de dados, a necessidade de mostrar as informações de uma forma de fácil compreensão muitas vezes nos obriga a utilizarmos esses recursos.

Baseados nessa metodologia veremos no decorrer deste artigo como construir gráficos a partir de informações armazenadas em um banco de dados. Utilizaremos o Delphi 7, que fará acesso através dos componentes da paleta dbExpress à base de dados *Employee.fdb*, que acompanha a instalação do Firebird. Para criar os gráficos utilizaremos o componente *DBChart* da paleta *Data Controls*.

Nos gráficos que iremos criar, vamos utilizar duas formas distintas de exibir os registros. Na primeira iremos exibir os re-

gistros absolutos que serão retornados de uma consulta SQL. E na segunda iremos parametrizar a quantidade de registros a ser exibida, onde o restante das informações não atingidas pelo parâmetro de limite da busca será mostrado em uma categoria "OUTROS" criada no gráfico com a devida porcentagem de valores a quantidade correspondida.

### Criando a aplicação

Utilizaremos no decorrer deste artigo para criação do exemplo de gráficos o Delphi 7, mas qualquer outra versão poderá ser utilizada. Crie uma nova aplicação no menu *File\New>Application*. Altere o nome do formulário para "frmPrincipal" e defina o *Caption* do formulário principal para "Gráficos com Win32". Salve a Unit como "uGráficos.pas" e o projeto salve como "Gráficos.dpr".

Crie agora um novo formulário, a partir do menu *File\New>Form*, que utilizaremos para exibir todos os registros



#### Maikel Marcelo Scheid

([maikelscheid@gmail.com](mailto:maikelscheid@gmail.com))

é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MSSQL. É membro da Equipe Editorial ClubeDelphi.

resultantes de uma pesquisa SQL à base de dados sem nenhum parâmetro de limitação de registros. Altere o nome para "frmCustomerCountry" e a propriedade *Caption* para "Clientes por País". Salve a Unit do formulário como "uCustomerCountry.pas".

Adicione também, através do menu *File|New>Data Module*, um Data Module ao projeto, que será utilizado para configuração do componente de conexão ao banco de dados e também dos componentes de consulta. Altere o nome do Data Module para "DM" e salve a *Unit* como "uDM.pas".

Adicione ao Data Module um componente *SQLConnection* ("sqlConexao") da paleta *dbExpress*. Altere a propriedade *LoginPrompt* para *False* e com um duplo clique abra o diálogo de configuração da conexão. Na caixa de diálogo que aparece clique no botão "+" e adicione uma nova conexão. Selecione, em *Driver Name*, a opção *Interbase* e em *Connection Name* digite "Employee" e confirme.

De volta ao diálogo faremos as configurações do caminho da base de dados e também do usuário e senha de acesso. Em *Database* informe o caminho para o banco de dados *Employee.fdb*, normalmente localizado em *C:\Arquivos de Programas\Firebird\Firebird\_2\_0\exemplos*, que acompanha a instalação do Firebird. Também é necessário incluir o nome de usuário em *User\_Name* e a senha padrão de acesso em *Password*. Caso não tenha modificado o usuário e senha do Firebird eles são *SYSDBA* e *masterkey*, respectivamente.

Também é de fundamental importân-

cia que se defina o dialeto de conexão em *SQLDialect* como dialeto 3. Defina também o *ServerCharSet* como *Win1252* (Figura 1).

## Criando e configurando a consulta de registros

Adicione ao Data Module um componente *SQLDataSet* ("sqlCustomerCountry") também da paleta *dbExpress*. Ligue a propriedade *SQLConnection* ao componente "sqlConexao" e altere a propriedade *Name* para "sqlCustomerCountry" e adicione a seguinte instrução *SQL* à propriedade *CommandText*:

```
select country.country as pais,
count(customer.cust_no) as clientes from
country, customer where customer.country =
country.country group by country.country
```

Na instrução *SQL* anterior estamos buscando dois campos dos registros do banco de dados. Um com o nome do país e outro com a contagem da quantidade de registros em cada país da seleção.

Com um duplo clique sobre o componente *SQLDataSet* abra o *Fields Editor* e com o botão direito do mouse selecione a opção *Add All Fields* para adicionar todos os campos retornados pela *SQL*.

## Criando o gráfico

Para criar o gráfico, volte ao *frmCustomerCountry* e adicione primeiramente um componente *Panel* da paleta *Standard*, alterando a propriedade *Align* para *alTop* e exclua o texto da sua propriedade *Caption*. Adicione ao formulário um componente *DBChart* da paleta *Data Controls* alterando

sua propriedade *Align* para *alClient*.

---

Nota: Não esqueça de adicionar o Data Module ao *Uses* do formulário para que possa ter acesso aos componentes de conexão com o banco. Para isso selecione *File|Use unit...* ou pressione *ALT + F11*.

---

Agora para realizar a configuração do componente, dê um duplo clique sobre o mesmo para que possa acessar suas propriedades de configuração. No diálogo de configuração do gráfico, a primeira escolha que temos a fazer é quanto ao tipo de gráfico. Utilizando o botão *Add*, selecione primeiramente o gráfico do tipo *Pizza* ("Pie") e confirme.

Aparecerá uma nova série selecionada, altere o nome da mesma para "Pie" utilizando o botão *Title*. Repetindo o processo selecione também o gráfico do tipo *Bar* e confirme. Altere seu nome para "Bar".

Pronto, você terá disponível agora dois tipos de gráficos (Figura 2).

Com o gráfico *Pie* selecionado, clique na aba *Series* ao lado da aba *Chart* e, dentre as abas de configuração, selecione *Data Source*. Nesta aba iremos configurar o acesso do componente de gráfico ao componente *sqlCustomerCountry* onde preparamos a instrução de consulta à base de dados.

Logo na primeira caixa de opções, selecione *DataSet*, a qual irá habilitar as demais opções de configuração. Na opção *DataSet* abaixo, ao clicar deverão aparecer os componentes aos quais o *DBChart* possui compatibilidade de conexão, no

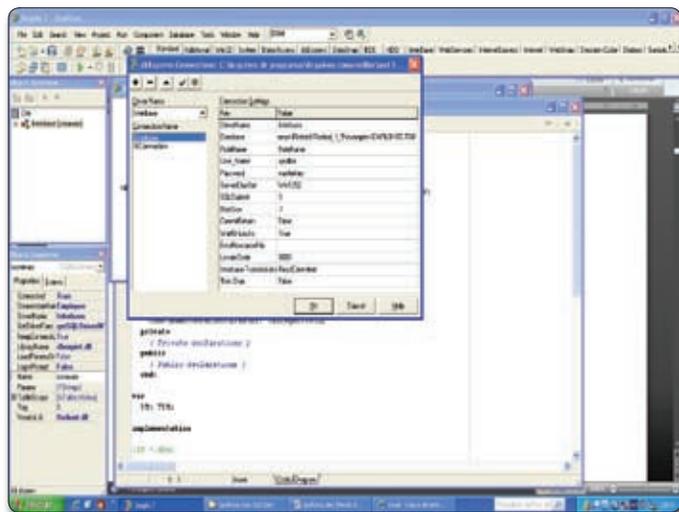


Figura 1. Configurando a conexão à base de dados

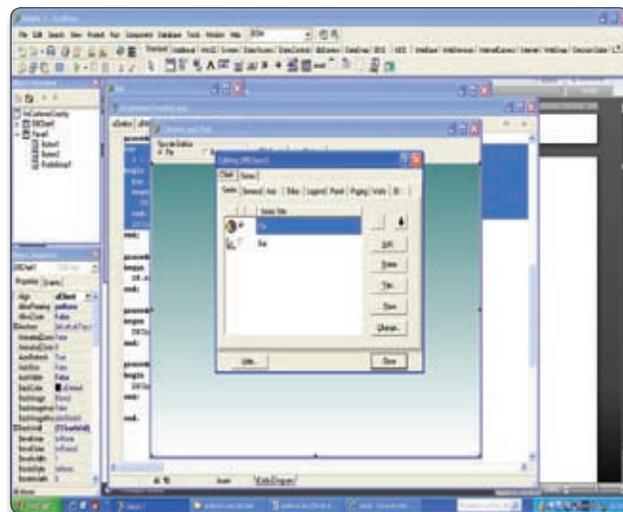


Figura 2. Adicionando os tipos de Gráficos

caso desse exemplo, irá aparecer somente o `sqlCustomerCountry` que configuramos no `DM`, que ao selecioná-lo irá habilitar outras duas configurações que precisam ser feitas.

Em `Labels` selecione `PAIS`, que corresponde ao campo onde serão informadas as descrições de países para exibição no gráfico. Na propriedade `Pie` selecione `CLIENTES`, que corresponde ao campo de contagem de ocorrências dos clientes para cada país que será exibido, de acordo com a instrução `SQL` configurada anteriormente (**Figura 3**).

Realizada a configuração da série `Pie`, selecione agora a série `Bar` nesta mesma aba `Series`. Na primeira opção de escolha

selecione novamente `DataSet` e ligue ao `sqlCustomerCountry` configurado no `Data Module`. As propriedades de configuração deste tipo de gráfico serão um pouco diferentes quanto à configuração do primeiro, pois este tipo de gráfico está solicitando um valor para a coluna `X` e outro valor para a coluna `Bar`.

Na propriedade `Labels` não se tem diferença, portanto selecione novamente o campo `PAIS`. Agora nas outras duas propriedades, iremos informar um valor apenas ao campo `Bar`, selecionando o campo `CLIENTES` (**Figura 4**).

Se fossemos informar o campo `CLIENTES` também na propriedade `X`, nosso gráfico iria agrupar os valores idênticos,

ou seja, se tivéssemos dois países com cinco clientes em cada um deles, uma única barra seria exibida ao valor correspondente e somente na legenda que apareceriam os nomes de ambos os países com o mesmo número de clientes.

### Configurações e aparência do Gráfico

Veremos agora como configurar algumas das muitas opções de configuração das propriedades do componente `DB-Chart`. Logo na aba `Marks`, dentro da aba `Series`, que corresponde especificamente ao tipo do gráfico selecionado, teremos as opções de configuração do corpo do gráfico, onde você poderá escolher de que forma serão exibidos os `Labels` para

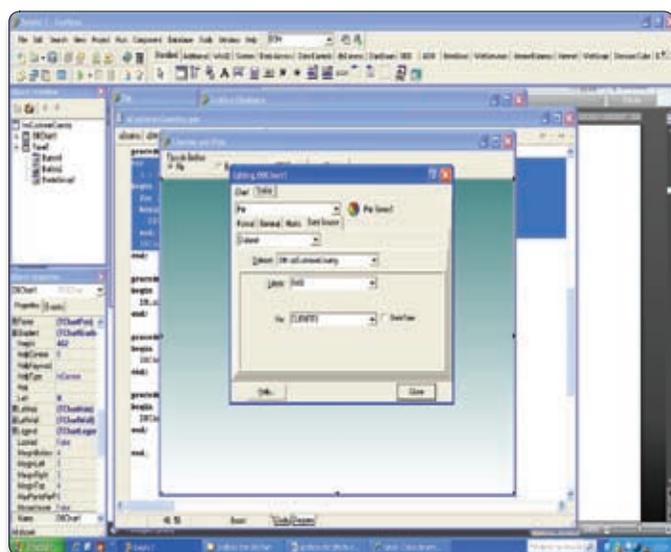


Figura 3. Configurando as fontes de informações do gráfico

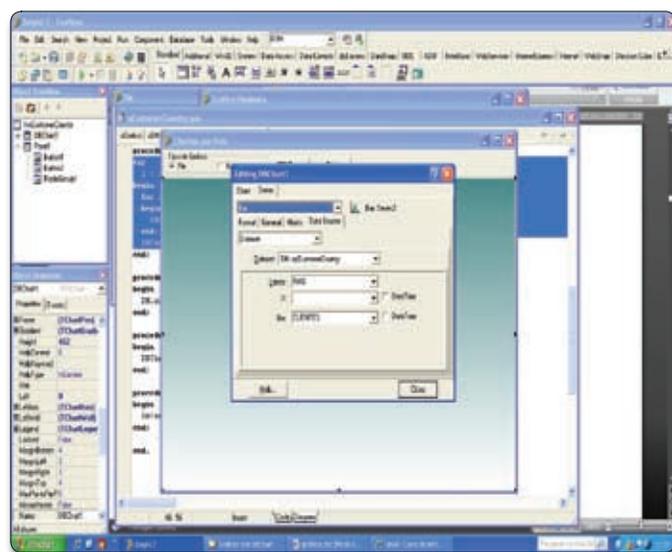


Figura 4. Configurando as fontes do gráfico de barras

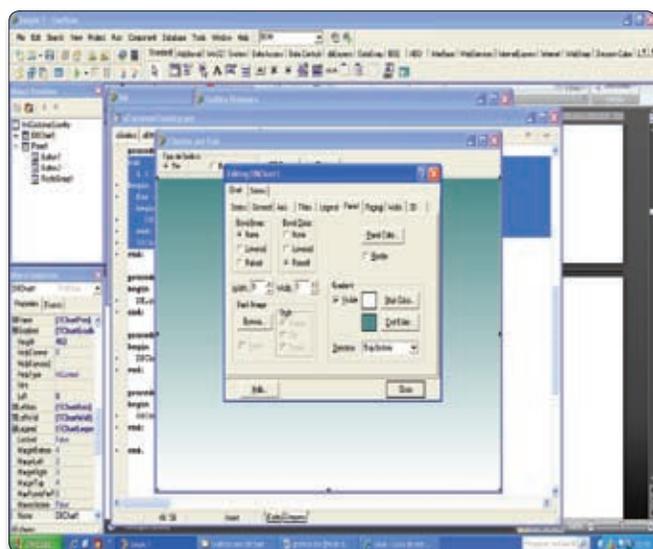


Figura 5. Configurações de fundo do corpo do gráfico

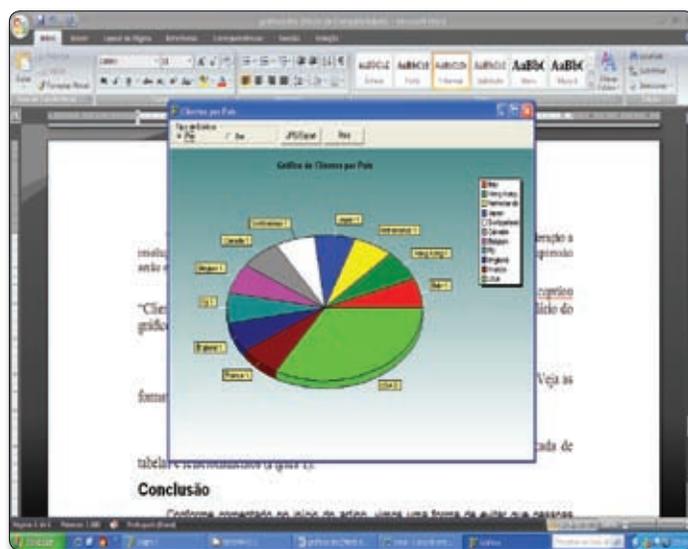


Figura 6. Exibição do gráfico de clientes por cidade

cada barra do gráfico. Se desejar, poderá exibir somente o valor (“Value”) ou a descrição e o valor (“Label and Value”), entre outras.

Poderá também alterar a cor de fundo (“Back Color”) dos Labels, tipo de fonte, tipo de borda, entre as demais configurações. Na aba *General* ao lado, você terá as opções para definir o estilo de formatação dos campos de valor. Na aba *Format* você encontrará inúmeras opções das configurações da aparência do gráfico individuais para cada tipo.

No gráfico do tipo *Pie* marque a opção *Circled Pie* que irá garantir um gráfico sempre em forma de círculo, independente da resolução em que o formulário que carrega o gráfico for aberto.

Voltando agora para a aba *Chart*, que corresponde às configurações gerais do gráfico independente do tipo selecionado, vá até a aba *Titles* e digite o texto

“Gráfico de Clientes por País” no campo *Memo* de edição do título. Altere também sua fonte e tamanho para destacar o título no gráfico.

Na aba *Panel* ative a opção *Visible no grupo Gradient do canto direito inferior*, a qual irá possibilitar que você configure uma combinação de duas cores para o fundo do gráfico.

Após selecionadas as cores, escolha a direção em que as mesmas serão exibidas (**Figura 5**). Todas as configurações já estarão valendo para o gráfico, utilize o botão *Close* para sair.

Você poderá ir testando todas as demais propriedades de configurações que não mencionei aqui neste artigo.

Ainda no mesmo formulário precisamos adicionar dentro do componente *Panel* um *RadioGroup* da paleta *Standard* para alternar de um tipo de gráfico para outro em tempo de execução.

Adicione a sua propriedade *Items* em duas linhas os valores “Pie” e “Bar”. Adicione também dentro do *Panel* dois componentes *Button* alterando o *Caption* do primeiro para “JPG Export” e do segundo para “Print”. Esses botões serão usados para exportar o gráfico para uma imagem no formato JPG e fazer a impressão direta do gráfico, respectivamente.

Adicione ao evento *OnClick* do *RadioGroup1* o código da **Listagem 1** que será o responsável por alterar os tipos de gráficos em tempo de execução. O código desabilitará todos os tipos de gráficos para que não ocorra a sobreposição de algum deles e após isso ativa um novo tipo de acordo com o índice selecionado.

No botão de exportação do gráfico para o formato JPG inclua a seguinte linha de código, onde chamamos o método *SaveToBitmapFile* do componente *DbChart*, passando um caminho onde a imagem será arquivada. Se preferir, poderá salvar também em outros formatos de imagens, como *Bitmap* por exemplo.

```
DBChart1.SaveToBitmapFile('C:\Grafico.jpg');
```

Ao componente de impressão, simplesmente adicione ao evento *OnClick* o código abaixo, que fará a impressão do gráfico na impressora padrão instalada.

```
DBChart1.Print;
```

---

**Nota:** Tanto na opção de exportar quanto para imprimir, leve em consideração a resolução de tela e do formulário onde o gráfico se encontra. As proporções de exportação e/ou impressão serão exatamente as mesmas as quais o gráfico se encontra no momento da requisição.

---

#### Listagem 1. Código para alteração do tipo de gráfico em runtime

```
procedure TfrmCustomerCountry.RadioGroup1Click(Sender: TObject);
var
  I : Integer;
begin
  for I:= 0 to DBChart1.SeriesCount -1 do
    DBChart1.Series[I].Active := False;
    DBChart1.Series[RadioGroup1.ItemIndex].Active := True
  end;
```

#### Listagem 2. Código exibição dos registros na montagem do gráfico dinâmico.

```
procedure TfrmGraficoDinamico.BitBtn1Click(Sender: TObject);
var
  soma: Integer;
begin
  if ((edtQntde.Text <> '') and (StrToInt(edtQntde.Text) > 0)) Then
  begin
    DM.cdsDinamicoGrafico.Open;
    with DM.cdsDinamicoGrafico do
    begin
      while not eof do Delete;
      DM.sqlDinamicoGrafico.Close;
      DM.sqlDinamicoGrafico.CommandText := 'select first ' + edtQntde.Text +
      ' skip 0 department.department ' + 'as departamento, ' +
      'count(employee.emp_no) as clientes ' + ' from department, employee ' +
      'where employee.dept_no = ' + ' department.dept_no group by ' +
      'department.department';
      DM.sqlDinamicoGrafico.Open;
      soma := 0;
      while not DM.sqlDinamicoGrafico.Eof do
      begin
        DM.cdsDinamicoGrafico.Append;
        DM.cdsDinamicoGraficoDEPARTAMENTO.Value := DM.sqlDinamicoGrafico.Fields[0].AsString;
        DM.cdsDinamicoGraficoCLIENTES.Value := DM.sqlDinamicoGrafico.Fields[1].Value;
        soma := soma + DM.sqlDinamicoGrafico.Fields[1].AsInteger;
        DM.cdsDinamicoGrafico.Post;
        DM.sqlDinamicoGrafico.Next;
      end;
      DM.sqlDinamicoGrafico.Close;
      DM.sqlDinamicoGrafico.CommandText := 'select cast(''OUTROS'' as varchar(70)) ' +
      ' as departamentos, ' + ' cast( (select (count(a.emp_no)) ' + 'from employee a) - ' +
      'IntToStr(Soma)+') as integer) as '+ 'CLIENTES from rdb$database';
      DM.sqlDinamicoGrafico.Open;
      DM.cdsDinamicoGrafico.Append;
      DM.cdsDinamicoGraficoDEPARTAMENTO.Value := DM.sqlDinamicoGrafico.Fields[0].AsString;
      DM.cdsDinamicoGraficoCLIENTES.Value := DM.sqlDinamicoGrafico.Fields[1].Value;
      DM.cdsDinamicoGrafico.Post;
    end;
  end
  else
  ShowMessage('O valor deverá ser superior a 0.');
```

## Rodando a aplicação

Retornando agora até o formulário principal, adicione um *Button* com o *Caption* “Clientes por País” e adicione ao evento *OnClick* o código para chamar o formulário do gráfico conforme a seguir:

```
frmCustomerCountry.Show;
```

Execute agora sua aplicação, exibindo o formulário onde criamos o gráfico. Veja as formatações e valores do gráfico construído (**Figura 6**).

## Gráfico dinâmico

Chamo esta forma de gráfico como dinâmica devido ao fato de você ter a liberdade de definir a quantidade de registros que deseja exibir, sendo que o restante dos mesmos, que não são atingidos pela condição de limite da quantidade, serão somados e adicionados ao gráfico em uma categoria "OUTROS". Sua construção não é muito complicada, portanto mãos a obra.

Vá até o Data Module do projeto e adicione três novos componentes, sendo eles:

**SQLDataSet:** altere a propriedade *Name* para "sqlDinamicoGrafico", ligue sua propriedade *SQLConnection* ao componente "sqlConexao" e adicione a seguinte instrução SQL a propriedade *CommandText*:

```
select first 5 skip 0 department.department as departamento,
count(employee.emp_no) as clientes from
department, employee
where employee.dept_no = department.dept_no
group by department.department order by 2
desc
```

**DataSetProvider:** altere o *Name* para "dspDinamicoGrafico" e conecte a propriedade *DataSet* ao *sqlDinamicoGrafico*;

**ClientDataSet:** altere o *Name* para "cdsDinamicoGrafico" e ligue a propriedade *ProviderName* ao *dspDinamicoGrafico*. Abra o *Fields Editor* e adicione todos os campos retornados pela SQL.

Crie agora um novo formulário, altere o nome para "frmGraficoDinamico" e o *Caption* para "Gráfico de empregados por Departamento". Salve a *Unit* do formulário como "uGraficoDinamico.pas". Adicione

agora ao formulário um componente *Panel* com sua propriedade *Align* definida para *alTop* e dentro do *Panel* adicione um componente *Edit* ("edtQntde") para receber a quantidade de registros a ser exibida e um *Button* com o *Caption* "Exibir Gráfico" para ativar a pesquisa. Adicione ao formulário um componente *DBChart* definindo sua propriedade *Align* para *alClient*, deixando os componentes organizados de forma semelhante a **Figura 7**.

Com um duplo clique sobre o gráfico, adicione uma nova série do tipo *Pie*, acesse a aba *Series* ao lado de *Chart* e faça a configuração na aba *DataSource*, de forma semelhante ao exemplo de gráfico anterior, porém agora não ligando o *DataSet* ao *SQLDataSet*, mas sim ao componente *ClientDataSet* configurado no Data Module. Na instrução SQL que adotamos para este exemplo, estamos selecionando o número de empregados por departamento. Atribua à propriedade *Labels* o campo DEPARTAMENTOS e à propriedade *Pie* o campo CLIENTES. Faça as demais configurações do gráfico da mesma forma que no gráfico do exemplo anterior.

Atribua ao evento *OnClick* do botão "Exibir Gráfico" o código da **Listagem 2**, que será responsável por fazer a pesquisa dos registros no banco de dados e armazenar os valores no *ClientDataSet*.

Ao final da seleção de todos os registros, de acordo com o limite de informações a serem exibidas, um cálculo será realizado e designará uma nova busca no banco de dados, que trará como resultados os

registros não atingidos pela primeira condição de pesquisa, adicionando um novo registro ao *ClientDataSet* com a descrição de OUTROS e o valor correspondente.

Volte agora para o formulário principal e adicione mais um botão, altere o *Caption* para "Empregados por Departamento" e adicione a seguinte linha de código ao evento *OnClick* do botão:

```
frmGraficoDinamico.Show;
```

Execute sua aplicação acessando o gráfico que irá trazer por padrão os 5 primeiros registros que deixamos configurados na instrução SQL, mas que ao receber um parâmetro de limite de valores exibirá o gráfico com registros selecionados e um último registro com o rótulo "OUTROS" e a quantidade a ele relacionado (**Figura 8**).

## Conclusão

Vimos nesse artigo exemplos práticos do uso e da empregabilidade de gráficos em nossos sistemas. Agora de uma forma fácil você poderá apresentar uma ferramenta a mais nos seus sistemas para seus clientes, possibilitando uma visualização mais entendível dos conteúdos e planilhas de informações. Até o próximo artigo. Abraços. ●

**ClubeDelphi PLUS** [www.devmedia.com.br/clubedelphi/portal.asp](http://www.devmedia.com.br/clubedelphi/portal.asp)

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Fabrício Desbessel que mostra como trabalhar com gráficos no Delphi.

[www.devmedia.com.br/articles/viewcomp.asp?comp=4906](http://www.devmedia.com.br/articles/viewcomp.asp?comp=4906)

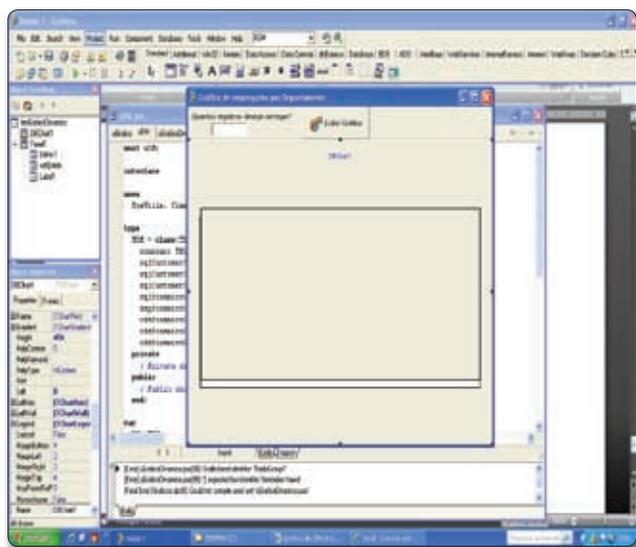


Figura 7. Componentes para montagem do gráfico dinâmico

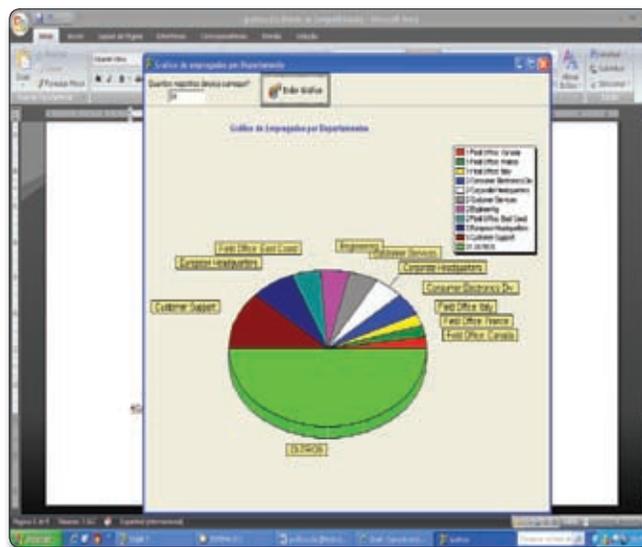


Figura 8. Gráfico dinâmico com limitação de registros



# DevMedia

GROUP

O conteúdo ao seu alcance!

+ de **600 artigos**  
impressos por ano

+ de **1500 vídeo-aulas**  
de acesso imediato

+ de **24 cursos** online

+ de **130 vídeos inéditos**  
todos os meses

Se você busca **conteúdo**  
de qualidade e aprendizagem rápida.

**Aqui é seu lugar!**

Acesse e confira:

[www.devmedia.com.br](http://www.devmedia.com.br)

## Nesta seção você encontra artigos para iniciantes na linguagem Delphi

### Aplicações Cliente/Servidor

#### Uso de múltiplos Data Modules

**Pablo Tøndolo de Vargas**

(pablodv@gmail.com)

é Acadêmico do Curso de Sistemas de Informação da Universidade Luterana do Brasil (ULBRA). Atua na área de desenvolvimento em Delphi para a empresa dotBR Soluções em TI.

**Fernando Sarturi Prass**

(fernando@dotbr.com.br)

é Mestre em Ciência da Computação pela UFSC. Professor da Universidade Luterana do Brasil (ULBRA) nos campus de Santa Maria e Cachoeira do Sul. Sócio-diretor da dotBR Soluções em TI (www.dotbr.com.br), empresa que presta serviços de desenvolvimento de sistemas e de consultoria em Bancos de Dados e Metodologias de Desenvolvimento.

No passado, principalmente quando se programava para MS-DOS, a quantidade de memória RAM disponível era irrisória, o que obrigava os programadores muitas vezes a usar técnicas complexas para economizar alguns poucos bytes de memória de leitura.

Atualmente os computadores possuem grande quantidade de memória RAM disponível, mas isto não é motivo para que se aloque memória desnecessariamente, já que nem sempre a aplicação rodará localmente. Por exemplo: uma aplicação instalada num servidor pode ser acessada simultaneamente por diversos usuários via Windows Terminal Services (WTS). Estes usuários podem estar em qualquer lugar do mundo, mas rodarão a aplicação diretamente no servidor. Se o sistema e o número de usuários forem grandes, e o desenvolvedor abusar da alocação de memória, pode haver problema de falta de memória mesmo em computadores

com mais de 1 GB de RAM.

Este artigo mostra passo a passo uma técnica simples e eficaz para criar *Data Modules* em tempo de execução em aplicações Cliente/Servidor, o que torna a inicialização da aplicação mais rápida e ao mesmo tempo consome menos memória RAM.

#### Entendendo a arquitetura

*Data Module* nada mais é do que um repositório de componentes não visuais assim como o próprio nome sugere. Sua finalidade é alocar componentes de acesso a dados a fim de separar regras de negócio da interface. Com isso tem-se uma maior facilidade na manutenção e/ou em uma possível migração de Cliente/Servidor (duas camadas) para *n-tiers* (três ou mais camadas).

A **Figura 1** ilustra como fica a arquitetura de uma aplicação usando *Data Modules*, onde os componentes de acesso a dados e as regras de negócios estariam neles,

ficando somente na parte do formulário os componentes visuais e os que não acessam dados.

## Criando a Aplicação

Essa aplicação será constituída de um formulário principal e 4 *Data Modules*. Portanto crie uma nova aplicação utilizando o menu *File\New>Application*. Salve o projeto na pasta de sua preferência. Para a *Unit* dê o nome de “uPrincipal.pas” e para o projeto dê o nome de “Employee.dpr”, por último altere a propriedade *Name* do formulário para “frmPrincipal”.

## Configurando os Data Modules

Adicione um *Data Module* usando o menu *File\New>Data Module* e salve-o com nome de “uDMConexao”, altere a propriedade *Name* para “dmConexao”. Coloque o *dmConexao* para que seja criado antes do *frmPrincipal*. Para isso acesse *Project>Options* ou use o atalho *Ctrl + Shift + F11* e na aba *Forms* mude a ordem para que o *dmConexao* esteja primeiro que o *frmPrincipal*.

Por padrão, todos os formulários e *Data Modules* são criados automaticamente quando o programa é inicializado, para alterar isso vá em *Tools\Environment Options>Designer* e desmarque a opção *Auto create forms & data modules*. Outra forma de fazer a mesma coisa é a partir da janela *Project>Options*, após criar o

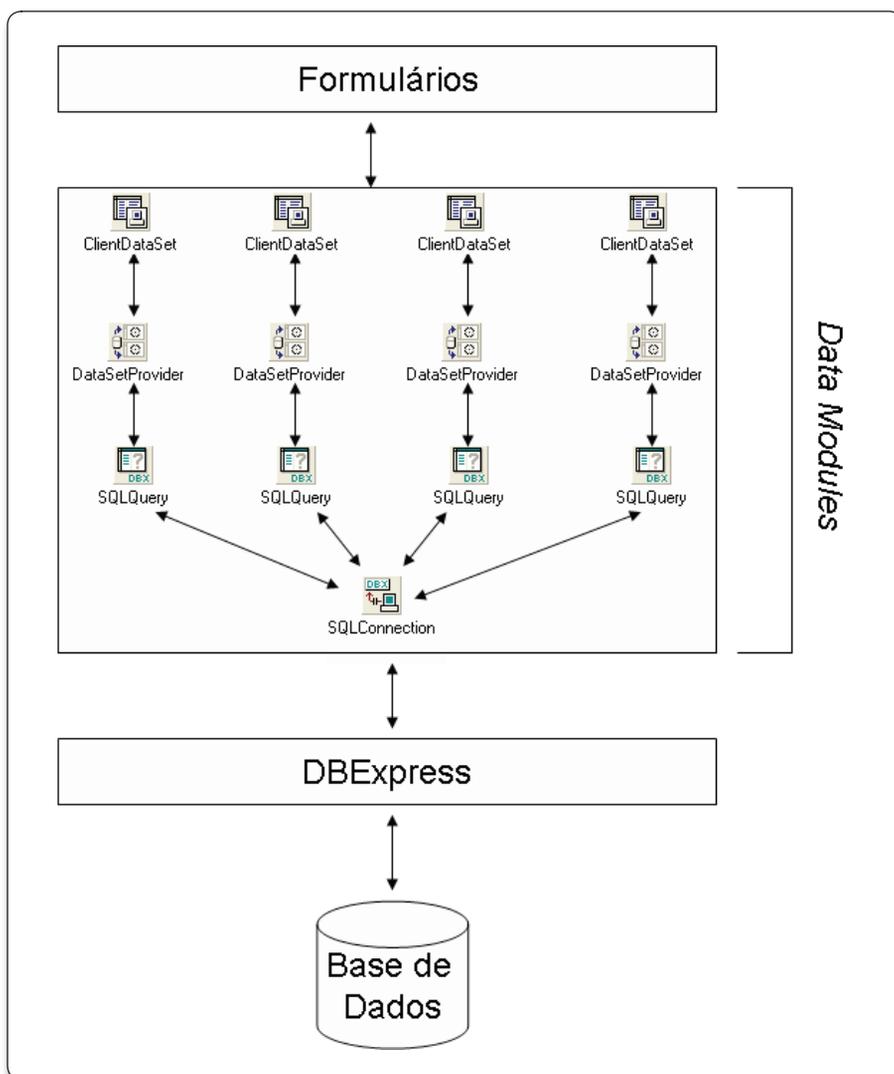


Figura 1. Arquitetura de uma aplicação com Data Module

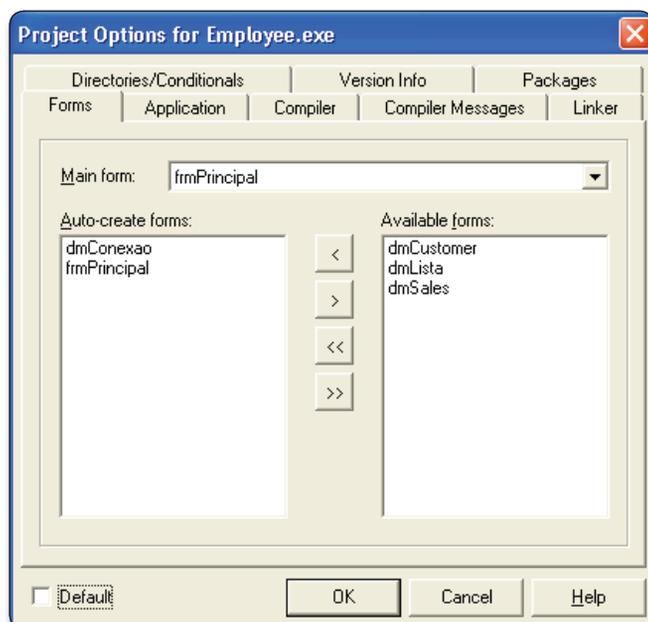


Figura 2. Opções do projeto com todos os Data Modules criados

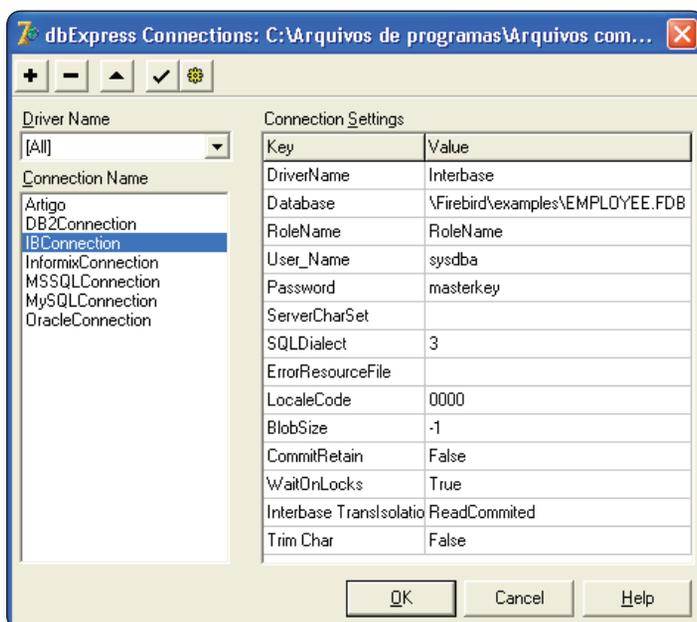


Figura 3. Exemplo de Conexão com o FireBird

formulário ou *Data Module* selecionar os que não serão criados com a aplicação e arrastá-los para o lado (*Available forms*). No final do desenvolvimento do aplicativo mostrado neste artigo, deverá ter-se a mesma configuração apresentada na **Figura 2**.

O próximo passo é configurar a conexão com a base de dados. Para este aplicativo será usada a base de exemplo que acompanha o FireBird. Para isso adicione um componente *SQLConnection* ("sqlConexao"), que está na aba *dbExpress*, ao *sqlConexao* dê dois cliques no componente para configurar suas propriedades.

Crie uma nova conexão clicando no botão "+". Na caixa de diálogo que se abre selecione o driver Interbase em *Driver Name*, dê um nome a conexão em *Connection Name* e confirme.

---

**Nota:** O Delphi traz por padrão algumas conexões previamente configuradas, como é o caso de *IBConnection*.

---

Após criar a conexão configure a propriedade *DataBase* informando o endereço completo para o arquivo *EMPLOYEE.FDB*, normalmente localizado na pasta *C:\Arquivos de Programas\Firebird\_1\_5\Firebird\Examples\empbuild\*. A propriedade *User\_Name* é o usuário que será usado para conectar no servidor, por padrão é *SYSDBA* e a senha é *masterkey* (atributo *Password*). Mude o *SQLDialect* para 3 e pressione *OK*, a configuração deverá ficar semelhante a mostrada na **Figura 3**.

No *Object Inspector* mude as propriedades *LoginPrompt* para *False* e ative a conexão mudando para *True* a propriedade *Connected*. Pronto, a conexão com o servidor de banco de dados está estabelecida.

Adicione um novo *Data Module* ao proje-

to. Ele conterá a *query* que será usada para mostrar os dados da tabela *Country*. Salve-o com o nome de "uDMLista", mude a propriedade *Name* para "dmLista" e adicione uma referência a *Unit dmConexao* usando para isso o menu *File>Use Unit* ou o atalho *Alt + F11*. Agora adicione um componente do tipo *SQLQuery* ao *Data Module*, renomeie-o para "qryCountry" e na propriedade *SQLConnection* coloque a conexão criada no *dmConexao*. Coloque a seguinte consulta na propriedade *SQL*:

```
select * from COUNTRY
```

Ative e desative a *query* para verificar se a consulta inserida está correta. Feito isso adicione mais dois componentes: um *DataSetProvider*("dspCountry") alterando a propriedade *DataSet* para "qryCountry". Um *ClientDataSet*("cdsCountry") com a propriedade *DataSetProvider* apontada para o *dspCountry*. Dê dois cliques no *cdsCountry* e com o botão direito do mouse selecione a opção *Add all fields*, com isso estará adicionando todos os campos da tabela ao *Fields Editor*.

Adicione outro *SQLQuery* que será utilizado para mostrar os dados da tabela *Customer*. Mude as propriedades *Name* para "qryCustomerLista", *SQLConnection* para "dmConexao.sqlConexao", e para o *SQL* adicione a instrução a seguir:

```
select * from CUSTOMER
```

Adicione agora um *DataSetProvider*("dspCustomerLista") e mude seu *DataSet* para "qryCustomerLista". Finalmente adicione um *ClientDataSet*("cdsCustomerLista"). Seu *ProviderName* deverá ser apontado para *dspCustomerLista*. Por fim dê dois cliques no *cdsCustomerLista* e com o botão direito do mouse selecione a opção *Add all fields* (**Figura 4**).

Adicione mais um *Data Module* para a

tabela *Customer* da nossa base de dados. Salve com o nome de "uDMCustomer.pas" e mude a propriedade *Name* para "dmCustomer". Adicione uma referência ao *dmConexao* usando *Alt + F11* assim como foi feito anteriormente.

Nesse ponto serão adicionados os componentes que permitem inserir novos registros na tabela. Para isso adicione três componentes sendo: um *SQLQuery*("qryCustomer"), *SQLConnection* como "dmConexao.sqlConexao" e propriedade *SQL* com a instrução a seguir:

```
select * from CUSTOMER
```

Um *DataSetProvider*("dspCustomer") com a propriedade *DataSet* apontada para *qryCustomer* e por último um *ClientDataSet*("cdsCustomer") com sua propriedade *ProviderName* igual a *dspCustomer*.

Assim como nos demais *ClientDataSets* adicione todos os campos da tabela usando a opção *Add all fields* encontrado no menu de contexto do *Fields Editor*, ou seja, clique duas vezes no *cdsCustomer* e em seguida com o botão direito escolha a opção mencionada (**Tabela 1**).

No evento *BeforeOpen* do *cdsCustomer* digite:

```
{ Cria o dmLista }
dmLista := TdmLista.Create(nil);
{ Abre o cdsCountry }
dmLista.cdsCountry.Open;
```

E no evento *AfterClose* digite:

```
{ Fecha o cdsCountry }
dmLista.cdsCountry.Close;
{ Libera de memória o dmLista }
FreeAndNil(dmLista);
```

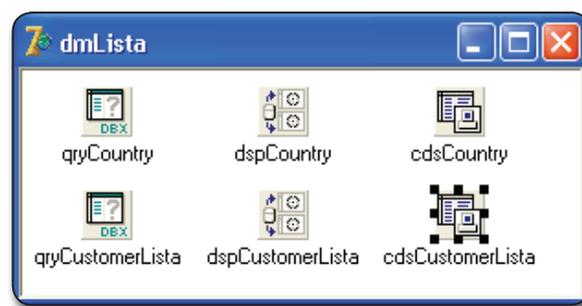
---

**Nota:** Não esqueça de adicionar a *Unit* do *Data Module* *dmLista* ao *Uses* do *Data Module* *dmCustomer*. Utilize *File>Use unit* ou *Alt + F11*.

---

Componente	Propriedade	Valor
SQLQuery1	Name	qryCustomer
	SQLConnection	dmConexao.SQLConnection1
	SQL	SELECT * FROM CUSTOMER
DataSetProvider1	Name	dspCustomer
	DataSet	qryCustomer
ClientDataSet1	Name	cdsCustomer
	DataSetProvider	dspCustomer

**Tabela 1.** Configuração dos componentes utilizados no *dmCustomer*



**Figura 4.** *dmLista* depois de configurado

Com estes dois eventos garante-se que o *dmLista* será criado quando for aberto o *cdsCustomer* e quando ele for fechado será liberado de memória, não permanecendo em memória quando desnecessário.

Adicione também a este *Data Module* mais um *SQLQuery* que desta vez será usado pelo *Generator CUST\_NO\_GEN*, que vai garantir que não se tenha chaves primárias repetidas para a tabela *Customer*, altere as propriedade *Name* para "qryGenCustomer", *SQLConnection* coloque a conexão criada no *dmConexao*, que neste caso é *sqlConexao*, e insira o seguinte select na propriedade *SQL*:

```
select GEN_ID(cust_no_gen,1) from
rdb$database
```

**Nota:** Generator é um objeto do banco de dados Firebird que, com o auxílio da função *GEN\_ID* também do banco, gera números seqüenciais, os famosos Auto Incrementados.

De dois cliques na *query* e adicione todos os *campos*. No evento *AfterInsert* do *cds-*

*Customer* digite o código da **Listagem 1**. Para tornar mais clara a utilização será criado agora mais um cadastro para a tabela *SALES* que tem relacionamento "n" para 1 com a tabela *CUSTOMER*. Seguindo a mesma idéia usada anteriormente para o



Figura 5. dmSales depois de configurado

**Listagem 1.** Código do evento *AfterInsert* do *cdsCustomer*

```
procedure TdmCustomer.cdsCustomerAfterInsert(DataSet: TDataSet);
begin
  { Abre a qryGenCustomer }
  qryGenCustomer.Open;
  { Atribui o valor do qryGenCustomerGEN_ID para o cdsCustomerCUST_NO }
  cdsCustomerCUST_NO.Value := qryGenCustomerGEN_ID.AsInteger;
  { Fecha a qryGenCustomer }
  qryGenCustomer.Close;
end;
```

Componente	Propriedade	Valor
SQLQuery1	Name	qrySales
	SQLConnection	dmConexao.SQLConnection1
	SQL	SELECT * FROM SALES
DataSetProvider1	Name	dspSales
	DataSet	qrySales
ClientDataSet1	Name	cdsSales
	DataSetProvider	dspSales

Tabela 2. Configuração dos componentes utilizados para o dmSales

## Impressão Rápida em Matriciais...

### RDprint 4.0

**O mais completo componente para impressão em MATRICIAIS !**  
**LIDERANÇA absoluta na sua categoria !**

Ideal para Notas Fiscais, Duplicatas, Boletos Bancários, etiquetas e relatórios em geral.

- Opção para impressão colorida
- Ajustes de margens para impressão gráfica
- Opção para ocultar a barra de progresso
- Variáveis PAGINAS, DATA, HORA e TÍTULO

**Novo form de SETUP com :**

- Mapeamento das impressoras e Modelos
- Seleção de páginas igual ao word (1-5,7,8)
- Opção para Inverter e Agrupar cópias na impressão

**Novo form de PREVIEW com:**

- Função para Procura de TEXTO no relatório
- ROLAGEM com salto automático de página
- ARRASTO da imagem do preview
- StatusBar com informações da impressão
- Novos ícones personalizados

Fone/Fax (14) 3454-7880  
[www.deltress.com.br](http://www.deltress.com.br)

\* Disponível para Delphi 5, 6, 7, 2005 e 2006 (VCL)  
 \* Compatível com todas as versões do Windows  
 \* Imprime em portas LPT / COM e USB (modo gráfico)

### Listagem 2. Código do botão Novo

```
procedure TfrmPrincipal.btnNovoClienteClick(Sender: TObject);
begin
  { Verifica se já foi instanciado a o dmCustomer }
  if dmCustomer = nil then
    { Instância a variável dmCustomer }
    dmCustomer := TdmCustomer.Create(nil);
    { Abre o cdsCustomer }
    dmCustomer.cdsCustomer.Open;
    { Insere um novo registro no cdsCustomer }
    dmCustomer.cdsCustomer.Insert;
end;
```

### Listagem 3. Código do botão Salvar

```
procedure TfrmPrincipal.btnSalvarClienteClick(Sender: TObject);
begin
  { Verifica se já foi criada o dmCustomer }
  if dmCustomer <> nil then
    begin
      { Guarda uma linha no cdsCustomer }
      dmCustomer.cdsCustomer.Post;
      { Salva as alterações guardadas no cdsCustomer para o banco }
      dmCustomer.cdsCustomer.ApplyUpdates(0);
      { Libera da memória o dmCustomer. }
      FreeAndNil(dmCustomer);
    end
  else
    { Caso não exista em memória ainda o dmCustomer emite uma mensagem }
    ShowMessage('Você deve clicar em novo antes de salvar um registro');
end;
```

### Listagem 4. Código do botão Cancelar

```
procedure TfrmPrincipal.btnCancelarClienteClick(Sender: TObject);
begin
  { Verifica se já foi criada o dmCustomer }
  if dmCustomer <> nil then
    begin
      { Cancela todas as alterações realizadas no cdsCustomer. }
      dmCustomer.cdsCustomer.Cancel;
      { Libera da memória o dmCustomer. }
      FreeAndNil(dmCustomer);
    end
  else
    { Caso não exista em memória ainda o dmCustomer emite uma mensagem. }
    ShowMessage('Você deve clicar no novo antes de cancelar um' + ' registro');
end;
```

cadastro da tabela *CUSTOMER* crie um novo *Data Module* salve com o nome de “uDMSales.pas” e mude a propriedade *Name* para “dmSales”. **Figura 5**

Adicione um componente *SQLQuery*, um *DataSetProvider* e um *ClientDataSet* configurando-os de acordo com a **Tabela 2**.

Dê dois cliques no *cdsSales* com o botão direito do mouse selecione a opção *Add All Fields*. Para o evento *BeforeOpen* do *cdsSales* digite:

```
{ Cria o dmLista }
dmLista := TDMLista.Create(nil);
{ Abre o cdsCustomerLista }
dmLista.cdsCustomerLista.Open;
```

E para o evento *AfterClose* do *cdsSales* digite:

```
dmLista.cdsCustomerLista.Close;
{ Fecha o cdsCustomerLista }
FreeAndNil(dmLista);
{ Libera de memória o dmLista }
```

Com isto tem-se somente em memória o *dmLista* enquanto o *cdsSales* estiver aberto.

## Configurando o formulário principal

Abra o *frmPrincipal* adicione uma referência aos *Data Modules* *dmCustomer*, *dmLista* e *dmSales*. Adicione dois *GroupBox*s mudando a propriedade *Name* para “gbxCustomer” e “gbxSales” respectivamente. Adicione também 4 *DataSources* alterando a propriedade *Name* de cada um deles para *dsCustomer*, *dsSales*, *dsCountryLista* e *dsCustomerLista*. Altere a propriedade *DataSet* de cada um deles apontando-os para seus respectivos *ClientDataSets* O *dsCustomer* receberá “dmCustomer.cdsCustomer”, para o *dsSales* coloque “dmSales.cdsSales”, no *dsCountryLista* receberá “dmLista.cdsCountry” e para o *dsCustomerLista* coloque “dmLista.cdsCustomerLista”.

Dentro do *gbxCustomer* adicione 10 *DBE-dits*, 11 *Labels* e um *DBLookupComboBox*. Para os componentes *DBE-dit* configure a propriedade *DataSource* para *dsCustomer* e a propriedade *DataField* faça com que cada um receba um campo diferente.

No *DBLookupComboBox* altere as seguintes propriedades: *Name* para “cbxCountry”, *ListSource* para “dsCountryLista”, *ListField* para “CURRENCY”, *KeyField* para “COUNTRY”, *DataSource* para *dsCustomer* e *DataField* para “COUNTRY”.

Dentro do “gbxSales” adicione 10 *DBE-dits*, 11 *Labels* e um *DBLookupComboBox*. Altere a propriedade *DataSource* dos *DBE-*

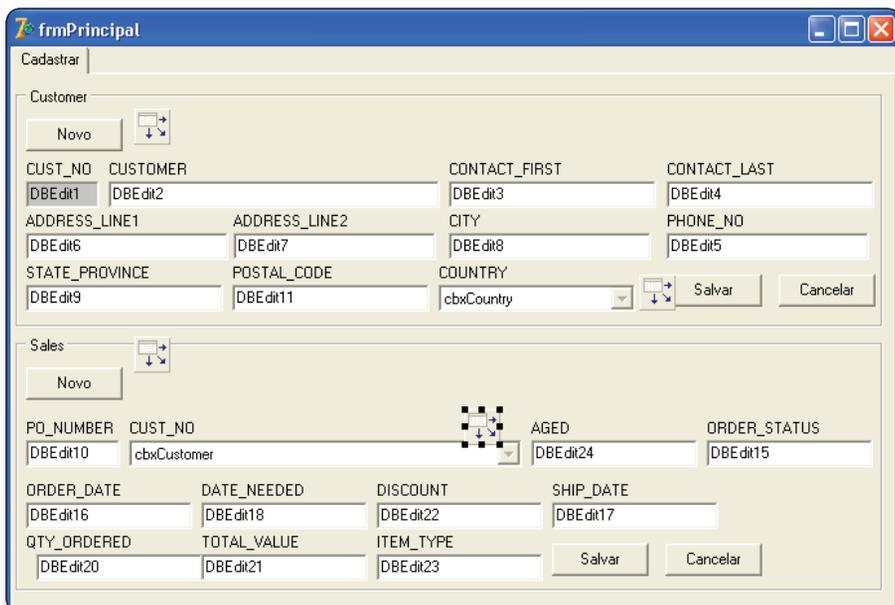


Figura 6. frmPrincipal depois de alterado para o cadastro da tabela Customer

dit para *dsSales* e ajuste o *DataField* para que nenhum *DBEdit* fique com o mesmo campo. Já para o *DBLookupComboBox* altere as seguintes propriedades: *Name* para *cbxSales*, *ListSource* para *dsCustomerLista*, *ListField* para "CUSTOMER", *KeyField* para "CUST\_NO", *DataSource* para *dsSales* e *DataField* para "CUST\_NO".

Após concluídas essas configurações o formulário principal deverá se parecer com a **Figura 6**.

## Codificando o exemplo

Faremos agora a parte de codificação do exemplo. Antes de codificar, insira 6 componentes *Button* sendo 3 no *GroupBox* superior com os nomes *btnNovoCliente*, *btnSalvarCliente* e *btnCancelarCliente*. Repita esses passos modificando os nomes dos 3 outros *Buttons* do *GroupBox* para *btnNovaVenda*, *btnSalvarVenda* e *btnCancelarVenda*.

Começaremos pelo botão novo chamado de *btnNovo*. Dê um duplo clique no *btnNovoCliente* que está dentro do *gbxCustomer* e insira o seguinte código da **Listagem 2**.

Para o evento *onClick* do *btnSalvar* que está dentro do *gbxCustomer* digite o código da **Listagem 3**.

Para o evento *onClick* do *btnCancelar* que está dentro do *gbxCustomer* digite o código da **Listagem 4**.

Em seguida codifique o *btnNovaVenda* que está dentro *gbxSales* digite o código da **Listagem 5**.

Já para o botão *btnSalvarVenda* que também está dentro do *gbxSales* digite o código da **Listagem 6**.

Por último o *btnCancelar* digite o seguinte código da **Listagem 7**.

### Listagem 5. Código do botão Nova Venda

```
procedure TfrmPrincipal.btnNovaVendaClick(Sender: TObject);
begin
  { Verifica se já foi criado o dmSales }
  if dmSales = nil then
    { Instancia a variável dmSales }
    dmSales := TdmSales.Create(nil);
    { Abre o cdsSales }
    dmSales.cdsSales.Open;
    { Insere um novo registro no cdsSales }
    dmSales.cdsSales.Insert;
end;
```

### Listagem 6. Código do botão Salvar Venda

```
procedure TfrmPrincipal.btnSalvarVendaClick(Sender: TObject);
begin
  { Verifica se já foi criado o dmSales }
  if dmSales <> nil then
    begin
      { Guarda uma linha no cdsSales }
      dmSales.cdsSales.Post;
      { Salva as alterações guardadas no cdsSales para o banco }
      dmSales.cdsSales.ApplyUpdates(0);
      { Fecha o cdsSales }
      dmSales.cdsSales.Close;
      { Libera da memória o dmSales }
      FreeAndNil(dmSales);
    end
  else
    { Caso não exista em memória ainda o dmCustomer emite uma mensagem }
    ShowMessage('Você deve clicar no novo antes de salvar um registro');
end;
```

### Listagem 7. Código do botão Cancelar Venda

```
procedure TfrmPrincipal.btnCancelarVendaClick(Sender: TObject);
begin
  { Verifica se já foi criada o dmSales }
  if dmSales <> nil then
    begin
      { Cancela todas as alterações realizadas no cdsSales }
      dmSales.cdsSales.Cancel;
      { Fecha o cdsSales }
      dmSales.cdsSales.Close;
      { Libera da memória o dmSales }
      FreeAndNil(dmSales);
    end
  else
    { Caso não exista em memória ainda o dmCustomer emite uma mensagem }
    ShowMessage('Você deve clicar no novo antes de cancelar um registro');
end;
```

## Conclusão

Embora os computadores de hoje possuam grande quantidade de memória RAM disponível, ainda é preciso que se tenha cuidado com o desenvolvimento, pois nem sempre as aplicações são executadas numa

rede local. Este artigo mostrou passo a passo uma técnica simples e eficaz para criar os componentes de acesso à dados em tempo de execução, o que torna a inicialização da aplicação mais rápida e ao mesmo tempo consome menos memória RAM. ●

+ de 80.000 membros cadastrados  
+ de 15.000 exemplos com fontes  
+ de 900 apostilas  
+ de 4.000 dicas  
Fórum Delphi  
Artigos

TOTALMENTE GRÁTIS

[www.delphi.eti.br](http://www.delphi.eti.br)

Um dos maiores sites de apoio a desenvolvedores Delphi do Brasil!!!



# Treinamento em ASP.NET – Parte 1

Primeiros passos com ASP.NET

O objetivo deste treinamento é mostrar a você, leitor, como criar aplicações para Internet por meio dos controles mais utilizados no dia-a-dia de uma página. A linguagem utilizada será o Delphi for .NET. A única ferramenta que você precisa é o Borland Developer Studio 8/2005/2006, na qual é possível criar desde o banco de dados até componentes, classes e WebServices. O ASP.NET vem conquistando enorme espaço entre os desenvolvedores devido a sua facilidade de uso, manutenção e performance.

O processamento das páginas ocorre no servidor, o que significa dizer que, quando o internauta navega nas páginas, estas são requisitadas ao servidor, que verifica a solicitação, realiza o parse das páginas, monta um HTML e as envia para o navegador que as solicitou.

E, já que mencionei HTML, não se preocupe, pois é possível desenvolver toda a aplicação sem usar nenhuma TAG de

HTML. Tudo o que você precisa saber é a linguagem Delphi for .NET. Cabe ressaltar que o Framework é o responsável por todo esse processo (por isso a necessidade de tê-lo instalado no servidor), juntamente, é claro, com o Internet Information Server.

No Borland Developer Studio 2006, selecione o menu *File|New>ASP.NET Web Application – Delphi for .NET* e crie um projeto do tipo ASP.NET Web Application chamado “DelphiMag”. Na caixa de diálogo (**Figura 1**) que se abre digite o nome da aplicação como citado anteriormente, configure seu caminho item *Location* e escolha o servidor no item de mesmo *Server*.

Clique no botão “OK” para criar automaticamente o site virtual dessa aplicação. Note que já existe um arquivo chamado *WebForm1.aspx*. Clique em *File>Save as...* e salve o arquivo como “Listas.aspx”. No fundo do documento, digite o texto “Controle de Listas ASP.NET” e utilize a barra



**Renato Haddad**

é Microsoft Most Valuable Professional (MVP). Autor de diversos treinamentos multimídia .NET e SQL Reporting Services para Microsoft Brasil e América Latina.

de ferramentas de formatação acima da página para melhorar a aparência do texto. Exiba a paleta de componentes do Delphi no menu *View>Tool Palette* (CTRL + ALT + P) caso não esteja visível e veja toda a lista de controles disponíveis.

Monte uma página com os seguintes controles: Um *Label* onde sua propriedade *text* será "Prato", um *TextBox* com a propriedade *ID* igual a "txtPrato", cinco *Buttons* com os *ID*'s iguais a "btnIncluir", "btnExcluir", "btnLimpar" e "btnPesquisar" respectivamente. Modifique também a propriedade *Text* de cada *Button* digitando o nome da função que cada um exercerá no exemplo. Em seguida adicione também um *ListBox* sendo que seu *ID* será ("lstCardapio") e *SelectionMode* igual a *Multiple*. Inclua um novo *Button* com *Text* igual a "Selecionados" e *ID* "btnSelecionados". Por último inclua um *Label* e altere suas propriedades *ID* para "lblItems" e *Text* para "". Observe na **Figura 2** uma sugestão de layout para o exemplo.

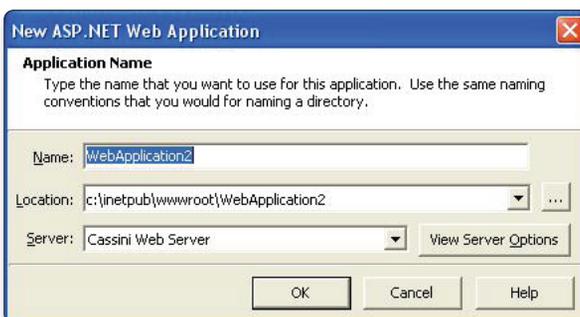
Para criar os códigos, dê um duplo clique em cada botão e digite os códigos relativos a eles apresentados na **Listagem 1**. A explicação dos códigos é mostrada nos próprios comentários das linhas.

Salve o arquivo e exiba novamente o formulário usando o atalho *F12*. No Delphi, defina esse arquivo como *Default Start Page* na janela *Project Manager*. Para compilar o formulário, selecione o menu *Project>Build DelphiMAG* ou pressione *CTRL + F9*. Para executar, selecione o menu *Run>Run* ou *F9*. Essa opção já salva, compila, abre o navegador e executa a aplicação. Veja o resultado da execução na **Figura 3**.

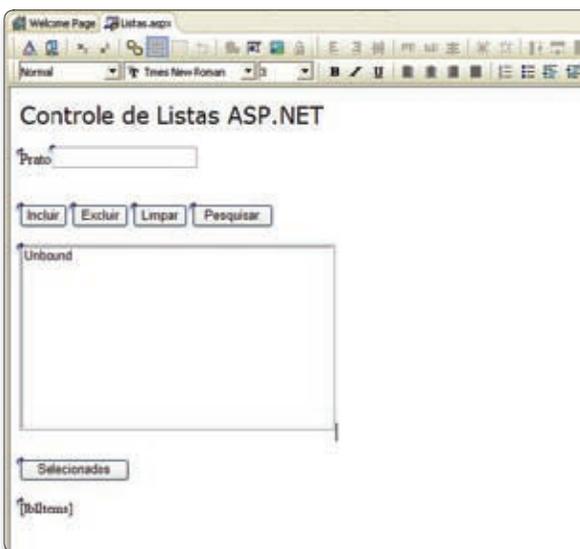
Você deve estar se perguntando onde é que entra o Delphi for .NET, ou seja, o código que criamos para cada botão. A resposta é simples: quando você compila a aplicação, é gerado um arquivo *DelphiMag.dll* contendo todos os códigos criados. Quando você abre uma página ASP.NET, ela é processada no servidor, que por sua vez identifica o navegador que a solicitou, monta um HTML e a envia ao navegador. As vantagens desse processo completo são muitas, dentre as quais destaco:

**Listagem 1.** Código dos botões Incluir, Excluir, Limpar, Pesquisar e Selecionados

```
procedure webListas.btnSelecionados_Click(sender: System.Object; e: System.EventArgs);
var
  I : Integer;
begin
  { Apaga o conteúdo do Label }
  lblItems.Text := '';
  { Adiona os itens selecionados ao Text do Label }
  for I := 0 to lstCardapio.Items.Count-1 do
    if lstCardapio.Items[I].Selected then
      lblItems.Text := lblItems.Text + '<br>' + lstCardapio.Items[I].Text;
end;
procedure webListas.btnPesquisar_Click(sender: System.Object; e: System.EventArgs);
begin
  { Desmarca qualquer item da lista }
  lstCardapio.SelectedIndex := -1;
  { Pesquisa o item na lista }
  lstCardapio.Items.FindByText(txtPrato.Text.Trim()).Selected := True;
  { Zera o campo }
  txtPrato.Text := '';
end;
procedure webListas.btnLimpar_Click(sender: System.Object; e: System.EventArgs);
begin
  { Limpa toda a lista }
  lstCardapio.Items.Clear();
end;
procedure webListas.btnExcluir_Click(sender: System.Object; e: System.EventArgs);
begin
  { Verifica se há um item selecionado }
  if lstCardapio.SelectedIndex >= 0 then
    { Remove o item da lista }
    lstCardapio.Items.RemoveAt(lstCardapio.SelectedIndex);
end;
procedure webListas.btnIncluir_Click(sender: System.Object; e: System.EventArgs);
begin
  { Verifica se o txtPrato contém algo digitado }
  if txtPrato.Text <> '' then
    { Inclui o item na lista }
    lstCardapio.Items.Add(txtPrato.Text.Trim());
    { Apaga o campo }
    txtPrato.Text := '';
end;
```



**Figura 1.** Criação de projeto ASP.NET no Delphi



**Figura 2.** Exemplo de formulário ASP.NET



### Listagem 2. Código do Page\_Load

```
procedure TWebForm1.Page_Load(sender: System.Object; e: System.EventArgs);
var
  ArrayMarca : Array[1..4] of string;
begin
  if not Page.IsPostBack then
  begin
    { Define um ArrayList com as opções }
    ArrayMarca[1] := 'Citroen';
    ArrayMarca[2] := 'Ford';
    ArrayMarca[3] := 'GM';
    ArrayMarca[4] := 'Volks';

    dropMarca.DataSource := ArrayMarca;
    { O DataBind preenche o controle }
    dropMarca.DataBind();
    { Insere um item na primeira opção do DropMarcas }
    dropMarca.Items.Insert(0, 'marca...');
  end;
end;
```

### Listagem 3. Código do evento SelectedIndexChanged do dropMarca

```
procedure TWebForm1.dropMarca_SelectedIndexChanged(sender: System.Object; e: System.EventArgs);
var
  tabModelos : DataTable;
  newRow : DataRow;
begin
  { Limpa o conteúdo do dropModelos }
  dropModelo.Items.Clear();
  { Define a tabela que será criada apenas na memória }
  tabModelos := DataTable.Create;
  { Define as colunas com os nomes e tipos }
  tabModelos.Columns.Add('codigo');
  tabModelos.Columns.Add('modelo');
  { Verifica qual o item selecionado }
  if dropMarca.SelectedItem.ToString() = 'GM' then
  begin
    { Cria uma nova linha }
    newRow := tabModelos.NewRow();
    { Define os conteúdos }
    newRow['codigo'] := '10';
    newRow['modelo'] := 'Vectra';
    { Adiciona a linha à tabela }
    tabModelos.Rows.Add(newRow);
  end
  else if dropMarca.SelectedItem.ToString() = 'Citroen' then
  begin
    { Cria uma nova linha }
    newRow := tabModelos.NewRow();
    { Define os conteúdos }
    newRow['codigo'] := '50';
    newRow['modelo'] := 'Xsara Picasso';
    { Adiciona a linha à tabela }
    tabModelos.Rows.Add(newRow);
  end
  else if dropMarca.SelectedItem.ToString() = 'Volks' then
  begin
    { Cria uma nova linha }
    newRow := tabModelos.NewRow();
    { Define os conteúdos }
    newRow['codigo'] := '32';
    newRow['modelo'] := 'Gol Special';
    { Adiciona a linha à tabela }
    tabModelos.Rows.Add(newRow);
  end
  else if dropMarca.SelectedItem.ToString() = 'Ford' then
  begin
    { Cria uma nova linha }
    newRow := tabModelos.NewRow();
    { Define os conteúdos }
    newRow['codigo'] := '80';
    newRow['modelo'] := 'Escort';
    { Adiciona a linha à tabela }
    tabModelos.Rows.Add(newRow);
  end;
  { Informa a origem do dropModelos que é a tabela
  definida com os respectivos modelos }
  dropModelo.DataSource := tabModelos;
  { Define o campo que será exibido no dropModelos }
  dropModelo.DataTextField := 'modelo';
  { Define o campo que será armazenado para cada item }
  dropModelo.DataValueField := 'codigo';
  { Preenche o dropModelos }
  dropModelo.DataBind();
end;
```

• O arquivo de layout ("Listas.aspx") fica separado do arquivo de códigos ("Listas.pas");

• Que a propriedade intelectual (códigos) fica protegida, pois está dentro da DLL;

• Quem se encarrega de montar um HTML que seja suportado pelo navegador é o Framework, e não você. Portanto, o código criado é apenas um, mas pode haver vários produtos finais.

No Delphi, adicione um novo formulário usando a opção *File|New|Other>New ASP.NET Files>ASP.NET Page* e dê o nome a ele de "Carros.aspx" usando para isso a opção *File>Save as....* Insira os seguintes controles: dois *DropDownList* com seus respectivos ID's iguais a "dropMarca" e "dropModelo". Marque como *True* a propriedade *AutoPostBack* do componente *dropMarca*. Isso é fundamental para que o formulário seja submetido ao servidor.

Insira também um *Calendar*. Clique com o botão direito nele e selecione *Auto format...*, selecione um layout de sua preferência e confirme. **Figura 4.**

Agora inclua um *Button* e defina sua propriedade *Text* como "Verificar Dados" e *ID* igual a "btnVerificar". Logo abaixo coloque um *Label* ("lblCarro"). Veja um exemplo de layout na **Figura 5.**

Como esses controles são utilizados com frequência, o funcionamento ocorre da seguinte forma: no momento em que o formulário é carregado, apenas na primeira vez é montado o *DropMarcas*. Assim que o internauta seleciona uma marca, é montado o *DropModelos* com os modelos específicos à marca selecionada. Em seguida, o internauta escolhe uma data no calendário e, quando pressiona o botão *Verificar*, os dados são capturados dos controles e seus conteúdos são exibidos no *Label*.

O *DropMarcas* será carregado assim que o formulário for aberto, portanto, digite o código da **Listagem 2** no *Page\_Load*.

Dê um duplo clique no *dropMarcas* e digite o código apresentado na **Listagem 3**. Como esse controle tem a propriedade *AutoPostBack* definida como *True*, assim que um item for selecionado, o formulário será enviado ao servidor, que montará o *dropModelo* de acordo com a marca selecionada.

Em seguida, crie o código para o *btnVe-*



ificar que exibe no `lblDados` o conteúdo da compra selecionada. Codifique como a seguir no evento `Click`:

```
lblDados.Text := 'Você selecionou a Marca: ' +
  dropMarca.SelectedItem.Text + ', o modelo: ' +
  dropModelo.SelectedItem.Text + ', código: ' +
  dropModelo.SelectedItem.Value +
  ', data de compra: ' +
  Calendar1.SelectedDate.ToShortDateString
```

Defina `Carros.aspx` como página `Default` assim como fizemos com a página `Listas.aspx`. Salve, compile e execute a página no navegador. Veja o resultado na **Figura 6**.

## Conclusão

O ASP.NET é atualmente o meio mais produtivo e rápido de se criar aplicações para Internet, além, é claro, de oferecer melhor performance e facilidade de manutenção. No próximo artigo, abordaremos outros controles e acesso a banco de dados. Aproveitem a leitura, e até lá! “No stress, think .NET”. ●



Figura 3. Exemplo de formulário ASP.NET em execução

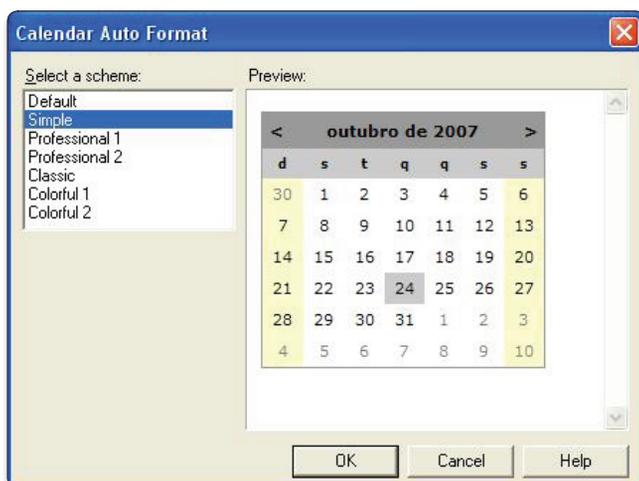


Figura 4. Escolha de layout para o componente Calendar

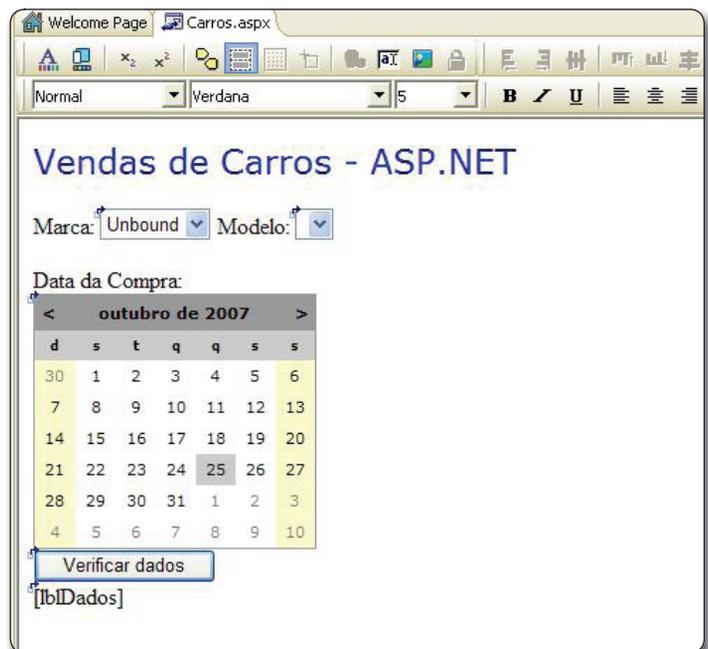


Figura 5. Exemplo de formulário de Carros

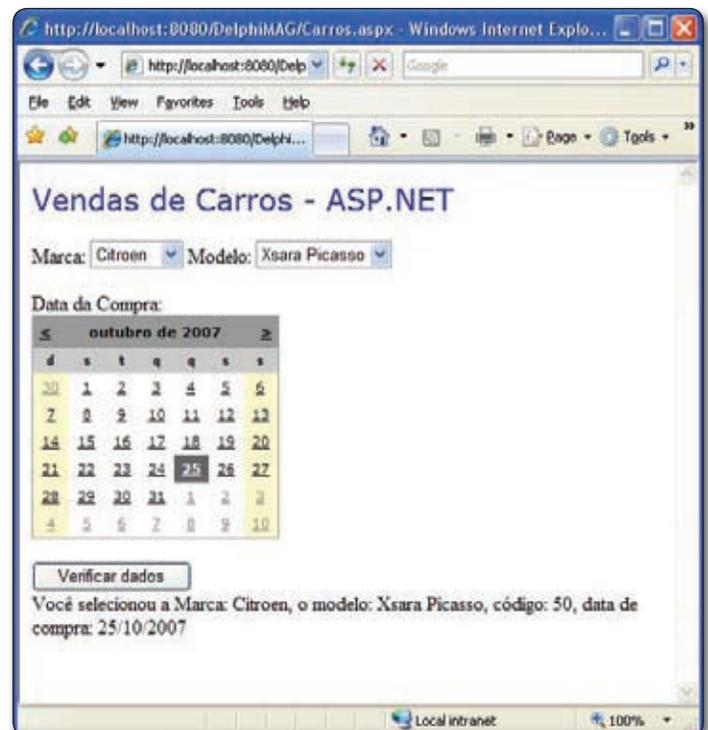


Figura 6. Formulários Carros.aspx em execução

## Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

### Delphi for PHP

#### Crie um site de publicação de notícias – Parte 3



**Fabrício Desbessel**

([fabricao@fabricao.pro.br](mailto:fabricao@fabricao.pro.br))

é professor de Linguagem de Programação do Curso Técnico em Informática do Colégio Frederico Jorge Logemann de Horizontina/RS e da FAHOR Faculdade Horizontina. Delphiano de coração, está sempre disposto a provar que com o Delphi sempre teremos a melhor solução. Site [www.fabricao.pro.br](http://www.fabricao.pro.br).

**D**ando seqüência ao mini-curso Delphi for PHP, onde estamos vendo como criar um site de notícias, vamos criar uma opção de envio de fotos para cada notícia. Na verdade, criaremos um gerenciador de arquivos, onde será possível enviar e excluir imagens para cada notícia.

Outra implementação será a segurança da aplicação onde criaremos login e restrição de páginas pelo nível do usuário. Criaremos uma forma de diferenciar usuários comuns de administradores através de níveis de acesso. Isso é importante, pois determinamos como e quem pode ter acesso ao sistema. Então, coloque a mão na massa e vamos à prática.

#### Upload e instalação do componente EditUploadFile

Para permitir o envio de fotos das notícias, é necessário criar uma forma de upload de arquivos do tipo jpeg. Infelizmente o Delphi For PHP não trouxe um

componente específico para isso, mas atendendo às solicitações dos usuários, o pessoal da CodeGear criou e disponibilizou o componente *EditUploadFile*. Baixe o componente do link [codecentral.borland.com/Item/24670](http://codecentral.borland.com/Item/24670) e descompacte-o em uma pasta de sua preferência. Localize o arquivo *edituploadfile.inc.php* e copie-o para a pasta de instalação da VCL, normalmente localizada em *C:\Arquivos de programas\CodeGear\Delphi for PHP\1.0\vcl*.

O arquivo *edituploadfile.package.php* deve ser copiado para a pasta *package* dentro do diretório da VCL, e o arquivo *edituploadfile.bmp* para dentro da pasta *icons* dentro de *packages*.

Agora é necessário instalar o componente no IDE. Para tanto, acesse o menu *Component>Packages*, clique no botão *Add*, localize o arquivo *edituploadfile.package.php*, clique em *Abrir* e depois confirme. Verifique a paleta de componente *Standard* para confirmar que o componente foi instalado corretamente. (Figura 1)

Agora abra o formulário *noticias.php* e insira um *ListBox* da paleta de componentes *Standard*. Insira um *EditUploadFile*, um *Button* com o *Caption* definido como "Enviar" para efetivamente fazer o envio da foto, pois o componente, na verdade, só cria um botão para procurar arquivos na máquina, e outro *Button* com o *Caption* "Excluir".

Com esses componentes o usuário poderá enviar fotos para um diretório que será criado, utilizando como nomenclatura o código da notícia, exibir os arquivos enviados em um *ListBox* e permitir que exclusões sejam efetuadas. Veja na **Figura 2** como ficou o layout do formulário de cadastro de notícias.

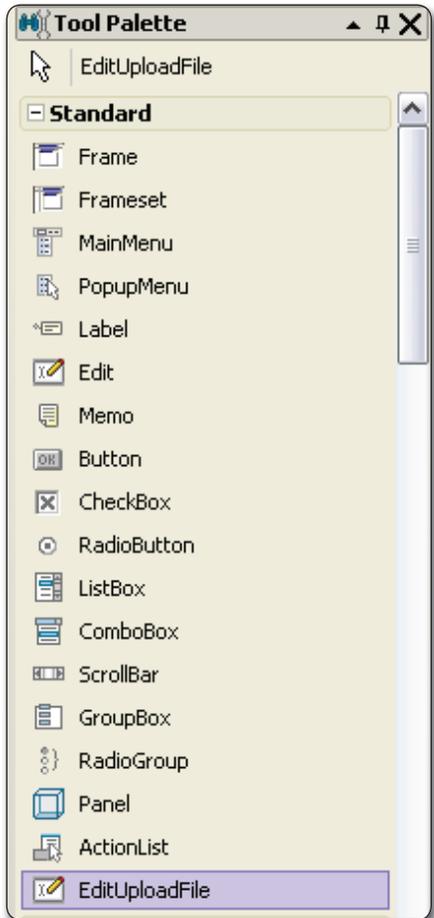
Agora vamos aos códigos. Primeiramente, codifique o botão *Enviar* conforme a **Listagem 1**.

No código da **Listagem 1** utilizou-se uma variável *\$diretorio* que deverá ser modificada conforme a localização do projeto em sua máquina. Achei interessante criar um subdiretório *fotos* para separar melhor os arquivos. Como co-

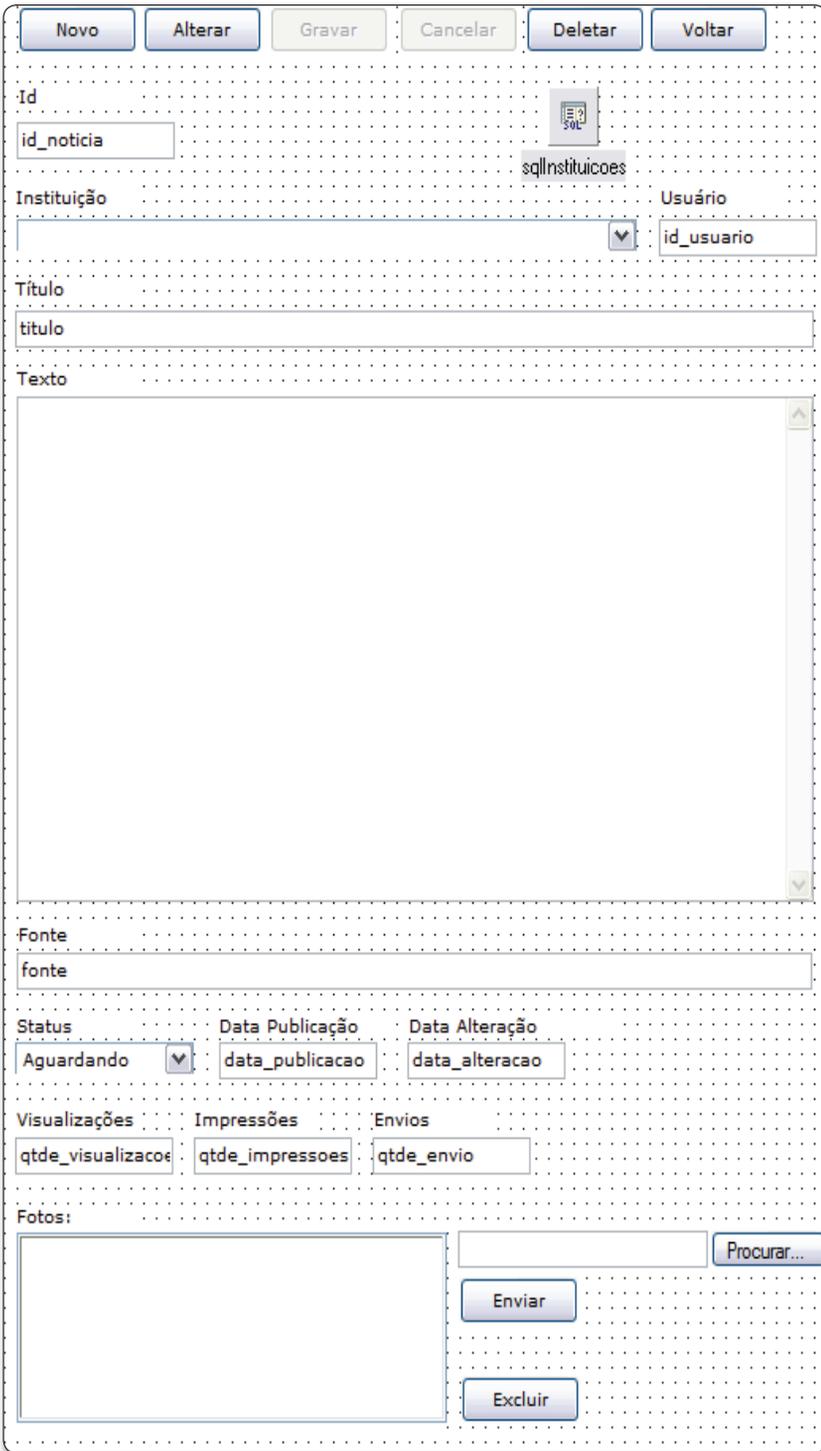
**Listagem 1.** Código do botão *Enviar*

```

global $UDados;
if ($_FILES['EditUploadFile1']['name'] != "")
{
    $diretorio = "C:/CD/Jornal/fotos/n".$UDados->tbnoticias1->id_noticia."/";
    if (!is_dir($diretorio)){
        mkdir($diretorio);
    }
    $nome_arquivo = $diretorio . $_FILES['EditUploadFile1']['name'];
    move_uploaded_file($_FILES['EditUploadFile1'] ['tmp_name'], $nome_arquivo);
}
    
```



**Figura 1.** Paleta de Componentes *Standard* com o *EditUploadFile*



**Figura 2.** Formulário de cadastro de notícias com componentes necessários para upload e exclusão de fotos

mentado anteriormente, pega-se o código da notícia para criar um subdiretório com o nome iniciando com a letra "n" e seguindo com o código propriamente dito. Lembre-se também que todos os usuários deverão ter permissão de escrita e leitura nessa pasta para que o sistema efetivamente faça o *upload* do arquivo. Inicialmente faz-se um teste para verificar se o diretório existe e, caso ainda não exista, cria-se o mesmo usando a função *mkdir*.

A função *move\_uploaded\_file* é a responsável por efetivamente fazer o *upload* do arquivo. Para utilizá-la, é necessário passar como parâmetro o arquivo e o caminho de destino. Para passar o parâmetro, usamos a variável *\$FILES* com o nome do componente, acrescentando *tmp\_name* para efetivamente pegar o nome físico do arquivo.

Antes de testar o envio de uma foto, precisamos alterar o formato de envio das informações para que o browser saiba que o formulário estará enviando dados e arquivos. Para isso acesse o evento *OnBeforeShow* do formulário *noticias.php* e acrescente a seguinte linha de código ao já existente. (código da parte II):

```
$this->FormEncoding="multipart/form-data";
```

Agora você já pode abrir uma notícia existente e fazer o teste enviando fotos para a mesma. Depois vá até o diretório "fotos" que está dentro do diretório da aplicação e veja se foi criada uma pasta iniciando com a letra "n" e terminando com o código da notícia. Para que as fotos de cada notícia apareçam no *ListBox*, codifique o seu evento *OnBeforeShow* conforme a **Listagem 2**.

No código da **Listagem 2** utiliza-se um laço de repetição (*while*) para percorrer o diretório em busca de arquivos. Inicialmente é preciso testar o *filetype* para impedir a listagem de subdiretórios e depois utiliza-se a função *getimagesize* para confirmar que o arquivo é de imagem. Isso impedirá que arquivos que não são imagens sejam exibidos no *ListBox*. Para finalizar, codifique o botão *Excluir* conforme a **Listagem 3**.

No código da **Listagem 3** verifica-se a existência de algum item selecionado no *ListBox* e, se existir, faz-se a exclusão do arquivo utilizando a função *unlink*.

Para segurança, é interessante fazer uma pergunta ao usuário para confirmar se ele tem certeza que deseja excluir o arquivo. Para fazer isso selecione o botão *Excluir*, acesse a aba *Javascript* no *Object Inspector* e, no evento *OnClick*, abaixo do comentário *//Add your javascript code here*, digite o seguinte código:

```
return(confirm("Você tem certeza que deseja excluir a foto selecionada?"));
```

Esse código vai incluir uma função Javascript que exibirá uma mensagem ao usuário com a opção de continuar a exclusão ou cancelá-la. Conforme a escolha do usuário, o *post* da página será executado ou não. Pronto! Agora você colocou à disposição do usuário a manutenção de fotos para as notícias de forma bem organizada, criando um diretório específico para cada notícia e sem limitar a quantidade. O usuário poderá enviar quantas fotos quiser para a notícia.

## Formulário Principal

Chegou a hora de criarmos o formulário principal, que será a primeira página dos usuários administradores do site. Acesse o menu *File|New>Form*, mude a propriedade *Name* para "UPrincipal" e salve a *Unit* como "principal.php".

Nesse formulário você pode colocar o logotipo de sua empresa ou cliente, bem como ajustar sua apresentação visual. Esse formulário servirá como caminho de acesso aos formulários da aplicação, e para isso acontecer, utiliza-se um componente *MainMenu*.

Adicione um *MainMenu* nesse formulário e, para criar as opções, acesse a propriedade *Items* que abrirá um editor conforme mostra a **Figura 3**.

Inicialmente, o editor já vem com o primeiro item adicionado bastando apenas colocar o texto. Para incluir um sub-item clica-se no botão *New SubItem*. Então o trabalho baseia-se em adicionar

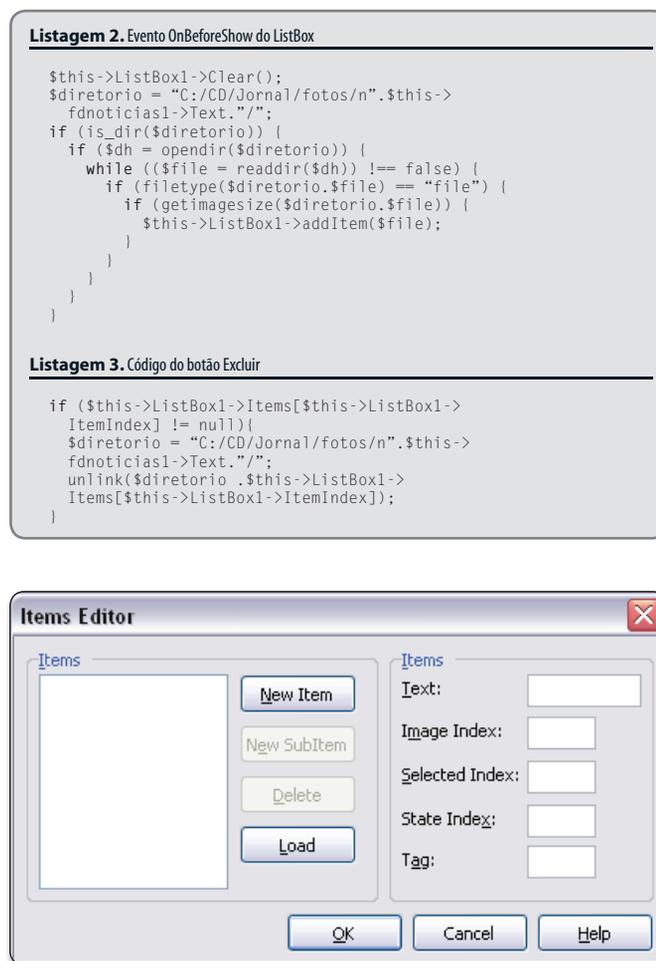


Figura 3. Editor de itens do menu

itens e sub-itens informando o *Text*, que é a informação que aparece no menu, e informar a propriedade *Tag*.

A propriedade *Tag* será utilizada para saber em qual menu o usuário realmente clicou, uma vez que o componente possui um único evento *OnClick*, bem diferente do *MainMenu* do Delphi 7.

Então, para esse sistema, montei um menu com a estrutura apresentada na **Tabela 1**. Veja também na **Figura 4** como o menu será estruturado. Utilize a mesma estrutura e, principalmente, os mesmos valores para a propriedade *Tag*, para não termos problemas nas codificações futuras.

Acesse o evento *OnClick* do menu e codifique conforme a **Listagem 4**.

Na **Listagem 4** há um teste para cada opção do menu, verificando a propriedade *Tag* através da variável *\$params* que o evento recebe. Nesse código não foi incluído ação para o menu "Sair" (Tag 41), pois isso será codificado depois da implementação do controle de acesso ao sistema.

### Controle de Acesso

Agora vamos criar um controle de acesso ao sistema, com formulário de login e, principalmente, restrição de acesso às páginas pelo nível do usuário.

Comece com a criação do formulário de login. Acesse o menu *File\New>Form*, altere a propriedade *Name* para "ULogin" e salve a *Unit* como "login.php". Adicione ao formulário dois *Label*'s e dois *Edit*'s para solicitar ao usuário o nome de usuário e senha. Altere a propriedade *Name* dos *Edit*'s para "edtUsuario" e "edtSenha", respectivamente. Essa alteração de *Name* é necessária para a codificação funcionar tanto para você como para mim. Adicione um *Button* e altere seu *Caption* para "Logar". Para não exibir a senha é necessário definir a propriedade *IsPassword* do *edtSenha* para *True*. Abaixo do botão insira mais um *Label*, altere seu *Name* para "lblMensagem" e apague sua propriedade *Caption*.

Abaixo coloque um outro *Label*, mude o *Name* para "lblUrl" e também apague o *Caption*. Esse último *Label* irá servir para guardar a URL da página que chamou o login e, conseqüentemente, para onde o usuário será redirecionado se o login for

aceito. Nesse label você pode configurar a propriedade *Visible* para *False*, escondendo essa informação do usuário final. No final dessa configuração o layout do formulário de login deve ser parecido com a **Figura 5**.

Antes de codificar precisamos adicionar um componente *Query* ("sqlUsuarios") ao formulário de login, para fazer a busca no banco de dados. Para ligá-lo à propriedade *DataBase* lembre-se de acessar o menu *File>Use unit* e escolher *dados.php*. Depois,

ligue a propriedade *DataBase* ao *UDados.dbjournal1*. Altere a propriedade *Name* para "sqlUsuarios" e na propriedade *SQL* insira a seguinte instrução:

```
select * from usuarios
```

Clique no botão Logar e codifique o evento conforme a **Listagem 5**.

Na **Listagem 5** passa-se um filtro para *sqlUsuarios* utilizando o campo e-mail, que servirá como nosso login, e também a senha. Então se o usuário não passar

```
Listagem 4. Evento OnClick do MainMenu

if ($params['tag']==11) {
    redirect('noticias.php');
}

if ($params['tag']==12) {
    redirect('pesNoticias.php');
}

if ($params['tag']==21) {
    redirect('instituicoes.php');
}

if ($params['tag']==22) {
    redirect('pesInstituicoes.php');
}

if ($params['tag']==31) {
    redirect('usuarios.php');
}

if ($params['tag']==32) {
    redirect('pesUsuarios.php');
}

if ($params['tag']==41) {
    redirect('sair.php');
}
```

Ítem	Sub-ítem	Tag
Notícias	Nova	11
	Pesquisar	12
Instituições	Nova	21
	Pesquisar	22
Usuários	Novo	31
	Pesquisar	32
Sistema	Sair	41

Tabela 1. Itens do menu



Figura 4. Formulário principal (principal.php)



Figura 5. Formulário de Login

corretamente o e-mail e a senha para o acesso, será retornada a mesma página de login exibindo uma mensagem de usuário ou senha incorreta. Para evitar *SQL Injection* (utilização de aspas para injetar instrução SQL) e burlar a segurança, utiliza-se a função *addslashes* que adiciona barras de escapes antes de aspas.

---

**Nota:** *SQL Injection* é uma técnica muito utilizada por Hackers para executar comandos SQL no banco a partir de controles de entrada de dados.

---

Se *sqlUsuarios* retornar algum registro o login será aceito e, portanto coloca-se em variáveis de sessão (*\$\_SESSION*) o id, nome e nível do usuário. Se o usuário chamou diretamente a página de login ele será redirecionado para página *principal.php*. Caso ele tenha tentado acessar uma outra página do sistema, o que resultou no direcionamento dele

para o login, ele será encaminhado para essa página. Para esse redirecionamento funcional precisamos ainda colocar o código a seguir no evento *OnBeforeShow* da página de login:

```
$this->lblUrl->Caption = $_GET['url'];  
$this->lblMensagem->Caption = "<font  
color='red'>".$_GET['erro']. "</font>";
```

No código anterior, buscamos valores passados no link da página através de campos de consulta ("*\$\_GET*"). Esses valores serão enviados quando alguma página chamar o login, informando a página de destino do usuário, após o login ser aceito. Nesse momento uma mensagem de sucesso é enviada para o usuário.

---

**Nota:** Note que nessa mensagem pode-se utilizar tags HTML para formatação de fontes. Isso é possível em qualquer componente que tenha a propriedade *Caption*.

---

Falta ainda fazer a exigência de login em todas as páginas. Para tanto, cria-se uma função em uma nova *Unit*. Então acesse o menu *File!New>Unit* ou usando o atalho *Ctrl+N* para criar um arquivo de *Unit* e salve-o com o nome "funcoes.php". Nesse arquivo poderemos adicionar funções que serão utilizadas por vários formulários da aplicação. Então, codifique a função "seguranca" conforme é exibido na **Listagem 8**.

A função *seguranca* verifica se o usuário já está logado, testando se existe uma variável de sessão chamada *nome* ("*\$\_SESSION['nome']*"). Lembrando que quando é efetuado o login, essa variável é criada na aplicação. Se ela não existir, o usuário é redirecionado através da função *redirect*. Foi necessário utilizar a função *substr* para tirar uma barra ("/") do nome da página que chamou o login. Se o usuário estiver logado, verifica-se seu nível de permissão buscando o valor de uma variável de sessão, e testando com o nível exigido como parâmetro. Por isso, para funcionar é interessante utilizar um parâmetro numérico para definir o nível e, com isso, coloca-se um valor maior para o usuário administrador que acessará as páginas.

Depois da função *seguranca* ter sido criada é necessário em cada página da aplicação, exceto *login.php*, no evento *OnBeforeShow* adicionar, a chamada da função abaixo:

```
seguranca($_SERVER['PHP_SELF'], 1);
```

Para a chamada da função funcionar é preciso acessar o menu *File>Use Unit* e escolher o arquivo *funcoes.php* pois é lá que estará a função. Na chamada utilizamos a variável *\$\_PHP\_SELF* para passar como parâmetro a página que está chamando a função, que é para onde o usuário será redirecionado depois do login.

O segundo parâmetro é o nível do usuário, onde pode-se definir para cada página um nível diferente. Nesse exemplo estou exigindo o nível "1" na página *principal.php*, nível "2" nas páginas de pesquisa e o nível "3" nas páginas de manutenção de dados. Então se um usuário de nível "1" tentar

---

**Listagem 5.** Evento *OnClick* do botão Logar

---

```
$this->sqlUsuarios->Filter="email='".addslashes(  
    $this->edtUsuario->Text)."' AND senha='".  
    addslashes($this->edtSenha->Text)."'";  
$this->sqlUsuarios->Open();  
if ($this->sqlUsuarios->RecordCount > 0){  
    $_SESSION['id_usuario'] = $this->sqlUsuarios->id_usuario;  
    $_SESSION['nome'] = $this->sqlUsuarios->nome;  
    $_SESSION['nivel'] = $this->sqlUsuarios->nivel;  
    if ($this->lblUrl->Caption != NULL){  
        redirect($this->lblUrl->Caption);  
    } else {  
        redirect('principal.php');  
    }  
} else {  
    $this->lblMensagem->Caption="<font color='red'>Usuário ou senha incorreta!  
    </font>";  
}  
$this->sqlUsuarios->Close();
```

---

**Listagem 6.** Função *seguranca*

---

```
function seguranca($pagina, $nivel){  
    if (!($_SESSION['nome'])){  
        redirect('login.php?url=' . substr($pagina, 1, strlen($pagina)-1));  
    } else {  
        if ($_SESSION['nivel'] < $nivel){  
            redirect('login.php?erro=Sem permissão para essa página');  
        }  
    }  
}
```

---

**Listagem 7.** Código acrescentado no *OnClick* do *MainMenu*

---

```
if ($params['tag']==41) {  
    $_SESSION['id_usuario']=NULL;  
    unset($_SESSION['id_usuario']);  
    $_SESSION['nome']=NULL;  
    unset($_SESSION['nome']);  
    $_SESSION['nivel']=NULL;  
    unset($_SESSION['nivel']);  
    redirect('login.php');  
}
```



acessar uma página de pesquisa ou manutenção será redirecionado para página de login onde será apresentada uma mensagem de erro, dizendo que o usuário não possui permissão para acessar essa página.

Pronto! Agora você já pode começar a fazer os testes. Crie três usuários, sendo um para cada nível. Lembre-se que o usuário de nível "1" poderá somente acessar a página *principal.php*. Os usuários de nível "2" poderão acessar a página *principal.php* e também as páginas de pesquisa, e os de nível "3" poderão acessar todas as páginas. Qualquer acesso indevido deve ser redirecionado para a página de login. Tente também executar uma página que não seja a de login para certificar-se que o login será exigido.

Na página *principal.php* coloque um *Label* e altere a propriedade *Name* para "lblUsuario". Nesse *Label* vamos exibir uma mensagem contendo o nome do usuário que logou-se na aplicação. Acesse o evento *OnBeforeShow* desse *Label* e coloque o seguinte código:

```
$this->lblUsuario->Caption =
    "Usuário logado: ".$_SESSION['nome'];
```

Veja na **Figura 6** um usuário devidamente logado no sistema.

Para finalizar vamos codificar a ação *Sair* do menu. Então acesse o evento *OnClick* do *MainMenu* do formulário principal e acrescente, ao código existente, o código da **Listagem 7**.

No código da **Listagem 7** passamos o valor *NULL* para todas as variáveis de sessão, bem como destruímos com o comando *unset*. Sua aplicação está com toda a parte administrativa funcionando.

**Conclusão**

Agora temos a aplicação com o *Backend* funcionando e preparado para receber as notícias. Na próxima etapa do minicurso vamos trabalhar na parte acessível por todos os usuários, o *Frontend*, onde serão aplicados *templates* ("modelos") e apresentaremos nossas notícias. Então, não perca a próxima parte. ●



Figura 6. Usuário logado no sistema

# PENSE...

QUANTO TEMPO  
VOCÊ GASTARIA  
PARA DESENVOLVER  
COBRANÇA COM BOLETOS  
BANCÁRIOS PARA  
APENAS UM BANCO  
NO SEU SOFTWARE

## COBREBEMX




-  56 BANCOS E MAIS DE 430 CARTEIRAS DE COBRANÇA PARA IMPRESSÃO E/OU ENVIO DE BOLETO BANCÁRIO POR EMAIL;
-  GERAÇÃO DE BOLETOS ON LINE;
-  GERAÇÃO E LEITURA DE ARQUIVOS (REMESSA/RETORNO) NOS PADRÕES FEBRABAN E CNAB;
-  MAIS DE 40 EXEMPLOS EM DIVERSAS LINGUAGENS DE PROGRAMAÇÃO

DOWNLOADS E INFORMAÇÕES EM [WWW.COBREBEM.COM](http://WWW.COBREBEM.COM)







# Introdução ao PHP – Parte 2

## Tipos de Dados e arrays

Ewald Geschwinde  
Hans-Juergen Schoenig

Nas seções anteriores, nós já vimos que o PHP suporta variáveis, mas por enquanto só aprendemos a utilizar números e seqüências. Como estruturas de controle, os tipos de dados são um componente fundamental de toda linguagem de programação.

Tal como a maior parte das linguagens de alto nível, o PHP trabalha normalmente sem tipos de dados e tenta reconhecer o tipo de dado de uma variável por conta própria. Contudo, o PHP fornece um conjunto de tipos de dados diferentes, como mostrado na **Tabela 1**. Nesta seção, aprenderemos a utilizar estes tipos de dados eficientemente.

### Variáveis e nomes

Como já vimos nos exemplos discutidos nas seções anteriores, as variáveis em PHP são marcadas com o caractere "\$". Diferentemente do Perl, o PHP marca todas as variáveis com um caractere "\$" e não usa os caracteres "@" e "%" como símbolos.

Em PHP, os nomes de variáveis são sensíveis, o que significa que faz diferença se o nome é escrito com caracteres minúsculos ou maiúsculos. O nome de uma variável pode iniciar com um caractere de "a" a "z" ou com o caractere "\_" ("underscore"). Depois do primeiro caractere, todas as demais letras e números, assim como o caractere "\_" serão permitidos. Contudo, recomendo fortemente utilizar só letras no nome da variável, pois caso contrário, os programas ficarão confusos e difíceis de entender, sobretudo quando se misturam letras maiúsculas

Tipo de dado	Descrição
Int, Integer	Numéricos, valores sem virgule
Double, real	Numéricos de ponto flutuante
String	Seqüência de caracteres
Array	Conjunto de valores
Object	Tipo de dado artificial, definido pelo usuário

**Tabela 1.** Tipos de dados no PHP

e minúsculas em nomes de variáveis. O exemplo a seguir, mostra que \$a não será interpretado como \$A:

```
<?php
$a = "Eu sou uma seqüência.";
$A = "Eu também sou uma seqüência.";
echo "a: $a; A: $A<br>\n";
?>
```

Como podemos ver no resultado, as duas frases não representam a mesma variável.

## Reconhecimento automático de tipo de dados

Quando uma variável é processada pelo PHP, o interpretador segue algumas regras pré-estabelecidas para reconhecer que tipo de dado uma variável pertence:

- Se uma seqüência de caracteres inicia e termina com aspas simples, será tratada como um literal do tipo string: \$a = 'abc' - atribui a seqüência "abc" a \$a, pois a seqüência está entre aspas simples;
- Se um número não contiver um ponto decimal, é tratado como um valor inteiro: \$a = 23, indica que o número inteiro 23 será atribuído a \$a. O valor 23 é um valor inteiro porque não contém nenhum ponto decimal;
- Se um número contiver um ponto decimal, será tratado como um número de ponto flutuante: \$a = 3.14159 faz com que o PHP atribua um valor ponto flutuante a \$a;

O conhecimento dessas três regras torna fácil a previsão do resultado de uma operação e facilita a escrita de aplicações sem erros de codificação.

## Verificação de tipos de dados e formatação explícita

Às vezes é necessário modificar explicitamente o tipo de dado de uma variável. Embora o PHP faça internamente a formatação de tipos em alguns casos poderia ajudar, caso seja necessário que uma variável seja atribuída com determinado tipo de dado. Para executar operações assim, o PHP fornece as funções necessárias. Antes de aprendermos sobre essas funções, veremos como podemos descobrir o tipo de dado de uma variável na **Listagem 1**.

Com o auxílio da função chamada *get-type*, é possível recuperar facilmente o tipo de dado de uma variável. Se utilizarmos as regras descritas na seção anterior,

### Listagem 1. Verificando o tipo de uma variável

```
<?php
$a = 3.14159;
$b = 12;
$c = "Eu sou uma seqüência.";
$d = array(1, 2, 3);

echo '$a is a ' . gettype($a) . " value<br>\n";
echo '$b is an ' . gettype($b) . " value<br>\n";
echo '$c is a ' . gettype($c) . " value<br>\n";
echo '$d is an ' . gettype($d) . " value<br>\n";
?>
```

### Listagem 2. Outra forma de verificar um tipo de dado de uma variável

```
<?php
$a = 3.14159;
$b = 12;
$c = "Eu sou uma seqüência.";
$d = array(1, 2, 3);

echo '$a: ' . is_double($a) . "<br>\n";
echo '$b: ' . is_integer($b) . "<br>\n";
echo '$c: ' . is_string($c) . "<br>\n";
echo '$d: ' . is_array($d) . "<br>\n";
echo '$d: ' . is_object($d) . "<br>\n";
?>
```

### Listagem 3. Exemplo de formatação de variáveis

```
<?php
$a = 3.14159;
$b = 12;

$a_cast = (int)$a;
$b_cast = (double)$b;

echo "a_cast: $a_cast<br>";
echo "b_cast: $b_cast<br>";
?>
```

será fácil prever o resultado do script:

```
$a is a double value
$b is an integer value
$c is a string value
$d is an array value
```

O primeiro valor é um valor ponto flutuante porque contém uma vírgula. O segundo é um valor inteiro porque não contém uma vírgula. O terceiro está entre aspas e por isso é tratado como um string. O último valor é um array, pois a expressão no lado direito do caractere "=", retorna um array de valores inteiros.

Outro modo de verificar o tipo de dado de uma variável seria utilizar a função específica de tipo de dados como mostrado na **Listagem 2**.

Utilizamos as funções *is\_double*, *is\_integer*, *is\_string*, *is\_array* e *is\_object*. Se o valor passado para a função tiver o tipo de dado apropriado, será retornado 1. A listagem a seguir, contém o resultado do script que acabamos de ver:

```
$a: 1
$b: 1
$c: 1
$d: 1
$d:
```

Todos os valores menos o último serão "true". A última linha mostra que \$d não

é um objeto.

Se executarmos uma formatação explícita, poderemos facilmente perder os dados armazenados na variável que desejamos formatar. Isto não representa um ponto fraco do PHP, pois a formatação não poderia ser feita de outra forma pelo interpretador PHP. (**Listagem 3**).

A variável \$a é um número de ponto flutuante que é formatado para número inteiro e exibido na tela. A variável \$b é formatada como double e exibida também na tela. Se observarmos o resultado, veremos que \$a\_cast não será tão preciso como \$a\_cast, pois não é permitido que um valor inteiro armazene os dígitos após a vírgula. As variáveis \$b\_cast e \$b serão iguais, pois os números inteiros também podem ser representados como valores double sem perda de dados:

```
a_cast: 3
b_cast: 12
```

Algumas regras gerais e dicas devem ser levadas em consideração quando forem executadas formatações. A **Tabela 2** contém uma compilação das informações mais importantes a respeito de formatações.

## Regras em PHP

Nessa tabela, podemos facilmente constatar os perigos que iremos enfrentar executando operações de formatação.

## Funções

As funções são a base de toda linguagem de programação. Elas são utilizadas para isolar certas tarefas de um programa e ajudar o programador a reutilizar o mesmo código em vários lugares sem necessidade de copiá-los. As funções são chamadas com parâmetros e executam um conjunto fixo de instruções. Se necessário, uma função pode retornar valores para a função ou objeto que a chamou.

Nós já trabalhamos com funções. Uma das funções com as quais já lidamos é a função *gettype*, que retorna o tipo de dado de uma variável. Esta função é um bom exemplo a respeito do que o tópico todo aborda: passamos uma variável e um valor é retornado.

Nos últimos anos, a expressão função tem sofrido alguma perda de popularidade, pois no paradigma da programação orientada por objetos, ela não mais existe.

## Funções de saída e trabalhando com strings

Assim como no Pascal, uma linguagem de programação especificada pelo professor Niklaus Wirth em 1970, o PHP suporta um tipo de dado chamado *string*. As *strings* consistem de um conjunto de

caracteres e muitas vezes são utilizadas para armazenar texto. Diferentemente do C/C++, a manipulação de *strings* é uma tarefa fácil em PHP, pois o programador não tem que se preocupar com o gerenciamento de memória e com as falhas de segmentação que ocorrem por causa de problemas de alocação de memória. Isto torna o PHP conveniente para o desenvolvimento de software rápido e confiável.

A lista de funções utilizadas para trabalhar com *strings* em PHP parece ser infinita. Nesta seção, tentaremos cobrir as mais importantes.

Para exibir caracteres ASCII na tela, o PHP provê uma função chamada *chr*. Simplesmente passamos o código ASCII para a função e o caractere desejado será retornado. O exemplo a seguir, mostra como o endereço de e-mail de um dos autores pode ser "facilmente" exibido utilizando a função:

```
<?php
echo chr("103").chr("117").chr("105").
chr("110").chr("116").chr("104").
chr("101").chr("114").chr("64").
chr("100").chr("101").chr("118").
chr("109").chr("101").chr("100").
chr("105").chr("97").chr("46").chr("99").
chr("111").chr("109").chr("46").
chr("98").chr("114")."<br>\n";
?>
```

E o resultado é

```
guinther@devmedia.com.br
```

Temos certeza que existem modos mais eficientes de exibir endereços de e-mail. Normalmente a função *chr* é utilizada

para exibir caracteres que não são encontrados no teclado.

A contrapartida da função *chr* é a função *Ord*, que pode ser utilizada para mostrar o valor ASCII de um caractere:

```
<?php
$a = "a";
echo Ord($a);
?>
```

Atualmente a encriptação de dados é uma tarefa extremamente importante, pois os padrões de segurança estão aumentando e os piratas poderiam ameaçar as redes de computador. Com a chegada de todos aqueles vírus que se propagam através das redes, a segurança tornou-se um tópico importante. No exemplo a seguir, veremos como uma string pode ser facilmente encriptada utilizando as funções embutidas do PHP:

```
<?php
$a = "Eu sou uma seqüência.";
echo crypt($a);
?>
```

Simplesmente passamos a string que desejamos processar para a função *crypt*, e o resultado será codificado utilizando o algoritmo DES. O resultado será uma seqüência de caracteres que não conseguiremos ler:

```
i34vSr3Aueg2A
```

Como já vimos, o código HTML é um conjunto de caracteres especiais que têm certo significado. Um bom exemplo para isto é o caractere "<", que é utilizado para definir o ponto de partida de uma tag HTML. O questão é: o que podemos fazer quando desejamos exibir um caractere "<" na tela? O problema pode ser facilmente resolvido utilizando a função chamada *htmlspecialchars*, a qual converte todos os caracteres que têm certo significado ou cuja utilização não é permitida no código HTML correto. O seguinte trecho de código mostra como isto funciona:

```
<?php
$b = "1 > 0";
echo htmlspecialchars($b);
?>
```

A expressão "1 > 0" será exibida na tela, porém, o código HTML gerado pelo PHP, terá um aspecto ligeiramente diferente:

```
1 &gt; 0
```

Tipo de Dados de Resultado	Tipo de dados antes da Formatação	Alterações
Integer	Double	Os dados à direita do ponto decimal são omitidos - o valor não será arredondado.
Integer	String	Se nenhum número for reconhecido, será retornado 0.
Double	Integer	Sem alteração.
Double	String	Se nenhum número for reconhecido, será retornado 0.
String	Integer	Retorna o número como uma seqüência de caracteres.
String	Double	Retorna o número como uma seqüência de caracteres.
Array	Object	Será diretamente convertido.
Array	Integer	Será criado um array de valores inteiros.
Array	String	Será criado um array de valores string.
Array	Double	Será criado um array de valores de ponto flutuante.
Object	Array	Será diretamente convertido.
Object	Integer	Será gerado um object com as mesmas propriedades de um valor inteiro.
Object	String	Será gerado um object com as mesmas propriedades de um valor string.
Object	Double	Será gerado um object com as mesmas propriedades de um valor double.

Tabela 2. Tipos de dados no PHP

O caractere ">", foi substituído pela seqüência "&gt;" e é exatamente isto que nós desejamos.

Outro caractere importante na lista de símbolos especiais é o símbolo "&". No código HTML o símbolo "&" deve ser codificado como "&amp;". Como podemos ver no exemplo a seguir, a função *htmlspecialchars* toma conta deste particular:

```
<?php
    $b = "Sr. Adriano Santos &
        Sr. Guinther Pauli são do Brasil";
    echo htmlspecialchars($b). "<br>\n";
?>
```

Se observarmos o código HTML gerado, podemos ver que o caractere '&' foi substituído por "&amp;".

```
Sr. Adriano Santos &amp;
Sr. Guinther Pauli são do Brasil.
```

A função *printf* é a função "antediluviana" para processamento de textos e strings. Ela permite exibir o texto formatado na tela. Os programadores C estarão familiarizados com esta função. A versão PHP da função *printf* é muito semelhante à função do C. Vejamos um exemplo na **Listagem 4**.

Na primeira linha definimos uma variável que contém o nome do autor. Na linha seguinte, desejamos exibir esta variável como parte de uma seqüência no texto. Por isso, utilizamos a função *printf*. Os caracteres "%s" são substituído por \$a. A string \$a é interpretada como um literal e é exibida na tela como um texto. A seguir, \$b é definido como um valor inteiro. Depois disto, \$b é exibido como valor inteiro utilizando '%d'.

Na linha seguinte \$b é exibido como um valor binário e como um valor octal. Esta é a primeira vez que utilizamos a função *printf* com mais de um parâmetro. Como duas substituições serão executadas, é necessário passar dois parâmetros para a função *printf*.

Uma linha depois, tentamos interpretar \$b como um valor inteiro exibido como um caractere. Em outras palavras, tentamos exibir na tela o caractere \$b pelo seu valor ASCII.

Finalmente, acrescentamos 2/12 a \$b (\$b += 2/12 é equivalente a \$b = \$b + 2/12) e o exibimos como um número de ponto flutuante utilizando '%f'. O número de ponto flutuante tem um comprimento total de cinco dígitos, sendo que três destes dígitos têm que estar do lado direito do

ponto decimal.

Executando o script, obteremos este resultado:

```
Este é o diário de Adriano.
Ele tem 30 anos de idade.
30 pode também ser escrito em 10111 (binário)
ou 27 (octal).
Vamos imprimir 30 novamente mas nesse momento
como um inteiro impresso como o caracter: #.
Para ser mais preciso: Adriano tem 30.166666666667
anos e 2 meses de idade Como ponto flutuante ele
seria representado assim: 30.167.
```

A função *printf* suporta uma grande variedade de parâmetros de formatação. A **Tabela 3**, contém uma lista de atalhos suportados pela função *printf*.

### Formatos aceitos pela função printf

A função *printf* retorna 0 ou 1 se o comando foi executado com sucesso. Se quisermos recuperar a string gerada pela função *printf*, devemos utilizar em seu lugar a função *sprintf*. Diferentemente da função *printf*, *sprintf* retorna a string que foi gerada mas não a exibe na tela. Vejamos um exemplo:

```
<?php
    $a = "Adriano";
    $result = sprintf ("Este é o diário de
        %s.", $a);
    echo "Result: $result<br>\n";
?>
```

Desta vez o resultado é atribuído a uma variável chamada \$result e exibido na tela utilizando o comando echo.

Se desejarmos que uma caractere "\" (barra invertida) seja colocado antes dos caracteres "\\", "+", "\*\*?", "[", "^", "]", "(", "\$" ou ")", podemos utilizar a função *QuoteMeta*. Caso estejamos construindo interfaces para outros produtos de software ou trabalhando com expressões regulares, esta função pode ser interessante, pois nos auxilia quando não queremos exibir caracteres especiais. Vejamos um exemplo de como a função pode ser utilizada:

```
<?php
    $a = "Is 1+2*3=7?";
    $result = QuoteMeta ($a);
    echo "$result";
?>
```

O resultado conterà várias barras invertidas:

```
Is 1\+2\*3=7\?
```

Se quisermos remover as barras invertidas inseridas por *QuoteMeta*, podemos utilizar a função *stripslashes*.

A função *strcmp* pode ser utilizada para comparar string e é outra função que foi

**Listagem 4. Exemplos de uso de printf**

```
<?php
    $a = "Adriano";
    printf ("Este é o diário de %s.<br>\n", $a);

    $b = 30;
    printf ("Ele tem %d anos de idade.<br>\n", $b);
    printf ("%b pode também ser escrito em %b ."
        "(binário) ou %o (octal).<br>", $b, $b);
    printf ("Vamos imprimir %b novamente, mas nesse
        momento ele é um inteiro".
        " impresso como um caracter: %c.<br>", $b, $b);

    $b += 2/12;
    printf ("Para ser mais preciso: %s e %b "
        " anos e dois meses de idade. "
        " Como número de ponto flutuante pode ser: "
        "%5.3f.<br>\n", $b);
?>
```

Formatação	Significado
%b	Interpretar o valor como um valor inteiro, mas exibi-lo como um valor binário.
%c	Interpretar o valor como um valor inteiro, mas exibi-lo como um valor caractere.
%d	Interpretar o valor como um valor inteiro, mas exibi-lo como um valor decimal.
%f	Interpretar o valor como um valor inteiro e exibi-lo como um número de ponto flutuante.
%o	Interpretar o valor como um valor inteiro, mas exibi-lo como um valor octal.
%s	Interpretar e exibir o valor como uma seqüência.
%x	Interpretar o valor como um valor inteiro, mas exibi-lo como um valor hexadecimal, utilizando letras em minúsculas.
%X	Interpretar o valor como um valor inteiro, mas exibi-lo como um valor hexadecimal, utilizando letras em maiúsculas.

Tabela 3. Variações da função printf

emprestada do C. O uso da função *strcmp* em PHP é muito semelhante à do C, como podemos ver no seguinte exemplo:

```
<?php
$a = "abc";
$b = "bcd";

echo strcmp($a, $b)."<br>";
echo strcmp($b, $a)."<br>";
echo strcmp($a, $a)."<br>";
?>
```

Nas duas primeiras linhas as variáveis foram definidas e serão comparadas uma com a outra nas três linhas seguintes. O primeiro comando da função *strcmp* retorna -1, pois as duas variáveis foram passadas para a função em ordem alfabética. Os parâmetros foram passados para o segundo comando em ordem inversa, portanto é retornado 1. Se ambos os valores forem iguais, a função *strcmp* retornará 0. Nas linhas seguintes, podemos ver a saída do script:

```
-1
1
0
```

A função *strlen* é largamente utilizada e pode ser aplicada para retornar o tamanho de um string. Em primeiro lugar, é definida uma string chamada *\$a*. No passo a seguir, o comprimento desta string é verificado e exibido na tela:

```
<?php
$a = "Esta é uma string";
echo strlen($a)."<br>\n";
?>$a
```

Em alguns casos poderia ser útil remover caracteres em branco do começo e do fim de um string. Para isto, podemos utilizar a função *trim*:

```
<?php
$b = " Eu sou uma string ";
echo inicio:".trim($b).":fim";
?>
```

A variável *\$b* contém uma seqüência com alguns espaços em branco no começo e no fim da mesma. Com a ajuda da função *trim*, esses espaços em branco são facilmente removidos:

```
inicio:Eu sou uma string:fim
```

O resultado não mais contém espaços em branco. Se quisermos apenas remover os espaços em branco à esquerda da string, a função *ltrim* pode ser utilizada. E se quisermos remover os espaços em branco à direita, a função *rtrim* pode ser utilizada.

Podemos converter strings em maiúsculas e vice-versa, utilizando duas funções simples do PHP chamadas *strtoupper* e *strtolower*. No exemplo a seguir, veremos como estas funções são utilizadas:

```
<?php
$a = "eu sou uma string";
$b = strtoupper($a);

echo "$b<br>\n";
echo strtolower($b);
?>
```

Em primeiro lugar, uma seqüência é definida. A seguir, é convertida para maiúsculas e atribuída a *\$b*. Finalmente ambas as seqüências são exibidos na tela:

```
EU SOU UMA STRING
eu sou uma string
```

Em muitos casos não desejamos converter todos os caracteres para letras maiúsculas ou minúsculas. Com a ajuda da função *ucfirst*, é possível converter só a primeira letra para maiúscula. A função *ucwords*, garante que cada palavra em uma string irá começar com maiúscula:

```
<?php
$a = "eu sou uma string ";

echo ucfirst($a)."<br>";
echo ucwords($a)."<br>";
?>
```

A saída do script não é de modo nenhum surpreendente:

```
Eu sou uma String
Eu Sou Uma String
```

O PHP suporta muitas funções relacionadas a strings ("textos"), porém, estão além do escopo deste artigo abordar todas essas funções detalhadamente. Decidimos mencionar apenas as funções mais importantes e as mais largamente utilizadas.

## Trabalhando com arrays unidimensionais

A capacidade de lidar fácil e eficientemente com arrays, é uma das características que fez com que PHP se difundisse nestes últimos anos. Não só os arrays podem ser facilmente manipulados, mas temos também disponíveis muitas funções relacionadas a arrays. Nesta seção aprenderemos sobre essas funções.

Os arrays podem ser *unidimensionais* ou *multidimensionais*. Um array unidimensional é uma estrutura de dados utilizada

para armazenar uma lista de valores. Nós já aprendemos nesse mesmo artigo, que os arrays constituem um dos tipos de dado primitivo do PHP. Contudo, os arrays são de certa forma diferentes de tipos de dados tais como *integer* ou *double*.

Um array não é um valor atômico, o que significa que é composto de um ou mais valores que podem, por sua vez, ser também arrays.

Começaremos com um exemplo, onde veremos como podemos descobrir se uma variável é um array:

```
<?php
$a = 12;
$b = array(30, "Adriano", "São Paulo", 1.67);

echo "a: ".is_array($a)."<br>";
echo "b: ".is_array($b);
?>
```

Podemos ver que foram definidos um valor inteiro e um array. O array compõe-se de um número inteiro, duas strings e um valor de ponto flutuante. Como podemos observar, não faz absolutamente nenhuma diferença quais valores serão inseridos em um array, pois o mesmo é apenas um container para armazenar todas as espécies de variáveis. Executando o script, podemos ver que a função *is\_array* retorna 1 caso a variável passada for um array:

```
a:
b: 1
```

Agora que aprendemos a criar arrays e descobrir se uma variável é um array, aprenderemos a acessar os componentes de um array. A primeira coisa a saber, é como descobrir quantos valores estão armazenados em um array. Isto pode ser feito utilizando a função *count*. Assim que conhecermos o número de componentes, é fácil exibirmos o conteúdo da tabela na tela:

```
<?php
$a = array(30, "Adriano", "São Paulo", 1.67);
$val = count($a);
echo "Nº de valores no array: $val<br><br>\n";
for($i = 0; $i < $val; $i++){
    echo "Valor número $i: $a[$i]<br>";
}
?>
```

A saída do script mostra que quatro registros foram encontrados no array. Finalmente, todos os registros são exibidos na tela. As várias células da tabela são acessadas utilizando parênteses e o índice do valor:

Número de valores no array: 4  
 Valor número 0: 30  
 Valor número 1: Adriano  
 Valor número 2: São Paulo  
 Valor número 3: 1.67

A exibição de todos os registros na tela, pode ser feita também de outra maneira:

```
<?php
$a = array(30, "Adriano", "São Paulo", 1.67);

foreach ($a as $x){
    echo "$x<br>";
}
?>
```

Depois de definir a tabela, foi incluída uma repetição que é executado para cada valor no array. Todo valor em `$a` é atribuído a `$x`, que é exibido na tela usando um comando `echo`:

```
30
Adriano
São Paulo
1.67
```

Um array não precisa ser consecutivamente numerado. Em muitos casos é útil utilizar strings para identificar os valores em um array:

```
<?php
$a = array(idade => 30, nome => "Adriano",
    residência => "São Paulo", altura => 1.67);

foreach($a as $x){
    echo "$x<br>";
}
?>
```

No exemplo anterior, um array associativo é inicializado e o conteúdo é exibido na tela como mostrado na seguinte listagem:

```
30
Adriano
São Paulo
1.67
```

Muitos podem já ter trabalhado com arrays associativos em Perl. No PHP, as coisas funcionam de forma bastante semelhante ao Perl. Um array associativo é apenas um array indexado com strings. A princípio deste tipo de array, é capaz de armazenar cada tipo de dado em um array e indexá-lo com qualquer dado desejado.

A recuperação de dados de um array associativo pode ser feita da mesma forma vista com arrays indexados com números (**Listagem 5**).

Todos os valores serão exibidos na tela:

```
Idade: 30 anos
Nome: Adriano
Residência: São Paulo
Altura: 1.67 centímetros
```

Nessa seção já mencionamos que um

#### Listagem 5. Array indexado

```
<?php
$a = array(idade => 30, nome => "Adriano",
    residência => "São Paulo", altura => 1.67);

echo "Idade: ".$a["idade"]." years <br>";
echo "Nome: ".$a["nome"]." <br>";
echo "Residência: ".$a["residência"]." <br>";
echo "Altura: ".$a["altura"]." centímetros<br>";
?>
```

array também pode conter outros arrays. O exemplo a seguir mostra um array que tem um elemento que por sua vez, contém outro array com dois elementos:

```
<?php
$a = array(nome => array("Adriano", "Santos"));
$b = $a["nome"];
echo "$b[0] - $b[1]<br>\n";
?>
```

Podemos inserir um array em outro array como faríamos com qualquer outro registro. O único aspecto que devemos levar em consideração, é a recuperação dos valores. Isso é ligeiramente mais difícil do que se tratássemos apenas de um valor comum.

Se executarmos o script, os valores que inserimos serão recuperados:

```
Adriano - Santos
```

Com a ajuda de arrays, podemos construir estruturas de dados complexas. Contudo, se as estruturas de dados ficarem muito complexas, é melhor utilizar o PHP orientado a objetos em lugar de arrays e procedimentos complexos.

Um comando importante quando trabalhamos com arrays, é o comando `reset`. Diferentemente do que poderíamos esperar o comando `reset` não é utilizado para remover elementos de um array, mas para posicionar o ponteiro no primeiro elemento do mesmo. Se estivermos trabalhando com um array, o ponteiro poderá não estar posicionado no primeiro elemento e teremos problemas, portanto, para usarmos com segurança comandos como `next` ("próximo") e `prev` ("anterior") necessitamos usar `reset`. No exemplo a seguir, veremos como definir um array, posicionar o ponteiro no primeiro elemento e exibir todos os registros na tela, inclusive as chaves:

```
<?php
$a = array(23, 16, 8, 99);
reset($a);
while (list($key, $val) = each ($a)){
    echo "$key: $val<br>\n";
}
?>
```

Como podemos verificar, cada valor no array tem uma chave.

```
0: 23
1: 16
2: 8
3: 99
```

Achar a chave de um valor é uma tarefa importante, pois a estrutura de dados pode ser colocada no índice utilizando qualquer outra variável.

A ordenação de dados é outra operação importante:

```
<?php
$a = array(23, 16, 8, 99);
sort($a);

foreach ($a as $val){
    echo "valor: $val<br>\n";
}
?>
```

Para construir estruturas de dados eficientes, é importante ordenar os dados antes de trabalhar com elas. A saída do script que acabamos de ver é exibida ordenada porque utilizamos o comando `sort` para modificar a ordem dos elementos.

```
valor: 8
valor: 16
valor: 23
valor: 99
```

Para ordenar os dados em ordem inversa, utilizamos a função `arsort` em vez de `sort`:

```
<?php
$a = array(23, 16, 8, 99);
arsort($a);
foreach ($a as $val){
    echo "valor: $val<br>\n";
}
?>
```

Como podemos ver, os dados serão exibidos agora em ordem decrescente:

```
valor: 99
valor: 23
valor: 16
valor: 8
```

Para obtermos um efeito contrário ao de ordenar um array devemos utilizar a função `shuffle` a qual coloca os valores de forma desordenada:

```
<?php
$a = range(1, 7);
shuffle($a);
while (list($key, $val) = each($a)){
    echo "$key: $val<br>\n";
}
?>
```

Em primeiro lugar, criamos um array que contém os valores 1 até 7 utilizando a função `range`. A seguir, os valores da tabela são misturados e todos os valores são exibidos na tela.

```
0: 3
1: 1
2: 2
3: 4
4: 5
5: 7
6: 6
```

Os valores não estão mais ordenados. Desordenar arrays pode ser útil em muitas situações. Imaginemos uma situação em que desejamos exibir registros em ordem aleatória. A função `shuffle` faz isto.

Nós vimos como um array pode ser acessado utilizando um índice ou um comando `foreach`. Outro modo de processar todos os valores de um array consiste em utilizar os comandos `reset` e `next`. O comando `next`, posiciona o ponteiro no próximo valor do array e retorna o valor atual. Utilizando uma repetição `while`, é fácil de percorrer o array (**Listagem 6**).

Como podemos ver a seguir, todos os valores serão exibidos na tela:

```
valor: 23
valor: 16
valor: 8
valor: 99
```

## Arrays Stacks

Os arrays do tipo `stacks`, são um tipo especial de array. Consistem em uma estrutura de dados fundamental e são largamente utilizados por muitos programadores. Se quisermos processar a comandos utilizando a notação `posfix`, os arrays `stacks` representam um instrumento essencial.

Duas operações são predefinidas para `stacks`:

- A operação `push`, adiciona um valor ao `stack`;
- A operação `pop` retorna o último valor que foi adicionado ao array e o remove da lista.

Em outras palavras, arrays `stacks` utilizam um algoritmo LIFO (LIFO = Last in first out - último a entrar primeiro a sair).

Para esclarecer o que acabamos de aprender vejamos a **Listagem 7**.

Na primeira linha, foi definido um array vazio. Nas três linhas seguintes, os valores são adicionados ao array `stack`, utilizando o comando `array_push`, e todos os valores são exibidos na tela. Então, dois valores são retirados da `stack` e exibidos na tela. Como podemos ver, o valor de retorno de `array_pop` é retornado e removido do array. Depois de ter adicionado três valores ao array e ter retirando dois valores do mesmo, somente restou um valor o qual é exibido na tela:

```
0: o primeiro valor
1: o segundo valor
2: o terceiro valor

removendo: o terceiro valor
removendo: o segundo valor

0: o primeiro valor
```

A contagem dos valores de um array pode ser feita utilizando a função `array_count_values`. A idéia desta função é a de contar a frequência com a qual um determinado valor ocorre na tabela.

Às vezes é necessário remover valores duplicados de uma tabela. A função `array_unique` faz este serviço. Vejamos alguns exemplos de ambas funções na **Listagem 8**.

Depois de definir um array contendo algum valor, um array é inicializado contendo a frequência com a qual vários valores em `$a` ocorrem. Na primeira coluna a chave é exibida. A segunda coluna diz-nos com que frequência a chave foi encontrada no array.

Em seguida a função `array_unique` é utilizada para remover entradas duplicadas. A chave para um valor é a posição na qual certo valor ocorreu no array original. O valor 9, por exemplo, foi encontrado na quarta e a última posição do array. Como isto é redundante, o último valor no array foi eliminado e só permaneceu o valor na posição número 4:

```
Retorno da listagem:
23: 3
12: 1
9: 2

Retorno da listagem única:
0: 23
2: 12
3: 9
```

A saída da função mostra o que é exibido quando o script é executado.

Quando trabalhamos com mais de um array pode ocorrer a necessidade de unirmos determinados arrays para algum fim. Para realizarmos isso, o PHP fornece

algumas funções poderosas e fáceis de utilizar. Uma das funções é chamada `array_merge` e é utilizada para combinar dois arrays:

```
<?php
$a = array(23, 12, 9);
$b = array(12, 13, 14);

$res = array_merge($a, $b);

while (list($key, $val) = each($res)){
    echo "$key: $val<br>\n";
}
?>
```

São definidos dois arrays, os quais serão fundidos em um só. Se executarmos o script, o resultado conterá todos os valores de ambos os arrays, tal como mostrado na listagem a seguir:

```
0: 23
1: 12
2: 9
3: 12
4: 13
5: 14
```

Ao contrário da função `array_merge`, a função `array_diff`, pode ser utilizada para obter um array diferença de outros dois arrays, tal como mostrado a seguir:

```
<?php
$a = array(23, 12, 9);
$b = array(12, 13, 14);

$res = array_diff($a, $b);

while (list($key, $val) = each($res)){
    echo "$key: $val<br>\n";
}
?>
```

Foram encontrados dois valores em `$a` que não existem em `$b`:

```
0: 23
2: 9
```

Se quisermos computar a intersecção de dois arrays em lugar da diferença, podemos utilizar a função `array_intersect`. Como com as funções `array_diff` e `array_merge`, devemos passar dois arrays para a função `array_intersect`:

```
<?php
$a = array(23, 12, 9);
$b = array(12, 13, 14);

$res = array_intersect($a, $b);

while (list($key, $val) = each($res)){
    echo "$key: $val<br>\n";
}
?>
```

Apenas um valor é retornado:

```
1: 12
```

O cálculo da soma de todos os valores em um array, pode ser feito de várias maneiras porém, a forma mais fácil é utilizar a função `array_sum`:

```
<?php
$a = array(23, 12, 9);
echo array_sum($a);
?>
```

O valor 44 resultante, será exibido na tela.

Dividir uma seqüência e gerar um array a partir da mesma é uma operação importante. Se tivermos que processar arquivos separados por tags, por exemplo, dividir strings e gerar arrays é um procedimento essencial. O PHP provê uma função chamada *explode*. A função *explode* é chamada com dois parâmetros. O primeiro diz ao PHP onde dividir a string mediante a definição de um padrão. O segundo parâmetro contém a variável que desejamos processar.

Ao contrário dela podemos usar a função *implode*, que faz exatamente o contrário da função *explode*. Vejamos um exemplo na **Listagem 9**.

No início do script, uma variável é definida, a qual será utilizada para gerar uma tabela na segunda linha. Um caractere espaço é designado como caractere separador. A seguir, o conteúdo de \$b é exibido na tela. Finalmente todos os valores em \$b são combinados novamente em uma seqüência. Mais uma vez, um caractere espaço é designado como caractere separador.

Se executarmos o script, a saída será exibida na tela:

```
0: Esta
1: é
2: uma
3: string
```

Esta é uma string

Como podemos ver na última linha do resultado, o array foi novamente combinado na string que tinha sido definida na primeira linha.

## Trabalhando com arrays multidimensionais

Os arrays multidimensionais são um bocado mais difíceis de manipular do que os arrays unidimensionais. Contudo, com algumas técnicas básicas, é possível executar operações complexas.

Estas técnicas podem ser utilizadas para processar tanto arrays unidimensionais quanto multidimensionais. Mesmo para estruturas de dados mais complexas, cada um desempenhará um papel importante (**Listagem 10**).

Nas quatro primeiras linhas, um array de duas dimensões é criado e quatro valores são atribuídos ao mesmo. O objetivo é exibir todos os registros na tela.

A listagem a seguir, mostra o que é exibido quando executamos o script:

```
Informações:
0: Array
1: Array
```

A listagem não contém o resultado desejado, pois cada um extrai uma lista de arrays em lugar de uma lista de valores únicos da tabela multidimensional. Para exibir os dados, devemos fazer algumas pequenas modificações no código conforme a **Listagem 11**.

Uma segunda repetição deve ser acrescentada para obter todos os campos dos

### Listagem 6. While

```
<?php
$a = array(23, 16, 8, 99);
reset($a);
$val = $a[0];
do{
    echo "valor: $val<br>\n";
}
while($val = next($a));
?>
```

### Listagem 7. Arrays Stacks

```
<?php
$a = array();
array_push($a, 'o primeiro valor');
array_push($a, 'o segundo valor');
array_push($a, 'o terceiro valor');

while(list($key, $val) = each($a)){
    echo "$key: $val<br>\n";
}
echo "<br>removendo: ". array_pop($a). "<br>";
echo "removendo: ". array_pop($a). "<br><br>";
while(list($key, $val) = each($a)){
    echo "$key: $val<br>\n";
}
?>
```

### Listagem 8. array\_count\_values e array\_unique

```
<?php
$a = array(23, 23, 12, 9, 23, 9);
$counted = array_count_values($a);

echo "Retorno da listagem: <br>\n";
while(list($key, $val) = each($counted)){
    echo "$key: $val<br>\n";
}

echo "<br>Retorno da listagem única: <br>\n";
$a = array_unique($a);
while(list($key, $val) = each($a)){
    echo "$key: $val<br>\n";
}
?>
```

### Listagem 9. Funções implode e explode

```
<?php
$a = "Esta é uma string";
$b = explode(" ", $a);

while(list($key, $val) = each($b)){
    echo "$key: $val<br>\n";
}
$c = implode(" ", $b);
echo "<br>$c<br>";
?>
```

### Listagem 10. Arrays multidimensionais

```
<?php
$a[0][0] = "João Morais";
$a[0][1] = "Maria Aparecida";
$a[0][2] = "Chamada de Viena";
$a[1][0] = "Satélite para Satélite";

echo "<br>Informações:</b><br>\n";
while(list($key, $val) = each($a)){
    echo "$key: $val<br>\n";
}
?>
```

arrays extraídos do array multidimensional. Se executarmos o script agora, o resultado conterà todos os valores:

```
Informações:
chave:0 - subchave: 0 - valor: João Morais
chave:0 - subchave: 1 - valor: Maria
Aparecida
chave:0 - subchave: 2 - valor: Chamando de
Viena
chave:1 - subchave: 0 - valor: Satélite para
Satélite
```

A seguir desejamos ordenar os valores no array. Para realizar este objetivo, podemos tentar acrescentar um comando *sort* após atribuírmos os valores ao array:

```
sort($a);
```

Isto não nos conduzirá ao resultado desejado, pois o PHP ordenará as chaves e não os valores:

```
Informações:
chave:0 - subchave: 0 - valor: Satélite para
Satélite
chave:1 - subchave: 0 - valor: João Morais
chave:1 - subchave: 1 - valor: Maria
Aparecida
chave:1 - subchave: 2 - valor: Chamando de
Viena
```

Se quisermos ordenar o conteúdo do array e não as chaves, devemos construir um array multidimensional, onde todos os eixos têm a mesma quantidade de valores. No caso, podemos construir uma matriz com quatro valores (matriz 2x2):

```
$a[0][0] = "João Morais";
$a[0][1] = "Maria Aparecida";
$a[1][0] = "Chamando de Viena";
$a[1][1] = "Satélite para Satélite";
```

Esta matriz pode ser facilmente ordenada utilizando o comando *array\_multisort*:

```
array_multisort($a[0], SORT_ASC, SORT_STRING,
$a[1], SORT_ASC, SORT_STRING);
```

Ambas as colunas serão ordenadas em ordem ascendente. Se executarmos o script agora, obteremos uma lista ordenada:

```
Informações:
chave:0 - subchave: 0 - valor: Maria
Aparecida
chave:0 - subchave: 1 - valor: João Morais
chave:1 - subchave: 0 - valor: Satélite para
Satélite
chave:1 - subchave: 1 - valor: Chamando de
Viena
```

Se não tivermos modificado a estrutura dos dados, teríamos experimentado dificuldades, pois o tamanho dos arrays seria inconsistente:

A função *array\_multisort*, aceita dois parâmetros para influir no modo em que os dados serão ordenados. O PHP suporta dois flags de ordenação:

- SORT\_ASC – Ordenação ascendente;
- SORT\_DESC – Ordenação descendente;

Além do mais, três parâmetros de tipo de ordenação são fornecidas:

- SORT\_REGULAR – Comparar os itens normalmente;

- SORT\_NUMERIC – Comparar os itens numericamente;
- SORT\_STRING – Comparar os itens como seqüências;

Como podemos ver, a função *array\_multisort* é uma função poderosa e flexível. Contudo, deve ser utilizada com cuidado, para garantir que o resultado de um processo de ordenação, irá conter exatamente os dados desejados.

Nós já vimos que um array multidimensional se compõe de muitos arrays que têm uma dimensão a menos do que o array pai. O conhecimento disto permitirá que executemos todas as operações que desejarmos executar. Trabalhando com arrays *stacks* multidimensionais, é importante saber o que acontece dentro do PHP e como os dados serão fornecidos pelo intérprete.

## Conclusão

Neste artigo vimos como trabalhar com tipos de dados no PHP, incluindo arrays. Na próxima edição continuaremos explorando a linguagem PHP. Até lá! ●

### Listagem 11. Alteração na exibição dos arrays

```
<?php
$a[0][0] = "João Morais";
$a[0][1] = "Maria Aparecida";
$a[0][2] = "Chamando de Viena";
$a[1][0] = "Satélite para Satélite";

echo "<b>Informações:</b><br>\n";
while (list ($key, $val) = each ($a)){
    while (list ($subkey, $subval) = each ($val)){
        echo "chave:$key - subchave: $subkey - valor: $subval<br>\n";
    }
}
?>
```



# Cursos Online

Assinatura

**ClubeDelphi PLUS**

Mais conteúdo DELPHI por muito menos!

A Revista **ClubeDelphi** oferece para seus assinantes uma série de **Cursos Online** de alto padrão de qualidade .  
Conheça abaixo os cursos já disponíveis..

Curso em destaque

## Aplicações Client/Server com dbExpress e Firebird

Confira neste curso online como criar uma aplicação client/server completa no Delphi 7, utilizando dbExpress e Firebird.

Aprenda também: Como trabalhar com um driver dbExpress específico e gratuito para o Firebird ; Como utilizar Orientação a Objetos, e técnicas como herança visual de formulários e relatórios ; saiba como colocar regras de negócios no banco de dados, usando Triggers e Stored Procedures; E crie relatórios com o Quick Report, Rave Reports e Report Builder, e muito mais.

Confira o plano de aula completo:  
[www.devmedia.com.br/clienteserver](http://www.devmedia.com.br/clienteserver)

**A sua melhor opção de aprendizagem!**

Assine a **ClubeDelphi** e comece já seu treinamento!  
[www.devmedia.com.br/assine](http://www.devmedia.com.br/assine)

Outros cursos disponíveis: [www.devmedia.com.br/curso](http://www.devmedia.com.br/curso)

- Curso Online - Sistema SysCash
- Criando uma Aplicação multi-camadas Completa com Delphi
- Aplicações client/server com Windows Forms no Delphi 2006
- Aplicações WEB com IntraWeb e Delphi 7 (Delphi Win32)



**DevMedia**  
GROUP

Mais Informações: [www.devmedia.com.br/central](http://www.devmedia.com.br/central) - Tel.: 21 2220-5375 / 2220-5435

