

Comprando esta edição tenha acesso na WEB a:
+380 vídeo-aulas +150 artigos e 6 cursos online

www.clubedelphi.net Feita para desenvolvedores

ClubeDelphi

Ano 7 · Edição 85 · R\$9,90 

10 Dicas de Firebird que você precisa saber



Na Web!

Curso Online: IntraWeb com Delphi 7

Confira no portal ClubeDelphi PLUS um curso online completo sobre programação Web com Delphi 7

- Criando sua primeira aplicação Web com Delphi 7
- Componentes do IntraWeb
- Criando uma aplicação Web que acessa Firebird
- Gráficos dinâmicos com IW
- Relatórios na Web
- Trabalhando com Templates IW
- Utilizando sessions no IW
- Construindo uma aplicação Web completa

Mini-curso
Crie uma aplicação completa usando UML, ASP.NET, POO e ECO – Parte 3

Favoritos
Implemente um menu favoritos na sua aplicação para armazenar os forms mais acessados

Coluna: Oráculo da ClubeDelphi
Super Dicas sobre API do Windows, Threads (processamento paralelo), Propriedades em Forms e RTTI

Validação com Imagem
Proteja seu site contra ataques usando esse poderoso recurso de validação, amplamente utilizado na Web

Web Services
Construa um disco virtual usando WS e técnicas avançadas de SOAP

Delphi for PHP
O poder RAD do Delphi agora também para o PHP – tudo sobre as novidades

Coluna Ask the Expert

Quantas cópias de seu software existem no mercado?

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

Comprove você mesmo!



ClubeDelphi

Ano 7 - 84ª Edição - 2007 - ISSN 1517990-7

Impresso no Brasil

Corpo Editorial

Diretor Editorial e Administrativo

Gladstone Matos
gladstone@medificio.com.br

Editor de Arte

Vinicius O. Andrade
viniciusoandrade@gmail.com

Capa

Antonio Xavier
jaxs_design@yahoo.com.br

Articlistas desta edição

Paulo Roberto Quicoli, Fernando Sarturi Prass, Fabrício Desbessel, Frank Ingermann, Paul Vinkenoog, Adriano Santos, Murilo Costa Monteiro Filho, Alberto Germano Sevarolli Trevezani e Marco Antônio Pereira Araújo

Editor Geral

Guinther Pauli
guinther@devmedia.com.br

Editor Técnico

Luciano Pimenta
lucianopimenta@clubedelphi.net

Revisão

Rafael de Castro Santos
fael.cast@gmail.com

Distribuição

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

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:
Kaline Dolabella
publicidade@devmedia.com.br

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:

Thiago Andrade - Atendimento ao Leitor
www.devmedia.com.br/central/default.asp
(21) 2220-5375

Kaline Dolabella
Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 2220-5375

Publicidade

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

Kaline Dolabella
publicidade@devmedia.com.br

Para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

Jeff Wendell
jeff@devmedia.com.br

EDITORIAL

Como está a saúde do seu banco de dados? E o desempenho? Amigo leitor, de nada adianta construir uma aplicação com layout arrojado, várias opções customizáveis, pesquisas sofisticadas, se só lembrar do banco de dados quando ele der problema. A boa funcionalidade de um BD é a peça chave para o sucesso de qualquer projeto e é importante que você tome os cuidados necessários, fazendo um trabalho preventivo ao invés de corretivo. Com o Firebird, a figura do DBA se tornou um pouco obsoleta. O próprio programador deve realizar o trabalho do DBA, verificando gargalos, detectando problemas de performance, analisando planos de execução e otimizando queries.

Nesta edição, os irmãos Prass destacam 10 importantes dicas que todos os desenvolvedores Delphi têm que saber ao trabalhar com o banco de dados open-source. Vale ressaltar que muitas dicas também valem para outros bancos. Aprenda a importância de um backup / restore, o porquê de analisar as estatísticas do banco, páginas de dados e índices, aprenda mais sobre a cache de leitura e escrita, análise do plano de execução, dicas sobre joins, índices e chaves-primárias.

Enfim, um artigo de cabeceira para quem quer ter sempre um banco robusto e saudável! Aproveite as dicas porque sem dúvida elas são valiosas!

Ainda nesta edição, o Fabrício introduz o novíssimo Delphi for PHP, a versão da ferramenta mais RAD de todos os tempos para umas das linguagens mais utilizadas na Web. É o melhor de dois mundos. Você vai poder programar para Web com os mesmos controles que já desenvolve no Desktop, em uma linguagem Orientada a Objetos. Crie aplicações Web em minutos!

Confira ainda muito sobre ASP.NET, ECO, POO e UML no mini-curso do Paulo, como criar um menu favoritos para a aplicação, SOAP / WeBServices e para finalizar, trago minha coluna "Oráculo da Clube", com algumas dicas interessantes.

Boa leitura e muito sucesso com o Delphi for PHP!



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>

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 o que você está achando da revista: que tipo de artigo você gostaria de ler, que 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!

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:

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

Luciano Pimenta - Editor do Site
lucianopimenta@clubedelphi.net

ÍNDICE

06 - Sistemas auto atualizáveis

Marcelo Rodrigo Lopes

12 - Atualização de estruturas de dados

Adriano Santos

20 - Delphi 2007 for Win32

Andreano Lanusse

24 - Desenvolvendo uma Loja Virtual - ASP.NET, UML e ECO - Parte 2

Paulo Roberto Quicoli

30 - Validações em duas camadas

Fabrício Desbessel

38 - DataGrid - Criando colunas dinamicamente

Marcos Santos e Alexandre Santos

46 - Análise de Requisitos com CaliberRM

Felipe La Rocca Teixeira e Marco Antônio P. Araújo

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 380 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 e utilize as informações abaixo: **LogIn: DVM.CD e Senha: KZW19**

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 **Guinther Pauli**
(envie as suas para guinther@devmedia.com.br)

Flash no Delphi

Gostaria de saber como exibir uma animação Flash em um formulário Delphi, para por exemplo personalizar minha tela de Splash ou colocar um banner no formulário principal?

Juliano Vieiro

Olá Juliano, obrigado. Para um exemplo, inicie uma nova aplicação no Delphi. Clique a seguir no menu *Component>Import ActiveX Control*. Na caixa de diálogo *Import ActiveX*, localize o item *Shockwave Flash* e clique no botão *Install*. Confirme a instalação do controle bastando clicar em *Ok*. A seguir, no editor, clique em *Install Package*. Um novo componente deve surgir agora na paleta *ActiveX*, chamado *ShockwaveFlash*. Agorá basta colocar o componente no form, e no evento *OnCreate* digitar o código a seguir (troque pelo nome do seu arquivo). O resultado é mostrado na **Figura 1**.

```
ShockwaveFlash1.LoadMovie(0, 'c:\bonequinho.swf');
```

Transactional DataModule no Delphi 2006

Muitos leitores têm nos escrito perguntando o porquê do item *Transactional Data Module* não estar presente no Delphi 2006, inclusive foi o caso do nosso amigo leitor



Figura 1. Flash no Delphi

Francesco Coutinho. Para habilitar essa opção no Delphi 2006, devemos acessar o seguinte item no registro:

```
HKCU\Software\Borland\BDS\4.0\Type Library\
```

Criar um novo valor de seqüência com o nome de "TransactionalWizards" e valor

"True" (**Figura 2**).

A opção deve agora aparecer na *Object Repository*, como mostra a **Figura 3**. Lembrando que antes você deve criar uma *ActiveX Library*. Pronto, agora você já pode criar seus projetos multicamadas usando COM+ no BDS 2006.

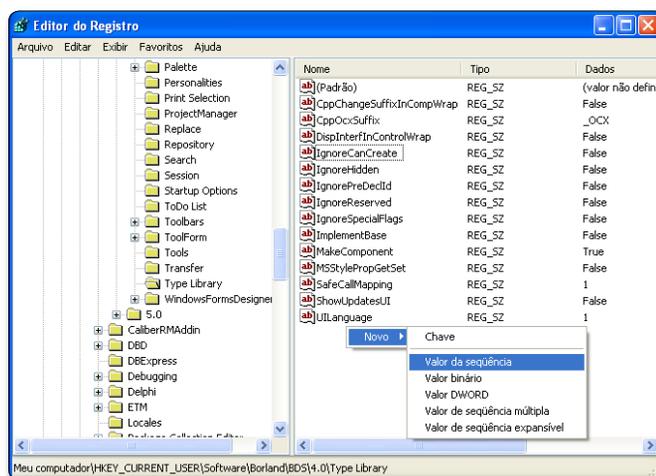


Figura 2. Adicionando um valor no Registro do Windows

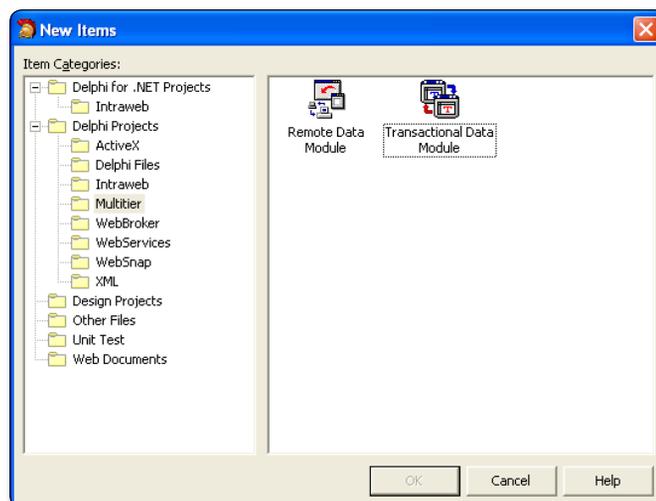


Figura 3. Item adicionado ao New Items do Delphi



Assinatura

ClubeDelphi PLUS

Mais conteúdo .NET por menos!

+380 vídeo aulas | 6 cursos online

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 380 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

Curso Online – IntraWeb com Delphi 7

Confira no portal ClubeDelphi PLUS um curso online completo sobre IntraWeb com Delphi 7

<http://www.devmedia.com.br/cursos/studentportal.asp?curso=15>

Últimas Vídeo-Aulas (Somente para assinantes)

Sistema completo com Delphi 7, dbExpress e Firebird 2.0

Acompanhe a criação de um sistema completo de gerenciamento de locadora, usando Delphi, dbExpress e Firebird 2.0. Luciano Pimenta, mostrará todos os passos necessários para a criação do projeto em todas as fases de implementação, desde a criação do banco de dados, até a migração para .NET.

Criando uma tela de consulta dinâmica – Parte I a III

Veja nessa vídeo aula de Paulo Quicoli, como criar um tela de consulta dinâmica.

Novidades do Delphi 2007 - Parte IV a V

Veja nessas aulas de Adriano Santos as principais novidades em relação ao novo release do Delphi, o Delphi 2007.

Aplicações MySQL com Delphi for PHP - Parte V

Veja nessas vídeo aulas de Fabricio Desbessel como trabalhar com MySQL no Delphi for PHP.

Compactação de arquivos com UPX

Veja nessa vídeo aula de Adriano Santos, como utilizar a ferramenta UPX para compactação de arquivos.

CURSOS ONLINE

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]

**Web
Mobile**

A revista do desenvolvedor Web e Wireless

Acesse já! www.devmedia.com.br/webmobile/pagina.asp

10 dicas sobre Firebird que todo desenvolvedor deve saber



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.



Fábio Sarturi Prass

(fabio@dotbr.com.br)

é Bacharel em Sistemas de Informação pelo Centro Universitário Franciscano e Analista de Sistema da Di Uno Informática.

O Firebird tornou-se um banco de dados de grande aceitação, principalmente pela comunidade Delphi. Grande parte desse sucesso deve-se a sua facilidade de uso e de configuração. Entretanto, apesar de ser bastante simples, o Firebird exige alguns cuidados por parte dos desenvolvedores, do contrário corre-se o risco de que seu desempenho seja reduzido significativamente.

Este artigo mostrará 10 dicas que tendem a garantir um bom desempenho ao banco de dados. Algumas são baseadas em textos que foram publicadas por outros autores (logo as fontes estão citadas, nos devidos lugares), outras surgiram da necessidade de resolver problemas encontrados pelo autor em consultorias realizadas em empresas de todo o país.

1. Backup e Restore

É provável que não exista uma dica melhor e mais simples sobre Firebird do que

essa: sempre que possível faça um *backup* e *restore*. Essas operações trazem uma série de benefícios, os principais são:

- As páginas de dados e índices são alocadas de forma contínua e aquelas que não são usadas são eliminadas;
- A árvore de índices é reconstruída e a seletividade dos índices recalculada;
- Registros eliminados (através do DELETE) são excluídos fisicamente (*Garbage Collection*).

Uma dica: caso o principal uso do banco de dados seja a leitura e não a inserção ou atualização de dados, pode-se usar o parâmetro USE_ALL_SPACE ao restaurar o *backup*.

Devido à forma como o Firebird trabalha com as versões de um mesmo registro, utiliza um modelo chamado *Versioning*, as páginas de dados do banco armazenam múltiplas versões dos registros nelas contidos. Quando um BD é restaurado, o banco reserva um espaço de aproximadamente 20% da página para

armazenar as novas versões dos registros casos os mesmos sejam alterados. Se o banco de dados será lido e não atualizado (ou pouco atualizado), não há necessidade de que esse espaço seja reservado (KARWIN, 1998). O comando completo é mostrado a seguir:

```
GBAK -c -use_all_space -user sysdba -pas
masterkey <arquivo_backup.fbk> <novo_banco.
fdb>
```

2. Análise das estatísticas do cabeçalho do banco

As estatísticas do cabeçalho do banco de dados fornecem uma série de informações importantes para avaliar a real situação do servidor. Elas podem ser obtidas através do utilitário de linha de comando GSTAT, presente na pasta *bin* do Firebird.

Parâmetro	Função
-a	Analisa os dados e as páginas de índice
-d	Analisa as páginas de dados
-h	Analisa a página de cabeçalho (header page)
-i	Analisa as "folhas" das páginas de índices (leaf pages)
-l	Analisa a página de log
-u	Nome do usuário para conectar no banco
-p	Senha para conexão
-r	Analisa os registros de versões
-t	Especifica o nome da tabela a ser analisada (para o caso de ser uma única tabela)

Tabela 1. Principais parâmetros disponíveis para o utilitário GSTAT

O GSTAT retorna uma série de informações sobre o banco solicitado, essas informações dependem dos parâmetros passados. O comando "gstat -help" apresenta as opções disponíveis, as principais delas estão na Tabela 1.

A Listagem 1, cujo comando para obtenção está logo a seguir, mostra parte do resultado da análise do cabeçalho do banco de dados EMPLOYEE que acompanha o Firebird e será usado em todos os exemplos deste artigo (a Tabela 2 explica as principais variáveis):

```
gstat -h -u sysdba -p masterkey
c:\firebird\examples\employee.fdb
```

Conforme mostra CANTU (2005) a análise das variáveis de transações podem apontar dois problemas importantes:

- Se a diferença entre *Oldest active* e *Next transaction* for muito grande é provável que as transações fiquem abertas por muito tempo. A solução é realizar um *sweep* manual através do utilitário GFIX (também presente na pasta *bin*):

```
gfix -sweep -user sysdba -password masterkey
<banco de dados>
```

- Uma diferença maior que 20.000 entre *Next Transaction* e *Oldest Transaction* pode indicar que o *sweep* automático está desligado ou configurado para um valor muito alto. Pode-se alterar o valor através do GFIX (parâmetro *housekeeping*):

```
gfix -housekeeping 15000 -user sysdba -password
<banco de dados>
```

Informação	Descrição
Page Size	Tamanho definido para as páginas do banco de dados.
ODS Version	<i>On Disk Structure Version</i> , versão da estrutura do arquivo do banco de dados, informa em qual versão do Firebird o banco foi criado. É atualizado sempre que um <i>backup</i> de uma versão mais antiga é restaurado num servidor mais recente (CANTU, 2005).
Oldest Transaction	Transação mais antiga que ainda não recebeu um <i>commit</i> .
Oldest Active	ID da transação mais antiga ainda ativa.
Next Transaction	ID da próxima transação a ser criada.
Implementation ID	Sistema operacional onde o banco de dados foi criado, 16 representa o Windows de 32-bit e 19 o Linux.
Shadow count	Número de arquivos de <i>shadow</i> (sombra) do banco de dados.
Page buffers	Tamanho do <i>cache</i> em número de páginas. Se estiver zero, indica que o valor utilizado é o padrão.
Database dialect	Dialeto atual do banco de dados.
Attributes	Atributos do banco de dados: <i>force write</i> : modo de escrita síncrona (<i>write-through</i>), sem <i>cache</i> . <i>no_reserve</i> : nenhum é espaço reservado para os registros de versão. <i>shutdown</i> : banco indisponível para conexões (exceto para o usuário SYSDBA).
Sweep Interval	Intervalo para realização do <i>sweep</i> automático.

Tabela 2. Principais informações obtidas no cabeçalho (header) do banco

3. Análise das estatísticas das páginas de dados e de índices

As estatísticas do banco fornecem também informações sobre as páginas de dados, as páginas de índices e registros das tabelas. Veja a Listagem 2 o resultado da execução do comando:

```
gstat -r -u sysdba -p masterkey <banco de dados>
```

Primeiro as informações obtidas sobre os registros de versões: média em *bytes* do tamanho dos registros armazenados (*Average Record Length*); total de registros na tabela, incluindo ativos e inativos (*Total Records*); média do tamanho dos registros temporário (*Average Version Length*); total de registros temporários existentes (*Total Versions*) e número máximo de versões que um registro possui (*Max Versions*). Quanto maior o valor de *Total Versions*, maior será o espaço que será liberado com um *backup/restore*.

Sobre páginas de dados e de índices tem-se: número total de páginas de dados já alocadas para a tabela (*Data Pages*); número de ponteiros atualmente alocados para páginas de dados da tabela (*Data Page Slots*, deve ter obrigatoriamente o mesmo valor de *Data Pages*); média geral de ocupação das páginas alocadas (*Average Fill*); e média por faixas da ocupação das páginas (*Fill Distribution*).

Se a *Average Fill* for menor que 60% ou se a maior parte das páginas de dados não estiverem nas faixas 60-79% e 80-99%, então um *backup/restore* se faz necessário. Por último, nas estatísticas dos índices tem-se: número de páginas entre a primeira página do índice e as páginas *leaf* (*Depth*); número de páginas *leafs* (*Leaf Buckets*); total de páginas de índices que compõem a árvore (*Nodes*); total de linhas que apresentam chaves duplicadas/repetidas (*Total Dup*); e média por faixas da ocupação das páginas (*Fill Distribution*).

Se a distribuição da maior parte das páginas de índices não estiver na faixa 80-99%, é necessário que o índice seja reconstruído (veja como na Dica 9). Se o valor do *depth* for maior do que três, talvez seja preciso aumentar do tamanho das páginas de dados (ver Dica 5).

Nota: páginas *leafs* são aquelas que contêm informações que apontam para onde estão as linhas (registros) da tabela;

4. Tamanho da página (Page size)

O tamanho da página padrão utilizada pelo InterBase era de 1 KB, no Firebird esse valor passou para 4 KB, esse aumento por si só já gerou um ganho automático de desempenho.

Hoje os sistemas ocupam cada vez mais espaço em disco, dessa forma o ideal é que se use 8 KB como tamanho da página de dados. As razões para o aumento do tamanho das páginas são explicadas por KARWIN (1998):

- *Menos fragmentos de registros divididos:* é comum existir registros que ocupem mais de 4 KB, o que faz com que sejam armazenados em múltiplas páginas forçando o servidor a ler várias páginas para recuperar a informação;

- *Melhor utilização dos índices Btrees:* um número menor de páginas será usado para armazenar os índices, quanto menor o número de páginas mais rápido o

processamento;

- *Operações de I/O mais contínuas:* o servidor armazena os registros na primeira página livre, o que pode significar que registros sucessivos não sejam armazenados em páginas próximas (uma página só armazena registros de uma mesma tabela), páginas maiores implicam em mais dados da mesma tabela numa mesma página;

- *Buffer maior é igual a mais memória de cache:* o cache do Firebird é alocado em número de páginas e não por uma quantidade fixa de bytes. Quanto maior o cache maior a chance dos dados serem encontrados nele.

A leitura das vantagens apresentadas pode levar o leitor a pensar que quanto maior o tamanho da página, melhor o desempenho, mas isso nem sempre é verdade. O Firebird suporta páginas de dados com até 16 KB, entretanto para a maioria dos bancos, 8 KB é o tamanho

ideal. Páginas maiores que 8 KB só devem ser usadas por bancos de dados muito grandes ou que contenham tabelas que possuem elevado acesso e cujos registros são maiores do que 8 KB.

O tamanho das páginas de dados pode ser alterado a qualquer momento através de um *backup/restore*. De toda forma, seja qual for o tamanho do banco de dados e o tamanho de página escolhido, é aconselhável que se façam testes com os outros valores para determinar qual o melhor para cada situação.

5. Cache de leitura

O Firebird mantém um *cache* em memória com as últimas páginas de dados e de índices utilizadas. Como em qualquer *cache*, o ganho de performance depende basicamente da repetição de acessos aos mesmos dados.

O tamanho do *cache* é definido em número de páginas de dados. Por padrão, o tamanho alocado é de 2048 páginas, valor que pode ser verificado no arquivo *firebird.conf* que se encontra na pasta onde o banco foi instalado (procure por *DefaultDbCachePages*). Se o tamanho de cada página estiver definido em 1 KB, então 2 MB de memória serão usados. Se o tamanho da página for 4 KB, então 8 MB de memória será usado (baseado no exemplo apresentado em PRENOSIL, 2002).

No InterBase o valor padrão do *cache* era de 256, pois na época as máquinas possuíam pouca memória. Nos servidores de hoje, onde há grande quantidade de memória disponível, é altamente recomendável que aumente o tamanho do *cache* para obter uma melhor performance. É possível fazer isso através do utilitário GFIX:

```
gfix -buffers <valor> <banco de dados>
```

O tamanho máximo permitido é 65535, entretanto não se deve usar *cache* maior do que 10000, pois quanto maior o valor maior a necessidade de processamento, o que pode causar perda de desempenho (PRENOSIL, 2002).

Além disso, não se deve usar um valor muito alto ao ponto do servidor utilizar a memória virtual (SWAP), pois isso anularia a vantagem de usar um *cache* visto que o acesso ao disco é muito mais lento do que o acesso à memória.

O valor também não deve ultrapassar o

Listagem 1. Resultado da execução do GSTAT no banco de dados EMPLOYEE

```
Database header page information:
Flags                0
Checksum             12345
Generation           160
Page size            4096
ODS version          10.1
Oldest transaction   156
Oldest active        157
Oldest snapshot     157
Next transaction     159
Bumped transaction   1
Sequence number      0
Next attachment ID  0
Implementation ID   16
Shadow count         0
Page buffers         0
Next header page    0
Database dialect     3
Creation date        Jan 17, 2006 12:07:17
Attributes           force write

Variable header data:
Sweep interval:     20000
*END*
```

Listagem 2. Estatísticas da tabela SALES do banco de dados EMPLOYEE

```
(...)
SALES (138)
Primary pointer page: 218, Index root page: 219
Average record length: 67.30, total records: 33
Average version length: 0.00
Total versions: 0, max versions: 0
Data pages: 1, data page slots: 1,
average fill: 68%
Fill distribution:
 0 - 19% = 0
20 - 39% = 0
40 - 59% = 0
60 - 79% = 1
80 - 99% = 0
Index NEEDX (3)
Depth: 1, leaf buckets: 1, nodes: 33
Average data length: 2.00, total dup: 11,
max dup: 6
Fill distribution:
 0 - 19% = 1
20 - 39% = 0
40 - 59% = 0
60 - 79% = 0
80 - 99% = 0
(...)
```

número de páginas do banco, pois uma página do disco ocupa uma página na *cache*, que nunca é duplicada. É preciso experimentar os valores do *cache* e analisar o resultado de cada caso. O ganho pode chegar 30% (KARWIN, 1998).

Nota: O tamanho do *cache* também pode ser definido durante o *restore* através do parâmetro `BUFFERS: gbak -c -buffers 1234 <demais parâmetros>`

Para finalizar, a forma como *cache* é alocado difere da versão *Classic* para a *SuperServer*. Na versão *Classic* cada usuário possui uma instância do servidor, logo cada usuário possui seu próprio *cache*.

Na *SuperServer* um único *cache* é compartilhado entre todos os usuários, o que é mais eficaz uma vez que se dois ou mais usuários requisitarem a mesma informação, quando o segundo o fizer, os dados já estarão em memória.

Usuários que utilizam a versão *Classic* precisam tomar um cuidado maior ao definir o tamanho do *cache* já que um *cache* é alocado para cliente conectado ao banco. Por exemplo: se o tamanho do *cache* é 5000 e o *page size* do banco é 4 KB serão alocados 20 MB de memória para cada cliente conectado, caso existam 10 clientes simultâneos 200 MB de memória serão alocados. Existindo dois bancos no servidor serão 400 MB, existindo 3 serão 600 MB e assim por diante.

6. Cache de escrita

O Firebird possui dois modos de escrita: com *cache* (chamado de escrita síncrona ou *write-through*) e sem *cache* (chamado de escrita assíncrona ou *write-back*). O primeiro modo é o padrão na plataforma Windows, o segundo na plataforma Linux.

Na escrita síncrona cada operação de escrita é imediatamente passada pelo servidor do banco para o Sistema Operacional (observe a linha contínua na **Figura 1**). Na escrita assíncrona o banco armazena os dados em memória e atrasa a gravação dos mesmos, ou seja, múltiplas operações de escrita para uma página do *cache* são executadas na memória para depois serem gravadas no disco, o que resulta num melhor tempo de resposta para as operações de escrita (observe a

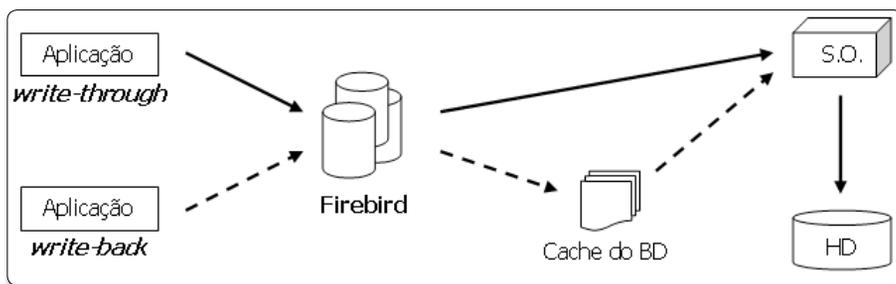


Figura 1. Representação dos modos de gravação *write-through* e *write-back*

linha tracejada na **Figura 1**).

Na escrita assíncrona a gravação física ocorre quando uma determinada quantidade de informação é coletada, um evento associado ocorre ou quando um intervalo de tempo tenha passado. Como se pode deduzir, o modo utilizado no Linux é mais eficiente do que o utilizado no Windows.

A configuração do modo como o *cache* deve se comportar pode ser alterada manualmente através do comando:

```
gfix -write async <banco de dados>
```

KARWIN (1998), diz que o uso de escrita assíncrona pode melhorar em cerca de 4 vezes o desempenho de aplicações padrões e em até 20 vezes aplicações que executam milhares de operações de escrita (INSERT, UPDATE, DELETE) em seqüência.

O autor alerta que o modo *write-back* somente deve ser usado em servidores estáveis e que contam com *nobreaks*, pois dados podem ser perdidos no caso do servidor sofrer uma queda de energia ou se for finalizado de maneira anormal. No modo *write-through*, isso não acontece.

7. Analise os Planos de Execução

A análise do PLAN de execução é o primeiro fator a ser levado em consideração para garantir o bom desempenho de um *query*, pois ele mostra como *engine* do banco de dados chega até os dados solicitados. Para analisar os planos de acesso será usado neste artigo o IBO-Console (www.mengoni.it/Downloads/IBOConsole.zip).

Como exemplo, a *query* que lista todos os funcionários que receberam um aumento maior ou igual a 7% (sete por cento):

```
select e.full_name, s.percent_change
from employee e, salary_history s
where
e.emp_no = s.emp_no and
s.percent_change >= 7
```

Ao analisar as estatísticas (**Figura 2**) pode-

se verificar que a tabela `SALARY_HISTORY` foi lida em sua totalidade (*full scan*).

Isso é indicado em PLAN pela palavra `NATURAL` após o "S" (alias que foi dado à tabela):

```
PLAN SORT (JOIN(S NATURAL, E
INDEX(RDB$PRIMARY7)))
```

O plano de execução também mostra que o acesso à tabela `EMPLOYEE` (alias "E") se deu através do índice da chave primária (`RDB$PRIMARY7`). Outros dados importantes das estatísticas são *Execution Time* que mostra quanto tempo a *query* levou para ser executada, *Reads* e *Writes* quantas leituras e escritas foram feitas (respectivamente) e *Rows Affected* que mostra quantas linhas retornaram.

Se um índice for criado para o campo `PERCENT_CHANGE`, o plano de execução será modificado:

```
PLAN SORT (JOIN(S INDEX (IDX_SALARY_HISTORY),
E INDEX (RDB$PRIMARY7)))
```

Ou seja, agora o acesso à tabela `SALARY_HISTORY` também se deu através do índice. Se o leitor verificar o tempo que as duas *query* levaram para ser executadas (*Execution Time*) notará que praticamente não houve diferença, isso ocorreu apenas porque a base de dados é limitada.

E se a tabela guardasse o histórico de aumento de salário ao longo de uma década de uma empresa com centenas de funcionários? Provavelmente a tabela de histórico teria milhares de linhas e aí o tempo ganho seria significativo.

8. Sempre que possível use Joins e não SubSelects

É inegável que o *subselect* fornece ao programador uma alternativa eficiente para a solução de diversos problemas, porém essa alternativa muitas vezes faz com que o desempenho da *query* caia significativamente.

A explicação para a queda de desempenho é que um *subselect* é na verdade um *loop* aninhado. Se a *query* possui dois *subselects*, são três *loops* aninhados e assim sucessivamente.

A solução é, sempre que possível, usar um *joins*. Veja o exemplo da **Listagem 3**.

As duas consultas retornam o mesmo resultado: a lista de pessoas que trabalham no Canadá. Se o leitor analisar os planos de execução de ambas verificará que a primeira é mais lenta, pois precisa “varrer” toda a tabela EMPLOYEE para encontrar o resultado, enquanto a segunda utiliza os índices.

9. Use índices corretamente

Índices, quando utilizados corretamente, podem aumentar consideravelmente a performance de um SELECT. A seguir algumas dicas sobre índices (RODRIGUES, 2006):

- Crie índices para as colunas mais usadas em JOINS, exceto para aquelas que fazem parte de uma chave estrangeira (*Foreign Key*), pois essas já possuem índices;
- Crie também índices para as colunas mais usadas em ORDER BY, mas lembre-se que índices são ordenados de forma ascendente e ou descendente, logo só serão usados se a ordenação do comando

for a mesma do índice;

- Não crie índices em tabelas que possuem poucos registros (menos de 50);
- Reconstrua os índices periodicamente, pois as tabelas que sofrem muitas atualizações tendem a ficar com índices desatualizados. Para tanto, basta usar os comandos:

```
alter index <nome do índice> inactive;  
alter index <nome do índice> active;  
set statistics index <nome do índice>;
```

• Índices prejudicam o desempenho em operações de atualização de dados (INSERT, UPDATE, DELETE) porque são atualizados pelo banco durante essas operações. Pode-se evitar esse transtorno desativando os índices antes de realizar atualizações em lote e ativando-os logo após a operação;

• Se for preciso criar um índice para um campo com baixa seletividade (por exemplo, um campo cujas opções são SIM/NÃO), a melhor opção é criar um índice composto pelo campo desejado seguido de outro campo usado na pesquisa.

Na **Listagem 4** temos alguns exemplos para verificar o desempenho dos índices na prática.

Analisando os planos de execução verifica-se que na primeira e na segunda consulta o índice composto já existente na tabela (NAMEX, formado pelos campos LAST_NAME e FIRST_NAME, nessa ordem) foi usado, mas na última não.

A situação muda se um índice for criado para o campo FIRST_NAME, com o código da **Listagem 5**.

O índice criado é usado na primeira consulta, mas não na segunda. E se forem usadas as funções UPPER e CONTAINING? (**Listagem 6**).

Mais uma vez o índice não é utilizado, de onde se conclui: um índice composto só é usado se todos os campos que fazem parte dele forem usados na *query* ou se o(s) primeiro(s) campo(s) do índice estiver(em) nela.

O comando LIKE utiliza índice desde que não se utilize o “%” no início do parâmetro; UPPER e CONTAINING não utilizam índices. Para finalizar execute e análise os planos de execução.

Uma pergunta: como explicar o fato Firebird ter usado o índice na segunda consulta, se no parágrafo anterior foi dito

Listagem 3. Utilizando Join ao invés de SubSelects

```
select first_name, last_name  
from employee  
where  
  job_code in (select job_code  
              from job j  
              where j.job_country = 'Canada')  
  
select first_name, last_name  
from employee e, job j  
where  
  e.job_code = j.job_code and  
  j.job_country = 'Canada'
```

Listagem 4. Exemplos para verificar desempenho de índices

```
select first_name, last_name  
from employee  
where first_name like 'K%' and last_name like 'B%'  
  
select first_name, last_name  
from employee  
where last_name like 'B%'  
  
select first_name, last_name  
from employee  
where first_name like 'K%'
```

Listagem 5. Criando um índice para o campo FIRST_NAME

```
create index idx_first_name on employee(first_name)  
  
select first_name, last_name  
from employee  
where first_name like 'K%'  
  
select first_name, last_name  
from employee  
where first_name like '%e%'
```

Listagem 6. Usando as funções UPPER e CONTAINING

```
select first_name, last_name  
from employee  
where upper(first_name) like 'K%'  
  
select first_name, last_name  
from employee  
where first_name containing 'nn'
```

Listagem 7. Utilizando CONTAINING e LIKE

```
select first_name, last_name  
from employee  
where first_name like 'B%' or first_name  
  containing 'nn'  
  
select first_name, last_name  
from employee  
where first_name like 'B%' and first_name  
  containing 'nn'
```

que a função CONTAINING não utiliza-va um índice? Veja a **Listagem 7**.

A presença do operador AND na segunda consulta faz com que um registro só seja mostrado se FIRST_NAME iniciar com 'B' e se contiver (em qualquer parte dele) 'nn'. Ciente disso, o otimizador do banco de dados executa primeiro a parte da consulta que utiliza o LIKE, fazendo uso do índice existente para o campo.

Isso reduz significativamente a quantidade de registros, e é somente nessa quantidade reduzida de registros que a segunda parte da consulta (que utiliza o CONTAINING) é executada.

Na verdade, embora o PLAN mostre que o índice foi utilizado, é preciso deixar claro que isso ocorreu apenas na primeira parte da busca, na segunda não houve uso de nenhum índice.

10. Não use chave-primária composta

Imagine o seguinte modelo de dados: aluno, curso e turmas (relação M:N entre aluno e curso). Para que o sistema esteja na terceira forma normal a tabela TURMA deve possuir uma chave primária composta pelas chaves primárias das tabelas ALUNO e CURSO ("Modelo

Normalizado" na **Figura 2**).

Entretanto, em termos de desempenho, essa não é a melhor opção. Encontrar um registro em uma tabela com chave composta é mais lento do que encontrar um registro numa tabela com chave primária simples, a explicação é óbvia: na primeira é preciso comparar dois valores, na segunda apenas um.

Nesse caso, o ideal é que se quebre a forma normal e crie-se uma chave primária simples (número inteiro seqüencial) e um índice composto único contendo as chaves primárias das demais tabelas ("Modelo Ideal" na **Figura 2**).

Conclusão

Este artigo mostrou que mesmo usando um banco de dados de fácil configuração, o Firebird exige por parte do seu administrador pequenos cuidados em relação às configurações como: tamanho da página de dados e de cache.

Saindo do lado do servidor e entrando no desenvolvimento propriamente dito, cabe ao programador tomar alguns cuidados para que o desempenho não seja afetado pela falta ou pelo mau uso dos índices. Para isso, cabe sempre analisar os planos de execução. ●

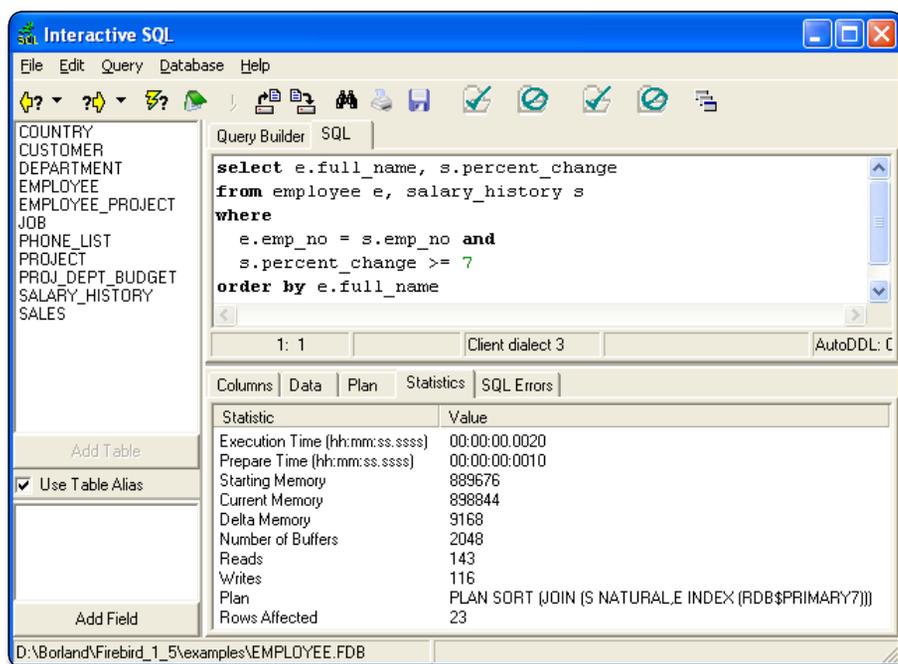


Figura 2. Estatísticas do plano de execução da query no IBOConsole

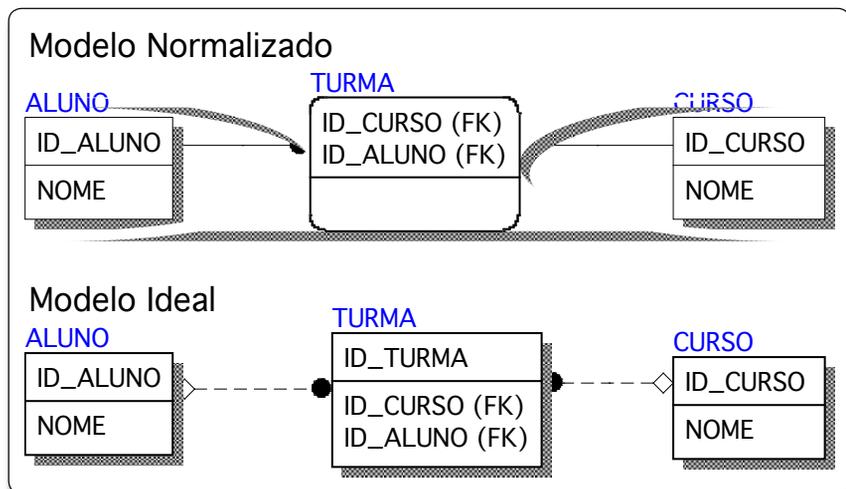


Figura 3. Modelo de dados Normalizado e Modelo de Dados Ideal

Referências Bibliográficas

KARWIN, Bill.
Ten Things You Can Do To Make InterBase Scream. Inprise Conference, Agosto de 1998.

CANTU, Carlos Henrique.
Entendendo o GSTAT – Parte I. DB Free Magazine, volume 1, Abril de 2005.

PRENOSIL, Ivan.
Cache settings. Disponível em www.volny.cz/iprenosil/interbase. Fevereiro de 2002

RODRIGUES, Anderson Haertel.
Índices, Performance e Estatísticas no Firebird. Disponível em www.comunidade-firebird.org. Acessado em Setembro de 2006.





Desenvolvendo uma Loja Virtual com ASP.NET, UML e ECO – Parte 3

Chegamos na parte final do nosso mini-curso e veremos como criar a página inicial do site *ECOStore*, onde o usuário poderá escolher que produto comprar e finalizar a compra. Em nossa página inicial vamos permitir ao usuário escolher seu produto através das categorias existentes.

Página Principal

Inicie um novo ECO WebForm acessando o menu *File>New>Other>New ASP.NET Files>Eco ASP.NET Page*, renomeie-o como “Principal.aspx”. Ligue o *RootHandle* ao *ECOSpace* através de sua propriedade *EcoSpaceType*.

Inclua no formulário o *User Control* criado anteriormente. Vamos adicionar agora um *Expression Handle*, guia *Enterprise Core Objects* da *Tool Palette* e o renomeamos para “*exphCategoria*”. Nele vamos digitar a expressão OCL para obter os objetos *categoria*.

Alterando sua propriedade *RootHandle* para o objeto *RootHandle* constante

da página, ligamos o *exphCategoria* ao *ECOSpace*. Altere para *True* a propriedade *AddExternalId*, para podermos utilizar o identificador do objeto e liberar o acesso ao ID do campo chave da tabela (classe). Compile o projeto.

Para definir agora que objetos e classes serão retornadas utilizamos uma expressão OCL que será especificada na propriedade *Expression*, em nosso caso queremos todas as categorias, portanto a expressão é “*Categoria.allInstances*”.

Adicione ao formulário um *ListBox* e vamos ligá-lo ao *exphCategorias*. Ajuste as seguintes propriedades do *ListBox*:

```
DataSource = exphCategorias  
DataTextField = Nome  
DataValueField = ExternalId  
AutoPostBack = True
```

Adicione mais um *Expression Handle*. Mude seu nome para “*exphProdutos*”, ligue-o ao *rhRoot* e ajuste a propriedade *AddExternalId* para *True*. Na propriedade *Expression*, digite:



Paulo Roberto Quicoli

(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 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.

Metodologia

A aplicação desenvolvida neste mini-curso é desenvolvida sobre um modelo multi-camadas, onde temos: banco de dados, aplicação web (o servidor), camada de persistência e interface (browser). O Modelo no entanto não segue o padrão MVC. Existe uma camada de negócio (Business Logic que é gerada pelo modelo UML construído pelo ECO. De fato, o ECO gera a maioria do código de negócio, com classes e regras de acordo com a definição do modelo.

Devido a utilização do ECO, a programação utilizada será completamente orientada a objetos (OO). As vantagens dessa abordagem são muitas:

- Separação lógica entre as partes da aplicação, o que torna o software mais flexível e fácil de manter;
 - Reutilização de código (por exemplo uma mesma validação ou regra de negócio pode ser reutilizada em diferentes tipos de interface de usuário);
 - Melhor uso de técnicas de POO, por utilizar o ECO, a programação se torna totalmente orientada a objetos e deixa de ser totalmente orientada a eventos;
 - Menor impacto na mudança de requisito e redução de custos. Uma solicitação de alteração, por parte do cliente, dependerá um esforço muito menor por parte da equipe de desenvolvimento (muitas vezes o ECO faz a maior parte do trabalho), se comparado a um software desenvolvido usando tecnologias tradicionais.
- Desvantagens:
- Investimento em aprendizado;
 - Queda na velocidade de produção da “primeira versão”;
 - O desenvolvimento fica “preso” ao ambiente ECO;

A lógica de negócio representada no modelo UML garante a independência de interface de usuário podendo ser exposta através de um webform, de um Webservice ou até mesmo de um Windows Form. Nesse caso as interfaces utilizarão as classes definidas no modelo UML criando uma maneira do usuário interagir com elas. Esta é a função de uma camada de apresentação, apenas permitir a interação do usuário com a lógica do sistema.

Para desenvolver utilizando essa metodologia é preciso ter em mente o conceito de camadas lógicas. Cada camada é responsável por uma área e elas trocam informações entre si. As áreas de uma aplicação são divididas de acordo com o entendimento de cada um, porém quanto ao ECO podemos relacionar as seguintes:

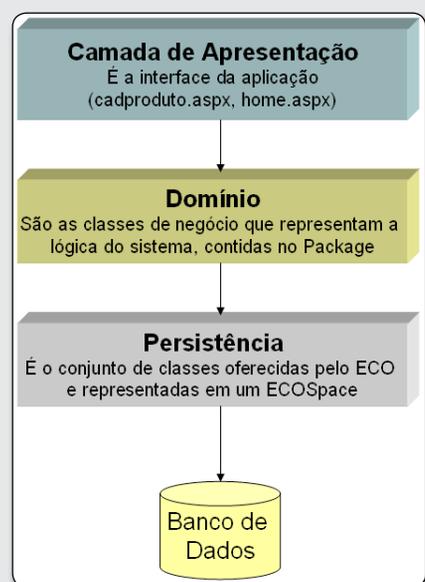
- 1. Banco de dados** – O Banco de Dados é utilizado para armazenamento dos objetos. O próprio ECO se responsabiliza pela adequação da estrutura do banco de dados em relação ao modelo de objetos.
- 2. Domínio** – As classes são armazenadas em um “pacote UML”, um arquivo PAS que contém além das classes de domínio, classes e atributos que definem o modelo UML da regra de negócio, como relacionamentos e constraints.
- 3. Persistência** – O ECO oferece um conjunto de classes para realização da persistência dos objetos no banco de dados, se encarregando da mesma, retirando assim do desenvolvedor essa obrigação. Veja um exemplo de como persistir um objeto “Usuario”.

```
procedure TWebForm1.btnSalvar_Click(
  sender: System.Object; e: System.
  EventArgs);
var
  obj: Usuario;
```

```
begin
  obj := Usuario.Create(EcoSpace);
  obj.Nome := txtUsuario.Text;
  obj.Endereco := txtEndereco.Text;
  obj.Email := txtEmail.Text;
  obj.Senha := txtSenha.Text;
  UpdateDatabase;
  Response.Redirect('Home.aspx');
end;
```

4. Apresentação: São as classes que utilizam as classes de domínio e de persistência. É a interface que o usuário vê e utiliza. Essas classes são do tipo da interface a ser construída, seja ela ASP.NET, Windows Forms etc.

Na figura a seguir vemos uma representação das divisões da aplicação



```
"Produto.allInstances->select(currentProduto
| currentProduto.Categoria.Nome = '0')"
```

Insira um *DataGrid* criando três colunas, sendo a primeira para a propriedade *Nome*, a segunda para o propriedade *UrlImagem* e a terceira sendo uma coluna do tipo *Hyperlink*.

Na coluna referente a *UrlImagem* configure a propriedade *Data Formatting expression* para "", desta maneira ao carregar o *DataGrid* será mostrada a figura. Na coluna *Hyperlink* ajuste sua propriedade *Text Field* e *URL field* para *ExternallId* e em *URL format string* informe "VerDetalhes.aspx?RootId={0}".

Configure as propriedades *DataSource* e *DataKeyField* do *DataGrid*, para *exphProdutos* e *ExternallId*. Veja na **Figura 1** um exemplo da página principal.

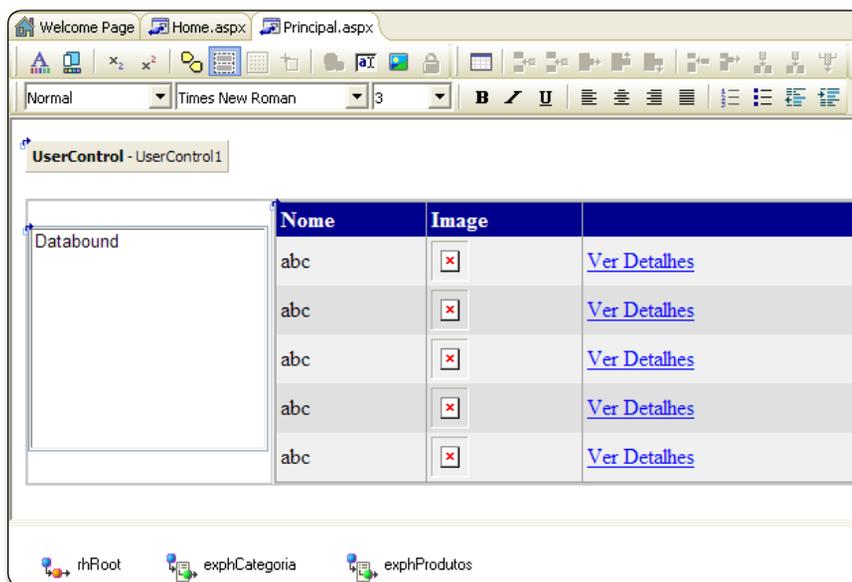


Figura 1. Exemplo da página principal

No evento *SelectedIndexChanged* do *ListBox* vamos refazer a sentença OCL para que sejam retornados os Produtos da categoria selecionada (**Listagem 1**).

Detalhes dos produtos

Vimos que no *DataGrid* anterior, temos uma coluna do tipo *Hyperlink* que aponta para uma página de detalhes do produto e é a partir dessa página que vamos permitir a compra do produto.

Inicie um novo ECO WebForm e renomeie como "VerDetalhes.aspx". Ligue o *rhRoot* ao *Ecospace* e em sua propriedade *StaticValueTypeName* escolha *Produto*, informando assim o tipo de objeto que o *Expression Handle* trará. Configure a página conforme **Figura 2**.

Nota: O layout é bem simples, apenas atente para o nome dos *Label*s e para as configurações do *TextBox* da descrição (propriedade *TextMode = Multiline*).

Para cada controle vamos realizar um *DataBinding* (propriedade *DataBindings* do controle) com sua propriedade específica contida no *rhRoot*, menos para o controle que exibirá a categoria (fizemos essa técnica na edição anterior).

Exibindo a categoria do Produto

Primeiramente adicione uma referência ao *Package_1Unit*, depois vamos alterar o *Page_Load*, conforme a **Listagem 2**.

Rode a aplicação, filtre as categorias na *Principal.aspx* e clique no link para abrir a página de detalhes do produto (**Figura 3**).

Criando o carrinho de compras

Inicie um novo ECO WebForm e renomeie como "Carrinho.aspx". Ligue o *rhRoot* ao *Ecospace*. Adicione um *DataSet*, o renomeie para "dsCarrinho" e nele vamos adicionar uma tabela em memória chamada "COMPRAS". Para isso acesse a propriedade *Tables* e clique em *Add* no editor. Em *Columns* crie os campos, conforme a **Tabela 1**.

Vamos usar aqui um *DataSet* por ser o objeto responsável por armazenar dados em memória.

De volta ao editor de tabelas (*DataTable*) do *Dataset*, definimos o campo

Listagem 1. Selecionando produtos pela categoria

```
procedure TWebForm1.ListBox1_SelectedIndexChanged(
sender: System.Object; e: System.EventArgs);
begin
  exphProdutos.Expression := 'Produto.allInstances-''+
  '>select(currentProduto | '+
  'currentProduto.Categoria.Nome = '' +
  ListBox1.SelectedItem.Text + ''''';
  DataBind;
end;
```

Listagem 2. Carregando a Categoria

```
procedure TWebForm1.Page_Load(sender: System.Object;
e: System.EventArgs);
var
  Id: string;
begin
  EcoSpace.Active := True;
  Id := Request.Params['RootId'];
  if Assigned(Id) and (Id <> '') then
    rhRoot.SetElement(ObjetoForId(Id));
  if not IsPostBack then
    begin
      DataBind;
      ( Adicione esse código )
      lblCategoria.Text := Produto(
rhRoot.Element.AsObject).Categoria.Nome;
    end;
end;
```

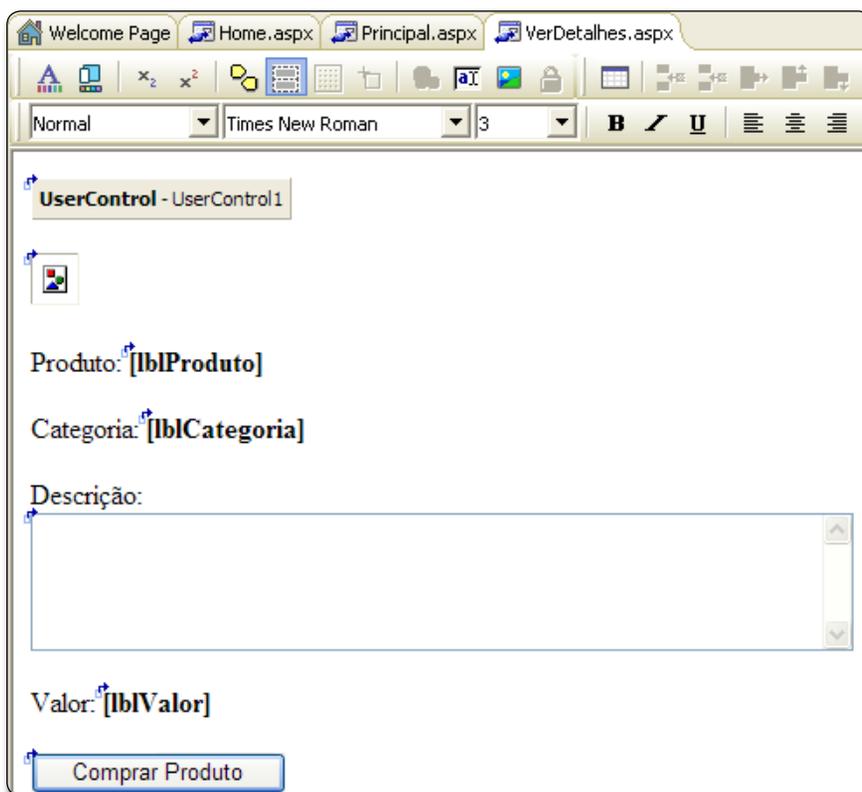


Figura 2. Página de detalhes

INTERNAL_ID_PRODUTO como chave primária do *DataSet* (propriedade *PrimaryKey*).

Utilizando Sessions

Todos sabemos que uma aplicação ASP.NET é *stateless*, ou seja, não mantém o estado de seus objetos. O uso do *Session* permite armazenar o estado de um objeto

Coluna	Tipo
INTERNAL_ID_PRODUTO	System.String
NOME_PRODUTO	System.String
VALOR_PRODUTO	System.Decimal
QTDE	System.Int32
SUBTOTAL	System.Decimal
TOTAL	System.Decimal

Tabela 1. Colunas do DataSet

entre requisições feitas pelo usuário. Se não utilizarmos o *Session* para armazenar o estado do carrinho a cada nova adição de produtos, perderíamos o conteúdo do carrinho.

Vamos criar na seção *private* um método chamado *GetSession*, que colocará o *dsCarrinho* na sessão. Na **Listagem 3** temos a implementação do método.

Adicionando produtos ao carrinho

O *Page_Load* já possui um código que trata se na URL da aplicação foi passado o *InternalId* de algum produto, e vamos utilizar esse código para preencher nosso carrinho. A primeira coisa a se fazer é especificar o tipo de objeto que o *ReferenceHandle* *rhRoot* representa.

Para isso, vamos alterar sua propriedade *StaticValueTypeName* para *Produto*, e altere para *True* também a propriedade *AddExternalId*. Vamos criar agora um método que adicionará o produto ao carrinho, verificando se o produto já existe e que calcula o TOTAL e SUBTOTAL do carrinho.

Na seção *private* declare um método chamado *AddItem*. Na **Listagem 4** temos sua implementação.

Após adicionar os itens ao carrinho é preciso exibi-los ao usuário. Para isso adicione um *DataGrid* e ligue-o ao nosso *DataSet*. Adicione manualmente as colunas do *DataGrid*, removendo os campos *INTERNAL_ID_PRODUTO* e *TOTAL*. Para a coluna *VALOR_PRODUTO* é preciso formatá-la como valor monetário. Assim, altere a propriedade *Data formatting expression* para "{0:c}".

Além de mostrar os produtos, é preciso exibir também o total. Altere a propriedade *ShowFooter* do *DataGrid* para *True* e no evento *ItemDataBound* digite o seguinte código:

```
if e.Item.ItemType = ListItemType.Footer then
  if GetSession.Tables[0].Rows.Count > 0 then
    e.Item.Cells[3].Text := 'Total: ' +
      System.&String.Format('{0:c}',
        GetSession.Tables[0].Rows[0]['TOTAL']);
```

Vamos alterar o *Page_Load* para que carregue o produto passado pela URL e já o exiba no *DataGrid*, usando o código da **Listagem 5**.

Vamos abrir a página *VerDetalhes.aspx* e adicionar ao lado do *Comprar Produto* um

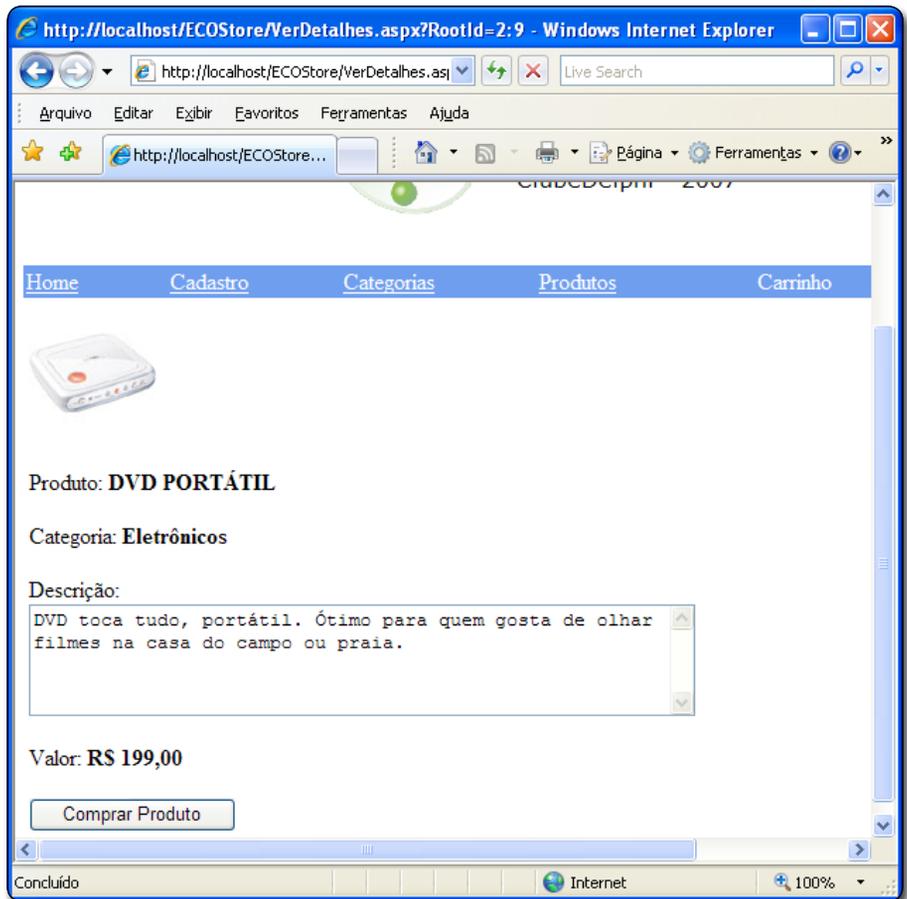


Figura 3. Página de detalhes do produto

Listagem 3. *GetSession* para armazenar o *DataSet*

```
function TWebForm1.GetSession: DataSet;
begin
  if (Session['CARRINHO'] = nil) then
    Session['CARRINHO'] := dsCarrinho;
  Result := Session['CARRINHO'] as DataSet;
end;
```

Listagem 4. Método *AddItem*

```
procedure TWebForm1.AddItem(ProdutoId, ProdutoNome,
  ProdutoValor, Qtde: string);
var
  dt: DataTable;
  row: DataRow;
  _qtde: System.Int32;
begin
  dt := GetSession.Tables[0];
  row := dt.Rows.Find(ProdutoId);
  if row = nil then
    begin
      row := dt.NewRow;
      row['INTERNAL_ID_PRODUTO'] := ProdutoId;
      row['NOME_PRODUTO'] := ProdutoNome;
      row['VALOR_PRODUTO'] := ProdutoValor;
      row['QTDE'] := Qtde;
      dt.Rows.Add(row);
    end
  else
    begin
      _qtde := row['QTDE'] as Int32;
      _qtde := _qtde + Convert.ToInt32(Qtde);
      row['QTDE'] := System.&Object(_qtde);
    end;
  dt.Columns['SUBTOTAL'].Expression :=
    'QTDE * VALOR_PRODUTO';
  dt.Columns['TOTAL'].Expression := 'SUM(SUBTOTAL)';
end;
```

Listagem 5. Incluindo o produto no DataSet

```
uses Package1Unit;
...
procedure TWebForm1.Page_Load(sender: System.Object; e: System.EventArgs);
var
  Id: string;
  CurrentProduto: Produto;
  QtdeComprada: string;
begin
  EcoSpace.Active := True;
  Id := Request.Params['RootId'];
  if Assigned(Id) and (Id <> '') then
  begin
    rhRoot.SetElement(ObjectForId(Id));
  end
  else
    Response.Redirect('Principal.aspx');
  if not IsPostBack then
  begin
    if Assigned(Id) and (Id <> '') then
    begin
      QtdeComprada := Request.Params['Qtde'].ToString;
      CurrentProduto := (rhRoot.Element.AsObject as Produto);
      AddItem(Id, CurrentProduto.Nome, CurrentProduto.Valor.ToString, QtdeComprada);
    end;
    DataGrid1.DataSource := GetSession;
    DataGrid1.DataKeyField := 'INTERNAL_ID_PRODUTO';
    DataGrid1.DataBind;
  end;
end;
```

Listagem 6. Removendo produtos do carrinho

```
procedure TWebForm1.DataGrid1_DeleteCommand(source: System.Object;
e: System.Web.UI.WebControls.DataGridCommandEventArgs);
var
  dt: DataTable;
  row: DataRow;
begin
  dt := GetSession.Tables[0];
  row := dt.Rows.Find(DataGrid1.DataKeys[e.Item.ItemIndex]);
  if row <> nil then
    dt.Rows.Remove(row);
  DataGrid1.DataSource := GetSession;
  DataGrid1.DataKeyField := 'INTERNAL_ID_PRODUTO';
  DataGrid1.DataBind;
end;
```

Listagem 7. Localizando o usuário

```
uses Package1Unit;
...
procedure TWebForm1.Button1_Click(sender: System.Object; e: System.EventArgs);
var
  CurrentUser: Usuario;
  objList: IElementCollection;
begin
  exphUsuario.Expression := 'Usuario.allInstances-'+
    '>select(currentUser | (currentUser.Nome = ''' + txtLogin.Text + ''' ) and '+
    '(currentUser.Senha = ''' + txtSenha.Text + '''))';
  if exphUsuario.GetList.Count = 0 then
    LabelErro.Text := 'Usuário inválido'
  else
  begin
    objList := exphUsuario.Element.GetAsCollection;
    CurrentUser := objList.Item[0].AsObject as Usuario;
    Session['USUARIO'] := CurrentUser;
    response.Redirect('carrinho.aspx');
  end;
end;
```

TextBox que será utilizado para indicar a quantidade que é comprada pelo cliente. Veja na **Figura 4** como ficou.

Agora no *Click* do botão é preciso chamar a página do carrinho de compras passando para ela os parâmetros necessários. Codifique o evento conforme o seguinte código (atente para o nome do *TextBox* adicionado anteriormente):

```
Response.Redirect('carrinho.aspx?RootId=' +
Request.Params['RootId']+'&Qtde=' +
txtQuantidade.Text);
```

Melhorando o carrinho

O usuário pode também retirar produtos do carrinho e devemos permitir isso. Abra a página *Carrinho.aspx* e adicione ao *DataGrid* um botão do tipo *Button Column> Delete*. No evento *DeleteCommand* adicione o código da **Listagem 6**.

Além de remover um produto do carrinho o usuário também pode incluir mais produtos ou pode encerrar a comprar. Vamos adicionar dois botões, um chamado "Continuar comprando" e "Finalizar compra". No *Click* do *Continuar comprando* retornamos para a página dos produtos usando o seguinte código:

```
Response.Redirect('Principal.aspx');
```

Rode a aplicação e faça alguns testes de inserção e exclusão de itens no carrinho (**Figura 5**).

Finalizando a compra

Para finalizar uma compra é preciso saber se temos algum usuário válido no sistema. Vamos criar agora uma página de login que será utilizada quando o usuário finalizar a compra, caso ele ainda não tenha efetuado a devida autenticação.

Inicie um novo *ECO WebForm* e nomeie como "Login.aspx". Ligue o *rhRoot* ao *EcoSpace*. Adicione um *Expression Handle*, mude seu nome para "exphUsuario". Ligue sua propriedade *RootHandle* ao *rhRoot* da página e mude para *True* a propriedade *AddExternalID*. Monte a página de login conforme a **Figura 6**.

No *Click* do *Entrar* faremos uma busca para localizar o usuário que possui o login e senha fornecido. Se encontrado adicionamos ele ao *Session* e retornamos ao carrinho, caso contrário, uma mensagem é exibida informando que o usuário não existe (**Listagem 7**).



Produto: [lblProduto]

Categoria: [lblCategoria]

Descrição:

Valor: [lblValor]

Quantidade:

Figura 4. Página de detalhes

Salvando a compra

Temos que criar as classes que representarão a venda. No *Project Manager* entre em *Model View* e dê um clique duplo sobre o *Package_1*. O diagrama de classes do nosso site surgirá. Adicione as classes e relacionamentos presentes na **Tabela 2**. Feito isso nosso diagrama de classe deve estar similar ao da **Figura 7**.

Voltamos agora ao *Project Manager* e com um duplo clique, abra o *EcoPersistenceMapperProvider.pas*, pressione F12 para ser exibido o modo de design. Veja que no rodapé temos o botão *Evolve Schema*. Clique nele para que as atualizações sejam realizadas no banco de dados (**Figura 8**).

Na *Carrinho.aspx* temos um *Finalizar compra*, nele vamos varrer o *DataSet* e inserir o conteúdo nos objetos, conforme o código da **Listagem 8**.

Ao terminar a inserção de objetos, retornamos à página *Principal.aspx*, mas poderia redirecionar para uma outra página que solicitasse os dados para pagamento, entrega etc. Agora é somente preciso editar no *User Control* incluindo o link para a página do carrinho de compras.

Classe: Item		Herda de: ClasseBase	
Atributo	Tipo		
Qtde	Integer		
Valor	Decimal		
Relacionamentos: Relacionar com a classe Produto			
End1.Name =	Item		
End2.Name =	Produto		
End2.Aggregation =	Composite		
Classe: Venda		Herdade de: ClasseBase	
Atributo	Tipo		
Data	DateTime		
Total	Decimal		
Relacionamentos: Relacionar com a classe Usuario			
Configuração do relacionamento			
End1.Name =	Venda		
End2.Name =	Usuario		
Relacionar com a classe Item			
Configuração do relacionamento			
End1.Name =	Venda		
End1.Multiplicity =	1		
Edit2.Name =	Itens		
End2.Multiplicity =	1..*		

Tabela 2. Adicionando classes e relacionamentos no diagrama



Figura 5. Carrinho de compras em execução

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



Validação de Objetos

As classes ECO possuem uma característica muito importante ao tratar de validação, a disponibilidade de uma propriedade chamada *Constraint*. A *Constraint* permite que seja definida uma ou mais validações para a classe, sendo essas compostas por uma mensagem mais a expressão OCL encarregada da validação.

Vamos definir para nossa classe produto uma *Constraint* que bloqueie produtos com nome vazio. No *Project Manager* selecione *Model View* e dê um clique duplo sobre o *Package_1*, o diagrama de classe será exibido.

Clicamos então sobre a classe *Produto* e em suas propriedades vemos *Constraints*. Clicamos nela e o editor é então mostrado, clique em *Add* e configure-o conforme **Figura 9**.

Abra o *CadProduto.aspx* e adicione um *Label* ao final da página renomeando para "LabelErros". Vamos utilizar esse *Label* para exibir as mensagens de erro dos objetos. Como a classe *Produto* possui uma *Constraint*, antes de salvar o objeto vamos obter essas *Constraints* e testá-las uma a uma, aquela que retornar *False* terá sua mensagem de erro exibida na tela através do *LabelErros*. Isso é feito na **Listagem 9**.

Através da interface *IObject* conseguimos acessar os metadados do modelo UML e executar a *Constraint* para o objeto passado como parâmetro. Altere o código do *Salvar* para executar a verificação das *Constraints* antes de salvar o produto, utilizando o código da **Listagem 10**.

Conclusão

Terminamos por aqui! Vimos nessa série de três artigos um pouco sobre o ECO e os recursos que são disponibilizados. O objetivo desse mini-curso é de despertar o interesse pelo ECO fazendo com que você, leitor, inicie uma pesquisa mais aprofundada sobre o assunto e aos poucos ir adotando a tecnologia.

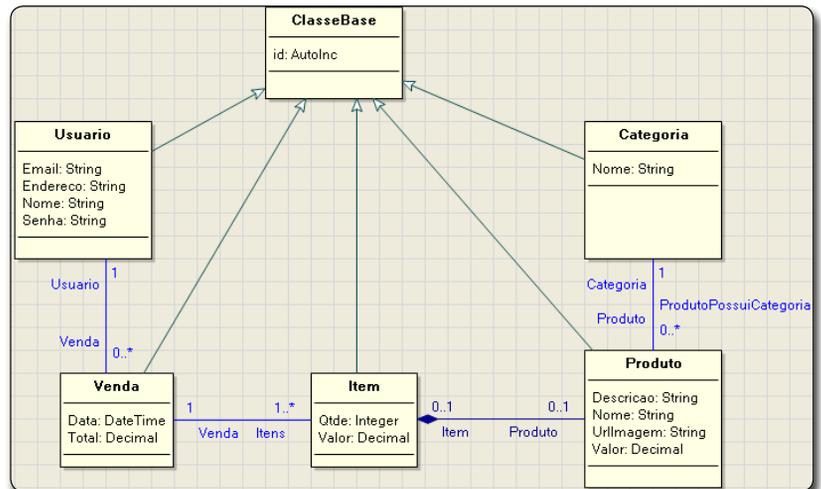


Figura 7. Diagrama de classes

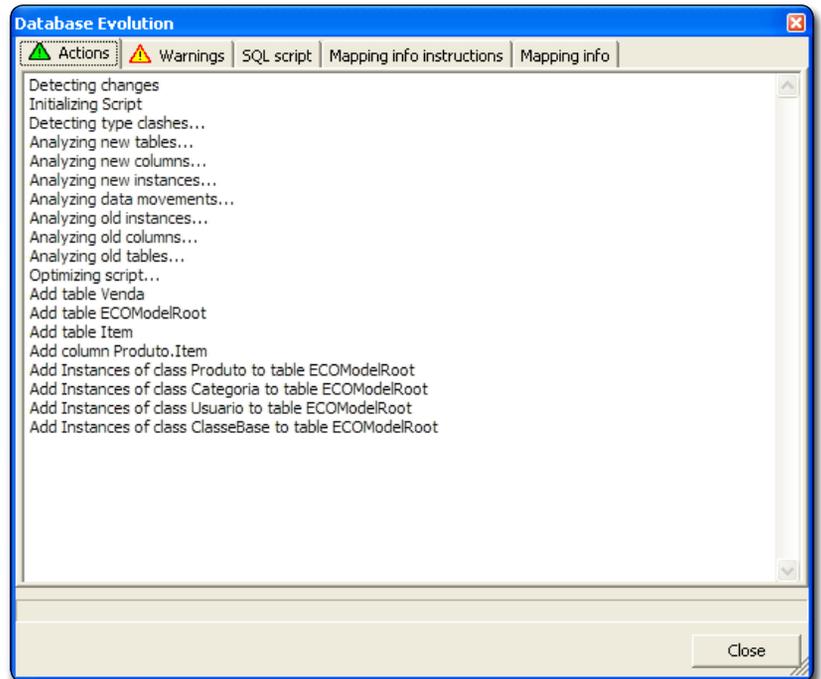


Figura 8. Evolve Database

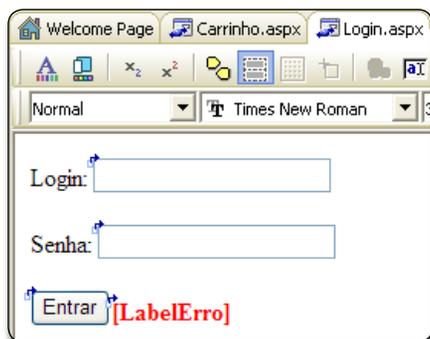


Figura 6. Página de login

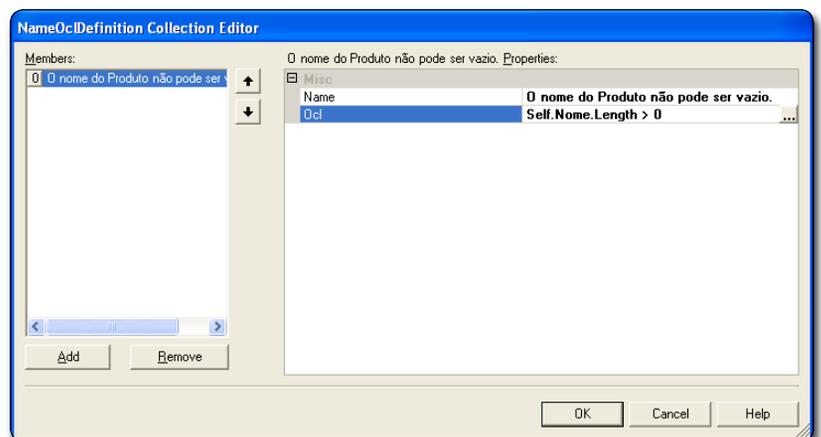


Figura 9. Definindo uma constraint

Usando a abordagem aqui apresentada, você pode agora desenvolver suas aplicações Delphi de forma mais orientadas a objetos (senão 100% OO), obtendo inúmeros

benefícios, como reutilização de código, desenvolvimento em camadas, separação da lógica de negócio entre outros.

Um grande abraço e até a próxima! ●

Listagem 8. Salvando a compra

```

procedure TWebForm1.btFinalizaCompra_Click(sender: System.Object; e: System.EventArgs);
begin
  if Session['USUARIO'] = nil then
    Response.Redirect('Login.aspx')
  else
    FinalizaCompra;
end;

procedure TWebForm1.FinalizaCompra;
var
  row: DataRow;
  lVenda: Venda;
  lItem: Item;
  obj: IObjetoInstance;
begin
  lVenda := Venda.Create(EcoSpace);
  lVenda.Data := DateTime.Now;
  obj := ObjectForId(Session['USUARIO'].ToString);
  lVenda.Usuario := obj.AsObject as Usuario;
  lVenda.Total := Convert.ToDecimal(GetSession.Tables[0].Rows[0]['TOTAL']);
  for row in GetSession.Tables[0].Rows do
  begin
    lItem := Item.Create(EcoSpace);
    lItem.Qtde := Convert.ToInt32(row['QTDE']);
    lItem.Valor := Convert.ToDecimal(row['VALOR_PRODUTO']);
    obj := ObjectForId(row['INTERNAL_ID_PRODUTO'].ToString);
    lItem.Produto := obj.AsObject as Produto;
    lItem.Venda := lVenda;
  end;
  UpdateDatabase;
  Response.Redirect('Principal.aspx');
end;
    
```

Listagem 9. Obtendo e testando as constraints

```

procedure TWebForm1.GetConstraintsForObject(
  Instance: IObjeto; var Constraints: ArrayList);
var
  Index: Integer;
  Ocl: IOclService;
begin
  Constraints.Clear;
  LabelErros.Text := '';
  if not Instance.Deleted then
    for Index := 0 to Instance.UmlType.Constraints.Count - 1 do
    begin
      Ocl := Instance.ServiceProvider.GetEcoService(
        typeof(IOclService) as IOclService;
      if (Ocl.EvaluateAndSubscribe(Instance,
        Instance.UmlType.Constraints.Item[
          Index].Body.Body, nil, nil).
        AsObject as Boolean = False) then
        begin
          Constraints.Add(Instance.UmlType.Constraints.
            Item[Index].Body.Body.ToString);
          LabelErros.Text := LabelErros.Text + #13 +
            Instance.UmlType.Constraints.Item[
              Index].Body.Body.ToString;
        end;
      end;
    end;
end;
    
```

Listagem 10. Verificando se o objeto pode ser salvo

```

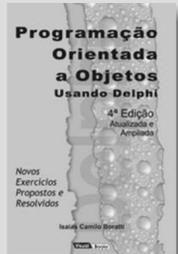
procedure TWebForm1.btSalvar_Click(
  sender: System.Object; e: System.EventArgs);
var
  lProduto: Produto;
  lConstraints: ArrayList;
begin
  ...
  btSalvar.Text := lProduto.Categoria.Nome;
  GetConstraintsForObject(lProduto.AsIObjeto,
    lConstraints);
  if lConstraints.Count = 0 then
  begin
    UpdateDatabase;
    Response.Redirect('ConsultaProduto.aspx');
  end;
end;
    
```



Microsoft SQL Server 2005
Fundamentos de Bancos de Dados
Bookman
R\$ 68,00

Programação Orientada a Objetos Usando Delphi
4ª Edição

Visual Books
R\$ 75,00



Dominando o PostgreSQL
Ciência Moderna
R\$ 39,20

Qualidade de Software
2ª Edição

Ciência Moderna
R\$ 179,00



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

Validação por imagem na Web

É cada dia mais comum o uso de imagens de validação em Web Sites. Mais que um quesito de segurança, é um artefato altamente recomendável para o controle de sua aplicação. Este artigo traz uma implementação desse tipo de validação, além de algumas sugestões para incrementar a segurança de aplicações com Delphi e ASP.NET, usando esse recurso.

Por que usar imagens de validação?

Atualmente, a maioria dos *bits* que trafegam na Internet, carregam conteúdos como *spam*, mídias ilegais, além de *robots*, que são aplicações que simulam requisições no servidor, ou seja, é como se alguém clicasse no botão *Submit* da sua aplicação.

Em virtude dessas ocorrências, cada vez mais é necessária a implementação de controles para incrementar a segurança dos sistemas usados na Internet. Abordaremos neste artigo, como criar uma inter-

face que solicite a digitação de um código de verificação que foi gerado através de uma imagem. Imagine um site de votação para um concurso (Figura 1).

Para não “cansar o braço” votando, um usuário, cria um *robot* que acessa a URL destino do formulário (*action*), enviando o código do seu candidato favorito (como, por exemplo, o link <http://www.website.com.br/vote.aspx?idCandidato=10>), que pode ser obtido através de uma simples consulta no código-fonte do *browser*.

Sendo assim, ele poderá fazer com que seu candidato ganhe o concurso, burlando a votação, pois o *robot* poderá ficar 24h por dia votando sem parar, em múltiplas instâncias de execução simultânea, inclusive. Esse é apenas um dos casos em que *robots* podem ser prejudiciais em aplicações Web.

Usando validação por imagem

Uma das implementações de segurança é o uso de imagens de validação em



Pedro Bajotto Filho

(bajotto@gmail.com)

é Analista de Sistemas com mais de 10 anos de experiência na área. Presta consultoria em Porto Alegre/RS, onde trabalha com C#, Java entre outras linguagens, nas plataformas Windows e Linux. Utiliza o framework .NET desde a primeira versão Beta.

formulários Web, pois no momento da escolha do candidato favorito, o usuário deverá digitar o código impresso na imagem, que será validado no servidor para confirmar o voto.

O uso bem planejado dessas imagens pode evitar o acesso dos *robots* em sua aplicação. As imagens de validação possuem um conteúdo, geralmente formado por caracteres aleatórios ou expressões, que devem ser digitados manualmente em uma caixa de texto, para validar o envio de informações no *submit* de um formulário.

Ao contrário de expressões em texto puro, que são embutidas no HTML e poderiam ser facilmente descobertas por um *robot*, o texto de uma imagem só é perceptível ao olho humano. Ou pelo menos deveria.

Entretanto, gerar essa imagem não é o suficiente. São conhecidas as técnicas que conseguem rastrear o conteúdo de uma imagem e obter o texto digitado (técnica conhecida muitas vezes como OCR). Por esse motivo, é necessário gerar uma imagem com atributos que “atrapalhem” o rastreamento do conteúdo.

A sugestão de implementação que veremos certamente poderá ser incrementada com o uso da criatividade do desenvolvedor, para criar um conteúdo que possa ser reconhecido pela visão, mas não por algum algoritmo de rastreamento de imagem.

A imagem que criaremos será gerada por uma aplicação ASP.NET, com *Respondo* do tipo *image/jpeg*. Em outras palavras,

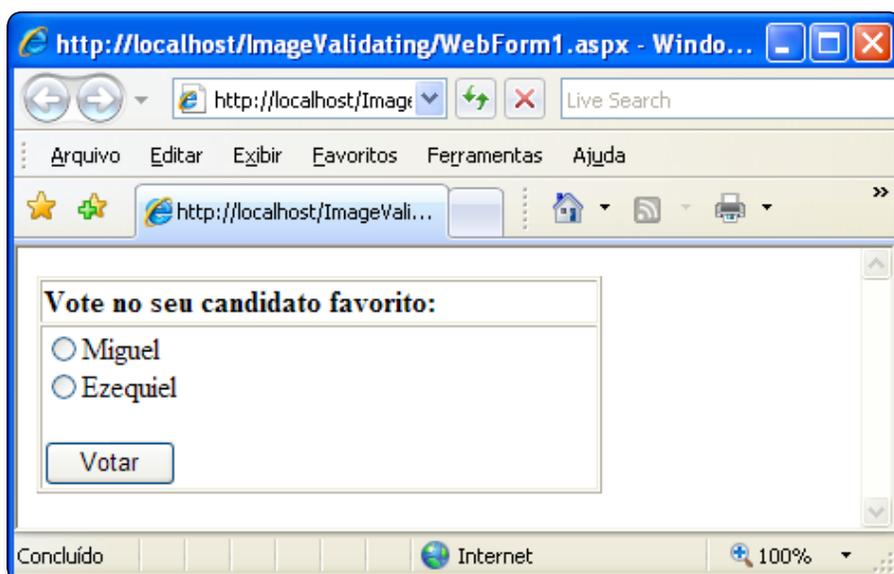


Figura 1. Formulário de cadastro de exemplo

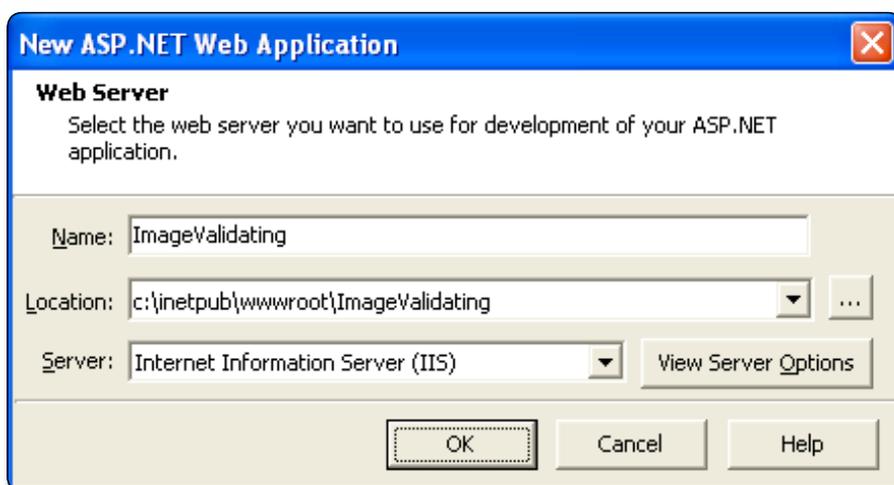


Figura 2. Criando o projeto ASP.NET

“ O Maior Portal para Desenvolvedores da América Latina.”



Acesse:

www.devmedia.com.br

Listagem 1. Método GenerateString

```
function TWebForm2.GenerateString: string;
var
  rnd: System.Random;
  validatingText: string;
  i: integer;
  chr: Char;
begin
  validatingText := '';
  for i := 0 to 6 do
  begin
    chr := Convert.ToChar(rnd.Next(65, 90));
    validatingText := validatingText + chr.ToString;
  end;
  Result := validatingText.ToUpper;
end;
```

Listagem 2. Page_Load da página

```
uses System.Drawing.Imaging, System.Drawing.Text, System.IO;
...
var
  x, y: integer;
  strValidation: string;
  rnd: System.Random;
  font: System.Drawing.Font;
  bitmap: System.Drawing.Bitmap;
  gr: System.Drawing.Graphics;
  Rectangle: System.Drawing.Rectangle;
  Pen: System.Drawing.Pen;
  Point: System.Drawing.Point;
begin
  strValidation := '';
  rnd := System.Random.Create;;

  Response.ContentType := 'image/jpeg';
  Response.Clear();
  Response.BufferOutput := True;

  strValidation := GenerateString;
  Session['strValidation'] := strValidation;

  font := System.Drawing.Font.Create('Arial', rnd.Next(17,20));
  bitmap := System.Drawing.Bitmap.Create(200, 50);
  gr := Graphics.FromImage(bitmap);

  Pen := System.Drawing.Pen.Create(Color.White);
  gr.FillRectangle(Brushes.BlueViolet, Rectangle.Create(0, 0, bitmap.Width, bitmap.Height));
  gr.DrawString(strValidation, font, Brushes.White, rnd.Next(70), rnd.Next(20));
  gr.DrawLine(Pen, Point.Create(0, rnd.Next(50)), Point.Create(200, rnd.Next(50)));
  gr.DrawLine(Pen, Point.Create(0, rnd.Next(50)), Point.Create(200, rnd.Next(50)));
  gr.DrawLine(Pen, Point.Create(0, rnd.Next(50)), Point.Create(200, rnd.Next(50)));

  for x := 0 to bitmap.Width - 1 do
    for y := 0 to bitmap.Height - 1 do
      if (rnd.Next(4) = 1) then
        bitmap.SetPixel(x, y, Color.White);

  font.Dispose;
  gr.Dispose;
  bitmap.Save(Response.OutputStream, ImageFormat.Jpeg);
  bitmap.Dispose();
end;
```

Listagem 3. Validando o conteúdo da imagem

```
if txtValidacao.Text = Convert.ToString(Session[
  'strValidation']) then
  lblResultado.Text :=
    'Seu voto foi computado com sucesso!'
else
  lblResultado.Text := 'Seu voto é inválido!';
```

teremos um documento ASPX, cujo conteúdo será a própria imagem. Agora que já vimos os fundamentos, vamos à prática.

Gerando a imagem

Abra o Delphi 8, 2005 ou 2006 e crie uma aplicação ASP.NET, dando o nome de "ImageValidating" para o projeto e clique em OK (Figura 2).

Adicione um novo arquivo ASPX (*New Items>New ASP.NET Files>ASP.NET Page*) e dê o nome de "image.aspx". Começaremos a implementação do código pelo método que definirá o conteúdo da imagem, ou seja, o texto que deverá ser validado através da digitação no *TextBox*. Adicione o *GenerateString* (daremos esse nome ao método), conforme a **Listagem 1**.

O código da listagem anterior gera uma *string* com sete caracteres randômicos (nesse caso letras maiúsculas), que irão compor a validação. Essa *string* será inserida em um *Session* que será invocado posteriormente para a validação. No *Page_Load* da página adicione o código da **Listagem 2**.

Conforme podemos observar na listagem anterior, o tipo de conteúdo de resposta do documento (*Response.ContentType*) será *image/jpeg*, ou seja uma imagem. Ainda salientamos que a imagem será descarregada de uma só vez, através do comando *Response.BufferOutput = true*.

Aqui usamos a classe *Graphic* para gerar a imagem. Através de seus métodos, como *DrawString*, *DrawLine* e *FillRectangle* criamos a imagem desejada dentro de um objeto *Bitmap*.

Entendendo o conteúdo da imagem

Conforme dito anteriormente, gerar a mensagem não é o suficiente, pois existem *robots* que acessam a URL de sua imagem com o intuito de rastreá-la e obter o conteúdo da mesma.

A primeira ação que devemos tomar é no sentido de criar uma imagem o mais randômica possível, ou seja, que menos possua padrões de posicionamento de caracteres, de tamanho de fonte etc.

No código anterior, veja que os tamanhos da fonte dos caracteres de validação não possuem um tamanho fixo, justamente para evitar que um *robot* seja configurado para rastrear o conteúdo de uma imagem cujo tamanho da fonte é conhecido.

Seguindo o mesmo raciocínio, a *string* de validação é "desenhada" na imagem. Novamente não temos um posicionamento definido para a mesma. Com esse cuidado evitamos que um *robot* acesse um ponto cardinal específico, onde ele saiba que a nossa *string* desenhada está localizada.

A seguir são tomadas ações para que a imagem não fique somente com os caracteres de validação impressos. Nesse caso, são inseridas algumas linhas inclinadas na horizontal, e a seguir um *loop* que "sorteia" pontos da tela, onde são inseridos *pixels*.

Tudo isso para, justamente, não deixar

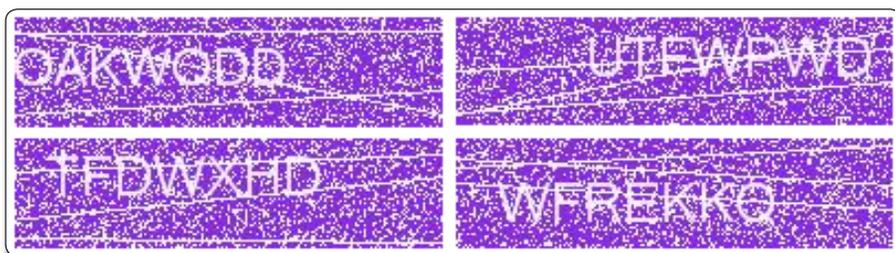


Figura 3. Exemplos de imagens geradas

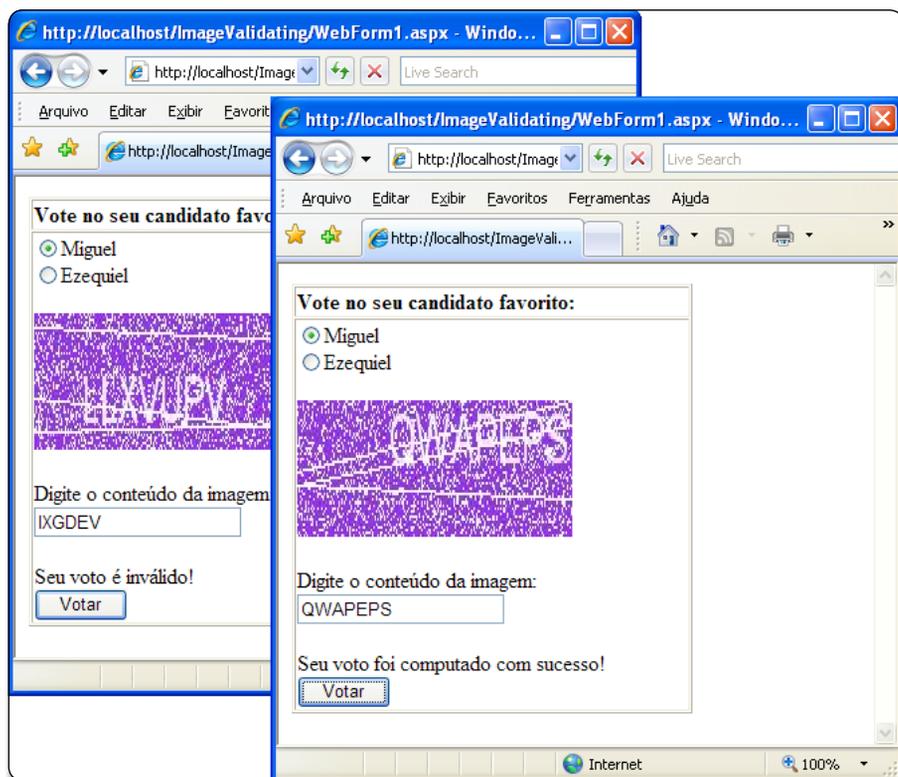


Figura 5. Validando o conteúdo digitado pelo usuário

a imagem com um padrão de formatação que possa ser previsto. Ao acessarmos a URL da imagem (Figura 3), podemos observar que a mesma não possui um padrão determinado, o que certamente torna o seu sistema mais seguro.

Inserindo a validação no formulário de votação

Volte à página *Default.aspx* e crie um layout semelhante ao da Figura 1. Adicione um *Image* e altere a propriedade *ImageUrl* para o *WebForm1*, que na realidade gera o conteúdo no formato de uma imagem *jpeg* (digite "image.aspx"). Adicione um *TextBox* para a digitação da *string* de validação e um *Label* para apresentar o resultado da validação. Veja como ficou a página na Figura 4.

Validando o conteúdo da imagem

Adicione o código da **Listagem 3** no evento *Click* do botão. Aqui simplesmente testamos se o valor digitado no *TextBox* corresponde ao valor armazenado em sessão. Ou seja, a digitação foi válida. Veja na Figura 5 o resultado da aplicação em execução.

Dicas para incrementar a sua imagem de validação

O código apresentado para a geração da imagem é uma sugestão, pois a verdadeira intenção é apresentar técnicas de proteção para sua aplicação. Conforme vimos anteriormente, a imagem gerada deve ser a mais randômica possível, ou seja, apesar de apresentar um número fixo de caracteres, esses devem ser apre-

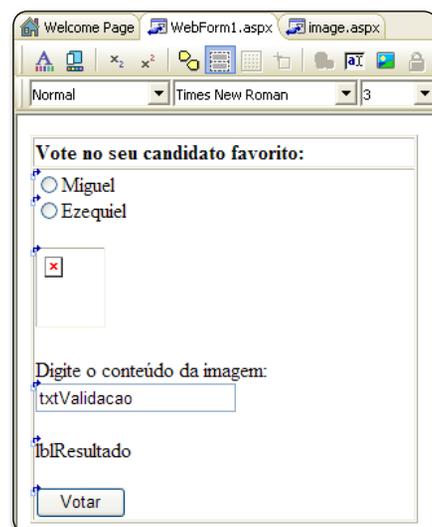


Figura 4. Inserindo a imagem no formulário de votação

sentados de forma não previsível.

Para tanto, incrementos como fazer com que a aplicação altere o tipo da fonte usada, alternância das cores, rotação, tanto do *background*, como da fonte e dos outros elementos, são ações que podem auxiliar ainda mais na segurança.

Lembre-se, ainda, que a legibilidade do conteúdo a ser validado é muito importante, pois o usuário precisa reconhecer visualmente os caracteres que vai submeter à validação, claro.

Conclusão

As aplicações que ficam disponíveis na Web devem ser protegidas para que os recursos de servidor estejam sempre acessíveis. Para tanto, é imprescindível que as técnicas aqui apresentadas sejam submetidas a uma análise que gradue o nível de segurança necessário que deve ser incorporado à solução desenvolvida.

É necessário, também, que o desenvolvedor analise e incremente, sempre que possível, o algoritmo de geração de imagem, para que suas técnicas não fiquem obsoletas (ou conhecidas) e para que eventuais vulnerabilidades sejam detectadas e corrigidas. ●

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo portal do assinante ClubeDelphi e assista a uma vídeo-aula de Luciano Pimenta que mostra como usar um componente para validação de imagens.

www.devmedia.com.br/articles/viewcomp.asp?comp=4378



Criando um menu Favoritos na aplicação

Antes de iniciar gostaria de transferir parte dos créditos deste artigo para meu colega de trabalho, Thiago Filiano, que ajudou a desenvolver essa rotina e hoje funciona perfeitamente em nossos sistemas comerciais.

Ao longo de minha carreira, já deparei com diversas solicitações de clientes. Isso é bastante comum e acaba fazendo com que o sistema evolua a crescer cada vez mais, adquirindo novas funcionalidades e agregando ainda mais valor.

Recentemente surgiu a necessidade de implementarmos uma rotina onde o usuário pudesse criar seus próprios atalhos para as telas que mais utiliza. Chamamos tal implementação de, naturalmente, *Favoritos*.

A idéia era permitir que o usuário pudesse incluir rapidamente um formulário como sendo Favorito e assim acessá-lo por meio de uma tecla de atalho ou menu. Neste artigo, mostrarei a técnica utilizada pra criar tal rotina. Mãos à massa!

Entendendo o exemplo

O exemplo consiste em criar um arquivo XML que receberá as telas que o usuário marcar como Favoritos. Ao carregar o sistema, o arquivo (*Favoritos.xml*) é lido e seus itens (formulários) são adicionados em um *PopupMenu* (*popFavoritos*), que por sua vez é vinculado a um *ToolButton* (*tbnFavoritos*) permitindo que o item seja acessado pelo botão ou através da tecla de atalho definida.

Fixaremos as teclas de atalho de F2 até F12 e o usuário não poderá modificá-las, ou seja, as teclas serão atribuídas automaticamente pelo sistema em ordem cronológica. Será permitido apenas excluir um atalho para ter a possibilidade de adicionar outro.

Desenhando a tela de exemplo

Vamos simular uma aplicação desenhando uma tela semelhante à **Figura 1**. Em nossa tela principal inclua um *MainMenu* e alguns itens conforme a **Figura 2**.



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.

Em seguida insira um *PopupMenu* (“popFavoritos”). Inclua também um *ToolBar* e alguns botões sendo que o último deles, *Favoritos* (“tbnFavoritos”) tem algumas características especiais (propriedades), como:

```
DropDownMenu = popFavoritos
Style = tbsDropDown
```

Vamos criar duas telas que servirão pra simular os cadastros de clientes e fornecedores. Para isso crie dois formulários clicando em *File>New>Form*. Salve o primeiro como “frmClientes.pas” e mude seu *Name* para “fClientes”.

Faça o mesmo para a segunda tela, dessa vez com o nome de arquivo de “frmFornecedores.pas” e seu *Name* como “fFornecedores”. Se preferir, insira alguns componentes nos formulários criados (Figura 3).

Registrando os forms

Os formulários do exemplo serão abertos usando o nome dos mesmos, por isso precisamos registrar as suas classes em nossa aplicação. No final da *unit* de cada formulário, antes do *end*, devemos incluir as seções *Initialization* e *Finalization* e em cada uma chamar *RegisterClass* e *UnRegisterClass*, respectivamente. O parâmetro dos métodos é o nome da classe que será registrada, por exemplo:

```
initialization
  Classes.RegisterClass(TfClientes);
finalization
  Classes.UnRegisterClass(TfClientes);
```

A função AbrirForm

Neste ponto do artigo precisamos escrever os códigos que farão a carga, utilização e manutenção do menu Favoritos. Vamos começar desenvolvendo uma função capaz de abrir os formulários utilizando apenas o nome deles. Isso se faz necessário, pois no arquivo XML, guardaremos o nome do formulário em *string* juntamente com a descrição.

Crie a função *AbrirForm* na seção *private* do formulário principal:

```
function AbrirForm(AForm: string): TForm;
```

Em seguida pressione CTRL+SHIFT+C pra criar o corpo da função. A função recebe uma *string* com o nome do formulário. Então localizamos sua classe na lista de formulários da aplicação, e caso seja

encontrado, o mesmo será criado.

Neste momento configuramos sua propriedade *KeyPreview* para *True* e associamos seu evento *OnKeyPress* à função *AdicionarAtalho* que será criada mais adiante. Por fim, chamamos o *ShowModal* pra mostrar a tela. Veja o código completo na **Listagem 1**.

Para que o usuário possa adicionar rapidamente a tela ao menu Favoritos, todos os formulários criados pela função *AbrirForm* possuirão o evento *OnKeyPress*

apontado para o procedimento *AdicionarAtalho*. O evento detecta o pressionamento das teclas CTRL+F12 (técnica para adicionar o formulário atual em Favorito), automaticamente o torna um item Favorito e poderá ser acessado pela tela principal de nossa aplicação.

A função AdicionarAtalho

Na seção *public*, crie a função *AdicionarAtalho* e pressione CTRL+SHIFT+C. Implemente a mesma, conforme a **Listagem 2**.

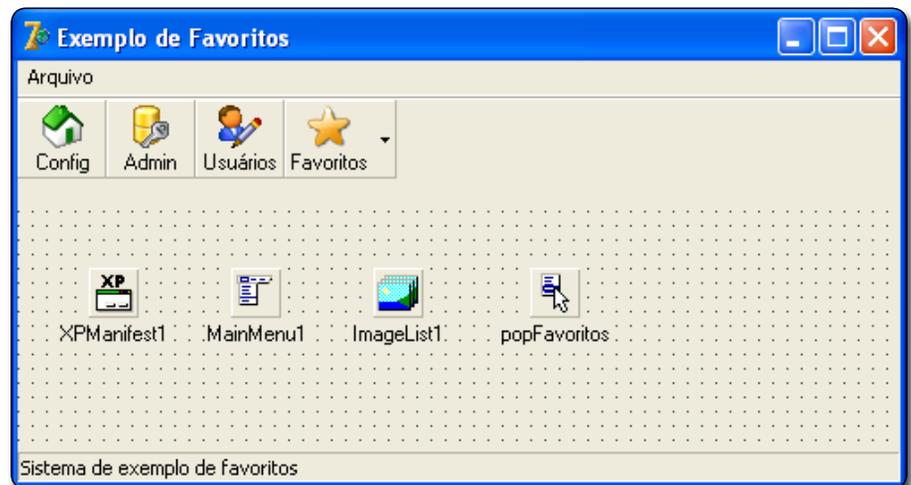


Figura 1. Tela de exemplo da aplicação

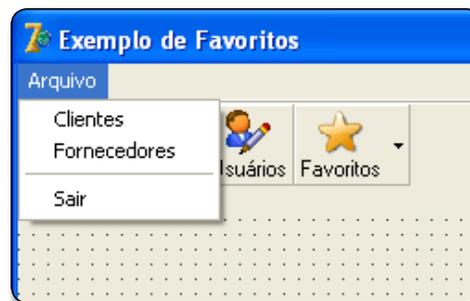


Figura 2. Menu principal da aplicação

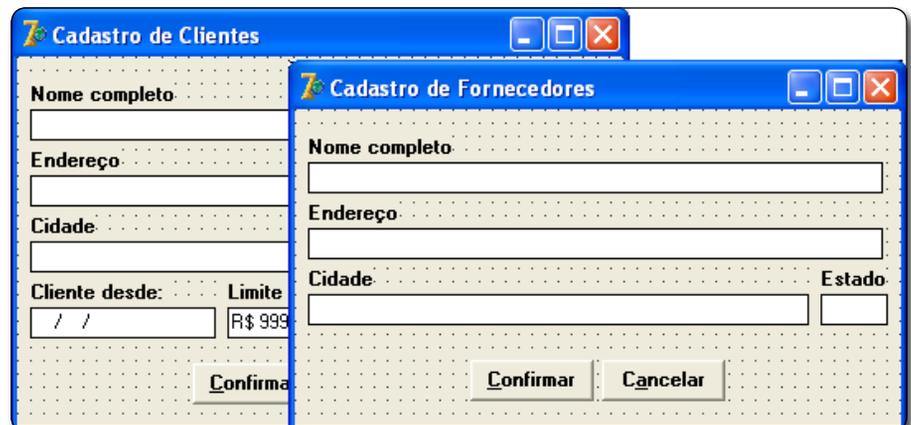


Figura 3. Exemplos de tela de cadastro de clientes e fornecedores

A explicação do código anterior é simples: primeiro verificamos se as teclas pressionadas são CTRL+F12 e em caso positivo criamos um *ClientDataSet*. Configuramos sua propriedade *Filter* com a string *FORM=i* + o nome do formulário aberto. Com o filtro ativo chamamos o *LoadFromFile* do componente, indicando o caminho para o arquivo *Favoritos.xml*.

Caso sejam retornados registros, significa que o formulário atual encontra-se como item de Favoritos, do contrário, limpamos o filtro e passamos para a próxima verificação. Verificamos se existe "vaga" para um novo item. Se houver menos que 11 registros no XML, podemos incluir um novo item.

Essa verificação é necessária, pois determinamos que fossem usadas apenas as teclas de F2 a F12 como atalhos que dão a soma de 11 itens. Em caso positivo basta inserir o item (*Append*), informar o nome do formulário e descrição nos campos presentes no XML e então salvar (*Post*). Para persistir os dados necessitamos chamar o *SaveToFile* do *ClientDataSet*

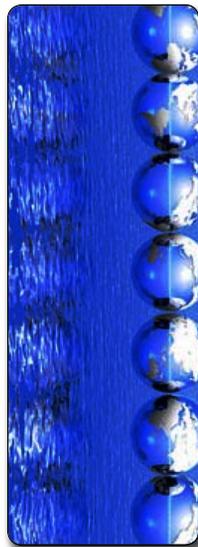
para que seja efetivada a inclusão do registro atual.

Criando o arquivo de favoritos

Para garantir que o *Favoritos.xml* exista no diretório da aplicação quando a mesma for aberta, criaremos agora uma *procedure* que cria a estrutura do arquivo. Implemen-

te a *CriarFavoritos* conforme a **Listagem 3**.

Pra quem nunca criou um arquivo XML a estrutura é simples e bastante parecida com a criação de arquivos *Paradox*. Criamos em tempo de execução um *ClientDataSet* e em sua propriedade *FieldDefs* adicionamos os campos FX, FORM e DESCRICAO conforme a **Tabela 1**.



Listagem 1. Função AbrirForm

```
function AbrirForm(AForm: string): TForm;
var
  Pc: TPersistentClass;
  frmChamado: TForm;
  i: Integer;
begin
  try
    Pc := GetClass('T' + AForm);
    if (Pc <> nil) then
      begin
        frmChamado := nil;
        i := 0;
        while i < Screen.FormCount do
          begin
            if Screen.Forms[i] is Pc then
              frmChamado := Screen.Forms[i];
              Inc(i);
            end;
          end;
          try
            frmChamado := TFormClass(Pc).Create(frmChamado);
            frmChamado.KeyPreview := True;
            frmChamado.OnKeyDown :=
              fPrincipal.AdicionarAtalho;
            frmChamado.ShowModal;
          except on E: Exception do
            begin
              MessageDlg('Ocorreu um erro ao carregar essa função.' +
                #13 + 'Erro original: ' +
                #13#13 + E.Message, mtWarning, [mbOk], 0);
              Result := nil;
              Exit;
            end;
          end;
          Result := frmChamado;
        end
      else
        begin
          MessageDlg('Ocorreu um erro ao carregar essa função.' +
            #13 + 'Janela: ' + AForm, mtWarning, [mbOk], 0);
          Result := nil;
        end;
        Screen.Cursor := crDefault;
      except
        Result := nil;
      end;
    end;
end;
```

Listagem 2. Função AdicionarAtalho

```
procedure TfPrincipal.AdicionarAtalho(
  Sender: TObject; var Key: Word;
  Shift: TShiftState);
var
  cdsFavoritos: TClientDataSet;
  NomeForm: string;
  function RetornarCaption(AForm: string): string;
  var
    Pc: TPersistentClass;
    frmChamado: TForm;
    i Integer;
  begin
    try
      Pc := GetClass('T' + AForm);
      frmChamado := nil;
      i := 0;
      while i < Screen.FormCount do
        begin
          if Screen.Forms[i] is Pc then
            frmChamado := Screen.Forms[i];
            Inc(i);
          end;
          if Length(frmChamado.Caption) > 30 then
            Result := Copy(frmChamado.Caption, 1, 30) +
              '...';
          else
            Result := frmChamado.Caption;
          end;
        except
          end;
      end;
    begin
      if (Key in [VK_F12]) and (ssCtrl in Shift) then
        begin
          cdsFavoritos := TClientDataSet.Create(fPrincipal);
          with cdsFavoritos do
            begin
              (Primeiro verifica se já existe ou não o
                formulário que está sendo adicionado)
              NomeForm := TComponent(Sender).Name;
              cdsFavoritos.Filter := 'FORM=' +
                QuotedStr(NomeForm);
              cdsFavoritos.Filtered := True;
              cdsFavoritos.LoadFromFile(ExtractFilePath(
                Application.ExeName) + 'Favoritos.xml');
              cdsFavoritos.Open;
              if not cdsFavoritos.IsEmpty then
                MessageDlg('Esta janela já foi adicionada ao '+
                  'seu favoritos.', mtWarning, [mbOk], 0)
              else
                begin
                  cdsFavoritos.Filtered := False;
                  cdsFavoritos.Filter := '';
                  cdsFavoritos.Filtered := True;
                  if cdsFavoritos.RecordCount = 11 then
                    MessageDlg('Limite de favoritos excedido!',
                      mtInformation, [mbOk], 0)
                  else
                    begin
                      cdsFavoritos.Append;
                      cdsFavoritos.FieldName('FORM').AsString :=
                        NomeForm;
                      cdsFavoritos.FieldName(
                        'DESCRICAO').AsString :=
                        RetornarCaption(NomeForm);
                      cdsFavoritos.Post;
                      MessageDlg('Favoritos salvo com sucesso.',
                        mtInformation, [mbOk], 0);
                      cdsFavoritos.SaveToFile(ExtractFilePath(
                        Application.ExeName) + 'Favoritos.xml',
                        dfXMLUTF8);
                    end;
                  end;
                end;
              cdsFavoritos.Free;
              CarregarFavoritos;
            end;
          end;
        end;
      end;
    end;
end;
```

Adicionamos também dois índices na propriedade *IndexDefs*. Em seguida chamamos o *CreateDataSet* e por fim *SaveToFile* para a criação física do arquivo.

Lendo o arquivo de favoritos

Agora implementaremos o código para ler o arquivo XML e atualizar o *popFavoritos*. Novamente inclua o procedimento na seção *public* com o nome de *CarregarFavoritos* e codifique como na **Listagem 4**.

Não há segredos aqui. Apenas carregamos o arquivo *Favoritos.xml* e lemos o seu conteúdo. Em tempo de execução limpamos a propriedade *Items* do *popFavoritos* e então inserimos novos itens. Cada item recebe uma tecla de atalho definida pela variável *Vk_F* incrementada durante o *loop*. O valor inicial dela é 113 que equivale a tecla F2 até a F12. Por fim apontamos o evento *OnClick* de cada item para a *FavoritosClick* que criaremos a seguir.

Ao término do *loop* incluímos também uma barra divisória e em seguida um item fixo chamado *Personalizar*. Esse servirá para que o usuário possa excluir os favoritos que não deseja mais. Seu evento é apontado para o próprio evento *OnClick* do *tbmFavoritos* que por sua vez chama a tela de manutenção dos favoritos.

Nota: Observe que cada item adicionado no *popFavoritos* recebe o nome do formulário (*fClientes*, *fFornecedores* etc.) em sua propriedade *Name*.

Abrindo um favorito

Como dito anteriormente, cada item do *popFavoritos* é apontado para o *FavoritosClick*. Esse procedimento abre o formulário gravado nos favoritos. Crie uma nova *procedure* como a seguir:

```
procedure TfPrincipal.FavoritosClick
(Sender: TObject);
begin
  AbrirForm(TMenuItem(Sender).Name);
end;
```

Campo	Tipo/Tamanho
FX	String (3)
FORM	String (20)
DESCRICA0	String (40)

Tabela 1. Estrutura do arquivo *Favoritos.xml*

Listagem 3. CriarFavoritos

```
procedure TfPrincipal.CriarFavoritos;
var
  cdsFavoritos: TClientDataSet;
begin
  { Cria o arquivo de favoritos obrigatório }
  if not FileExists(ExtractFilePath(
    Application.ExeName) + 'Favoritos.xml') then
  begin
    cdsFavoritos := TClientDataSet.Create(fPrincipal);
    with cdsFavoritos do
      begin
        Close;
        with FieldDefs do
          begin
            Clear;
            Add('FX', ftString, 3, False);
            Add('FORM', ftString, 20, False);
            Add('DESCRICA0', ftString, 40, False);
          end;

          with IndexDefs do
            begin
              Clear;
              Add('PRIMARIO', 'FORM', [ixPrimary, ixUnique]);
              Add('FILTRO', 'FX;FORM;DESCRICA0', [ixCaseInsensitive]);
            end;
            CreateDataSet;
            SaveToFile(ExtractFilePath(Application.ExeName) + 'Favoritos.xml', dfXMLUTF8);
            cdsFavoritos.Free;
          end;
        end;
      end;
end;
```

Listagem 4. Procedure CarregarFavoritos

```
procedure TfPrincipal.CarregarFavoritos;
var
  i: Integer;
  Item: TMenuItem;
  Vk_F: Integer;
  cdsFavoritos: TClientDataSet;
begin
  popFavoritos.Items.Clear;
  { 113 = F2 - 0 resto é sequencial }
  Vk_F := 113;
  i := -1;
  cdsFavoritos := TClientDataSet.Create(fPrincipal);
  cdsFavoritos.LoadFromFile(ExtractFilePath(Application.ExeName) + 'Favoritos.xml');
  cdsFavoritos.Open;

  if not cdsFavoritos.IsEmpty then
  begin
    while not cdsFavoritos.Eof do
      begin
        Inc(i);
        Item := TMenuItem.Create(Self);
        Item.Caption := cdsFavoritos.FieldName('DESCRICA0').AsString;
        popFavoritos.Items.Insert(i, Item);
        popFavoritos.Items[i].Name := cdsFavoritos.FieldName('FORM').AsString;
        popFavoritos.Items[i].ShortCut := Vk_F;
        popFavoritos.Items[i].OnClick := FavoritosClick;
        Inc(Vk_F);
        cdsFavoritos.Next;
      end;
      Inc(i);
      Item := TMenuItem.Create(Self);
      Item.Caption := '-';
      popFavoritos.Items.Insert(i, Item);
      popFavoritos.Items[i].Name := 'barral';

      Inc(i);
      Item := TMenuItem.Create(Self);
      Item.Caption := 'Personalizar...';
      popFavoritos.Items.Insert(i, Item);
      popFavoritos.Items[i].OnClick := tbmFavoritosClick;
    end;
  end;
else
  begin
    Item := TMenuItem.Create(Self);
    Item.Caption := '<favoritos vazio>';
    popFavoritos.Items.Insert(0, Item);
    popFavoritos.Items[0].Name := 'vazio';
  end;
  cdsFavoritos.Close;
  cdsFavoritos.Free;
end;
```

Listagem 5. Eventos OnShow e OnClick no MainMenu

```
procedure TfPrincipal.FormShow(Sender: TObject);
begin
  CriarFavoritos;
  CarregarFavoritos;
end;

procedure TfPrincipal.CadastrodeFornecedores1Click(
  Sender: TObject);
begin
  AbrirForm('fFornecedores');
end;

procedure TfPrincipal.CadastrodeClientes1Click(
  Sender: TObject);
begin
  AbrirForm('fClientes');
end;
```

Listagem 6. Excluindo um Favorito do ClientDataSet

```
if Key = VK_DELETE then
begin
  cdsFavoritos.Delete;
  cdsFavoritos.SaveToFile(ExtractFilePath(
    Application.ExeName) + 'Favoritos.xml',
    dfXMLUTF8);
end;
```

A explicação é bem simples. Chamamos *AbrirForm* usando o nome do formulário que está contido na propriedade *Name* do item de menu. Ex.: O item de menu referente ao cadastro de clientes terá gravada a string *‘fClientes’* na propriedade *name*.

Finalizando o form principal

Pra finalizar a tela principal vamos fazer apenas alguns retoques. Insira no evento *OnShow* uma chamada para *CriarFavoritos* e *CarregarFavoritos*. Em seguida codifique também os eventos *OnClick* de cada item de menu (*MainMenu*) referentes aos cadastros de clientes e fornecedores. Deve ser usada a função *AbrirForm* em cada item. (Listagem 5).

Agora basta codificar o botão *Favoritos* (*tbnFavoritos*) de forma que ele possa abrir a janela de manutenção (vista a seguir):

```
procedure TfPrincipal.
tbnFavoritosClick(Sender: TObject);
begin
  fFavoritos := TfFavoritos.Create(Self);
  try
    fFavoritos.ShowModal;
  finally
    FreeAndNil(fFavoritos);
  end;
end;
```

Manutenção de Favoritos

Pra finalizar nosso exemplo basta criarmos um novo formulário pra que o usuário possa excluir os *Favoritos* que não deseja mais. Crie uma nova janela e salve-a como *“frmFavoritos.pas”* e dê o nome de *“fFavoritos”*. Adicione um *DBGrid*, um *ClientDataSet* (*“cdsFavoritos”*) e um *DataSource* (Figura 4).

Nota: Faça as devidas ligações entre *DBGrid*, *DataSource* e *ClientDataSet* pra que possamos visualizar os dados.

Nessa tela vamos codificar apenas os eventos *OnShow* e *OnDestroy* da janela e *OnKeyDown* do *DBGrid*. No evento *OnShow* apenas carregamos o arquivo XML, usando o seguinte código:

```
cdsFavoritos.LoadFromFile(ExtractFilePath(
  Application.ExeName) + 'Favoritos.xml');
cdsFavoritos.Open;
```

E no evento *OnDestroy* fechamos o *ClientDataSet*:

```
cdsFavoritos.Close;
```

No *DBGrid* apenas detectamos se o usuário pressionou a tecla *Del* e excluimos o registro. Em seguida salvamos nova-

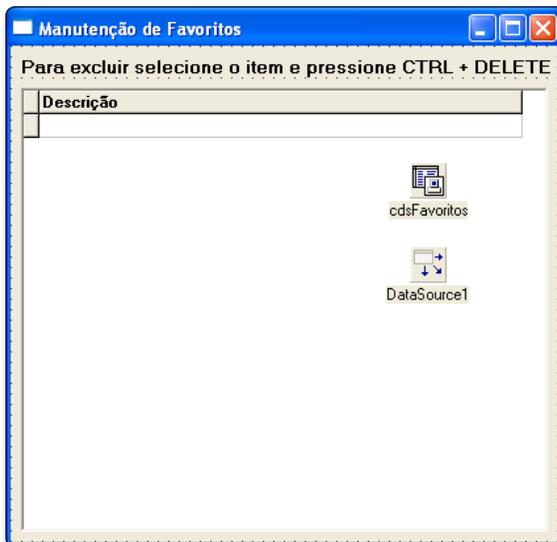


Figura 4. Manutenção de favoritos



Figura 5. Cadastro de clientes adicionado ao favoritos

mente o arquivo usando o *SaveToFile* do *ClientDataSet*, conforme a **Listagem 6**.

Nota: Não esqueça de adicionar no formulário principal todas as *units* que serão usadas por ele: *fClientes*, *fFornecedores* e *fFavoritos*. Em *Project>Options* deixe apenas o *fPrincipal* como criação automática (*Auto create*).

Pronto, agora basta compilar o exemplo e testar. Nas **Figuras 5 e 6** podemos ver o sistema funcionando.

Conclusão

Neste artigo vimos como desenvolver um menu Favoritos, permitindo que o usuário possa adicionar atalhos para as telas mais utilizadas por ele no sistema. Muitas vezes é necessário criar artifícios para que o sistema torne-se ágil e fácil de trabalhar. Com o exemplo criado, o usuário pode acessar rapidamente cada tela marcada como Favoritos. ●



Figura 6. Menu favoritos

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

* 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)

RDprint Setup

Configuração da Impressão

Impressora: HP DeskJet 870Cxi Propriedades...

Modelo: Gráfico - Compatível com Windows Visualizar

Intervalo de Páginas:

Todas

Página Atual

Páginas: 1-5,7

Imprimir: Todas as páginas do intervalo

Cópias:

Número de Cópias: 3

Agrupar

Ordem inversa

Digite a páginas e/ou intervalos separados por vírgula. Por exemplo: 1,3,5-12

Deltress
Informática

Fone/Fax (14) 3454-7880
www.deltress.com.br

Página: 2 de 19
87%
Impressora: HP DeskJet 870Cxi
Gráfico
* O RDprint 4.0 não imprime gráficos !



SOAP - Aprenda técnicas avançadas em um exemplo passo a passo – Parte 1

As aplicações distribuídas vieram para ficar, e o modelo *n-tier* é adotado pelas empresas cada vez mais. Aliado a isso, no cenário atual, nota-se a presença forte da internet nas soluções de TI como canal de comunicação. E quando se pensa em multicamadas e internet, é impossível não lembrar de uma tecnologia em franca expansão: os Web Services.

Neste artigo, que está dividido duas partes, aprenderemos algumas técnicas avançadas do protocolo SOAP e de sua implementação feita pela CodeGear, e veremos como empregá-las em nossas aplicações.

A aplicação

Como de costume, vamos utilizar um exemplo prático de aplicação para ilustrar os conceitos aprendidos. Faremos um disco virtual, um aplicativo onde o usuário poderá enviar arquivos para um servidor remoto, listá-los e recuperá-los

posteriormente.

Todos esses processos (envio, consulta e recebimento) poderão ser realizados de qualquer máquina com acesso ao servidor de aplicação.

O que você vai precisar

Será usado o Delphi 2006 Architect para o desenvolvimento da solução. Nada impede, porém, que você utilize alguma versão anterior que forneça os componentes do protocolo SOAP (em outras palavras, a paleta *WebServices*). Atente apenas para um ponto: muitos *bugs* da implementação foram corrigidos pela CodeGear e a versão 2006 do produto vem com muitas delas já instaladas.

Então, é altamente recomendável que você aplique essas atualizações no seu Delphi. Vamos ver como fazer isso no próximo tópico. É claro que você também pode usar a novíssima versão 2007, se a tiver instalada.

Também será necessário ter IIS (*Internet*



Michael Benford

(michael@devmedia.com.br)

é Acadêmico do curso de matemática da Universidade Federal Fluminense e desenvolvedor Delphi e .NET da QSi, Qualidade em Sistemas de Informação. Programa em Delphi há 5 anos, desenvolvendo aplicações cliente/servidor, multicamadas, Web, utilitários de uso geral, componentes, experts e aplicativos não-comerciais. É colunista do portal da DevMedia, articulista e membro da comissão editorial da revista ClubeDelphi. Palestrante da primeira edição do Developer's Webdays e da BorCon Brasil 2006.

Information Services) instalado, uma vez que o servidor de aplicação será desenvolvido sobre a tecnologia ISAPI. E como faremos acesso a banco de dados, será preciso ter um SGBD configurado. Neste artigo será usado o Firebird 2.0.

Nota: É esperado que o leitor já tenha alguma experiência com SOAP antes de ler este artigo. Vários artigos já foram publicados na revista ClubeDelphi, e, como referência, eu sugiro a edição 70, onde Guinther Pauli abordou o tema de forma brilhante.

Por último, você precisa instalar o *HTTPTRIOEx*, desenvolvido por mim, disponível para download junto com os fontes desse artigo. Ele é essencialmente igual ao original que acompanha o Delphi, de fato, ele herda diretamente desse último, exceto pelo novo evento *OnBeforeExecuteStream*, necessário para algumas técnicas abordadas neste artigo.

Para instalá-lo, apenas abra o seu pacote no Delphi e clique sobre a opção *Install*.

Atualizando o SOAP

Como mencionado anteriormente, é extremamente importante atualizar a implementação do SOAP com os últimos releases disponíveis. Você pode encontrá-los no site da CodeGear, ou através dos *news groups* da empresa (veja a seção *Links*).

Quando este artigo foi escrito, o último pacote disponibilizava as correções feitas no Delphi 2007 para as versões anteriores do produto. É provável que quando você esteja lendo este artigo, um novo *update* tenha sido liberado. Novamente, é importante visitar periodicamente o site da CodeGear para ter acesso a essas

informações.

Para aplicar as alterações, basta substituir as *units* antigas pelas novas (faça backup antes!). Caso alguma exista alguma mudança na interface dos componentes, talvez seja preciso reinstalar os pacotes do SOAP no IDE do Delphi.

Em todo caso, um documento de instalação sempre é distribuído junto com os arquivos.

Nota: É possível que o primeiro *update* do Delphi 2007 traga modificações na implementação do SOAP que descartem a necessidade de instalar o *HTTPTRIOEx*, citado no tópico anterior.

Preparando o banco de dados

Nossa aplicação vai precisar de um banco de dados para armazenar os usuários e os arquivos enviados por eles. Como já dito, vamos usar o Firebird 2.0 com SGBD. Entretanto, como o foco desse artigo é apenas o protocolo SOAP, não mostraremos como criar o banco e seus artefatos.

Basta você baixar os exemplos disponíveis e fazer uso do banco existente.

Iniciando os trabalhos

Com tudo preparado, já podemos começar a desenvolver nossa aplicação. Abra o Delphi, clique em *File>New>Other* e selecione o item *Delphi Projects>WebServices*. Marque o item *SOAP Server Application* e clique em OK. Quando for solicitado que você escolha o tipo de servidor a ser criado, selecione a caixa *ISAPI/NSAPI Dynamic Link Library*.

Clique em *Sim* quando aparecer a mensagem *Create Interface for SOAP module*. Na caixa seguinte, preencha o campo *Service name* como "DiscoVirtual" e em *Unit identifier* digite "uDiscoVirtual".

Mantenha as demais opções como estão e clique em OK.

O Delphi gerará todos os arquivos necessários para a criação de uma DLL ISAPI, e também uma interface para a classe remota. Salve todas as *units* em uma pasta chamada "server", com exceção do arquivo "uDiscoVirtualIntf", que deve ser armazenado em outro diretório, chamado "common". Salve o *WebModule* como nome "uWebModule", e o projeto como "DiscoVirtualServer".

Abra a *uDiscoVirtualIntf* e monte a interface como mostrado na **Listagem 1**.

Copie todos esses métodos, acesse o *uDiscoVirtualImpl* e cole-os na seção *public* da classe *TDiscoVirtual*. Pressione Ctrl+Shift+C para o Delphi criar o cabeçalho dos mesmos. À medida que avançarmos no artigo, vamos codificar cada um deles.

Pronto, já temos o esqueleto do nosso servidor de aplicação implementado. Vamos agora criar o projeto do programa cliente. Então, abra o *Project Manager*, clique de direita sobre o *Project Group* existente e selecione *Add New Project*. Na caixa *New Items*, vá para a pasta *Delphi Projects* e clique duas vezes sobre *VCL Forms Application*.

Dê ao formulário principal o nome "frmMain" e troque sua propriedade *Caption* para "Disco Virtual SOAP". Em seguida, crie uma pasta chamada "client" e salve todos os arquivos dentro dela. A unit do formulário deve se chamar "uFrmMain" e o projeto "DiscoVirtualClient".

Agora, clique no menu *Project* e selecione *Add to Project*. Localize o arquivo *uDiscoVirtualIntf* e adicione-o à solução. Por fim, salve o *Project Group* com o nome "DiscoVirtual".

Nesse momento a sua estrutura de arquivos deve estar parecida com a **Figura 1**.

Listagem 1. Interface do servidor de aplicação

```
IDiscoVirtual = interface(IInvokable)
['{972B0D6B-4B41-4136-859F-5B94DD697341}']

{ <-- 0 seu GUID será diferente! }
procedure Login(const UserName, Password: string); stdcall;
procedure Logout; stdcall;
function GetFileList: OleVariant; stdcall;
procedure UploadFile(const FileName: string;
  FileData: TByteDynArray); stdcall;
function DownloadFile(const FileName: string): TByteDynArray; stdcall;
procedure DeleteFile(const FileName: string); stdcall;

end;
```

ClubeDelphi PLUS

www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como usar Web Services com Delphi.

www.devmedia.com.br/articles/viewcomp.asp?comp=522

ClubeDelphi PLUS

www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como usar Web Services com Delphi.

www.devmedia.com.br/articles/viewcomp.asp?comp=908

Criando a interface gráfica do cliente

O primeiro módulo que vamos efetivamente construir é o cliente consumidor do Web Service. Então, selecione o projeto *DiscoVirtualClient* no *Project Manager* e abra o formulário principal. Adicione a ele um *PageControl* (paleta *Win32*) e modifique a propriedade *Align* para *alClient* e *TabPosition* para *tpBottom*.

Depois, crie duas páginas, definindo a propriedade *Caption* da primeira para "Principal", e a da segunda para "Trace". Insira um *StatusBar* no formulário, troque sua propriedade *Name* para "StatusBar" e crie dois painéis. O primeiro deve ter as seguintes propriedades configuradas: *Width*: 200 e *Text*: "Desconectado".

Prosseguindo, selecione a aba *Principal* do *PageControl* e adicione os controles de acordo com a **Figura 2**.

Dica: Se você estiver utilizando o Delphi 2006 ou 2007, pode fazer uso das *VCL Guidelines*, um recurso no IDE introduzido nessa versão que ajuda a alinhar os controles uns com os outros e com o formulário.

Nota: Observe na **Figura 2** os nomes que devem ser dados aos controles, com exceção dos *Labels*, que, como não são referenciados programaticamente, podem permanecer como sugerido pelo IDE (*Label1*, *Label2* etc.). Nesse caso, então, dar a eles nomes descritivos, fica a cargo do leitor.

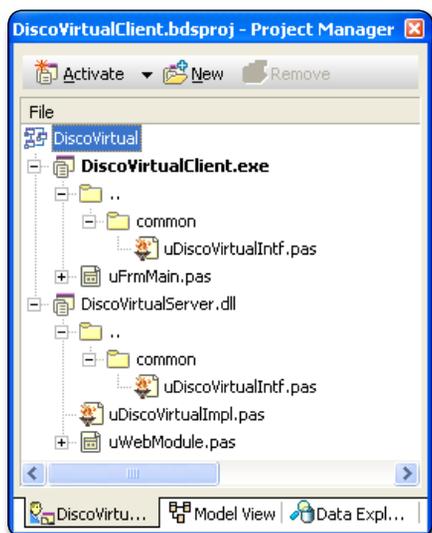


Figura 1. Arquivos da solução no Project Manager

Vá agora para a aba *Trace* e configure-a como mostrado na **Figura 3**.

Bom, já temos a interface gráfica do cliente pronta e não a modificaremos mais. De agora em diante só resta escrever muito, muito código!

Autenticação com SOAP

Na aplicação de disco virtual, um requisito básico é que cada usuário tenha sua própria pasta remota de arquivos. Isso significa que o *Usuário A* não enxerga os arquivos enviados pelo *Usuário B*, e

vice-versa. Para garantir esse comportamento, precisamos de um mecanismo de autenticação para impedir o acesso não autorizado aos arquivos remotos.

A maneira mais comum de conseguir isso é através de um nome de usuário e uma senha. Outro uso da autenticação em Web Service é impedir que métodos remotos sejam executados sem autorização.

Entretanto, informar as credenciais em cada chamada remota não é uma solução muito elegante. Para contornar esse problema, o protocolo SOAP ofe-

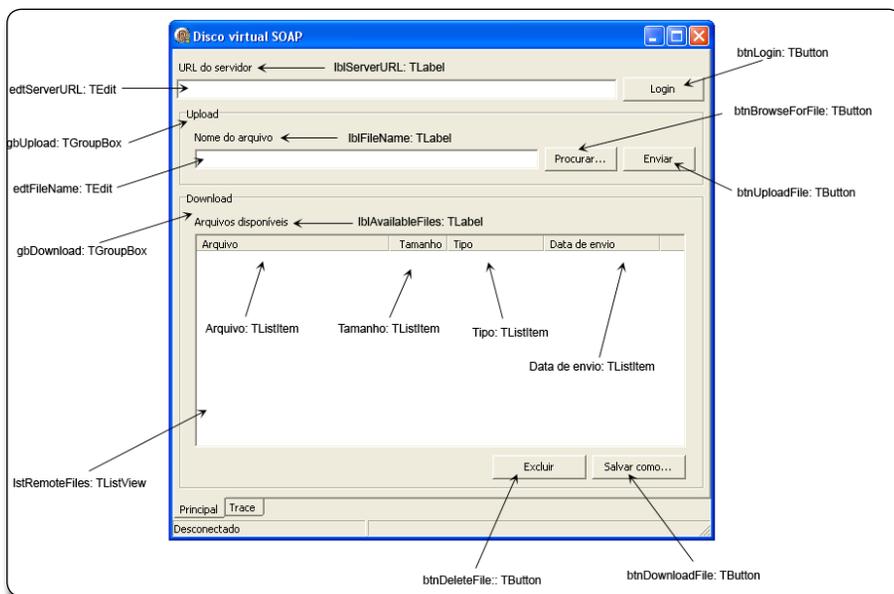


Figura 2. Aba Principal e seus controles

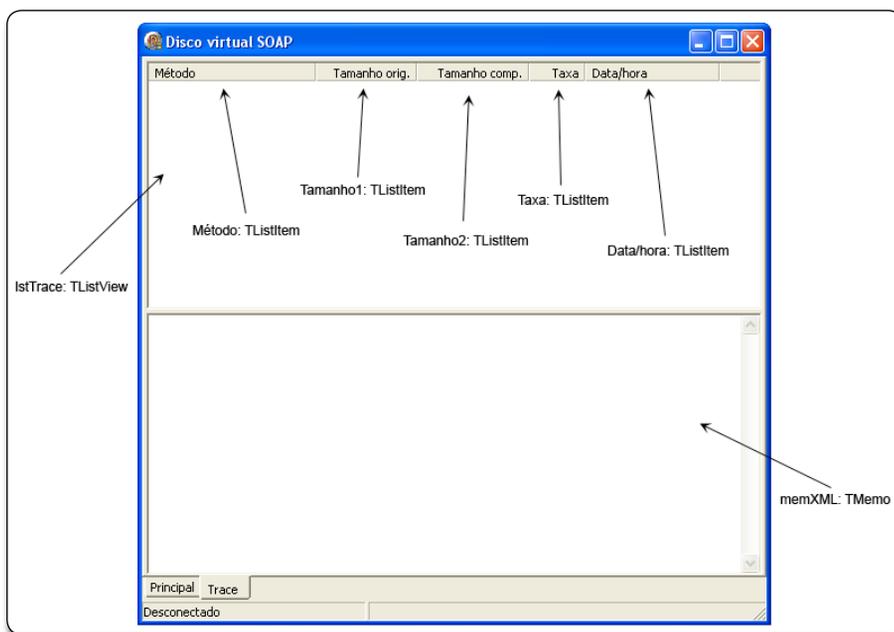


Figura 3. Aba Trace e seus controles

rece um recurso: os cabeçalhos SOAP, ou *SOAP Headers*. Nesse contexto, um cabeçalho é um container de informação que é enviado juntamente com o corpo da mensagem, a cada execução de um método remoto.

Pode-se armazenar nesse container qualquer informação, inclusive, é claro, credenciais de acesso. Vamos então, desenvolver a autenticação da nossa aplicação. Selecione o projeto do *DiscoVirtualServer* e adicione a ele um Data Module (*File>New>Delphi Files>Data Module*), nomeando-o para "DataAccess".

Em seguida, adicione um *SQLConnection (dbExpress)* e estabeleça uma conexão com o banco de dados preparado anteriormente.

Nota: Lembre-se de alterar a propriedade *VendorLib* para *fbclient.dll*.

Adicione ao Data Module os seguintes componentes: um *SQLDataSet*, um *DataSetProvider* e um *ClientDataSet*. Faça as ligações necessárias entre eles e ao final aponte a propriedade *Connection* do *SQLDataSet* para a conexão criada.

Nesse momento o seu Data Module deve se parecer com a **Figura 4**.

Dica: Confira na **Figura 4** os nomes que cada componente deve ter.

Configure então a propriedade *CommandText* do *SQLDataSet* para "select ID, NOME, SENHA from USUARIO where NOME = :NOME". Em seguida, dê um duplo clique sobre o componente, e no editor de campos, clique de direita e selecione *Add all fields*. Selecione o campo ID e altere sua propriedade *Required* para *False*. Agora, clique duas vezes também sobre o *ClientDataSet* e adicione todos os campos disponíveis.

Em seguida, abra o código-fonte do arquivo do projeto, e remova a linha de criação do Data Module. Isso é importante, pois vamos criá-lo sob demanda. Salve todas as alterações realizadas, e, quando o IDE lhe pedir um nome para a unit do Data Module, digite "uDataAccess".

Vá agora para *uDiscoVirtualImpl* e localize o método *Login*. Em seguida, codifique-o como mostrado na **Listagem 2**.

Salve todos os arquivos. Crie agora um

diretório virtual no IIS para hospedar nossa DLL ISAPI. Chame-o de "ClubeDelphi", e lembre-se de dar permissão de execução para aplicações ISAPI/CGI.

Em seguida, volte ao Delphi e clique no menu *Project>Options*, selecione o item *Directories/Conditionals* e no campo *Output directory* digite o caminho para a pasta mapeada no IIS anteriormente (no meu caso, *C:\Inetpub\ClubeDelphi*). Por fim, compile o servidor para gerar a DLL.

Vamos então fazer o cliente chamar o método *Login*. Selecione o projeto *DiscoVirtualClient* no *Project Manager*, clique em *File>New>Form*. Desenhe o formulário como mostrado na **Figura 5**.

Configure a propriedade *PasswordChar* do *edtPassword* para "*". Aproveite ainda para definir para *True* as propriedades

Default do *OK*, e *Cancel* do *Cancelar*. Por fim, troque o nome do formulário para "frmLogin", modifique sua propriedade *BorderStyle* para *bsDialog* e salve a unit como "uFrmLogin".

Antes de continuarmos, retire o formulário da lista de criação automática do projeto. Para isso, vá ao menu *Project>Options*, selecione a aba *Form* e arraste o *frmLogin* para a lista *Available forms*. Mais uma vez, salve todas as alterações.

Muito bem. Agora, selecione o formulário principal e adicione o *HTTPRIOEx (WebServices)*. Como dito anteriormente, ele faz o mesmo trabalho do *HTTPRIO* da VCL, com a diferença de possuir um evento a mais. Veremos mais à frente a função desse novo atributo. Crie agora uma propriedade chamada "AppServer",

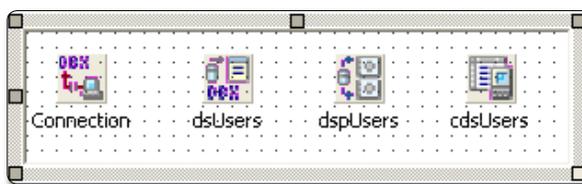


Figura 4. DataModule com os componentes de acesso a dados

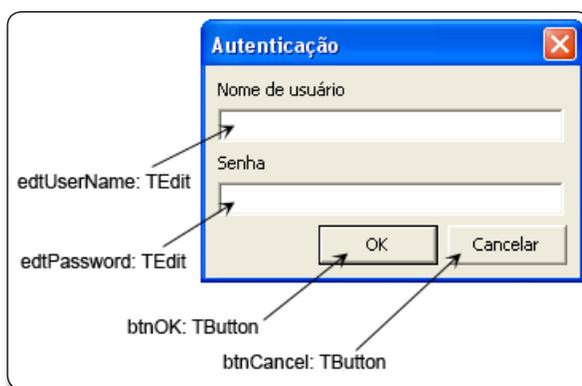
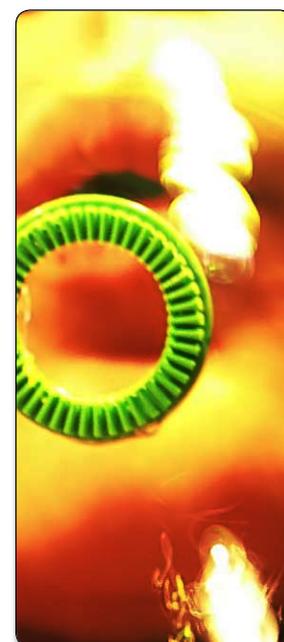


Figura 5. Caixa de autenticação



na seção *public* da classe do formulário:

```
property AppServer: IDiscoVirtual read  
GetAppServer;
```

Pressione Ctrl+Shift+C para o Delphi criar o *GetAppServer*. Implemente-o com o código a seguir:

```
function TfrmMain.GetAppServer: IDiscoVirtual;  
begin  
  Result := HTTPRIOEx1 as IDiscoVirtual;  
end;
```

Precisamos referenciar a *uDiscoVirtualIntf* para termos acesso à interface *IDiscoVirtual*. Inclua, então, à cláusula *uses* da seção *interface* do arquivo.

Nota: Se você usar a opção *Use unit* do menu *File* do Delphi, a unit será adicionada à cláusula *uses* da seção *implementation*, ficando indisponível para a propriedade *AppServer*.

A propriedade que criamos serve para facilitar o acesso ao servidor de aplicação. É através dela que executaremos todos os métodos remotos a partir do cliente. Como primeiro exemplo de uso, dê dois cliques sobre o *Login* e digite o código da **Listagem 3**.

Adicione *uFrmLogin* à cláusula *uses* da unit para a caixa de autenticação ficar disponível no código. Declare também o campo *FLoggedOut* na seção *private* da classe do formulário, que deve ser do tipo *Boolean*. No evento *OnCreate* do formulário principal, inicialize-o com *True*.

Certo de que o IIS esteja em execução, compile e rode a aplicação cliente. Digite no campo "URL do servidor" o endereço da DLL ISAPI criada:

```
http://localhost/clubedelphi/  
discovirtualserver.dll/SOAP/IDiscoVirtual
```

Efetue o login com o usuário "clubedelphi" e a senha "123". Se tudo correr bem, você verá escrito "Autenticado (clubedelphi)" na barra de status.

Nota: Esse usuário está disponível no banco de dados que você baixou.

Exceções remotas

O método *Login* pode levantar duas exceções: uma para o caso de o nome de usuário não existir no banco de dados, e outra para quando a senha informada ser

inválida. Quando uma delas ocorrer no servidor de aplicação, haverá uma conversão para o tipo *ERemotableException*, que é uma classe especial, capaz de ser serializada e enviada para o cliente.

Assim, a única coisa que diferenciará uma da outra será a mensagem de erro. Se quisermos tratar a exceção no lado do cliente, teremos que comparar as *strings*, o que não é seguro e elegante de fazer. Precisamos de um tipo específico de exceção para cada caso.

Selecione então o projeto do servidor e abra *uDiscoVirtualIntf*. Adicione na seção *type* as seguintes linhas:

```
EUsernameNotFound = class(ERemotableException);  
EInvalidPassword = class(ERemotableException);
```

Na seção *initialization* da unit, adicione também o código da **Listagem 4**.

RegisterXSClass registra uma classe como remota, e *RegisterException* associa a exceção com a interface do Web Service. Preci-

samos registrar as classes das exceções que criamos para que fiquem acessíveis do lado do cliente e possam ser tratadas. Ambos as rotinas são estáticas e não precisam de uma instância da classe para funcionarem.

Volte ao método *Login* e substitua as referências a *Exception* pelas classes que acabamos de criar. Em seguida, recompile o servidor e vá para o projeto do cliente. No bloco *try..except* existente no evento *OnClick* do *Login*, complemente o tratamento de erros com as linhas da **Listagem 5**.

Nota: *MessageBox* é um *wrapper* para a API do Windows de mesmo nome. Você encontrará o código dela nos fontes do artigo.

Enviando e recebendo SOAP Headers

Muito bem, já conseguimos executar o método *Login* remotamente. Continuando com o desenvolvimento da autenticação via SOAP, abra a *uDiscoVirtualIntf* e abaixo

Listagem 3. Realizando o login no servidor

```
procedure TfrmMain.btnLoginClick(Sender: TObject);  
begin  
  if FLoggedOut then  
  begin  
    HTTPRIOEx.URL := edtServerURL.Text;  
    with TfrmLogin.Create(nil) do  
      try  
        if ShowModal = mrOK then  
          try  
            AppServer.Login(edtUserName.Text, edtPassword.Text);  
            StatusBar.Panels[0].Text := Format('Autenticado (%s)', [edtUserName.Text]);  
            btnLogin.Caption := 'Logout';  
            edtServerURL.Enabled := False;  
            FLoggedOut := False;  
          except  
            raise;  
          end;  
        finally  
          Release;  
        end;  
      end;  
    end;  
  end;
```

Listagem 4. Seção Initialization

```
RemTypeRegistry.RegisterXSClass(EUsernameNotFound);  
RemTypeRegistry.RegisterXSClass(EInvalidPassword);  
InvRegistry.RegisterException(TypeInfo(IDiscoVirtual), EUsernameNotFound);  
InvRegistry.RegisterException(TypeInfo(IDiscoVirtual), EInvalidPassword);
```

Listagem 5. Alterando a autenticação do usuário

```
(...)  
except  
  on E: EUsernameNotFound do  
    MessageBox('O usuário informado não foi encontrado', MB_OK + MB_ICONERROR);  
  on E: EInvalidPassword do  
    MessageBox('A senha informada é inválida', MB_OK + MB_ICONERROR);  
end;  
(...)
```

Listagem 6. Classe do cabeçalho de autenticação

```
TAuthenticationHeader = class(TSOAPHeader)  
private  
  FSessionID: string;  
published  
  property SessionID: string read FSessionID write FSessionID;  
end;
```

da declaração da interface *IDiscoVirtual*, adicione o código da **Listagem 6**.

Essa é a classe que usaremos para autenticar o cliente no servidor, após a realização de um login bem-sucedido. Repare que ela deve herdar de *TSOAPHeader* e ter suas propriedades publicadas na seção *published* (isso é necessário porque a implementação SOAP do Delphi faz uso de RTTI para serializar os objetos remotos).

Entretanto, para que a classe possa trafegar de forma transparente entre os dois *endpoints* da aplicação (cliente e servidor), precisamos registrá-la, como fizemos com as exceções do tópico anterior. Para fazer isso, temos o método estático *RegisterHeaderClass*, da classe *InvRegistry*. Então, adicione a linha a seguir na seção *initialization* da *unit*:

```
InvRegistry.RegisterHeaderClass(
  TypeInfo(IDiscoVirtual),
  TAuthenticationHeader);
```

Resta-nos agora apenas preparar o cabeçalho para ser enviado nas mensagens SOAP. Vamos lembrar: o usuário executa o método remoto *Login*, que o autentica no banco de dados. Em caso positivo, o servidor cria um cabeçalho SOAP com um ID único e o envia de volta ao cliente. Depois, em cada chamada remota que o cliente realizar, ele envia esse ID, e o servidor pode verificar se o usuário já está autenticado, permitindo a execução do procedimento.

Para anexar um cabeçalho em uma mensagem SOAP, usamos o *Send*, da interface *ISOAPHeaders*. Para ver isso na prática, selecione o projeto do servidor e abra a *unit uDiscoVirtualImpl*. Acrescente ao final do código de *Login* o código a seguir:

```
(...)
Header := TAuthenticationHeader.Create;
Header.SessionID := '123';
(Self as ISOAPHeaders).Send(Header);
end;
```

Nota: Por enquanto vamos usar um código fixo como ID de sessão.

Descarregue a DLL do servidor no IIS, e recompile o projeto. Vá agora ao cliente e modifique o evento *OnClick* do *Login*, adicionando o código da **Listagem 7** exatamente antes da linha *FLoggedOut := False*:

Para retirar o cabeçalho da mensagem, utilizamos o *Get*, ao contrário de *Send*. Note que é importante verificar se há um *Header* de fato, através do *Assigned*. Por fim, armazenamos o ID da sessão obtido

no *FSessionID*, que você deve declarar na seção *private* da classe do formulário. Esse campo deve ser do tipo *string*, obviamente. Repare que também estamos exibindo o identificador recebido do servidor no segundo painel da barra de status.

Nesse ponto já conseguimos enviar e receber cabeçalhos SOAP. O que precisamos fazer agora é, a cada chamada de um método remoto no cliente, enviar junto, no *Header* da mensagem, o ID da sessão que armazenamos. E, no servidor, verificar antes de executar o código se esse ID é válido.

Por partes, já que estamos no cliente, cuidemos primeiro dele. Crie um método na classe do formulário chamado *PrepareHeader*. Implemente-o (Ctrl+Shift+C) com o código da **Listagem 8**.

Nota: O *Send* pode causar confusão num primeiro momento, porque ele não envia nada de fato, apenas adiciona um item

ao cabeçalho da mensagem (pode haver vários cabeçalhos em uma mensagem SOAP). O envio só é feito efetivamente quando um método remoto é executado.

Agora, adicione a linha destacada a seguir no evento *OnClick* do *Login*:

```
end;
AppServer.GetFilesList; { Adicione essa linha }
finally
...
end;
```

Selecione então o projeto do servidor, localize o *GetFileList* em *uDiscoVirtualImpl* e adicione a linha de código a seguir:

```
CheckHeader;
```

Como você pode perceber, precisamos também criar o *CheckHeader*. Adicione-o então à seção *private* da classe, com o código da **Listagem 9**.

Pressione Ctrl+Shift+C e digite o código da **Listagem 10**.

Essas linhas não devem ser novas para

Listagem 7. Passando o ID da sessão

```
(...)
(HTTPRIOEx as ISOAPHeaders).Get(TAuthenticationHeader,
  TSOAPHeader(Header));
if Assigned(Header) then
begin
  FSessionID := Header.SessionID;
  StatusBar.Panels[1].Text := FSessionID;
end;
FLoggedOut := False;
(...)
```

Listagem 8. Preparando o cabeçalho para ser enviado

```
procedure TfrmMain.PrepareHeader;
var
  Header: TAuthenticationHeader;
begin
  Header := TAuthenticationHeader.Create;
  Header.SessionID := FSessionID;
  (HTTPRIOEx as ISOAPHeaders).Send(Header);
end;
```

Listagem 9. Implementando o método CheckHeader

```
TDiscoVirtual = class(TInvokableClass, IDiscoVirtual)
private
  procedure CheckHeader; overload;
  procedure CheckHeader(var SessionID: string);
  overload;
(...);
end;
```

Listagem 10. Verificando a existência do cabeçalho na mensagem SOAP

```
procedure TDiscoVirtual.CheckHeader;
var
  SessionID: string;
begin
  CheckHeader(SessionID);
end;
procedure TDiscoVirtual.CheckHeader(var SessionID: string);
var
  Header: TAuthenticationHeader;
begin
  (Self as ISOAPHeaders).Get(TAuthenticationHeader, TSOAPHeader(Header));
  if not Assigned(Header) then
    raise ERemotableException.Create('Cabeçalho de identificação não encontrado');
  if Header.SessionID <> '123' then
    raise ERemotableException.Create('Identificar de sessão inválido');
  SessionID := Header.SessionID;
end;
```

nós: estamos recuperando o cabeçalho da mensagem SOAP, via *Get*, e verificando se o ID de sessão informado nele é válido. Temos duas versões do método: uma para simplesmente verificar o cabeçalho da mensagem e outra para fazer isso e também retornar o identificador de sessão encontrado. Isso se mostrará útil nos próximos tópicos deste artigo.

Para testar tudo que fizemos até agora, descarregue a DLL do servidor no IIS e recompile-o. Em seguida, alterne para o projeto do cliente e execute-o. Ao realizar o login, você deve receber uma men-

sagem de erro com o texto da primeira exceção levantada no *CheckHeader* da **Listagem 10**.

Isso aconteceu porque não enviamos nenhum cabeçalho para o servidor. Volte ao evento *OnClick* do *Login* e inclua antes de *AppServer.GetFilesList*, uma chamada para *PrepareHeader*. Rode novamente a aplicação e dessa vez o erro não deve mais ocorrer. Para testar a segunda exceção, troque o ID da sessão, ou no cliente ou no servidor, para alguma coisa diferente de 123.

Agora que temos implementados os métodos *PrepareHeader* no cliente e *Che-*

ckHeader no servidor, vamos recapitular o que devemos fazer para contemplar o mecanismo de autenticação:

1. Realizar o login com credenciais de usuário (cliente);
2. Autenticar o usuário e retornar um identificador de sessão (servidor);
3. Salvar o identificador gerado (cliente);
4. Enviar a cada chamada remota o identificador no cabeçalho SOAP da mensagem (cliente);
5. Verificar se o identificador recebido é o mesmo gerado na autenticação (servidor).
6. Como você pode perceber, o único método que pode ser invocado sem depender de um identificador de sessão é, por razões óbvias, o próprio método *Login*.

Antes de continuarmos com o assunto de autenticação, vamos ver algo interessante.

Rastreando as chamadas ao servidor

Durante o andamento deste artigo, vamos precisar ver as mensagens SOAP trocadas com o servidor para entender melhor como as coisas funcionam. Uma maneira de fazer isso é através de um *sniffer*, monitorando a porta 80. Outra, através da nossa própria aplicação. Advinha qual vamos escolher?

Você deve ter reparado quando criamos a interface gráfica da aplicação cliente que uma segunda aba foi criada, chamada *Trace*, e que dentro dela foram colocados dois controles: um *ListView* e um *Memo*. Eles serão usados para exibir os pacotes de requisição e resposta que enviaremos e receberemos do servidor. Vamos implementar esse comportamento agora.

O *HTTPRIO* possui, entre seus eventos, dois importantes: *OnBeforeExecute* e *OnAfterExecute*. O primeiro é disparado antes da mensagem de requisição ser enviada ao Web Service, e o segundo, quando a respectiva resposta retorna. Ambos os eventos trazem em seus parâmetros o conteúdo do pacote sendo enviado ou recebido.

Entretanto, o formato desses parâmetros não é compartilhado: *OnBeforeExecute* passa um *WideString*, enquanto *OnAfterExecute*, um *TStream*. Segundo a Borland, houve razões no passado para isso ser assim. Deixando de lado a inconsistência na interface dos controles,

Listagem 11. Logando os pacotes enviados e recebidos

```
procedure TfrmMain.LogMessages(Upload: Boolean;
  AMethodName: string; Stream: TStream;
  OriginalSize, CompressedSize: Integer);
var
  TempStream: TStringStream;
begin
  TempStream := TStringStream.Create('');
  try
    Stream.Position := 0;
    TempStream.CopyFrom(Stream, 0);

    with lstTrace.Items.Add do
      begin
        if Upload then
          ImageIndex := 0
        else
          ImageIndex := 1;
          Caption := AMethodName;
          SubItems.Add(IntToStr(OriginalSize) + ' B');
          SubItems.Add(IntToStr(CompressedSize) + ' B');
          SubItems.Add(IntToStr(((OriginalSize - CompressedSize) * 100) div OriginalSize) + '%');
          SubItems.Add(DateTimeToStr(Now));
          SubItems.Add(TempStream.DataString);
        end;
      finally
        TempStream.Free;
      end;
    end;
end;
```

Listagem 12. Chamando o método LogMessages

```
procedure TfrmMain.HTTPRIOEx1BeforeExecuteStream(const MethodName: string; SOAPRequest: TStream);
var
  TempStream: TMemoryStream;
  OriginalSize, CompressedSize: Integer;
begin
  TempStream := TMemoryStream.Create;
  try
    SOAPRequest.Position := 0;
    TempStream.CopyFrom(SOAPRequest, 0);

    OriginalSize := SOAPRequest.Size;
    CompressedSize := SOAPRequest.Size;
    LogMessages(True, MethodName, TempStream, OriginalSize, CompressedSize);
  finally
    TempStream.Free;
  end;
end;

procedure TfrmMain.HTTPRIOEx1AfterExecute(
  const MethodName: string; SOAPResponse: TStream);
var
  TempStream: TMemoryStream;
  OriginalSize, CompressedSize: Integer;
begin
  TempStream := TMemoryStream.Create;
  try
    OriginalSize := SOAPResponse.Size;
    CompressedSize := SOAPResponse.Size;
    SOAPResponse.Position := 0;
    TempStream.CopyFrom(SOAPResponse, 0);
    LogMessages(False, MethodName, TempStream, CompressedSize, OriginalSize);
  finally
    TempStream.Free;
  end;
end;
```

existe um problema com o argumento do primeiro evento.

Ser um *WideString* compromete recursos como criptografia e compressão dos pacotes, veremos isso em detalhes mais à frente.

Nota: É quase certo que no primeiro *hotfix/update* do Delphi 2007 essa inconsistência seja eliminada.

Para trabalhar apenas com *streams*, eu criei o *HTTPTRIOEx*, que você baixou e instalou no início do artigo. Como mencionado, ele publica o evento *OnBeforeExecuteStream*, que passa o conteúdo da mensagem como um *TStream*, e não mais como *WideString*.

Bom, vamos voltar ao aplicativo. Crie um método na seção *private* da classe do formulário, e chame-o de "LogMessages". Dê a ele a seguinte assinatura:

```
procedure LogMessages(Upload: Boolean;
  AMethodName: string; Stream: TStream;
  OriginalSize, CompressedSize: Integer);
```

Pressione Ctrl+Shift+C e implemente-o com o código da **Listagem 11**.

Estamos adicionando ao *ListView* o método remoto passado para a função. Usamos a classe *TStringStream* para converter os *bytes* do *stream* informado como parâmetro para uma *string*. Em seguida, fazemos uma conta para calcular o percentual de compactação dos pacotes. Não se preocupe com isso agora, veremos o assunto mais à frente.

Implemente agora o evento *OnClick* do *lstTrace* para o código a seguir:

```
if Assigned(lstTrace.Selected) then
  memXML.Lines.Text := lstTrace.Selected.
  SubItems[4]
else
  memXML.Lines.Clear;
```

Adicione agora um *ImageList* ao formulário e inclua nele duas imagens para representar o envio e recebimento dos pacotes, respectivamente. Eu utilizei uma seta para

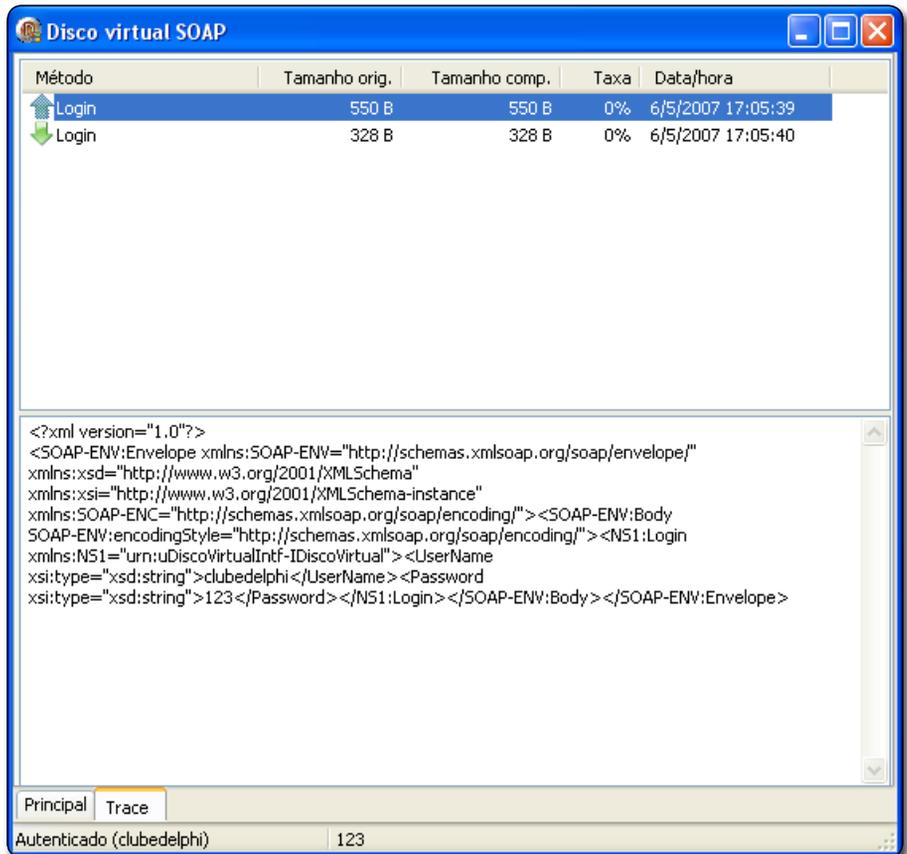


Figura 6. Visualizando os pacotes SOAP enviados e recebidos

cima e outra para baixo. Associe o *ImageList* à propriedade *SmallImages* do *lstTrace*.

Por fim, implemente os eventos *OnBeforeExecuteStream* e *OnAfterExecute* com o código da **Listagem 12**.

Compile e execute novamente a aplicação, efetue o login e em seguida vá para a aba *Trace* (**Figura 6**).

Repare no XML do método remoto sendo invocado, juntamente com seus parâmetros. De agora em diante, consulte sempre esses *logs* para entender melhor como funciona o transporte das informações enviadas e recebidas do *Web Service*.

Conclusão

Na primeira parte deste artigo, começamos a construir nosso disco virtual, um aplicativo onde o usuário poderá enviar arquivos para um servidor remoto, listá-los e recuperá-los posteriormente. Criamos a interface gráfica do cliente, vimos um pouco sobre autenticação com SOAP, exceções remotas, envio e recebimento de SOAP Headers e como rastrear as chamadas ao servidor. Não perca a próxima parte deste artigo, na próxima edição, onde veremos gerenciamento de sessão, envio de arquivos via SOAP, compactação e muito mais. Um abraço e até lá!

+ 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

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



Oráculo da ClubeDelphi



Guinther Pauli

(guinther@devmedia.com.br)

Atua no ramo de tecnologia e programação há mais de 17 anos, é autor de mais de 100 artigos publicados e 200 vídeo-aulas e do livro "Delphi – Programação para Banco de Dados e Web". É Bacharel em Sistemas de Informação, Microsoft Certified: MCP, MCAD, MCS.D.NET, Borland Certified: Delphi 6, 7, 2005, 2006, Web e Kylix. Editor Geral das Revistas .net Magazine, ClubeDelphi, WebMobile (.NET) e Mr.Bool. Palestrante em vários eventos pelo Brasil, como TechDay SP,RJ,POA,BH, Web Days e todas as edições da Borland Conference.

Na edição 62 da Revista ClubeDelphi, publiquei uma coluna chamada "Oráculo da ClubeDelphi". A idéia seria apresentar situações reais que são enfrentadas no dia a dia de um desenvolvedor Delphi ou que não fossem tão triviais, não aquelas que você encontra em uma simples busca no Google ou nos demos do Delphi, mas problemas mais "bruxos".

O que apresento aqui, em mais uma versão da coluna, são quatro dúvidas para alguns problemas relacionados ao desenvolvimento com o Delphi. Antes de iniciar, lembrem-se que jamais se pergunta para um desenvolvedor Delphi se "dá para fazer determinada coisa", mas "para quando ele pode fazer", porque não há limites para quem tem um Delphi aberto na sua frente.

Missão Impossível?

Em certa ocasião, trabalhei para o CPD de uma universidade em Santa Maria no

RS. Desenvolvíamos alguns sistemas internos, como o controle financeiro e vestibular. Porém, o sistema de biblioteca era terceirizado. Os alunos tinham um cartão pessoal (e intransferível) que utilizavam para retirar livros na biblioteca, através de uma leitura do cartão por código de barras.

Alguns alunos mais "espertinhos", e sem cadastro na instituição, utilizavam o cartão de outras pessoas para retirarem livros. A solução foi fazer toda a universidade tirar foto para ser examinada no momento da retirada do livro, já que o cartão não possuía foto.

Aí começaram os problemas. Foi requisitado ao CPD que construísse um sistema que exibisse a foto do usuário no terminal de empréstimo justamente no momento em que o cartão de código de barras fosse passado no sistema. A matrícula aparecia em um *TextBox / Edit* do sistema externo e precisaríamos interceptar essa matrícula, localizar a foto, e

exibir no canto da tela. Porém, tínhamos algumas pedras no caminho. Como iria fazer o sistema exibir a foto, se:

1. Não tínhamos os fontes do sistema de biblioteca;
2. Ele não foi feito em Delphi;
3. Não tínhamos como modificá-lo.

A API do Windows é repleta de surpresas. Apesar de usar tipos exóticos e não ser OO, você pode fazer um sistema inteiro, totalmente funcional, só usando API. Para algumas situações, precisamos recorrer a esse artifício para solucionar problemas mais complexos ou não suportados diretamente pela VCL do Delphi.

Bom, como não existem limites para um desenvolvedor Delphi, topamos o desafio, e achamos essa solução na API do Windows. Vou simular aqui uma solução bastante semelhante ao que fiz na época.

No exemplo deste artigo construí primeiramente um formulário chamado "Server", com um "TextBox" para ser informado um texto (nesse caso um nome). Esse form representa o form de empréstimo onde o usuário fornecia a matrícula. Como precisava ser em outra linguagem, construí a aplicação em Windows Forms com C#. A **Figura 1** apresenta o layout do form, que não tem código.

Em seguida, construí o formulário Delphi, que no caso representaria o form que

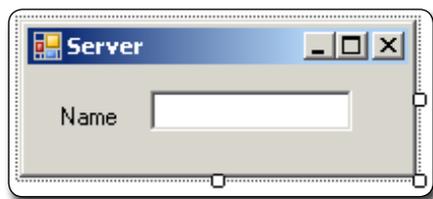


Figura 1. Formulário em C#

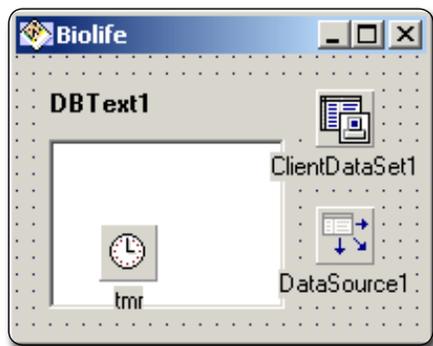


Figura 2. Formulário Delphi

exibiria a foto. O layout está na **Figura 2**. Um *ClientDataSet* está ligado ao arquivo *biolife.xml* dos demos do Delphi, pela propriedade *FileName*. Um *DBText* e um *DBImage* ligam, via *DataSource*, aos campos *Common_Name* e *Graphic* da tabela *biolife*. O *Timer* é explicado a seguir. O objetivo é: quando um nome ou iniciais forem digitados no *TextBox* na aplicação "Server", vamos dar um *locate* no respectivo registro no form Delphi e exibir os dados (como se fosse a foto da biblioteca, o procedimento é o mesmo).

A grande pergunta para achar a solução para o problema: "Como descobrir o que está digitado no *TextBox*?". Vamos começar pelo código do *Timer* (**Listagem 1**).

No Windows, toda janela tem uma *Handle*, que na verdade é um número. Exato, tipos no Windows são números! O primeiro passo foi localizar a janela *Server* pelo seu nome, usando a API *FindWindow*. O *Handle* da janela é armazenado na variável *Programa*. Pronto, já temos acesso

à janela servidora. A seguir, chamamos *GetWindowRect* para obter as dimensões do form *Server*.

EnumChildWindows é uma função interessante. Ela funciona assim, passamos para ela a *Handle* do form desejado e um ponteiro para uma função de *CallBack*. Ou seja, para cada controle filho que existir no form encontrado, a função de *CallBack* será chamada passando os devidos parâmetros. É o que o nome da função diz, "enumere as janelas (ou controles) filhos". Em uma dessas iterações, será localizado o nosso tão desejado *TextBox*. *ListaFilhos* é nossa função de *callback* e é exibida na **Listagem 2**.

Lembrando que como essa função é chamada para cada controle que está dentro do form *Server*, precisamos saber se o controle varrido é o *TextBox* desejado. A técnica usada consiste em testar a posição do controle. O primeiro passo é capturar a posição e tamanho do controle filho. Comparamos então esses valores com as

Listagem 1. Código do evento OnTimer

```
implementation
var
  MainForm: TMainForm;
  Texto: string;
  PRect: TRect;
  ....
procedure TMainForm.tmrTimer(Sender: TObject);
var
  Programa: THandle;
begin
  Programa := FindWindow(nil, pchar('Server'));
  if Programa <> 0 Then
  begin
    GetWindowRect(Programa, PRect);
    EnumChildWindows(Programa, @ListaFilhos, 0);
    if Texto <> '' then
      ClientDataSet1.Locate('Common_Name', Texto, [loCaseInsensitive, loPartialKey]);
  end;
end;
```

Listagem 2. Função de CallBack

```
function ListaFilhos(Win: THandle;
  lp: LPARAM): Boolean; stdcall;
var
  pc: pchar;
  loop: integer;
  str: string;
  Pos: TRect;
begin
  result:=true;
  getwindowrect(win, Pos);
  if PRect.left - pos.left = -65 then
  if PRect.top - pos.Top = -35 then
  begin
    (* Aloca buffer *)
    str:='';
    for loop := 1 to 300 do
      str:=str + ' ';
    (* Obtém um ponteiro para o buffer *)
    pc:=strnew(pchar(str));
    (* Manda uma mensagem para a janela *)
    sendmessage(win, WM_GETTEXT, 300, cardinal(pc));
    (* Texto é o desreferenciamento do ponteiro *)
    Texto := string(pc);
    strdispose(pc); (* Libera buffer *)
  end;
end;
```

dimensões do controle pai, e se as diferenças relativas forem as informadas (valor do deslocamento X e Y), então localizamos o *TextBox*, representado pela variável *Win*. Observe que o valor informado para o deslocamento precisa ser "hard-coded".

A seguir, alocamos um *buffer* de dados com *StrNew* e enviamos uma mensagem ao controle *TextBox* (com *sendMessage*) solicitando que ele coloque o seu texto em nossa variável que será enviada de volta à aplicação Delphi. O código da mensagem é *WM_GETTEXT*.

Em outras palavras, é como se o programa falasse: "Form Server, já descobri que

tu tens um *TextBox* que contém algo que eu preciso, estou te enviando um pacote para você colocar nele o conteúdo que necessito, depois mande-me de volta, please". Pronto! Conseguimos recuperar o texto do *TextBox*.

E isso finaliza nosso exemplo. O *Locate* no Delphi que já codificamos no *Timer* se encarrega de exibir o "peixinho" correto de acordo com suas iniciais. Veja o resultado na **Figura 3**.

Processamento paralelo "sem" TThread

Quase todo mundo sabe que para processar uma tarefa em segundo plano em

Delphi é preciso usar *Threads*. Criamos uma classe descendente de *TThread*, sobrescrevemos o método virtual abstrato *Execute* e codificamos nossa tarefa. Mas sinceramente, isso não é tão produtivo como colocar um componente no form e configurar um evento, certo? Muitas vezes, deixando um pouco o purismo da OO de lado, é bem mais fácil aderir ao EDP (*Event Driven Programming*) para ganhar velocidade e produtividade no desenvolvimento.

Quando comecei a estudar o .NET Framework 2.0, conheci um interessante controle chamado *BackgroundWorker*. Ele funciona de forma muito simples, você o coloca no form, no seu evento *DoWork* codifica o que gostaria de executar em segundo plano e depois simplesmente chama seu método *RunWorkerAsync* para executar o código de forma assíncrona (processamento paralelo). Bem mais RAD que a solução da VCL.

Mas o que não dá para fazer com Delphi? Talvez chover (ainda), então resolvi criar meu próprio *BackgroundWorker* para VCL Win32. O desenvolvimento ficou muito mais RAD e simples, e orientado a eventos, nada de criar classes descendentes de *TThread* (não que isso seja ruim, é apenas uma outra forma de ver as coisas). Vamos lá!

Primeiro criamos um *Package*, acessando o respectivo item no *Object Repository* (*File>New>Other>Package*). Salve o pacote como *pkBackgroundWorker.dpk*. No editor que aparece, clique em *Add* para adicionar um novo componente ao pacote, preenchendo as informações conforme a **Figura 4**. Observe que demos o apropriado nome de *TBackgroundWorker* à nova classe, que descende de *TComponent*. Clique em *OK*. Codifique a classe como mostrado na **Listagem 3**.

Se o leitor observar atentamente o código, vai perceber a "jogada". O que fizemos foi encapsular a classe *TThread* dentro de um componente, para que você literalmente nunca mais precise vê-la. Esse é um padrão de projeto bastante conhecido, chamado *Wrapper*.

A nossa classe *TBackgroundWorker* tem uma propriedade do tipo evento chamado *DoWork* (exatamente como no .NET), e seu respectivo campo privado *FDoWork*. Veja que adicionamos eventos a um componente como propriedades do tipo

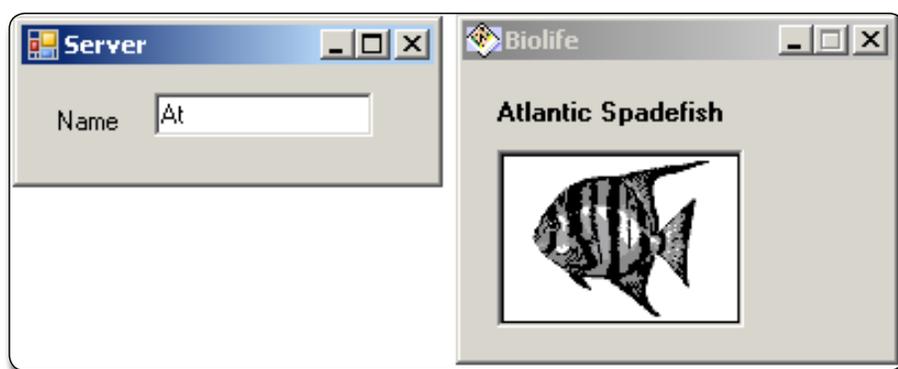


Figura 3. Pesquisando em um form Delphi a partir de um form C#

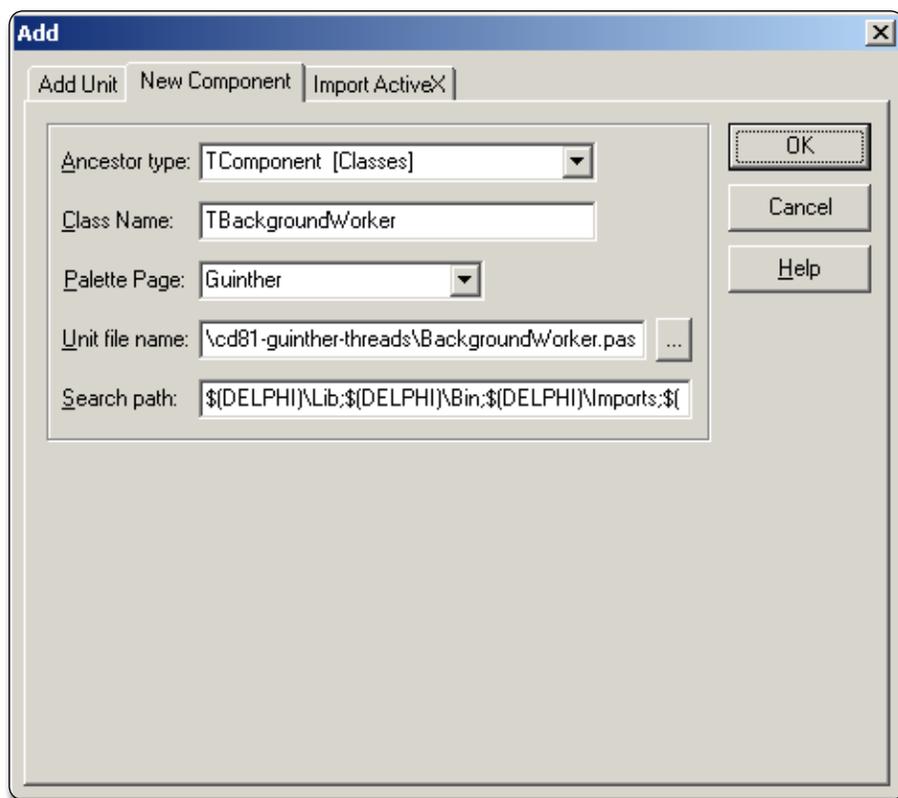


Figura 4. Criando o componente TBackgroundWorker

TNotifyEvent (para o tipo mais básico).

O método *RunWorkerAsync* nada mais faz do que verificar se o evento *DoWork* está apontando para alguma coisa e o chama (esse seria o código que o usuário do componente gostaria de executar em paralelo). Porém, ao invés de chamá-lo diretamente (síncrono), passa um ponteiro para o evento para ser chamado a partir de nossa *Thread* interna, que chama o evento de forma assíncrona. A *Thread* recebe esse evento na variável *FMethod* e o chama no método virtual *Execute*.

Voltando ao editor do pacote, clique no botão *Install* para instalar o componente no IDE. Crie uma nova aplicação e vamos fazer nosso primeiro exemplo com o novo *TBackgroundWorker*. Coloque no form um botão e um *Edit*. No botão digite:

```
BackgroundWorker1.RunWorkerAsync;
```

Coloque o controle *TBackgroundWorker* no form e no seu evento *DoWork* digite o código da **Listagem 4**. O código tem um loop infinito, que atribui um número randômico ao *Edit*. Em uma situação normal, isso travaria a aplicação, aqui ela continua funcionando normalmente. Você pode mover o form, redimensioná-lo etc. Ou seja, a aplicação não “trava”! E não foi preciso criar *TThread*, você fez tudo isso usando o melhor do RAD do Delphi.

Clicando em *View>Debug Windows>Threads* (*Ctrl+Alt+T*) podemos comprovar que realmente existem duas linhas de execução na aplicação (*Threads*). Uma é a principal, outra é a que está processando nosso algoritmo de randomização (**Figura 5**).

Para um exemplo mais real, que tal abrir aquele relatório ou query enorme e pesada em background? Como exemplo, criei uma tabela “*Foo*” no Firebird e gerei 100 mil registros aleatórios, e coloquei um *SimpleDataSet* do dbExpress associado a essa tabela. Colocamos um segundo *TBackgroundWorker* e no seu evento *DoWork* digitamos simplesmente:

```
procedure TForm1.BackgroundWorker2DoWork(
  Sender: TObject);
begin
  SimpleDataSet1.Open;
end;
```

Coloque um botão para chamar o método *RunWorkerAsync* do *BackgroundWorker2*. Rode a aplicação e faça o teste. Veja que a aplicação não trava mesmo abrindo uma query tão pesada. E me-

Listagem 3. Código do TBackgroundWorker

```
unit BackgroundWorker;

interface

uses
  SysUtils, Classes;

type
  TMyThread = class(TThread)
  private
    FMethod: TNotifyEvent;
  protected
    procedure Execute(); override;
  public
    constructor Create(CreateSuspended: boolean; AMethod: TNotifyEvent);
  end;

  TBackgroundWorker = class(TComponent)
  private
    FDoWork: TNotifyEvent;
  protected

  public
    procedure RunWorkerAsync;
  published
    property DoWork: TNotifyEvent read FDoWork write FDoWork;
  end;

  procedure Register;

implementation

procedure TBackgroundWorker.RunWorkerAsync;
begin
  if Assigned(DoWork) then
    TMyThread.Create(false, DoWork);
end;

procedure Register;
begin
  RegisterComponents('Guinther', [TBackgroundWorker]);
end;

{ TMyThread }

constructor TMyThread.Create(CreateSuspended: boolean;
  AMethod: TNotifyEvent);
begin
  inherited Create(CreateSuspended);
  FMethod := AMethod;
end;

procedure TMyThread.Execute;
begin
  FreeOnTerminate := True;
  FMethod(nil);
end;

end.
```

Listagem 4. Evento DoWork

```
procedure TForm1.BackgroundWorker1DoWork(Sender: TObject);
var
  I: integer;
begin
  randomize;
  while true do
    Edit1.Text := IntToStr(Random(10000));
end;
```

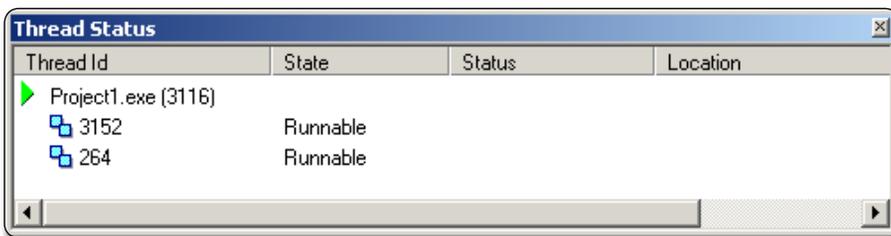


Figura 5. Threads rodando simultaneamente

lhor, aproveite para clicar também no botão *Random* e comprove que podemos ter múltiplas tarefas sendo executadas simultaneamente (**Figura 6**).

Nota: Para mais informações sobre Threads, consulte meu artigo na edição 20 ou o utilitário *ThreadManager* na minha página de projetos no CodeCentral da CodeGear, no link <http://cc.codegear.com/Author/222668>

Adicionando uma propriedade em um TForm no Object Inspector

O uso de Propriedades é um dos principais fundamentos da Programação Orientada a Objetos. E não pense que você precisa criar sistemas enormes e complexos para desfrutar das maravilhas da OO. Mesmo em situações muito simples, uma técnica OO bem empregada pode tornar seu código muito mais simples, elegante, modular e reaproveitável.

Lembro-me do meu primeiro exemplo com propriedades. Construir um form que tinha uma *StatusBar*. Então, a cada ação do usuário ou mudança de estado do sistema, eu mudava a descrição no controle para indicar o que estava acontecendo. Ex.:

```
StatusBar1.SimpleText := 'Inserindo registro';
...
StatusBar1.SimpleText := 'Dados gravados com sucesso';
...
```

Mas e se um dia eu resolvesse trocar a *StatusBar* por outro controle, por exemplo um *Panel*? Ou exibir as mensagens em caixas de mensagem? Precisaria obviamente trocar as referências em vários lugares.

Resolvi o problema encapsulando o acesso à *StatusBar* em um único ponto central do código, criando uma propriedade somente escrita, como mostra a **Listagem 5**.

Agora, para atribuir uma mensagem à *StatusBar*, simplesmente escrevemos:

```
Mensagem := 'Dados gravados com sucesso';
```

Se uma modificação de controle fosse feita, um único local do código precisaria ser modificado.

Mas veja que essa propriedade não aparece no *Object Inspector*. Esse ponto é um pouco mais complexo. Vamos então criar um segundo exemplo que vai adicionar uma propriedade a um formulário, em tem-

Listagem 5. Propriedade em um form

```
TForm1 = class(TForm)
  StatusBar1: TStatusBar;
private
  procedure SetMensagem(const Value: string);
public
  property Mensagem: string
    write SetMensagem;
end;

implementation
...
procedure TForm1.SetMensagem(const Value: string);
begin
  StatusBar1.SimpleText := Value;
end;
```

Listagem 6. Usando SetProperty

```
uses TypInfo;
...
procedure TForm1.Button1Click(Sender: TObject);
begin
  SetProperty(
    FindComponent(Edit1.Text), //componente
    Edit2.Text, //propriedade
    Edit3.Text); //novo valor
end;
```

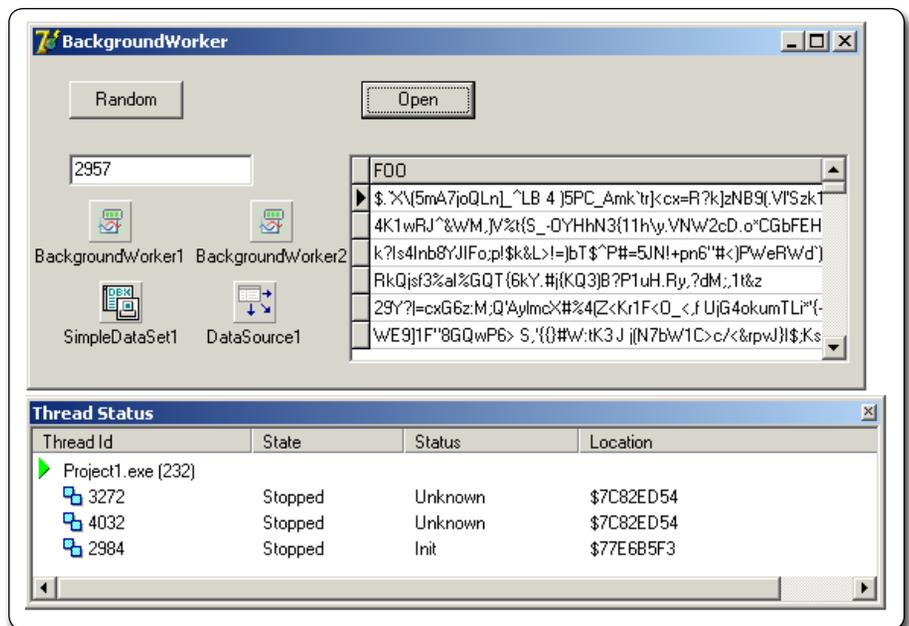


Figura 6. Múltiplas tarefas sendo executadas em paralelo

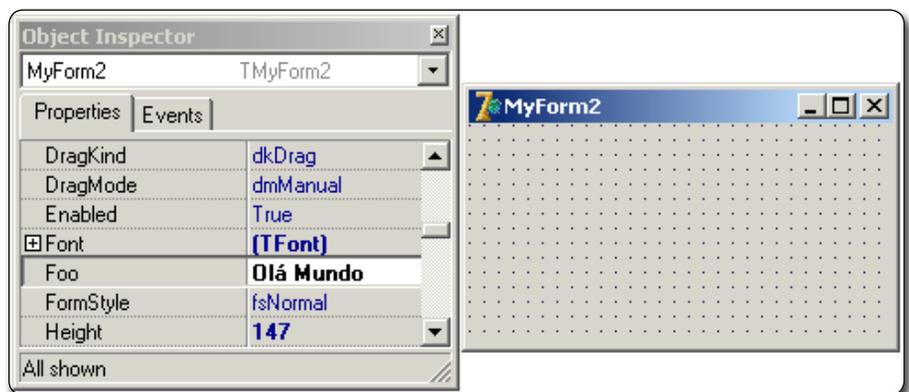


Figura 7. Adicionando uma propriedade a um formulário

po de design. Siga os seguintes passos.

1. Crie um pacote e adicione nele um formulário. Chame-o de "MyForm" e dê o nome da unit de "uMyForm.pas". Para ver como criar pacotes, veja o item 1 deste artigo. Chame o pacote de "pkMyForm.dpk";

2. Na unit do form, declare o seguinte:

```
// aqui é o final da declaração da classe
procedure Register;
...
uses
    DesignIntf, DesignEditors;
...
procedure Register;
begin
    RegisterCustomModule(TMyForm, TCustomModule);
end;
```

3. Declare uma propriedade de exemplo no form, e após pressione **Shift+Ctrl+C** (repare que ela tem que ser *published*):

```
published
    property Foo: string
        read FFoo write SetFoo;
end;
```

4. No editor do pacote, na seção *Requires*, adicione o arquivo *designide.dcp* que se encontra no diretório *Lib* do Delphi;

5. Instale o pacote no IDE clicando em *Install*;

6. Abra o form, clique de direita sobre

ele e acione a opção para adicioná-lo ao repositório (*Add to Repository*). Preencha as informações do caixa de diálogo e clique em *Ok*.

Para testar, inicie uma nova aplicação, abra o *Object Repository* e localize nosso form adicionado. Atenção, certifique-se que a opção *inherit* esteja marcada ao invés de *copy*. A propriedade deverá então aparecer no form em design time, totalmente funcional (**Figura 7**).

RTTI

A RTTI (*RunTime Type Information*) é um importante fundamento do Delphi. É amplamente utilizado pelo IDE, desde o momento que você abre o Delphi. Por exemplo, o Delphi usa RTTI para ler e salvar arquivos DFM e configurar propriedades de seus controles. Com RTTI, podemos obter informações de tipo em tempo de execução, e resolver uma série de problemas que muitas vezes não são supridos diretamente pela linguagem.

Você pode não saber, mas já usou RTTI com Delphi. Se já usou os operadores *is* e *as*, então fez uso da técnica. Mas os recursos desse poderoso artifício não param por aí. A unit *typinfo* traz uma série de rotinas que

podem ajudar e muito o desenvolvedor a resolver problemas do dia a dia, de maneira simples e direta (sugiro o leitor abrir os fontes da unit e examiná-la).

Neste último tópico, vou propor dois exemplos que fazem uso avançado de RTTI. O primeiro será um form que o usuário poderá personalizar controles, e o segundo será um "mini-IDE", onde o usuário poderá desenhar forms. Quem sabe você não usa a idéia para que seu próprio usuário desenhe a tela do sistema da maneira como ele deseja, e você só importa para o Delphi? Vamos lá!

Em uma nova aplicação, coloque no formulário principal três *Edits* e um *Button* dentro de um *Panel*, e logo abaixo deles, alguns controles a sua escolha, como *Labels*, *Edits*, *ComboBoxes*, *ListBoxes*, *Buttons* etc. Seu formulário deve estar semelhante à **Figura 8**. No *OnClick* do botão *Setar* digite o código da **Listagem 6**.

SetPropValue é uma importante rotina da unit *typinfo* que permite configurar o valor de qualquer propriedade de qualquer controle baseado no **nome** do componente, sem depender do seu tipo. É uma rotina muito flexível e bastante utilizada para resolver uma série de problemas em

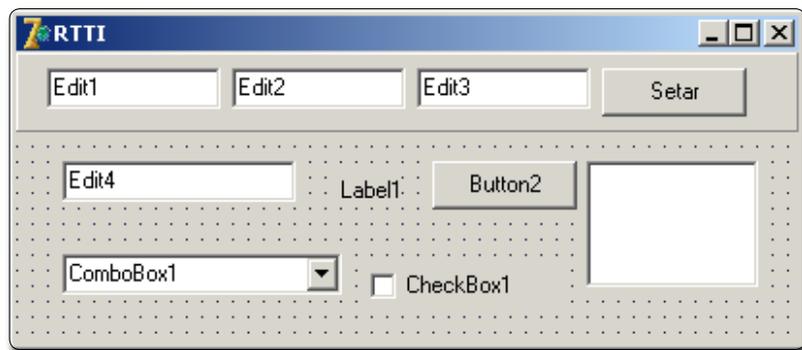


Figura 8. Formulário para teste de RTTI

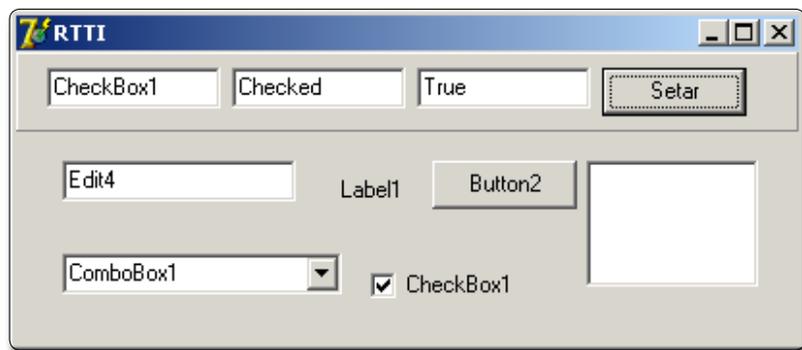


Figura 9. Usando RTTI

.net PLUS www.devmedia.com.br/msdn/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma videoaula de Guinther Pauli que mostra como usar TThreads.

www.devmedia.com.br/articles/viewcomp.asp?comp=1733

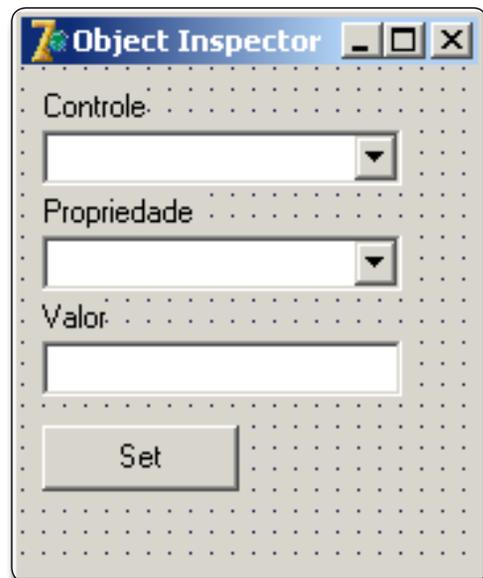
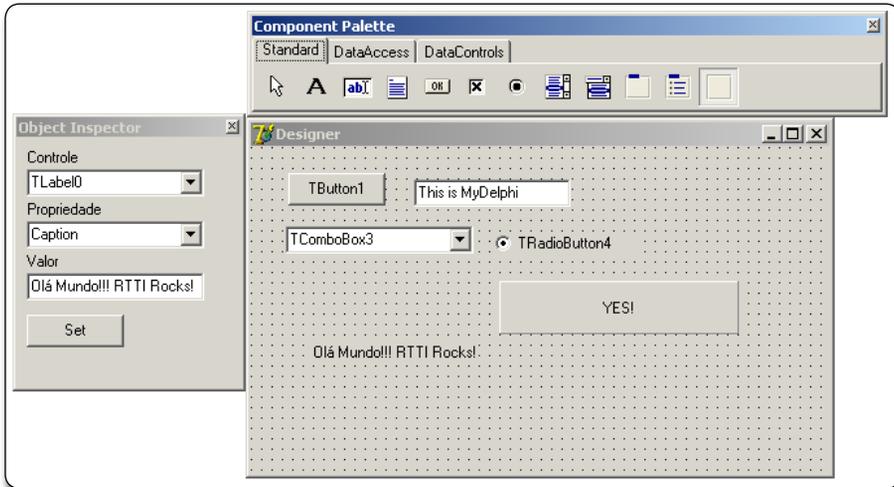


Figura 10. Object Inspector



Figura 11. Paleta de componentes



Listagem 7. Código da unit UFormOI.pas

```

...
type
  TFormOI = class(TForm)
  ...
  public
    procedure GetComponetProperties(AComponent: TComponent);
    procedure GetComponentList;
  ...
  implementation
  ...
  uses TypInfo, uFormDesigner;
  procedure TFormOI.GetComponentList;
  var
    i: integer;
  begin
    ComboBox1.Items.Clear;
    for i := 0 to FormDesigner.ControlCount - 1 do
      ComboBox1.Items.Add(FormDesigner.Controls[i].Name)
  end;
  procedure TFormOI.GetComponetProperties(
    AComponent: TComponent);
  var
    PropCount, i : Integer;
    PropList : PPropList;
    PropName : string;
  begin
    ComboBox2.Items.Clear;
    PropCount := GetPropList(AComponent.ClassInfo, PropList) - 1;
    for i := 0 to PropCount do
      begin
        PropName := PropList[i].Name;
        ComboBox2.Items.Add(PropName)
      end;
  end;
  procedure TFormOI.ComboBox1Change(Sender: TObject);
  var
    obj: TComponent;
  begin
    Obj := FormDesigner.FindComponent(ComboBox1.Text);
    GetComponetProperties(Obj);
  end;
  procedure TFormOI.Button1Click(Sender: TObject);
  begin
    SetPropValue(FormDesigner.FindComponent(
      ComboBox1.Text), ComboBox2.Text, Edit1.Text);
  end;
  procedure TFormOI.FormShow(Sender: TObject);
  begin
    GetComponentList;
  end;

```

estruturas de programas que precisam depender de vários tipos de dados.

Execute a aplicação e faça um teste, digite no primeiro *Edit* o nome de um componente que esteja no form, no segundo o nome de uma de suas propriedades e no terceiro um valor. Ao clicar no botão, o valor é alterado, tudo pelo nome e em tempo de execução! No exemplo da **Figura 9**, modifiquei o *Checked* do *CheckBox* para *True*. Faça outros testes, mude *Labels*, posições, tamanhos etc.

Seguindo essa mesma idéia, vamos um pouco mais longe, vamos criar um “Delphizinho”. Será uma aplicação semelhante a anterior, só que porém teremos uma paleta de componentes e um *mini-object inspector*. O usuário será capaz de mover os controles pela tela, e para finalizar, o formulário desenhado é persistido no disco.

Primeiro, crie uma aplicação com três formulários, com os nomes de “FormDesigner”, “FormOI” e “FrmPalette”. Certifique-se que todos estejam com *Visible = True*. O *FormOI* pode ser visto na **Figura 10**, nele temos dois *ComboBoxes*, três *Labels* e um *Button*. O *FormDesigner* ficará em branco por enquanto.

O *FrmPalette* (**Figura 11**) tem a aparência da paleta de componentes do Delphi. Para construí-la, usamos um *PageControl*, dentro de cada aba existe uma *ToolBar*, onde cada componente é representado por um botão (atenção, o nome do botão deve ser o nome da classe que representa, exemplo *TLabel*, *TButton*). Para este exemplo, coloque somente os controles básicos do Delphi.

No *FrmPalette*, declare a seguinte variável pública:

```

public
  SelectedClass : TClass;

```

Dando um clique em qualquer botão da *ToolBar*, digite o código a seguir. Depois, faça com que todos os demais botões apontem para o mesmo evento.

```

procedure TFrmPalette.PaletteClick(
  Sender: TObject);
begin
  SelectedClass := GetClass((Sender as
    TComponent).Name);
end;

```

Esse é todo o código do form. Basicamente o que estamos fazendo é usando RTTI para obter uma referência para uma

classe na qual o usuário clicou, pois quando ele clicar no *FormDesigner*, devemos criar um objeto desse tipo.

A **Listagem 7** mostra o código do *FormOI*, o nosso *Object Inspector*. O método *GetComponentList* varre o *FormDesigner* procurando por componentes, e vai preenchendo o primeiro *ComboBox* com os seus nomes. Quando o usuário seleciona um componente nessa lista, devemos gerar a lista de suas propriedades, novamente usando RTTI através da função *GetPropertyList*. Colocamos essas propriedades no segundo *ComboBox*. No botão, configuramos o valor da propriedade escolhida com base no valor digitado no *Edit*.

A **Listagem 8** mostra o código do *FormDesigner*. O método *NewControl*, como o nome sugere, criar um novo controle baseado no botão selecionado na paleta de componentes, usando RTTI. Isso acontece quando o usuário clica com o mouse sobre o form, veja o método *FormMouseDown*.

O *ControlMouseDown* é interessante, é nele que permitimos que o usuário arraste controles pela tela em runtime, semelhante ao designer do Delphi. Para isso, usamos a mensagem *WM_SysCommand* da API do Windows. O *Create* e o *Close* do form se encarregam de salvar e carregar o formulário, ou seja, persisti-lo. E por fim, no *initialization* da unit, precisamos registrar as classes que vamos trabalhar na aplicação. A **Figura 12** mostra nosso “Delphizinho” em execução.

Conclusão

Neste artigo vimos como a API pode resolver problemas que vão além dos limites da VCL. Nosso *BackgroundWorker* vai ajudá-lo agora a criar processamento paralelo em suas aplicações de forma extremamente RAD. Formulários agora têm propriedades no *Object Inspector*, tornando-os muito mais flexíveis. E finalmente, vimos que a RTTI é realmente uma técnica poderosa para explorar alguns recursos mais “obscuros” do Delphi.

Amigos leitores, não deixem de enviar suas dúvidas por e-mail para que possamos trazer sempre mais e mais dicas interessantes aqui nesta coluna. Espero que tenham aproveitado o conhecimento adquirido e lembrem-se, não existem limites para quem tem um Delphi. Um grande abraço e até a próxima! ●

Listagem 8. Código da unit *UFormDesigner.pas*

```

...
type
  TFormDesigner = class(TForm)
  procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
  procedure FormClose(Sender: TObject;
    var Action: TCloseAction);
  private
  procedure NewControl(ClassRef: TClass; x, y: integer);
  procedure ControlMouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
  public
  constructor create (AOwner: TComponent); override;
  end;
...
implementation

uses uFrmPalette, uFormOI;

{$R *.dfm}

type
  TGetProtected = class (TControl)
  end;

procedure TFormDesigner.NewControl(ClassRef: TClass; x, y: integer);
var
  NewObj : TControl;
begin
  if ClassRef<>nil then
  begin
    NewObj:=TControlClass(
      FrmPalette.SelectedClass).Create(self);
    NewObj.Parent:=self;
    NewObj.top:=y;
    NewObj.left:=x;
    TGetProtected(NewObj).OnMouseDown:=ControlMouseDown;
    NewObj.Name:=NewObj.ClassName+IntToStr(NewObj.ComponentIndex);
    FrmPalette.SelectedClass:=nil;
    formOI.GetComponentList;
  end;
end;

procedure TFormDesigner.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  NewControl(FrmPalette.SelectedClass,x,y);
end;

procedure TFormDesigner.ControlMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  ReleaseCapture;
  TControl(Sender).Perform(WM_SysCommand, $F012, 0);
end;

constructor TFormDesigner.create(AOwner: TComponent);
var
  i: integer;
begin
  if FileExists('Designer.dfm') then
  begin
    inherited CreateNew(AOwner) ;
    ReadComponentResFile('Designer.dfm', self) ;
    for i := 0 to controlCount - 1 do
      TGetProtected(Controls[i]).OnMouseDown:=ControlMouseDown;
    end
  else
    inherited Create(AOwner);
  end;
end;

procedure TFormDesigner.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  WriteComponentResFile('Designer.dfm', self) ;
end;

initialization
  RegisterClass(TButton);
  RegisterClass(TEdit);
  RegisterClass(TLabel);
  RegisterClass(TMemo);
  RegisterClass(TCheckBox);
  RegisterClass(TRadioButton);
  RegisterClass(TListBox);
  RegisterClass(TComboBox);
  RegisterClass(TGroupBox);
  RegisterClass(TRadioGroup);
  RegisterClass(TPanel);
  RegisterClass(TDBGrid);
  ...
//adicione no uses as units respectivas a esses tipos

```



O poder RAD do Delphi agora também para o PHP

Salve Delphianos! Este artigo, em primeiro lugar, vai novamente comprovar nossa força. Digo isso, pois teremos a partir de agora, mais uma plataforma de desenvolvimento: o Delphi para PHP.

Confesso que ultimamente tinha vários projetos onde não utilizava nenhuma versão do Delphi. Eram projetos com orçamentos muito baixos que impediam investimentos em infra-estrutura de hardware e licenças para aplicações Web, de certa forma simples.

Utilizava várias ferramentas para criação de sites. Algumas até facilitavam na geração de páginas de manutenção em banco de dados, mas qualquer coisa a mais ocasionava horas de pesquisa na linguagem PHP.

Agora os meus e os seus problemas acabaram: Delphi for PHP! Com o Delphi for PHP novamente comprovaremos nossa força, pois, de forma rápida e simples poderemos criar aplicações em

PHP, levando em conta toda a nossa experiência em Delphi, adquirida em anos de trabalho.

Para quem nunca trabalhou com Delphi e desenvolve em PHP, certamente gostará dessa novidade que simplifica em muito a realização de tarefas cotidianas. Imagine arrastar o nome de uma tabela para um formulário e tudo ficar pronto, como em um passe de mágica, sem escrever uma linha de código sequer.

Ainda, imagine se essa tabela estiver representada por uma grade (*Grid*), já “zebrada” (com cores diferentes), com ordenação dos registros através de um simples clique no título da coluna e com a opção de ocultar colunas. Está achando demais? Bem, para finalizar essa grade ainda permite a edição dos registros e seu update no banco, utilizando Ajax. Tudo isso sem escrever uma linha de código!

Os desenvolvedores mais antigos em PHP poderiam dizer que se trata de um gerador de código e que o mesmo deve



Fabrício Desbessel

(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.

gerar um código enorme. Bem, é aí que eles se enganam, pois o Delphi for PHP utiliza todo o poder de Orientação a Objetos (OO) disponível no PHP 5, o que otimiza em muito o desenvolvimento e as aplicações resultantes. A grande dificuldade está em trabalhar totalmente orientado a objetos, o que não era necessário nas outras versões do PHP.

O Delphi for PHP veio para revolucionar o mundo PHP e, com certeza, acelerar o desenvolvimento de aplicações, tornando-se o primeiro IDE totalmente RAD (*Rapid Application Development*) para PHP, com design de formulários, editor de código, debug da aplicação, *wizard* para publicação e com toda a VCL traduzida para PHP (provavelmente o ponto mais notável).

Instalação

A instalação do Delphi for PHP é bem simples e sem mistérios. Você pode baixar a versão *Trial* no site da CodeGear (www.codegear.com/downloads/free/delphi.php). Para o *download* é necessário o cadastro no site que depois será utilizado para enviar a chave de ativação. Depois de concluído o

download execute a instalação, bastando passar por todas as telas (*Next*). Antes de entrar no Delphi for PHP abra seu e-mail e verifique o e-mail da CodeGear com a chave de ativação do *Trial*. Salve o arquivo em anexo no seu diretório de configurações, normalmente em *C:\Documents and Settings\SeuNome*.

Após isso já será possível abrir o Delphi for PHP. Caso solicite serial e chave revise o local onde foi salvo o arquivo da chave, recebido por e-mail. A instalação é completa, instalando tudo que é necessário para criar, compilar e debugar as aplicações.

Como servidor Web é instalado o Apache 2, que funcionará na porta 3569, não conflitando com o IIS, caso o tenha instalado. Junto com o Apache é instalada a versão 5.1.3 do PHP. Todos os softwares e componentes estão na mesma pasta do Delphi for PHP.

Conhecendo o Delphi for PHP

Na **Figura 1** temos o IDE do Delphi for PHP. A princípio parece o IDE do Delphi 2006. Trata-se de um IDE específico, com muita coisa que existe no Delphi 7.

Para começar temos o *Code Explorer* (1) que facilita a navegação no código da aplicação. Abaixo temos o *Object Inspector* (2) com as propriedades e eventos dos componentes. O interessante é que no Delphi for PHP o *Object Inspector* apresenta mais uma aba, a *JavaScript* para manipulação de eventos na aplicação cliente.

Na direita (3) temos a paleta de componentes (*Tool Palette*), com os componentes da VCL, já conhecidos das outras versões do Delphi. Na direita e acima (4) temos o *Project Manager* e o *Data Explorer* onde faremos nossas conexões com os bancos de dados. No topo (5) temos a barra de botões padrões para abrir projetos, salvar e compilar, bem como os menus. O IDE é leve e bem intuitivo. Você vai gostar.

VCL for PHP

O segredo do Delphi for PHP está na tradução da VCL para PHP. Essa tradução foi realizada pela empresa Qadram (www.qadram.com) que é uma parceira da Borland/CodeGear. Trata-se de um framework para o desenvolvimento de aplicações Web e é Open Source com a licença LGPL.

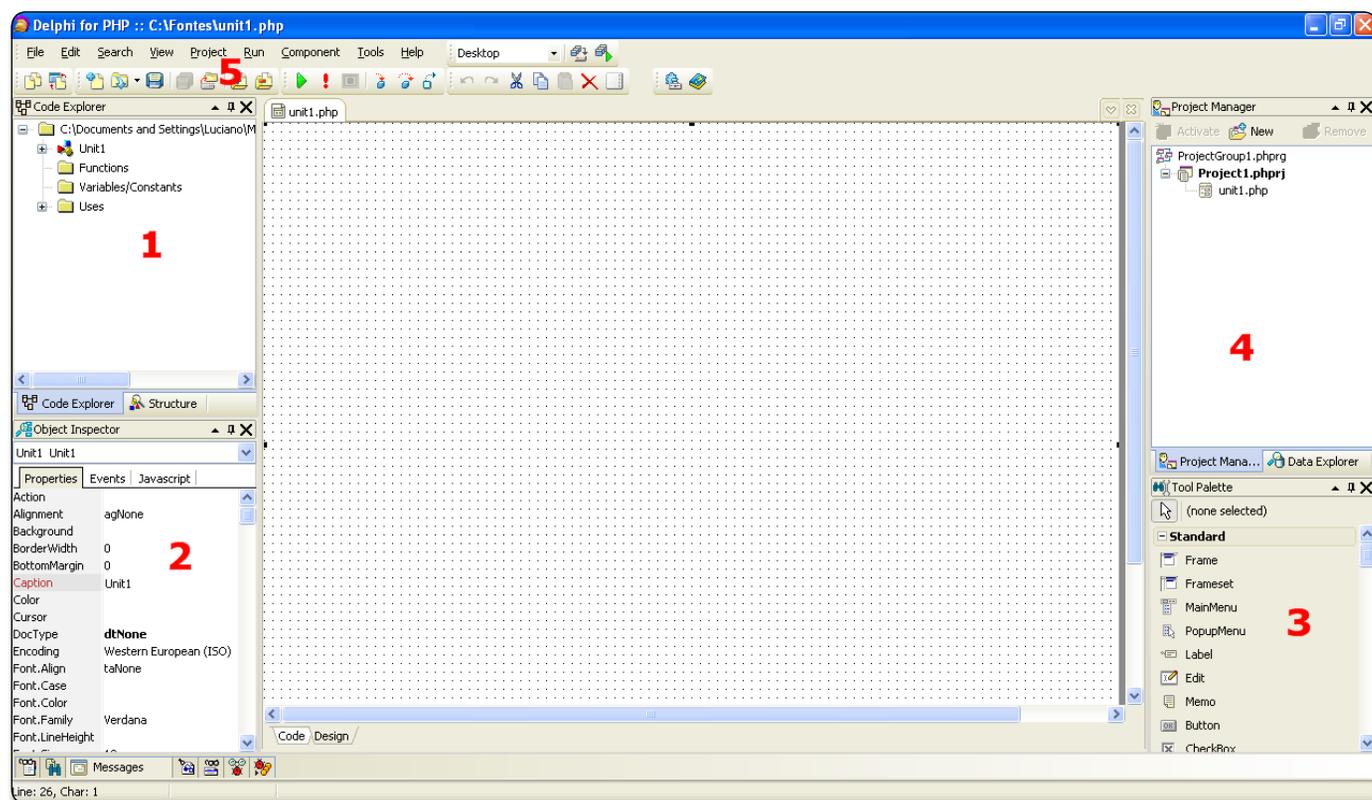


Figura 1. IDE do Delphi for PHP

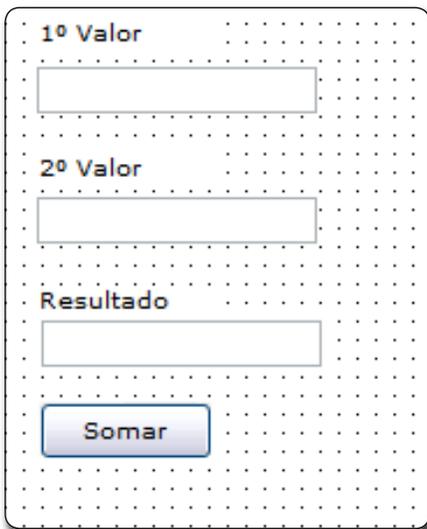


Figura 2. Layout da primeira aplicação

Por se tratar de software *Open Source* é essencial que se verifique periodicamente o surgimento de novas versões e correções de *bugs*, é claro. Já ouvi alguns comentários sobre *bugs*, visitando o site foi fácil achar as correções e verificar que existia uma outra versão estável.

Então, lembre-se disso: trata-se de um *framework Open Source* que possui atualizações e que, para não ter problemas, é bom utilizar sempre a última versão estável.

Primeira aplicação

Agora vamos ao que interessa, vamos criar nossa primeira aplicação simples com o Delphi for PHP, uma aplicação para somar dois valores. Inicie uma nova aplicação através do menu *File>New>Application*. Adicione um *Label* e altere a propriedade *Caption* para "1º Valor".

Logo abaixo, adicione um *Edit*. Repita esses passos para colocar mais dois *Labels* e dois *Edits*. No *Caption* dos *Labels* coloque "2º Valor" e "Resultado", respectivamente. Abaixo de tudo acrescente um *Button* e mude o *Caption* para "Somar". A aparência deve ficar igual à Figura 2.

Nota: Observe que o desenvolvimento com a VCL for PHP é praticamente idêntico ao modelo de desenvolvimento com a VCL do Delphi, ou seja, totalmente visual e drag & drop.

Clique duas vezes sobre o *Button* e adicione o seguinte código:

```
$this->Edit3->Text =
$this->Edit1->Text + $this->Edit2->Text;
```

A primeira coisa a ser explicada no código é o sinal "*->*": trata-se do nosso conhecido operador *ponto* ("*.*"), utilizado para acessar componentes, propriedades e métodos. No PHP Orientado a Objetos utiliza-se um traço seguido do sinal de maior. Outra coisa a ser explicada é o *\$this* que representa a própria instância. Chamar *\$this* é a mesma coisa que dizer: aqui->Edit2->Text. Imagine algo semelhante ao *self* da Delphi Language. No PHP o sinal de atribuição é simplesmente o símbolo do igual (=).

Na primeira vez que você for compilar será exigido o salvamento dos arquivos. Nesse momento você deve cuidar alguns detalhes: salve os arquivos na mesma unidade que o Delphi for PHP está instalado (normalmente no C:\). Se você não fizer isso a aplicação apresentará erros de *JavaScript* durante a execução, podendo não funcionar corretamente. Também leve em conta não utilizar acentuações ou espaços nas pastas onde

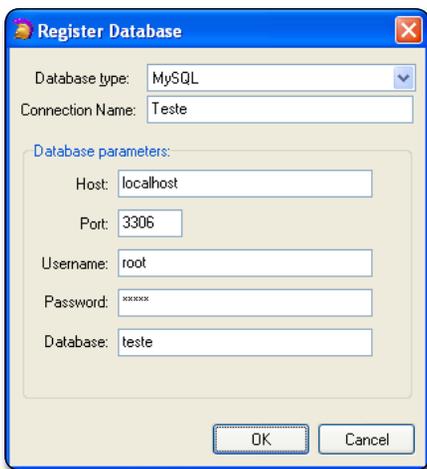


Figura 3. Tela de registro do banco de dados

ClubeDelphi PLUS 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 instalar o MySQL e o MySQLFront para criar aplicações acessando banco no Delphi for PHP.

www.devmedia.com.br/articles/viewcomp.asp?comp=4985

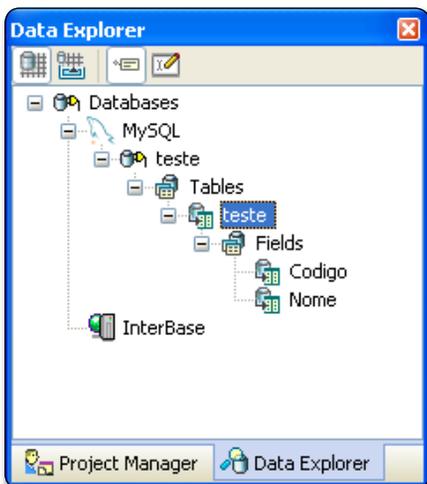


Figura 4. Data Explorer exibindo os campos da tabela

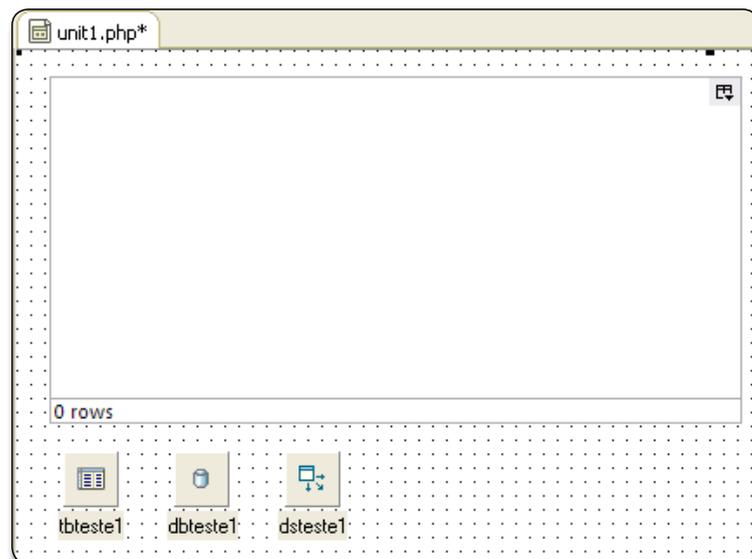


Figura 5. Tela em tempo design

ficará a aplicação e nos arquivos. Depois de salvo, basta compilar a aplicação e realizar o teste.

Pronto! Está feita nossa primeira aplicação PHP! Simples, fácil e RAD como o Delphi, não?

Nota: Se você não for debugar a aplicação é recomendável que utilize o botão *Run Without Debugging* (representado por um ponto de exclamação em vermelho) que compila e executa a aplicação de forma mais rápida.

Conexão com Banco de Dados

O Delphi for PHP aceita conexões com os bancos de dados *Interbase* da *CodeGear* e o *Mysql* que é *Open Source*. Agora vamos criar uma aplicação que conecte ao *Mysql*.

Crie um banco de dados teste que contenha uma tabela com um campo "Codigo" que é auto-incremento e um campo "Nome" do tipo *Varchar* (50). Crie uma nova aplicação através do menu *File>New>Application*.

Acesse o *Data Explorer* e clique com o botão direito sobre o ícone do *Mysql*. No menu de contexto escolha a opção *Register Database*. Preencha o campo *Connection Name* colocando um nome para sua conexão.

Informe no *Host* o nome da máquina que é o servidor *Mysql*, sendo que, se for local, informe *localhost*. No *Username* informe o nome do usuário com acesso

e sua senha. No campo *Database* informe o nome do banco. Veja na **Figura 3** a tela de registro do banco de dados. Após a conexão com o banco torna-se possível a navegação nas tabelas bem como nos campos de cada uma (**Figura 4**).

Agora clique no nome da tabela e arraste para o formulário da aplicação. Com isso será criado um *DBGrid*, ligado ao *DataSet* que estará ligado ao *Table* que por sua vez

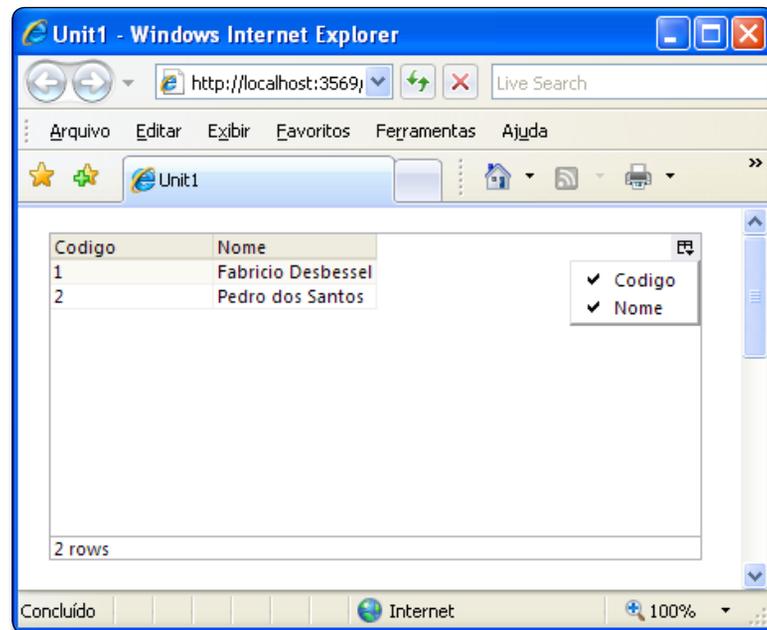


Figura 6. Aplicação de banco de dados em execução

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Fabricio Desbessel que mostra como trabalhar com parâmetros em consultas no Delphi for PHP.

www.devmedia.com.br/articles/viewcomp.asp?comp=5214

Web Mobile

WebMobile Edições Anteriores

Não fique na dúvida. Complete sua coleção!

Se você perdeu alguma edição da WebMobile, esta é a sua oportunidade de adquirir os números atrasados com o menor preço.

São três planos para você escolher, um deles é a medida certa pra você.



www.devmedia.com.br/anteriores

está ligado ao *Database* que faz a conexão com o banco (Figura 5).

Agora basta salvar sua aplicação e compilar para testar. Para melhorar nosso teste, insira alguns registros na tabela pelo gerenciador *MysqlFront*. Com aplicação rodando você pode fazer tudo aquilo que comentei na introdução do artigo: clicar no título das colunas para mudar a forma de ordenação, clicar no botão que aparece no canto superior direito da *Grid* para selecionar ou ocultar algumas colunas e, por último clicar na grade a alterar um registro, bastando sair do registro alterado para que ele seja atualizado no banco de dados (sem uma linha de código). Na Figura 6 a aplicação rodando.

O Delphi for PHP utiliza a mesma nomenclatura existente em aplicações Delphi 7 ou anteriores com BDE, para os nomes de componentes de conexão.

Delphi for PHP Update 1

Já está disponível o Delphi for PHP Update 1 que corrige vários bugs, inclusive os comentados nesse artigo. Então aproveite e faça o download através do link: <http://dn.codegear.com/article/36406>

Assim ainda temos o *Query*, para fazer uma consulta e *StoredProc* para executar uma *Stored Procedure* no banco. Todos localizam-se na paleta *Data Access*.

Publicando a aplicação

Com a aplicação pronta, precisamos fazer o *Deployment* da mesma. O Delphi for PHP possui um *wizard* para auxiliar nesse processo. Você pode acessá-lo através do botão *Deployment Wizard* ou através do menu *Tools* (Figura 7).

A idéia do *wizard* é detectar todas as classes da VCL for PHP utilizadas na aplicação para copiá-las junto com os arquivos. Clique em *Next* na primeira

tela e, na segunda defina um diretório de destino para o *deploy*. Clique novamente em *Next* e finalize. Verifique o diretório e veja que foram copiados todos os arquivos da aplicação e uma pasta *vcl* com todas as classes.

Conclusão

O Delphi for PHP é uma ótima ferramenta e facilita muito o desenvolvimento de aplicações PHP. Precisamos dedicar um tempo para estudo dessa ferramenta. Vale a pena investir. Continue acompanhando na ClubeDelphi mais artigos sobre o Delphi for PHP e no site várias vídeo-aulas. ●

Problemas na publicação

Para não ter problemas ao publicar a aplicação é preciso ter em mente que o servidor que hospedará a aplicação deve estar rodando o PHP 5 em uma versão igual ou superior a 5.1.3.

A maioria dos provedores no Brasil já está rodando o PHP 5, mas com o PHP 4 em conjunto. Portanto, verifique com seu provedor se não é necessário alterar o nome dos arquivos de sua aplicação para que fiquem com a extensão *php5*. Fiz alguns testes e foi necessário fazer isso. Tive que renomear todos os arquivos, mudando sua extensão. Ainda nos testes de publicação encontrei um bug que sempre

me retornava o seguinte erro: "The Input Filter PHP extension is not setup on this PHP installation, so the contents returned by Input is *not* filtered." Mesmo a versão do PHP estando certa e todas as extensões instaladas, permanecia o erro. Entrei no site da VCL for PHP e encontrei uma correção no link: vcl4php.svn.sourceforge.net/viewvc/*checkout*/vcl4php/trunk/vcl/system.inc.php?revision=5. Substituí o arquivo no diretório *C:\Arquivos de programas\CodeGear\Delphi for PHP\1.0\vd*, fiz novamente o *Deploy*, publiquei e funcionou.

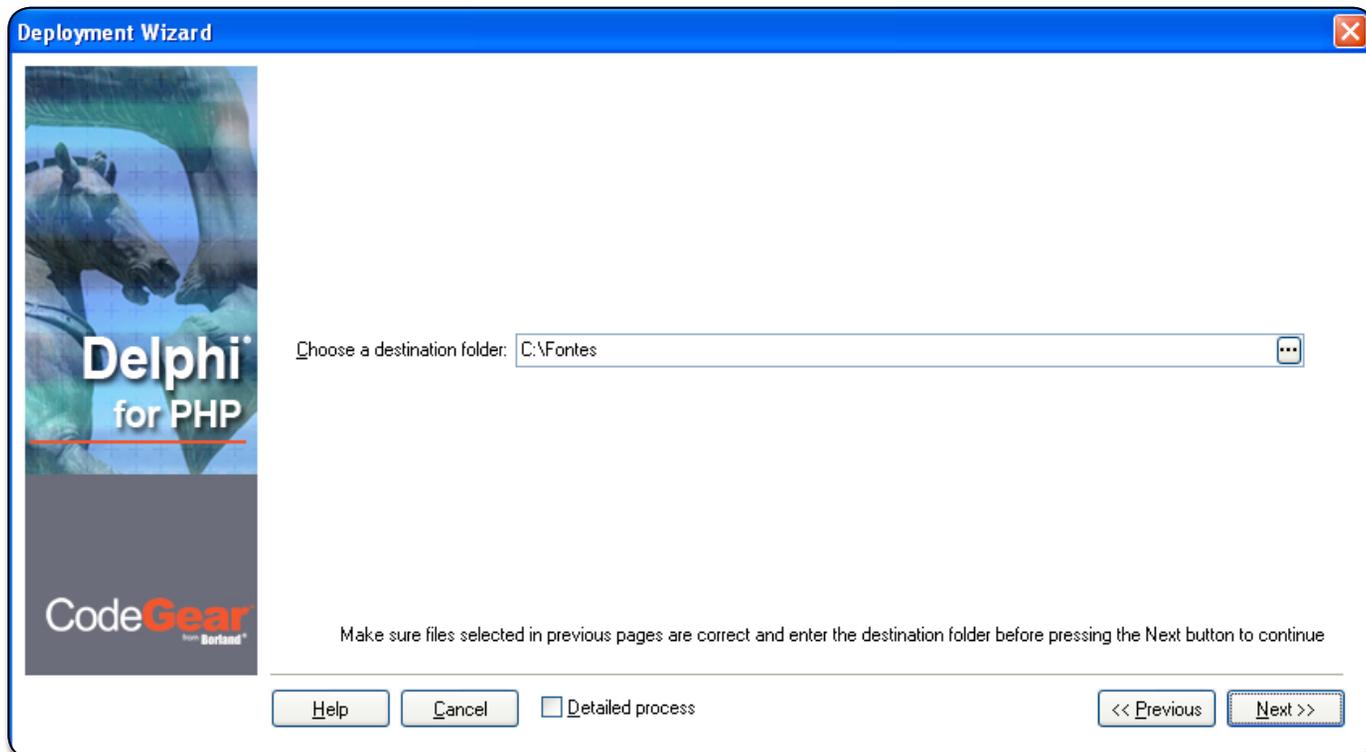


Figura 7. Fazendo o deployment da aplicação

FAÇA COMO A ALOG FOQUE EM SEU NEGÓCIO

A ALOG é totalmente focada em serviços de Data Center. Dedicada-se 24 horas por dia, 7 dias por semana a cuidar do negócio dos nossos 320 clientes corporativos.

ALOG desenvolveu no Brasil o conceito de **HOSTING GERENCIADO**: um serviço de excelência que garantirá a sua empresa estabilidade, agilidade, redundância, e escalabilidade. Para que isso seja verdade os clientes da ALOG contam com **Suporte Incondicional***, um conceito revolucionário em atendimento no Brasil.

ENTENDA O HOSTING GERENCIADO

CAMADAS DE NEGÓCIO	DESCRIÇÃO			
Processo de Negócio	Atividades cotidianas do cliente			
Aplicação do Cliente	Softwares de Gestão (Ex: ERP, CRM, outros)			
Ambiente da Aplicação	.NET, JAVA, SQL Server, MySQL, Oracle, Exchange			
Sistema Operacional	Windows, Linux, Unix, Solaris			
Servidor ou equipamento	Servidores RISC/CISC, Appliances, Roteadores, Switches			
Rede	IP, Wan, Rede Local			
Infra-estrutura Básica	Energia, HVAC, Segurança - 2N+1			
	ESTRATÉGIA DO CLIENTE	COLOCATION	HOSTING	HOSTING GERENCIADO

PLANOS DE HOSTING GERENCIADO

GERENCIADO 1

- Pentium D 2.8 GHz
- 512 RAM DDR2
- 2 HD SATA 80 GB
- 450 GB de Transf. Mensal
- Firewall e Backup

**BOOT REMOTO
GERENCIAMENTO****

R\$1.050,00 mensais
INSTALAÇÃO IMEDIATA***

GERENCIADO 2

- Dual Xeon Dual Core 3.0 GHz
- 1GB RAM DDR2 533
- 2 HD SATA RAID I 300 GB
- 1.000 GB de Transf. Mensal
- Firewall e Backup

**BOOT REMOTO
GERENCIAMENTO****

R\$2.850,00 mensais
INSTALAÇÃO IMEDIATA***



Excelência em Hosting
www.alog.com.br :: 0800 282 3330

Seu potencial. Nossa inspiração.™

Microsoft

ANTES DE ENCARAR UMA BATALHA
NA INTERNET, ESCOLHA A ARMA CERTA.

ENFREENTE OS DESAFIOS



Com o Visual Studio[®], você tem os controles ASP.NET e AJAX para comandar qualquer browser e ainda aumentar suas habilidades para construir sites mais interativos. Veja todas as dicas e informações em www.enfrenteosdesafios.com.br



Visual Studio

© 2007 Microsoft Corporation. Todos os direitos reservados. Microsoft, Microsoft Visual Studio 2005 e "seu potencial. Nossa inspiração." são marcas, registradas ou não, da Microsoft Corporation nos Estados Unidos e/ou em outros países.