

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



Desenvolvendo uma Aplicação Completa com PocketStudio

Aprenda a criar aplicações para o sistema operacional PalmOS – Parte 4



Ricardo C. Boaro

(rboaro@aquasoft.com.br)

trabalha com desenvolvimento de sistemas em Delphi há mais de 10 anos e PocketStudio há 3 anos. Atualmente é gerente de informática na Di Hellen Indústria de Cosméticos, e atual como instrutor certificado Borland na Aquasoft Tecnologia da Informação parceira da Borland, em Porto Alegre – RS. Borland Instrutor, Delphi 7, 2007 e Certified.

Continuando o artigo da edição anterior, iremos criar a última tela de nossa aplicação que será a tela de digitação de itens do pedido. Após concluirmos a criação da mesma iremos desenvolver um conduit para sincronizarmos com o Desktop e trocarmos informações. O objetivo do nosso conduit será enviar para o Desktop as tabelas de *itens de pedido* e *pedidos*, e receber do Desktop as tabelas de *produtos* e *clientes*. Sempre que o usuário conectar o cabo USB no Palm e no Desktop selecionar o *HotSync* o sincronismo será realizado e os dados atualizados, a regra é sempre primeiro enviarmos os dados do Palm para o Desktop e depois recebermos os dados do Desktop para o Palm. Mãos à obra, vamos criar a tela de *Itens de Pedido* para depois entrarmos em detalhes do Conduit.

Criando a tela de digitação de itens do pedido

Vamos adicionar um novo formulário para criarmos a tela de digitação de

itens do pedido, no menu principal do PocketStudio em *File|New>Form*, após adicionarmos o mesmo vamos salvá-lo clicando em *File|Save* ou pelas teclas de atalho *CTRL + S*, salvaremos ele com o nome “UfrmPedidoItem.pas” para indicarmos que é a *Unit* de itens do pedido. Feito isso altere seu *Name* para “FrmItensPedido” e *Caption* para “Itens do Pedido”.

Desenharemos uma tela semelhante à **Figura 1**. Para isso inclua (todos da paleta *Forms*) um componente *List* (“LstItens”), quatro *Label*’s com os *Caption*’s “Total do Pedido”, “Qtde.” e “Preço”. Altere a propriedade fonte do primeiro *Label* para *ftBold*. Em seguida adicione ao lado de cada *Label* um componente *Field* representando um campo da tabela. Incluiremos *FldTotPed*, *FldCodProd*, *FldDescricao*, *FldQtde* e *FldPreco*. Os botões chamemos de *BtnVoltar*, *BtnProcurar*, *BtnNovo*, *BtnGravar* e *BtnExcluir*. Se optar por utilizar *Bitmap*’s para representar as

ações nos botões, não esqueça de incluir quatro Bitmap's e nomeá-los como *BmpProcurar*, *BmpNovo*, *BmpGravar* e *BmpExcluir*. Caso tenha dúvidas da utilização de imagens nos botões, retorne aos artigos anteriores e veja como realizamos a tarefa de inclusão de Bitmap's nas telas anteriores.

Antes de codificarmos os botões vamos escrever algumas funções que irão nos auxiliar nas validações dos dados informados e limpar a tela quando o usuário clicar no botão *Novo* para incluir um novo item no pedido.

Além disso trabalharemos com um novo conceito, o de manipulação de listas. Logicamente o componente *List* que incluímos na janela listará os *Itens do Pedido* conforme adicionarmos. Existem várias maneiras de "alimentarmos" a lista para apresentar os itens. Uma delas seria fazer um laço("loop") e percorrer a tabela de itens do pedido adicionando-os à lista, para isso usaríamos a função *AddCopy* da *PSList*, mas isso é uma operação demorada, e que consome muita memória, sendo assim utilizaremos duas funções que são as chamadas "receitas de bolo", pois montamos a estrutura e podemos utilizar em todas as nossas listas alterando apenas as tabelas e os *Field's*.

A grande vantagem dessas funções é que elas não usam a memória, pois trabalham com os métodos de desenho na



Figura 1. Tela dos itens do pedido

Listagem 1. Procedimentos para listagem, carregamento e limpeza das listas

```
procedure FListaItens(ItemNum: Int16; Bounds: RectanglePtr; var ItemsText: PChar);
var
  Buffer: Array[0..100] of Char;
  ItemInicial: Int16;
begin
  { Se o item solicitado for menor que ZERO, não fazemos nada! }
  if ItemNum < 0 then
    exit;
  { Nós movemos para o índice informado, dentro do
    banco de Itens tomando como base nosso o índice
    absoluto do primeiro registro do Pedido + o
    Número do Item a ser apresentado }
  ItemInicial := 0;
  PSDatabase.MoveTo(DBItip, ItemInicial + ItemNum);

  { Procuramos pelo Produto no banco de Produtos }
  if not ProdutosDB.ProcuraCodigo(PSDatabase,
    FieldUInt16(DBItip, Itp_CodProd)) then
    StrCopy(Buffer, 'Produto não encontrado!')
  else
    StrCopy(Buffer, PSDatabase.FieldStringPtr(DBPro, Prod_DescProd));
  { Desenhamos a Descrição do Produto }
  WinDrawTruncChars(Buffer, StrLen(Buffer),
    Bounds.TopLeft.x, Bounds.TopLeft.y,
    Bounds.Extent.x);
end;

procedure CarregaArrayItem;
var
  Numero: UInt16;
  Buffer: Array[0..30] of Char;
  RecNo: Int32;
  VITotal: Double;
  NumItens: Int16;
begin
  { Zera Valor Total do Pedido }
  VITotal := 0;
  { Zera variável de contagem de Itens }
  NumItens := 0;
  { Recupera o Código do Pedido }
  PSField.Text(FldNumPed, Buffer, PSField.TextLength(FldNumPed) + 1);
  { Utiliza a Busca Binária para encontrar o Primeiro Item }
  Numero := StrAtol(Buffer);
  RecNo := DBSeek(DBItip, Itp_NumeroPed, @Numero, ftUInt32, False);

  { Se não encontrar, não apresenta nada }
  if RecNo = -1 then
    begin
      { Valor Total do Pedido é ZERO }
      PSField.SetText(FldTotPed, '0.00');
      { Atualiza a Lista }
      LstSetListChoices(PSList.Ptr(LstItensPed), nil, NumItens);
      LstSetDrawFunction(PSList.Ptr(LstItensPed), @FListaItens);
      LstDrawList(PSList.Ptr(LstItensPed));
      exit;
    end;
  { Movemos o cursor do até o primeiro registro encontrado }
  PSDatabase.MoveTo(DBItip, RecNo);
  { Salva o índice absoluto do primeiro Item do Pedido }
  ItemInicial := RecNo;
  { Interage pelos registros do Banco e Carrega Array }
  while not PSDatabase.EOF(DBItip) do
    begin
      if PSDatabase.FieldUInt16(DBItip, Itp_NumeroPed) <> Numero then
        break;
      { Soma Valor Total }
      VITotal := VITotal +
        (PSDatabase.FieldDouble(DBItip, Itp_Prec) * PSDatabase.FieldUInt16(DBItip, Itp_Qtde));
      { Incrementamos o índice da Lista }
      Inc(NumItens);
      PSDatabase.Next(DBItip);
    end;
  { Coloca o Valor Total do Pedido no campo específico }
  FormatFloat(Buffer, VITotal, 2);
  PSField.SetText(FldTotPed, Buffer);
  { Atualiza a Lista }
  LstSetListChoices(PSList.Ptr(LstItensPed), nil, NumItens);
  LstSetDrawFunction(PSList.Ptr(LstItensPed), @FListaItens);
  LstDrawList(PSList.Ptr(LstItensPed));
end;

procedure Limpa;
begin
  { Limpamos os Campos do formulário }
  PSField.SetText(FldCodProd, '');
  PSField.SetText(FldDescricao, '');
  PSField.SetText(FldQtde, '');
  PSField.SetText(FldPreco, '');
end;
```

Listagem 2. Código de validação

```
function Valida: Boolean;
var
  Buffer: Array[0..30] of Char;
begin
  Result := False;
  { Validamos se o campo Código do Produto é válido }
  PSField.Text(FldCodProd, Buffer, PSField.TextLength(FldCodProd)+1);

  if not ProdutosDB.ProcuraCodigo(StrAToI(Buffer)) then
  begin
    ShowMessage('Produto Inválido!');
    Exit;
  end;

  { Verificamos se a quantidade é maior do que Zero }
  PSField.Text(FldQtde, Buffer, PSField.TextLength(FldQtde) + 1);
  if StrAToI(Buffer) <= 0 then
  begin
    ShowMessage('Quantidade Inválida!');
    PSField.SetFocused(FldQtde, True);
    Exit;
  end;

  { Verificamos se o valor é maior que zero }
  PSField.Text(FldPreco, Buffer, PSField.TextLength(FldPreco) + 1);
  if FLPAToF(Buffer) <= 0 then
  begin
    ShowMessage('Valor Unitário Inválido!');
    PSField.SetFocused(FldPreco, True);
    Exit;
  end;
  Result := True;
end;
```

Listagem 3. Código da procedure MostraRegistro

```
procedure MostraRegistro;
var
  Buffer, StrQtdEmb: Array[0..30] of Char;
begin
  { Usamos as funções da PSDatabase(FieldUInt16, FieldStringPtr, etc.) para recuperar a
    informação dos campos }
  { Código do Produto }
  StrIToA(Buffer, PSDatabase.FieldUInt16(DBItp, Itp_CodProd));
  PSField.SetText(FldCodProd, Buffer);

  { Mostramos os dados do Produto no formulário }
  if ProdutosDB.ProcuraCodigo(StrAToI(Buffer)) then
  begin
    { Código }
    StrIToA(Buffer, PSDatabase.FieldUInt16(DBPro, Prod_Codigo));
    PSField.SetText(FldCodProd, Buffer);

    { Descrição }
    PSField.SetText(FldDescricao, PSDatabase.
      FieldStringPtr(DBPro, Prod_DescProd));

    { Unidade }
    StrCopy(Buffer, PSDatabase.FieldStringPtr(DBPro, Prod_UN));
    PSField.SetText(FldPdiUnidade, Buffer);
  end;
  { Quantidade }
  StrIToA(Buffer, PSDatabase.FieldUInt16(DBItp, Itp_Qtde));
  PSField.SetText(FldQtde, Buffer);

  { Vamos formatar o valor do campo Itp_Preco antes de mostrá-lo
    O PalmOS não tem função de formatação de campos com casas decimais,
    por isso usamos uma função customizada chamada FormatFloat e colocamos o
    valor formatado no Field }
  FormatFloat(Buffer, PSDatabase.FieldDouble(DBItp, Itp_Preco), 2);
  PSField.SetText(FldPreco, Buffer);

  { Se mostramos dados de um registro, não estamos incluindo }
  { Sendo assim setamos a variável de controle do banco de dados }
  ItemPedDB.bItemPedInclui := False;
end;
```

tela como o *WinDrawTruncChars*. Dessa forma os itens são desenhados na janela e não inseridos na lista o que é extremamente rápido, com a vantagem de usar pouquíssima memória. Por isso, inclua as procedures *FListaltens* e *CarregaArrayItem* à área *Implementation* do formulário de Itens de Pedido. Logo em seguida inclua o procedimento *Limpa*, todas presentes na **Listagem 1**. Fiz questão de comentar cada linha no próprio código, portanto veja as explicações das listagens em formato de comentários no código-fonte.

Para garantirmos a integridade dos dados vamos criar uma função para validação dos dados antes de salvarmos os mesmos. Faremos consistências simples, mas importantes, como por exemplo se o código do produtos existe, se a quantidade informada é maior que zero e se o preço informado é maior que zero. Essas validações garantem dados íntegros. O código dela está na **Listagem 2**, também devidamente comentada.

Outra função importante para nossa *Unit* é a *MostraRegistro*. Como seu nome diz ela irá mostrar os dados do produto. (**Listagem 3**).

Sempre que clicarmos no botão de consultar iremos chamar a função *ConsultaPro*, ela recebe como parâmetro um código do tipo *UInt16*, e utiliza a função *ProcuraCodigo* da base de dados para localizar o produto. Após isso mostra na tela as informações e coloca o foco no *Field* da quantidade (**Listagem 4**).

Nota: Não esqueça de que para compilar o sistema, a *Unit* do formulário de Itens de Pedido precisa ter acesso às demais *Unit's* do projeto. Por isso, encontre a palavra *Implementation* e, abaixo dela, declare uma seção *Uses* se não existir e nela inclua as *Unit's*: *PedidosDb*, *ProdutosDb*, *LibAll*, *uCabPedido*, *uDadosProdutos* e *ItemPedDb*.

Após criarmos todas essas funções para auxiliarmos no perfeito funcionamento da tela de *Itens do Pedido* vamos codificar os botões, iniciando pelo botão de Voltar. Sua codificação é simples e já a fizemos em outras telas. Basta fazer uma chamada ao método *FrmGotoForm* informando qual o formulário que deverá

ser chamado, nesse caso o formulário principal. Veja a seguir:

```
FrmGotoForm(FrmCabPedido);
```

Já o *BtnConsultaPro*, fará a consulta no banco para encontrar um produto. Utilizamos nesse caso uma variável *Buffer* para armazenar o código do produto procurado e informar a procedure *ConsultaPro* para pesquisa. Digite o código da **Listagem 5** no evento *OnClick* do botão em questão.

No código do botão *Novo* precisamos apenas configurar a variável de controle de banco de dados de *Itens* para indicarmos que estamos incluindo um novo registro e limpar a tela, para que o usuário possa entrar com as informações. Para que isso seja feito, basta incluir o código a seguir no botão *Novo*. Veja que o procedimento é bem simples.

```
ItemPedDB.ItemPedInclui:= True;
Limpa;
```

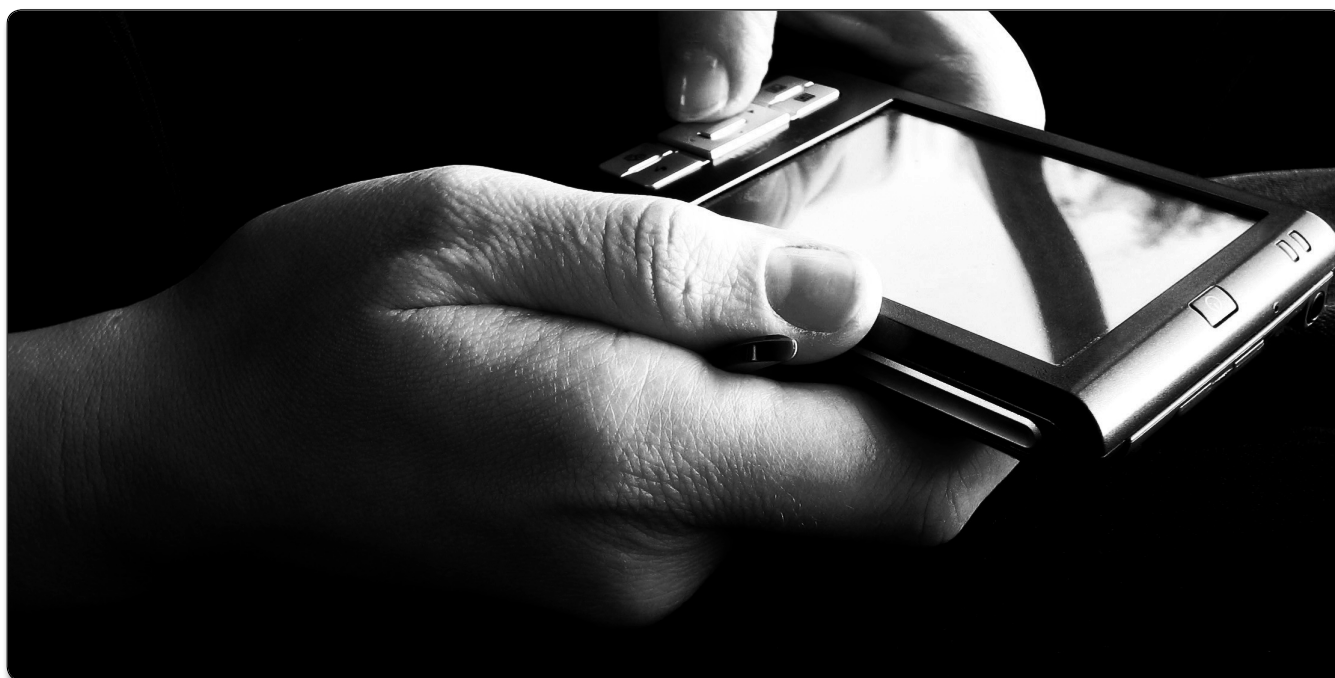
Sem dúvida o código mais trabalhoso será o código do botão *Gravar*, onde passaremos o valor dos *Field's* para os campos da base de dados *ItemPedDB*. Antes disso note que estamos utilizando a função de validação. Confira na **Listagem 6**

Listagem 4. Código da função de consulta

```
procedure ConsultaPro(Codigo: UInt16);
var
  Buffer, StrQtdEmb: Array[0..40] of Char;
begin
  { Procura pelo Produto }
  if not ProdutosDB.ProcuraCodigo(Codigo) then
  begin
    ShowMessage('Produto não encontrado!');
    exit;
  end;
  { Converte o Código do Produto }
  StrIToA(Buffer, PSDatabase.FieldUInt16(DBPro,
    Prod_Codigo));
  PSField.SetText(FLdCodProd, Buffer);
  { Descrição }
  PSField.SetText(FLdDescricao, PSDatabase.
    FieldStringPtr(DBPro, Prod_DescProd));
  { Unidade }
  StrCopy(Buffer, PSDatabase.FieldStringPtr(DBPro,
    Prod_Un));
  PSField.SetText(FLdPdiUnidade, Buffer);
  { Preço }
  FormatFloat(Buffer, PSDatabase.FieldDouble(DBPro,
    Prod_Preco), 2);
  PSField.SetText(FLdPreco, Buffer);
  { Coloca o foco no campo Quantidade }
  PSField.SetFocused(FLdQtde, True);
end;
```

Listagem 5. Código do botão *BtnConsultaPro*

```
procedure BtnConsultaProSelect;
var
  Buffer: Array[0..10] of Char;
  Codigo: UInt16;
begin
  { Recuperamos o valor do FieldCodProd e passamos
    para a variável Buffer }
  PsField.Text(FLdCodProd, Buffer, PsField.
    TextLength(FLdCodProd) + 1);
  { Converte o valor da variável Buffer de
    String para Integer }
  Codigo:= StrAToI(Buffer);
  { Por fim chamamos a função ConsultaPro passando
    a variável Codigo como parâmetro }
  ConsultaPro(Codigo);
end;
```



Listagem 6. Código do botão gravar

```
procedure BtnGravar;
var
  Buffer: Array[0..30] of Char;
  Codigo, RecPos: UInt16;
begin
  { Valida os campos do formulário }
  if not Valida then
    exit;

  { Se estivermos Incluindo colocamos o registro em modo de edição }
  if ItemPedDB.bItemPedInclui then
    begin
      { Recuperamos o Número do Pedido }
      PSField.Text(FldNumPed, Buffer, PSField.TextLength(FldNumPed) + 1);
      Codigo := StrAToI(Buffer);

      { Recuperamos a posição que o registro deve ser inserido antes de adicionar o novo Registro }
      RecPos := DBInsertionPos(DBItip, Itp_NumeroPed, @Codigo, ftUInt32, True);
      PSDatabase.Append(DBItip);
    end
  else
    PSDatabase.Edit(DBItip);

  { Pegamos os dados dos campos do Form e colocamos nos campos do }
  { banco de dados de acordo com o Índice deles }
  PSField.Text(FldNumPed, Buffer, PSField.TextLength(FldNumPed) + 1);
  PSDatabase.SetFieldUInt32(DBItip, Itp_NumeroPed, StrAToI(Buffer));

  { Código do Produto }
  PSField.Text(FldCodProd, Buffer, PSField.TextLength(FldCodProd) + 1);
  PSDatabase.SetFieldUInt16(DBItip, Itp_CodProd, StrAToI(Buffer));

  { Quantidade }
  PSField.Text(FldQtde, Buffer, PSField.TextLength(FldQtde) + 1);
  PSDatabase.SetFieldUInt16(DBItip, Itp_Qtde, StrAToI(Buffer));

  { Valor Unitário }
  PSField.Text(FldPreco, Buffer, PSField.TextLength(FldPreco) + 1);
  PSDatabase.SetFieldDouble(DBItip, Itp_Preco, FLPAToF(Buffer));

  { Grava as informações no banco de dados }
  PSDatabase.Post(DBItip);

  { Depois de Incluir, moveremos o registro para o local correto, }
  { encontrado anteriormente pela função DBInsertionPos }
  if ItemPedDB.bItemPedInclui then
    DBMoveRecord(DBItip, RecPos);

  { Voltamos a incluir um Registro }
  ItemPedDB.bItemPedInclui := True;

  { Limpamos os campos do Item }
  Limpa;

  { Carrega novamente o Array para refletir a mudança no banco de dados de Itens de Pedido }
  CarregaArrayItem;
end;
```

Listagem 7. Código do botão excluir

```
procedure BtPdiExcluirSelect;
begin
  { Se estivermos incluindo, retornamos um erro }
  if PedidosItemDB.bPedidoItemInclui then
    begin
      ShowMessage('Não existe Item selecionado!');
      exit;
    end;

  { Solicita confirmação }
  if FrmAlert(AlertConfirma)=0 then
    Exit;

  { Apaga o Registro }
  PSDatabase.Delete(DBPdi);

  { Voltamos a incluir um Registro }
  PedidosItemDB.bPedidoItemInclui := True;

  { Limpamos os campos do Item }
  Limpa;

  { Carrega novamente o Array para refletir a }
  { mudança no banco de dados }
  { de Itens de Pedido }
  CarregaArrayItem;
end;
```

o código completo do botão Gravar. O código está totalmente comentado para que possa entender perfeitamente tudo o que estamos fazendo.

O último botão a ser codificado é botão de Excluir. Basicamente verificamos o estado da tabela, que pode estar em inserção ou alteração, nesse caso retornamos uma mensagem de erro ao usuário e abortamos. Caso contrário solicitamos uma confirmação do usuário e preparamos para exclusão. O segredo todo está na chamada ao método *DBMoveRecord* que faz efetivamente a exclusão no banco de dados. Em seguida apenas “informamos” ao banco que deve retornar ao estado normal. Por fim recarregamos os dados em tela (**Listagem 7**).

Para finalizarmos a codificação da *Unit de Itens do Pedido*, vamos codificar o evento *OnOpen* do *Form*. Selecionando o formulário, vamos ao *Object Inspector* na aba *Events*. Clique duas vezes no evento *OnOpen* e vamos codificá-lo conforme o código da **Listagem 8**.

Construindo o conduit

O primeiro passo é instalarmos o *TurboSync* no Delphi. Para isso abrimos o Delphi 7 e selecionamos *Component\Install Packages*. Em seguida clique em *Add* e localize o arquivo *TurboSyncD7Trial.bpl* no diretório onde descompactou os arquivos baixados e descompactados anteriormente.

Após a instalação do pacote, vamos no menu *File\New>Other>New>Dll Wizard*. O próximo passo é adicionarmos um *Data Module* que será o objeto responsável por armazenar nossos componentes de acesso a dados. Use o menu *File\New>Data Module* e salve-o como “uDmConduit.pas”. Criada a dll, devemos adicionar a *Unit Forms* na *Uses* da dll, e na seção *begin..end* criarmos nosso *Data Module* quando executada. Salve nosso projeto como “ConduitDelphi.dpr”.

Feito isso devemos adicionar os componentes que farão a transferência dos dados. Após a instalação do *TurboSync*, foi criada uma aba chamada *Conduit*, localize-a e adicione o primeiro componente que é o *TabdConduit*, e o *TabdConduitDB*. O conduit deve ter apenas um *TabdConduit*, e outros

TabdConduitDB para cada tabela que será sincronizada. Para o nosso exemplo devemos adicionar os seguintes componentes configurados conforme a **Figura 2**.

Note que estamos trabalhando com os componentes da paleta BDE do Delphi. Podemos utilizar qualquer base de dados para fazer a importação para o banco de dados do *Desktop*. Na **Figura 2** temos a imagem do *DMConduit* com os componentes adicionados.

Os componentes *Table* da paleta BDE estarão conectados através de um *alias* que irá apontar para uma base de dados Paradox. Não vou entrar em detalhes aqui sobre a criação do *alias* e das tabelas do *Desktop*. A próxima configuração que devemos fazer é criar os *Field's* dos componentes *TabdConduitDB*, *ItemPedDB* e *PedidosDB*, é uma etapa muito importante pois precisamos analisar o tipo de cada campo no PalmOS para definir o mesmo no componente. Os campos são criados na propriedade *Field's* do componente, clicando em *Add*. O primeiro que vamos montar é o do *PedidosDB*, devemos configurar os campos dos *Field's Editor's* de *PedidosDb* e *ItensPedidoDB* como mostrado nas **Tabelas 1 e 2**. Para isso selecione o *TTabdConduitDb* apropriado, clique duas vezes em sua propriedade *Field's* e na janela que se abre adicione os campos necessários.

Em seguida faça as devidas configurações para criação das tabelas em Paradox e inclua nos *Field's Editor's* de cada *TTable*

Campo	Tipo
Itp_NumeroPed	pftLongint
Itp_CodProd	pftSmallint
Itp_Qtde	pftSmallint
Itp_Preco	pftDouble

Figura 2. Tipos dos Fields do ItemPedDB

Campo	Tipo
Ped_NumeroPedido	pftLongint
Ped_CodCli	pftSmallint
Ped_Emissao	pftDate
Ped_Cond	pftSmallint
Ped_Total	pftDouble

Tabela 1. Fields Editor da tabela PedidosDB

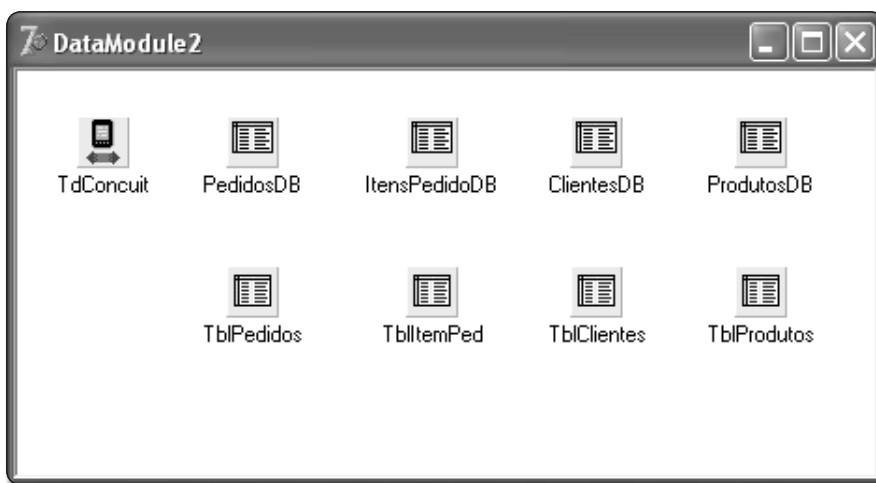


Figura 2. Data Module do nosso aplicativo conduit

Listagem 8. Código do evento OnOpen do formulário

```
procedure FrmItemsPedidoOpen;
var
  Buffer: Array[0..30] of Char;
begin
  PSForm.Draw;

  { Carrega número do Pedido no campo do formulário de Itens de Pedido }
  StrToA(Buffer, PedidoAtual);
  PSField.SetText(FldNumPed, Buffer);

  { Carrega os Itens do Pedido }
  CarregaArrayItem;

  { Coloca o Banco de Dados em Modo de Inclusão }
  ItemPedDB.bItemPedInclui := True;
end;
```

Listagem 9. Exemplo de importação de dados

```
procedure TDataModule2.DataModuleCreate(Sender: TObject);
begin
  PedidosDb.Open;
  TblPedidos.Open;
  while not PedidosDb.EOF do
  begin
    with TblPedidos, PedidosDB do
    begin
      TblPedidos.Append;
      FieldByName('Ped_NumeroPedido').AsFloat := FieldByName('Ped_NumeroPedido').AsFloat;
      FieldByName('Ped_CodCli').AsFloat := FieldByName('Ped_CodCli').AsFloat;
      FieldByName('Ped_Emissao').AsDateTime := FieldByName('Ped_Emissao').AsDateTime;
      FieldByName('Ped_Cond').AsFloat := FieldByName('Ped_Cond').AsFloat;
      FieldByName('Ped_Total').AsFloat := FieldByName('Ped_Total').AsFloat;
    end;
    Next;
  end;
end;
```



Nota do DevMan

Para que o componente TurboSync seja instalado corretamente em seu Delphi, será necessário ter as dll's sync20.dll e hslog.dll. Esses arquivos são instalados juntamente com o aplicativo Palm Desktop. O Palm Desktop é um programa utilizado para sincronizarmos o dispositivo móvel com o Desktop. Com ele podemos inclusive cadastrar contatos, endereços, telefones, e quaisquer outros dados diretamente no computador e transferi-los para o dispositivo móvel, poupando tempo.

O Palm Desktop é encontrado no CD de instalação do seu Palm, caso não tenha um Palm, poderá baixar gratuitamente em sites ligados a comunidades que usufruem desse tipo de dispositivo, como por exemplo o www.palmbrasil.com.br que possui uma enorme quantidade de aplicativos e utilitários tanto para o Palm quanto para Desktop.



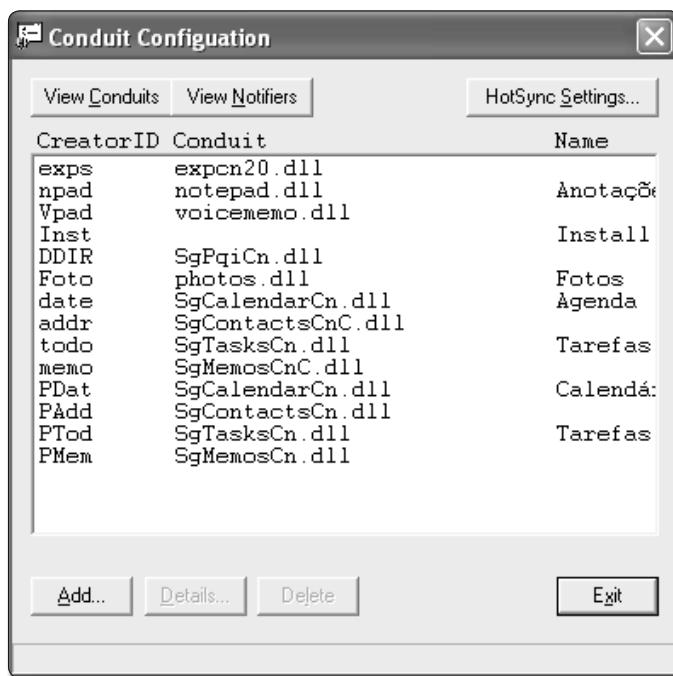


Figura 3. Tela de registro do conduit

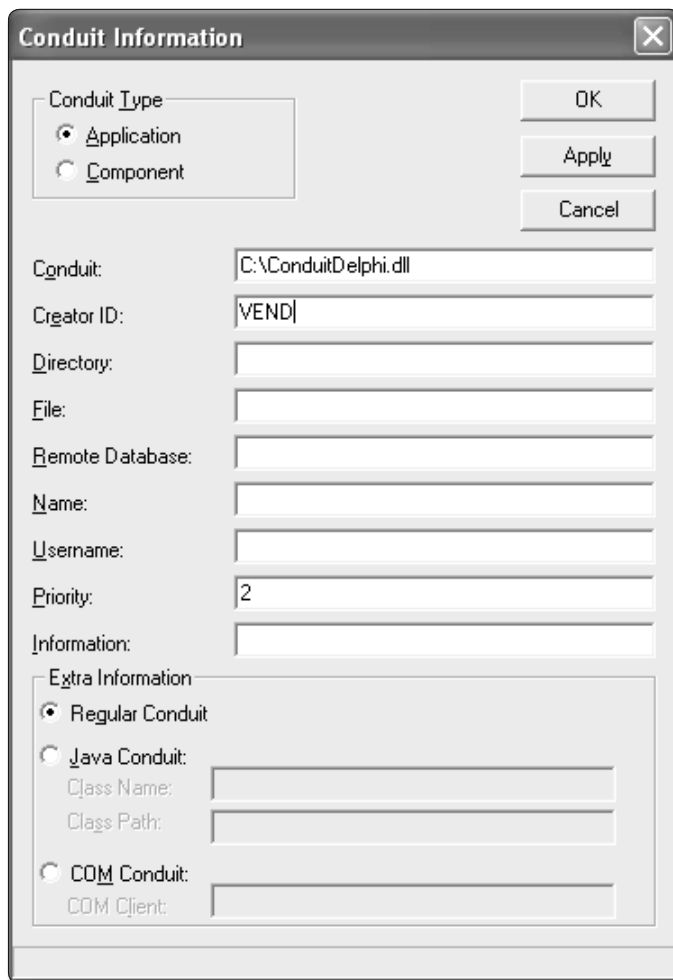


Figura 4. Tela de registro do conduit

o campos necessários. Feita a configuração dos campos dos componentes *TabdConduitDB*, e criadas as tabelas do Paradox no *Desktop*, devemos criar funções para ler os dados das tabelas do *Palm* e gravar nas tabelas no *Desktop* e vice-versa. Lembrando que a aplicação que construímos possui quatro tabelas, *Pedidos* e *Itens* que serão sincronizados do *Palm* para o *Desktop* e *Produtos* e *Clientes* que serão sincronizados do *Desktop* para o *Palm*.

Como estamos trabalhando com *DataSets*, lembrando que o componente *TabdConduitDB* é um *DataSet*, será um processo muito simples. Essas funções devem ser criadas na *Unit* do *DmConduit*, para cada tabela devemos criar uma função para facilitar o entendimento e a manutenção quando necessário. Os fontes do conduit estão disponíveis juntamente com os exemplos deste artigo. Não vou entrar em detalhes aqui sobre a construção das funções, pois os fontes estão comentados e fáceis de entender. Porém, para que se tenha uma noção do que é feito veja a importação de uma das tabelas do sistema na **Listagem 9**. Note que estamos usando apenas os métodos padrões de *TDataSet*, tais como *Insert* e *Edit*, em conjunto com um laço *while not...eof do*. O mais importante do conduit é entender o seu funcionamento.

Conduit

Após concluirmos o desenvolvimento de nossa aplicação para o *Palm*, partimos para o desenvolvimento da aplicação *Conduit*. Uma aplicação *conduit* pode ser desenvolvida em qualquer linguagem que tenha suporte a objetos COM e existem vários exemplos na internet. Uma aplicação dessas, como seu nome já diz, tem a função de exportar os dados do *Desktop* para o *Palm* e do *Palm* para o *Desktop*. Nosso exemplo será construído utilizando o Delphi 7, e um componente chamado *TurboSync*. Utilizando esse componente trabalharemos da mesma forma que trabalharmos com um *DataSet* do Delphi. O *TurboSync* pode ser baixado no endereço www.tabdee.ltd.uk/Downloads/TurboSync.html.

O conduit é muito simples de ser instalado por ser um pacote ("BPL"), a única consideração com relação ao componente disponibilizado, é que por

ser *Trial* ele funcionará apenas enquanto o Delphi estiver rodando, sendo assim devemos lembrar desse detalhe na hora que fazemos o sincronismo.

A aplicação que iremos desenvolver é uma dll. E para ficar claro como o *Palm* irá “saber” que precisa executá-la, o segredo é registrarmos a mesma no processo do *HotSync*.

Para registrarmos nossa dll no processo do *HotSync* devemos adicioná-la no registro do Windows. Para isso estou disponibilizando um executável, chamado de *RegRegistro.exe*, que irá fazer o trabalho para nós, já que alterar o registro do Windows é uma operação delicada. O aplicativo encontra-se disponível juntamente com os fontes de nossa aplicação no site da DevMedia.

Outro segredo para que o sincronismo funcione perfeitamente é que a dll registrada tenha o mesmo *CreatorID* registrado no registro do Windows e na aplicação instalada no *Palm* e informar a pasta onde ela está. A **Figura 3** mostra o nosso *conduit* já registrado,

sendo *CreatorID* o *creator* da aplicação, que definimos na *Unit* principal do projeto, ou seja, *Vendas*. Para facilitar sua localização basta procurarmos pela diretiva {\$CREATOR 'VEND'}. O *Conduit* é a pasta onde está a dll que iremos criar.

Para que possamos então fazer o registro da aplicação, basta abrir o arquivo *RegRegistro.exe* (**Figura 4**) e clicar em *Add*. Uma nova janela será aberta e é nesse momento que preenchemos os campos *Conduit*, *Creator ID* e *Priority*.

Conclusão

Neste artigo finalizamos nossa aplicação de vendas, construindo a tela para digitarmos os itens do pedido e o *Conduit*. Nos quatro artigos sobre desenvolvimento de aplicações com PocketStudio, desenvolvemos uma aplicação simples para controle de vendas, porém abordando os principais conceitos do desenvolvimento para PalmOS com PocketStudio. A intenção dos artigos não foi desenvolver uma aplicação

robusta e comercial e sim mostrar o caminho para tal. Com a aplicação construída com certeza você terá condições de desenvolver a sua aplicação robusta e comercial aplicando os conceitos aqui apresentados. Grande abraço a todos e até a próxima. ●



Nota do DevMan

HotSync é um *conduit* pré-instalado juntamente com o aplicativo *PalmDesktop*, que vem com o CD de instalação do dispositivo. Sempre que executarmos o sincronismo entre o *Palm* e o *Desktop* o *HotSync* será responsável por controlar o que será sincronizado.

Dê seu feedback sobre esta edição!

A *Clubedelphi* tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback



A EDIÇÃO QUE VOCÊ PRECISA
ESTÁ ESGOTADA?

SEUS PROBLEMAS
ACABARAM!!

Seja um assinante Gold!

Com a assinatura Gold você já pode consultar online todos os artigos publicados na sua revista desde a edição nº 1.



Saiba Mais! Acesse:

www.devmedia.com.br/assgold

Para mais informações:

www.devmedia.com.br/central

Assinatura

Gold

Atenção: Já encontram-se disponíveis as assinaturas GOLD das revistas *WebMobile* e *.net Magazine*.