



# ASP.NET e DataSnap

## Criando clientes Web para servidores Midas



### Guinther Pauli

([guinther@devmedia.com.br](mailto:guinther@devmedia.com.br))

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.

O DataSnap tem sido amplamente utilizado para a criação de aplicações distribuídas. O Delphi oferece um mecanismo quase que imbatível em produtividade nesse sentido. Podemos criar aplicações multicamadas de uma forma RAD e simples, sem nos preocuparmos com detalhes internos à tecnologia, como protocolos de comunicação.

Muitos desenvolvedores gastaram horas, dias, meses, construindo um servidor de aplicação robusto que concentre todo o acesso a dados e regras de negócio em uma única camada, compartilhada por todas as aplicações clientes.

Quando falamos em DataSnap, no entanto, estamos falando em VCL. Muitos sem dúvida sentiram a necessidade de oferecer não só um cliente desktop para o servidor de aplicação, mas uma interface Web, de forma que clientes pudessem acessar a solução multicamadas de qualquer local do planeta, usufruindo

de todos os recursos e códigos implementados no servidor de aplicação.

O Delphi for .NET deu suporte a uma poderosa tecnologia para o desenvolvimento de aplicações Web: o ASP.NET. O ASP.NET é sem dúvida hoje a tecnologia mais robusta para o desenvolvimento de soluções Web, e muitos desenvolvedores Delphi começaram a utilizá-la para novas soluções.

Mas como ficam as aplicações multicamadas existentes, onde todo o BD, acesso a dados e regra de negócio já estão implementados? Não seria interessante oferecer uma opção Web para a mesma arquitetura, convivendo em harmonia com aplicações cliente desktop já existentes?

Esse é o propósito deste artigo: mostrar como construir uma interface Web para um servidor DataSnap, usando a tecnologia mais robusta para esse propósito, o ASP.NET. Clientes desktop Win32 e Web poderão conviver lado a lado e poderão

compartilhar a mesma camada de negócio e acesso a dados, ou seja, o mesmo servidor de aplicação.

## Preparando o banco de dados

Neste exemplo vou utilizar um banco de dados no Firebird 2.0, mas sinta-se a vontade para utilizar o banco de dados de sua preferência. Para facilitar, vamos criar uma tabela com uma estrutura bastante simplificada dentro de um novo banco de dados, chamado "CUSTOMER.FDB".

Para criar o banco e a tabela "CUSTOMERS" você pode utilizar o IBExpert ([www.ibexpert.com](http://www.ibexpert.com)). A estrutura da tabela é mostrada na **Figura 1**. O *script* de criação da tabela pode ser visto na **Listagem 1**.

Ao criar a tabela, veja que definimos um *Sequence* para incrementar automaticamente o campo chave, nesse caso *CustomerId*. Se estiver usando o modo de designer do IBExpert para criar a tabela, basta marcar a opção *AutoInc* e marcar a seguir os itens *Create Generator/Sequence* e *Create Trigger*.

Criada a tabela, adicione alguns registros nela para facilitar nossos testes quando formos criar a aplicação no Delphi.

## Criando o servidor DataSnap

No Delphi, clique em *File>New>Other>Multitier>Transactional DataModule*. Na caixa de diálogo que aparece, digite "DM" para o nome da *CoClass*. Uma *CoClass* é na verdade uma classe que implementa uma interface COM. Salve a unit com o nome de "uDM.pas" e o projeto com o nome de "ServerComPlus.dpr", dentro de um diretório de mesmo nome.

Usando componentes dbExpress, configure uma conexão para o banco de dados criado, através do editor de conexões do *SQLConnection*. Lembre-se de colocar *localhost* na frente do caminho do banco para que seja usado TCP/IP e aplicações Web possam acessar o banco (elas são *multi-thread*, conexões locais não funcionam). Configure o *LoginPrompt* para *False*.

Coloque um *SQLDataset* ("sqlCustomers") e aponte-o para a conexão. Na propriedade *CommandText* retorne os registros da tabela CUSTOMERS conforme o *select* da **Listagem 2**.

Coloque um *DataSetProvider* ("dspCus-

tomers") apontando para esse *SQLDataSet*. Configure o parâmetro do *SQLDataSet* para *Input* e *ftString*. Dê também um valor padrão de "%" para ele.

Dê um *Build* no projeto. Agora vamos instalar o servidor no catálogo do COM+. Para isso, podemos usar o próprio IDE do Delphi, através do menu *Run>Install COM+ Objects*.

Na caixa que aparece, marque o objeto DM e na outra caixa, na opção *Install into a new Application* digite "ServerComPlus". Clique OK e OK. Pronto, nosso servidor está instalado e pronto para ser acessado.

## Requisitos para acessar o COM+ a partir do ASP.NET

Três etapas precisam ser feitas para podermos acessar o servidor COM+ DataSnap a partir de aplicações ASP.NET. A primeira diz respeito à segurança do aplicativo, a segunda é relativa à compatibilidade dos aplicativos ASP.NET com o COM+ e a terceira diz respeito a interoperabilidade entre .NET e Win32, mais

especificamente ADO.NET e DataSnap / ClientDataSet.

Vamos ao primeiro passo. Toda vez que um cliente instancia um servidor COM+, por padrão o objeto assume a identidade do usuário interativo, ou seja, do usuário logado no Windows.

Quando acessamos uma página Web, o ASP.NET juntamente com o IIS (servidor Web) utiliza um usuário padrão para instanciar o aplicativo, chamado *IUSR\_NomeDaMaquina*, a menos que a autenticação esteja configurada no aplicativo para usar algo diferente.

O problema é que esse usuário, que será automaticamente usado pela nossa aplicação ASP.NET, não possui privilégios para instanciar objetos COM. Nesse caso temos várias soluções. Uma delas seria

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

Accesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Guinther Pauli que mostra como criar servidores COM+.

[www.devmedia.com.br/articles/viewcomp.asp?comp=983](http://www.devmedia.com.br/articles/viewcomp.asp?comp=983)

Fields	Description					
<b>COMPANY VARCHAR(50) NOT NULL</b>						
#	PK	Field Name	Field Type	Size	Not Null	AutoInc
	1	CUSTOMERID	INTEGER		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
		FIRSTNAME	VARCHAR	50	<input checked="" type="checkbox"/>	<input type="checkbox"/>
		COMPANY	VARCHAR	50	<input checked="" type="checkbox"/>	<input type="checkbox"/>

**Figura 1.** Estrutura da tabela CUSTOMERS

### Listagem 1. Código de criação da tabela CUSTOMERS

```
CREATE SEQUENCE GEN_CUSTOMERS_ID;

CREATE TABLE CUSTOMERS (
  CUSTOMERID INTEGER NOT NULL,
  FIRSTNAME VARCHAR(50) NOT NULL,
  COMPANY VARCHAR(50) NOT NULL
);

ALTER TABLE CUSTOMERS ADD CONSTRAINT
PK_CUSTOMERS PRIMARY KEY (CUSTOMERID);

CREATE TRIGGER CUSTOMERS_BI FOR CUSTOMERS
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.CUSTOMERID IS NULL) THEN
    NEW.CUSTOMERID = GEN_ID(GEN_CUSTOMERS_ID,1);
  END
```

### Listagem 2. Código SQL do SQLDataSet

```
select
  CUSTOMERID,
  FIRSTNAME,
  COMPANY
from
  CUSTOMERS
where
  FIRSTNAME like :FIRSTNAME
```

trocar a conta para acesso anônimo, o que não seria uma boa idéia. Outra seria configurar a identidade do aplicativo ASP.NET, usando o *impersonate*.

A que escolhi e acredito ser a mais segura e fácil de implementar foi dizer ao COM+ para rodar o objeto servidor com uma conta de um usuário específico. Para fazer isso, acesse os *Serviços de Componentes* dentro do *Painel de Controle>Ferramentas*

*Administrativas*. O que você vai ver é uma interface que permite configurar várias opções dos objetos registrados, como pacotes, componentes etc.

Localize nossa aplicação, que deve se chamar *ServerComPlus*, e dê um clique de direita para abrir as propriedades. Na aba *Identity* configure as opções como mostrado na **Figura 2**, indicando uma conta válida (pode ser o seu usuário) na

opção *This User*. A seguir, desmarque a opção mostrada na **Figura 3**.

A segunda modificação é na aplicação ASP.NET. Você deve modificar a página que vai acessar o aplicativo COM+, acessando o código-fonte do arquivo ASPX. A diretiva *Page* deve conter agora o atributo *AspCompat*:

```
<%@ Page language="c#" Debug="true"
Codebehind="WebForm1.pas"
AutoEventWireup="false"
Inherits="WebForm1.WebForm1"
AspCompat="true" %>
```

Essa configuração permite que a página ASP.NET utilize o mesmo modelo de *thread* utilizado pelo servidor COM+, nesse caso por padrão é *single thread apartment* (STA).

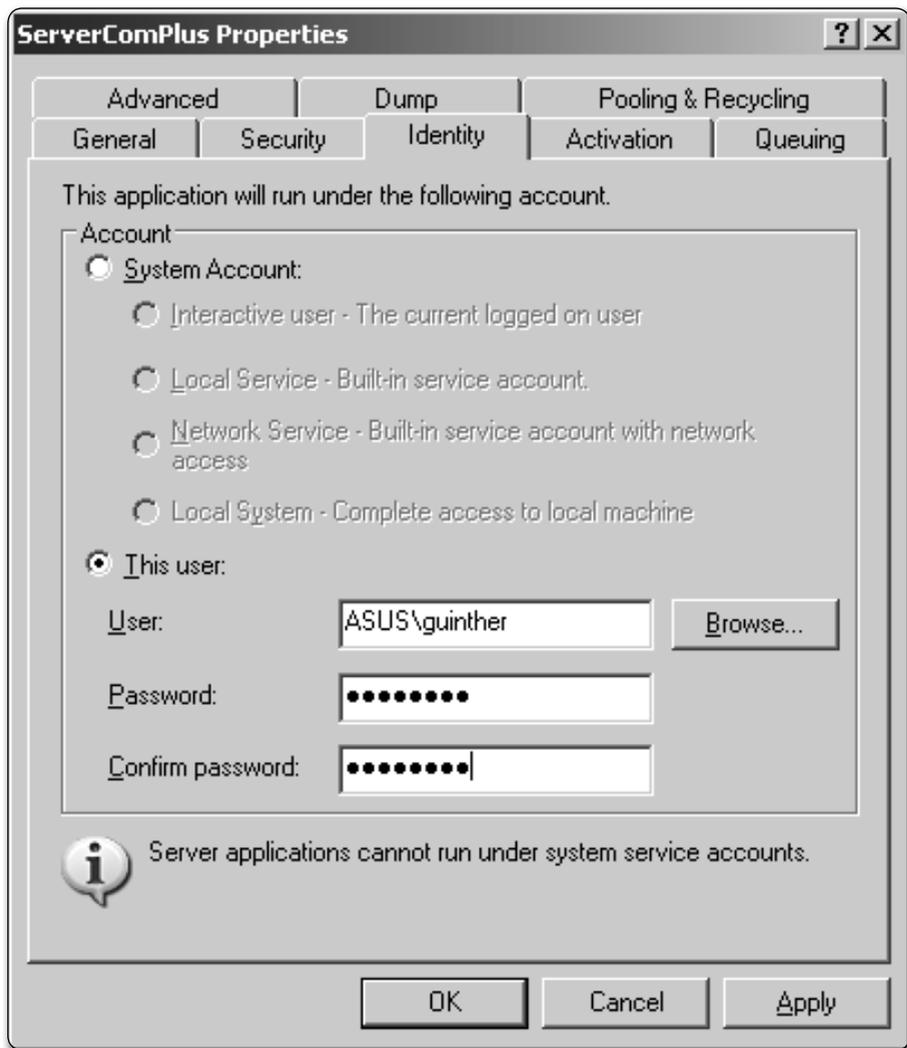
O terceiro passo, e mais complicado, é prover a comunicação entre ASP.NET e DataSnap. O grande problema é que um aplicativo ASP.NET não vai reconhecer os formatos de *DataPacket* enviados por um servidor DataSnap.

Uma solução seria usar XML, mas isso envolveria criar um servidor SOAP e não COM+. Veremos como resolver esse problema a seguir.

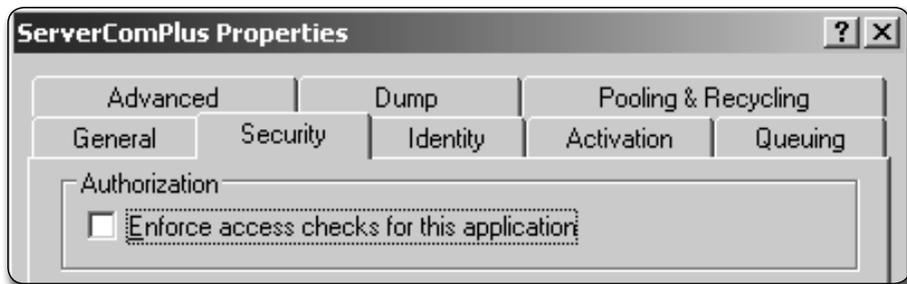
---

**Nota:** Antes de prosseguir, eu realmente sugiro que você crie uma aplicação cliente para teste em Win32, para ver se está tudo funcionando no seu servidor DataSnap COM+. Isso porque a maioria das mensagens de erro que possam vir a acontecer na aplicação ASP.NET, devido a um problema no servidor, não descreverão a causa exata do problema.

---



**Figura 2.** Configurando a conta do usuário COM+



**Figura 3.** Desmarcando o Enforce Access Checks

### Criando a aplicação cliente ASP.NET

Vamos partir para o último passo, que é a criação da aplicação cliente. Usando Delphi for .NET inicie uma nova aplicação ASP.NET. Já na página principal, no arquivo ASPX, faça a alteração na diretiva *AspCompat* como descrito anteriormente. Desenhe um formulário como mostra a **Figura 4**.

Os nomes dados aos componentes são: "btSelect" e "btApply" para os botões, "tbFind" para o *TextBox* e "DG" para o *DataGrid*. Para colocar o botão *Edit* no *Grid*, basta selecionar o *DG* e no *Object Inspector* acessar a opção *Property Builder*. No editor, acesse *Columns* e em *Button Column* adicione uma coluna do tipo *Edit, Update e Cancel*.

Agora precisamos obter os dados do servidor DataSnap. Para isso, no *Click* do *Select* digite o código da **Listagem 3**. O código do *GridDataBind* é mostrado na **Listagem 4**.

O método da **Listagem 3** mostra um personagem novo na história. Para obter os dados do servidor DataSnap, utilizamos um *DataSet* do ADO.NET para colocar os dados em memória (sessão). Porém, o formato de dados desse *DataSet* não é o mesmo que é devolvido por um *IAppServer* do servidor DataSnap. O formato usado pelo *ClientDataSet* em aplicações multicamadas se chama *DataPacket*, e não é compatível com o *DataSet* do ADO.NET.

Para resolver o problema, criei uma classe descendente de *DataSet* que é capaz de acessar um servidor DataSnap, chamada apropriadamente de *MidasDataSet*. Eu adicionei nessa classe algumas propriedades básicas para comunicação Midas, como você pode ver no código as propriedades *RemoteServer* e *ProviderName*, como se fosse um *ClientDataSet*.

A classe se comunica como um servidor COM+ por meio da interface *IAppServer* e recebe os dados em *DataPackets*, transformando-os em um *DataTable* do ADO.NET que fica interno ao *DataSet*.

Com isso, podemos fazer o *DataBind* para um *DataGrid*. Para facilitar, eu coloquei a classe *MidasDataSet* em uma unit separada, explicada a seguir.

### A classe MidasDataSet para comunicação COM+/.NET

Para implementar a classe uma nova unit deve ser criada, chamada "Midas

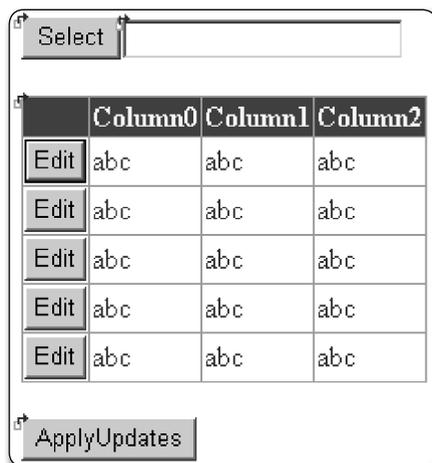


Figura 4. Web Form principal da aplicação

AdoDataSet.pas". O seu código é mostrado na **Listagem 5**.

**Nota:** Para facilitar e reduzir o código, declarei alguns membros como atributos simples na classe, mas o ideal é que você crie-os na forma de propriedades.

Para acessar o servidor DataSnap, foi necessário importar alguns namespaces da VCL for .NET, mostrados no início da unit. Você precisa referenciar os *assemblys* a partir do *Project Manager*, através da opção *Add Reference*. Os *assemblys* necessários são *Borland.VclDbRtl.dll*, *Borland.VclDSnap.dll* e *Borland.VclDSnapCon.dll*.

#### Listagem 3. Método para obter os dados

```
procedure TWebForm1.btSelect_Click(
  sender: System.Object; e: System.EventArgs);
var
  ds: MidasDataSet;
begin
  ds := MidasDataSet.Create();
  ds.RemoteServer := 'ServerCOMPlus.DM';
  ds.ProviderName := 'dspCustomers';
  ds.Param := tbFInd.Text;
  ds.Open(true);
  session['ds'] := ds;
  GridDataBind();
end;
```

#### Listagem 4. Método para preencher o DataGrid

```
procedure TWebForm1.GridDataBind;
begin
  DG.DataSource := session['ds'] as DataSet;
  DG.DataBind();
end;
```

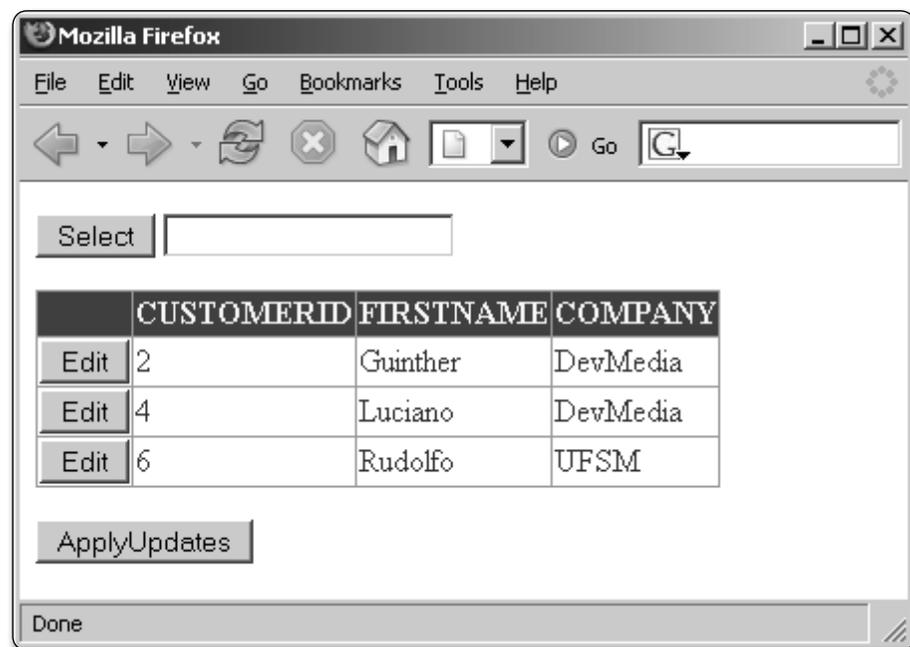


Figura 5. Aplicação ASP.NET acessando servidor DataSnap

### Listagem 5. Código da unit MidasAdoDataSet

```
unit MidasAdoDataSet;
interface
uses
  Borland.Vcl.MConnect, Borland.Vcl.Db, Borland.Vcl.DBClient, Borland.Vcl.Variants, System.Data;
type
  MidasDataSet = class (DataSet)
  private Cds: TClientDataSet;
  procedure Convert();
  procedure OpenCds();
  public
    RemoteServer: string;
    ProviderName: string;
    Param: string;
    procedure Open(KeepCdsInSession: boolean);
    procedure ApplyUpdates();
  end;
  MidasConnection = class(System.&Object)
  public
    DCom: TDCOMConnection;
  public
    constructor Create(ServerName: string);
  end;
implementation
  procedure MidasDataSet.ApplyUpdates;
  var
    row: DataRow;
    Key: TField;
    i: Integer;
  begin
    for row in Tables[0].Rows do
      if row.RowState = DataRowState.Modified then
        begin
          Key := cds.Fields[0];
          cds.Locate(Key.FieldName, VarArrayOf([row[key.FieldName].ToString()]), []);
          cds.Edit();
          for i := 1 to cds.FieldCount - 1 do
            cds.Fields[i].AsString := row[cds.Fields[i].FieldName].ToString();
            cds.Post();
          end;
          / Testar aqui as opções de insert e delete /
          cds.ApplyUpdates(0);
        end;
      procedure MidasDataSet.Convert();
      var
        i: integer;
        col: DataColumn;
        dt: DataTable;
        row: DataRow;
      begin
        cds.Open();
        dt := DataTable.Create();
        Tables.Clear();
        Tables.Add(dt);
        for i := 0 to cds.Fields.Count - 1 do
          begin
            col := DataColumn.Create;
            col.ColumnName := cds.Fields[i].FieldName;
            dt.Columns.Add(col);
          end;
          while not cds.eof do
            begin
              row := dt.NewRow;
              for i := 0 to Cds.FieldCount - 1 do
                begin
                  row[cds.Fields[i].FieldName] := cds.Fields[i].AsString;
                end;
              dt.Rows.Add(row);
              cds.Next;
            end;
            AcceptChanges();
          end;
          procedure MidasDataSet.Open(KeepCdsInSession: boolean);
          begin
            OpenCds();
            Convert();
            if not KeepCdsInSession then cds.Free();
          end;
          procedure MidasDataSet.OpenCds();
          var
            Con: MidasConnection;
          begin
            con := MidasConnection.Create(RemoteServer);
            cds := TClientDataSet.Create(nil);
            cds.ProviderName := ProviderName;
            cds.RemoteServer := Con.DCom;
            cds.FetchParams();
            if Param <> '' then Cds.Params[0].AsString := Param;
            cds.Open();
          end;
          constructor MidasConnection.Create(ServerName: string);
          begin
            inherited Create();
            DCom := TDCOMConnection.Create(nil);
            DCom.ServerName := ServerName;
          end;
        end;
      end;
end;
```

---

**Nota:** Se o leitor observar, o que fiz aqui foi exatamente o inverso da técnica que demonstrei na edição passada, onde na ocasião foi preciso transformar um *DataSet* do ADO.NET em um *Data Packet*.

---

Executando a aplicação e clicando no botão, já podemos testar o sistema (**Figura 5**). Você pode fornecer um parâmetro para o *like* implementado no servidor, digitando um texto no *TextBox*. Para simplificar o exemplo, a classe *MidasDataSet* considera que o *DataSet* no servidor possui nenhum ou um único parâmetro.

---

**Nota:** Se a aplicação não rodou, verifique atentamente todos os passos realizados e também o código. Nesse tipo de solução, a mensagem de erro que você obtém pode não indicar o erro exato. Experimente trocar simplesmente o nome do *ProviderName* no código e veja que receberá a mirabolante mensagem "Catastrophic failure" ou "Falha Catastrófica".

---

## Edição e ApplyUpdates

Após os dados serem obtidos do servidor, pela nossa classe *MidasDataSet*, eles são guardados em sessão (memória) para poderem ser manipulados pelo *DataGrid* do ASP.NET. Para isso, usamos o objeto *Session*.

Se observar bem, quando colocado em memória, nosso *MidasDataSet* terá dois *ResultSets*: o *ClientDataSet* originalmente obtido a partir do *DataPacket* e os dados do *DataTable*, que são as informações convertidas para ADO.NET.

Guardamos o *ClientDataSet* original em *cache* juntamente com o *MidasDataSet* por um único motivo: precisamos dele no *ApplyUpdates*. Quando chamamos o *ApplyUpdates* do nosso *MidasDataSet*, todos os valores alterados no *DataTable* são repassados de volta ao *ClientDataSet*, e então voltamos a fazer uma comunicação *DataSnap* com o servidor de aplicação.

---

**Nota:** Caso não vá alterar o *DataSet*, usando-o apenas para consulta, passe *False* para o parâmetro *KeepCdsInSession* do *Open*. Crie esse parâmetro com o propósito de otimizar a solução caso não vá precisar editar os dados.

---

Para testar o *Apply*, para habilitar a edição na interface da aplicação, primeiramente crie o código da **Listagem 6** para o *EditCommand* e *CancelCommand* do *DataGrid*. O código simplesmente coloca o grid em edição baseado na célula que o usuário clicou (botão *Edit*).

**Nota:** O *DataGrid* do ASP.NET, na sua versão 1.1, não suporta a edição automática como estamos acostumados a fazer com um *DBGrid* na VCL. Então tudo deve ser feito manualmente, como obter os dados editados e jogar na *DataSet*.

Quando o usuário clica na *Update* após a edição de um registro no Grid, precisamos capturar os valores digitados nas células e devolver para o respectivo registro no *Data Set* (nesse caso o *MidasDataSet*). Isso é feito no evento *UpdateCommand* do *DataGrid* (**Listagem 7**). O código do *Apply* simplesmente chama o *ApplyUpdates* do *MidasDataSet* que estava em sessão (**Listagem 8**). A **Figura 6** mostra a edição em execução.

**Nota:** Para não estender o exemplo, eu omiti as operações de inserção e exclusão. Mas você pode facilmente se basear no código do *Update* para criar essas operações.

## Conclusão

O exemplo aqui demonstrado é simples, mas ilustra o poder da combinação proposta. Como possíveis melhorias, sugiro que o leitor incremente o *MidasDataSet* para que se pareça ainda mais com um *ClientDataSet*, usando e abusando dos recursos de *IApp-Server*. O primeiro passo está dado.

A solução aqui apresentada vai agora permitir que você crie soluções Web baseadas em ASP.NET reaproveitando todo o código da sua camada intermediária, ao mesmo tempo que mantém a compatibilidade com clientes desktop.

Dessa forma, poderá oferecer ambas as soluções, por exemplo, permitindo o acesso Win32 a partir de uma intranet, e o acesso Web a partir da Internet. ●

### Listagem 6. Eventos *EditCommand* e *CancelCommand* do *DataGrid*

```
procedure TWebForm1.DG_EditCommand(
  source: System.Object;
  e: System.Web.UI.WebControls.DataGridCommandEventArgs);
begin
  DG.EditItemIndex := e.Item.ItemIndex;
  GridDataBind();
end;

procedure TWebForm1.DG_CancelCommand(
  source: System.Object;
  e: System.Web.UI.WebControls.DataGridCommandEventArgs);
begin
  DG.EditItemIndex := -1;
  GridDataBind();
end;
```

### Listagem 7. Evento *UpdateCommand* do *DataGrid*

```
procedure TWebForm1.DG_UpdateCommand(
  source: System.Object;
  e: System.Web.UI.WebControls.DataGridCommandEventArgs);
var
  ds: DataSet;
  row: DataRow;
begin
  ds := session['ds'] as DataSet;
  row := ds.tables[0].rows[e.Item.ItemIndex];
  row['FirstName'] := (e.Item.Cells[2].Controls[0] as TextBox).Text;
  row['Company'] := (e.Item.Cells[3].Controls[0] as TextBox).Text;
  DG.EditItemIndex := -1;
  GridDataBind();
end;
```

### Listagem 8. *ApplyUpdates*

```
procedure TWebForm1.btApply_Click(
  sender: System.Object; e: System.EventArgs);
begin
  (Session['ds'] as MidasDataSet).ApplyUpdates();
  GridDataBind();
end;
```

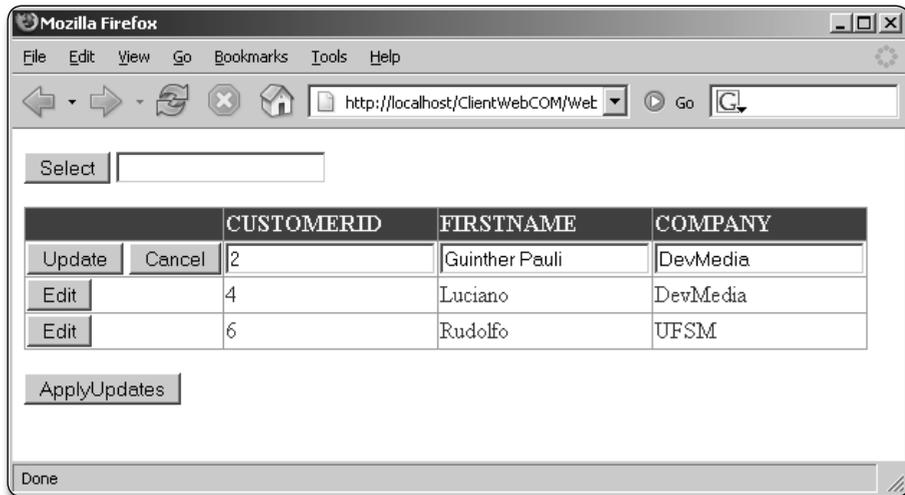


Figura 6. Testando a edição

### Links

**Treinamento On-Line em ASP.NET e Delphi 2005/2006 da ClubeDelphi**  
[www.devmedia.com.br/curso/ecommerce2005](http://www.devmedia.com.br/curso/ecommerce2005)

