

Como escanear e armazenar documentos com Delphi



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.

Certamente uma das maiores vantagens em informatizar uma empresa hoje, resume-se em armazenar em bancos de dados milhares de informações geradas no dia-a-dia, tais como dados de vendas, cotações, relatórios de movimentação de mercadorias, extratos bancários e uma série de outros relatórios importantes para o bom andamento dos negócios.

Porém, o que vemos é um grande acúmulo de papéis, muitos deles gerados pelo próprio sistema da empresa, já que fica muito mais fácil visualizá-los impressos. Pensando em diminuir a quantidade de papel, algumas empresas costumam digitalizar seus documentos e armazená-los no computador para posterior análise ou apenas no intuito de arquivá-los.

Neste artigo veremos como desenvolver um cadastro de documentos e usufruir de um recurso pouco utilizado no dia-a-dia: o escaneamento de imagens. Vamos colocar a mão na massa.

Nota: Para este artigo é necessário basicamente um scanner de mesa devidamente instalado, tanto fisicamente quanto logicamente, através do software que acompanha o aparelho.

Entendendo o projeto

Nosso projeto terá como objetivo interagir com o software do scanner instalado no Windows. Faremos com que o projeto invoque o aplicativo e aguarde até que o usuário conclua a digitalização da imagem.

Logo em seguida a imagem é transferida para um *DBImage* que é salvo automaticamente em nossa base de dados. Para isso, usaremos um componente *freeware* pouco conhecido que faz todo o trabalho de captura e interação com o usuário. Ele tem a função de “conversar” com a biblioteca *TWAIN_32.DLL* presente no sistema operacional.

Instalando o componente

Junto com o exemplo deste artigo segue o pacote (DPK) de instalação do *AcquireImage* (*TAcquireImage.dpk*). Basta abrir o Delphi, clicar em *Open*, localizar o arquivo e em seguida compilar e instalar (**Figura 1**).

Nota: O *AcquireImage* foi testado nas versões 6, 7 e 2006 do Delphi.

Não esqueça de adicionar o caminho dos arquivos fonte ou DCU no *Library Path* do Delphi, no menu *Tools>Environment Options>Library>Library Path*.

Criando o banco de dados

Em seguida criaremos a base de dados que fará parte do nosso exemplo. A estrutura do banco (tabela DOCUMENTOS) pode ser vista na **Tabela 1**. O *script* de criação do banco encontra-se na **Listagem 1**.

Listagem 1. Criação do banco de dados e tabela

```
SET SQL DIALECT 3;
SET NAMES WIN1252;
CREATE DATABASE <caminho>\DOCUMENTOS.FDB'
USER 'SYSDBA' PASSWORD 'masterkey'
PAGE_SIZE 4096
DEFAULT CHARACTER SET WIN1252;

CREATE TABLE DOCUMENTOS (
  NOME_ARQUIVO VARCHAR(100) NOT NULL,
  DESCRICAO VARCHAR(50) NOT NULL,
  DOCUMENTO BLOB SUB_TYPE 0 SEGMENT SIZE 80
);

ALTER TABLE DOCUMENTOS
ADD CONSTRAINT PK_DOCUMENTOS
PRIMARY KEY (NOME_ARQUIVO);
```

Nota: Veja que o nome do arquivo (NOME_ARQUIVO) é a chave primária da tabela.

Desenvolvendo o projeto e acesso a dados

Vamos “desenhar” uma tela simples de cadastro com os típicos botões *Incluir*, *Alterar*, *Gravar*, *Cancelar* e *Excluir*. Inclua também no layout um botão *Escanear Documento* e outro botão para *Sair*. Para visualizar os dados basta incluir um conjun-

Campo	Tipo/Tamanho
NOME_ARQUIVO	Varchar(100) Not Null
DESCRICAO	Varchar(50) Not Null
DOCUMENTO	BLOB SUB_TYPE 0

Tabela 1. Tabela DOCUMENTOS

Componente	Nome	Propriedade = Valor
DBGrid	dbgDocumentos	DataSource = dtsDocumentos
DBEdit	dbeDescricao	DataSource = dtsDocumentos; FieldName = DESCRICAO
DBEdit	dbeNomeArquivo	DataSource = dtsDocumentos; FieldName = NOME_ARQUIVO
DBImage	dblImagem	DataSource = dtsDocumentos; FieldName = DOCUMENTO

Tabela 2. Componentes visuais para a tabela DOCUMENTOS

to de componentes *Data-aware* como segue na **Tabela 2**.

E para o acesso ao banco de dados vamos usar, como costume chamar, o quarteto fantástico do dbExpress. Insira no formulário um *SqlConnection* e um *SqlDataSet* da paleta *dbExpress* e dê os nomes de “sqlConexao” e “sdsDocumentos”, respectivamente.

No *SqlConnection*, clique duas vezes e crie uma nova conexão apontando para o arquivo *Documentos.fdb* criado anteriormente. Em seguida selecione o *SqlDataSet* e em sua propriedade *SqlConnection* aponte-o para para *sqlConexao*.

Adicione os componentes *DataSetProvider* (“dspDocumentos”), *ClientDataSet* (“cdsDocumentos”) e *DataSource* (“dtsDocumentos”), todos da paleta *Data Access*. Configure-os de acordo com a **Listagem 2**.

Listagem 2. Configurando o acesso ao banco de dados

```
SdsDocumentos
CommandText:
  "SELECT * FROM DOCUMENTOS ORDER BY NOME_ARQUIVO"
SqlConnection: sqlConexao
dspDocumentos
DataSet: sdsDocumentos
cdsDocumentos
ProviderName: dpsDocumentos
dtsDocumentos
DataSet: cdsDocumentos
```

Para concluir, adicione os botões necessários para as operações citadas anteriormente: *Incluir*, *Alterar*, *Gravar*, *Cancelar*, *Excluir*, *Escanear Documento* e *Sair*.

Insira também um *RadioGroup* (“rdgTipoImagem”). Faremos uso dele para escanear a imagem em formato *Bitmap*, *Jpeg* ou direto para a memória. Por isso, em sua propriedade *Items* adicione as seguintes *strings*: “Bitmap”, “JPEG” e “Clipboard (Memória)”.

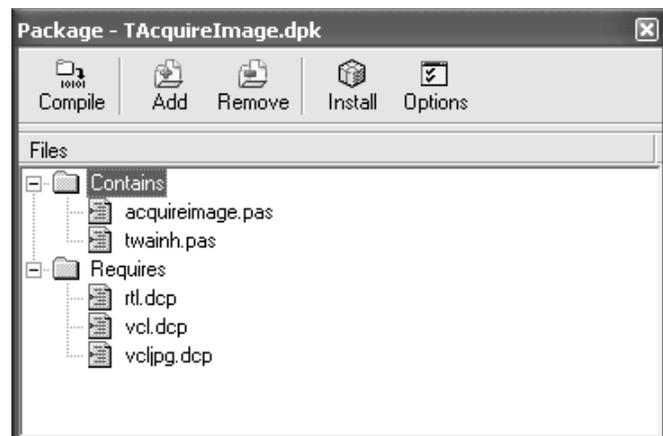


Figura 1. Instalando o AcquireImage



Listagem 3. Código do btnCapturar

```

procedure TfrmPrincipal.btnCapturaClick(Sender: TObject);

var
  Jpg: TjpegImage;
  Bmp: TBitmap;
  Flag: Boolean;
  S: string;

begin
  Flag := False;

  with aiScanearImagem do
  begin
    if LoadTWainModule then
    begin
      try
        OpenSourceManager;
        S := GetSource(False);
        SelectSource(S);
        OpenSource;

        case rdgTipoImagem.ItemIndex of
          0: AcquireBmp(Bmp);
          1: AcquireJpg(Jpg, 50);
          2: AcquireToClipboard;
        end;

        Flag := True;
      finally
        CloseTWainSession;
        UnloadTWainModule;
      end;
    end;
  else
    MessageDlg(
      'Erro ao carregar a biblioteca TWAIN_32.DLL',
      mtError, [mbOk], 0);
  end;

  if Flag then
  case rdgTipoImagem.ItemIndex of
    0: dbImagem.Picture.Bitmap := Bmp;
    1: dbImagem.Picture.Bitmap.Assign(Jpg);
    2: dbImagem.Picture.Bitmap.
      LoadFromClipboardFormat(cf_BitMap,
        Clipboard.GetAsHandle(cf_BitMap), 0);
  end;
end;
end;

```

Ainda se tratando de componente, falta inserir o responsável por interagir com o scanner. Clique na paleta *Others* e insira um *AcquireImage* no formulário. Nesse ponto do artigo nossa tela deverá se parecer com a **Figura 2**.

Não entraremos em detalhes quanto ao desenvolvimento do formulário principal, pois basta incluir rotinas simples para Inclusão, Alteração, Exclusão, Gravação etc. Vamos focar no *AcquireImage*.

Entendendo o componente

O *AcquireImage* funciona basicamente como uma ponte para a biblioteca TWAIN_32.dll que faz todo o trabalho de escanear e guardar a imagem em memória através do aplicativo do scanner. Por isso, internamente o mesmo faz referência à DLL e a carrega para que possamos usá-la.

Quando solicitado, o componente invoca o programa do scanner e aguarda até que o usuário cancele ou solicite a digitalização. Dessa forma, seu resultado é retornado ao *AcquireImage* que por sua vez o armazena.

Capturando a imagem do documento

O que faremos é usar a imagem capturada para atualizar o campo DOCUMENTO do tipo BLOB no banco de dados e salvar, fazendo com que a figura seja salva para futuras consultas. Clique duas vezes no *btnCapturar* e digite o código da **Listagem 3**.

Veremos agora o detalhamento do código de captura começando pelo método *Lo-*

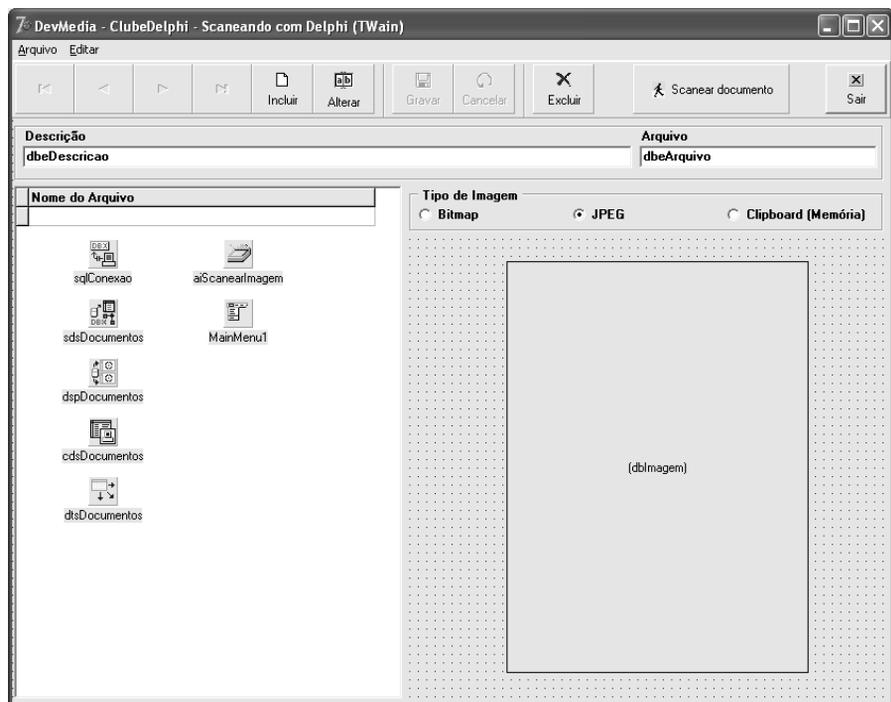


Figura 2. Formulário principal da aplicação



Figura 3. Gerenciador de dispositivo (fonte) a ser utilizado

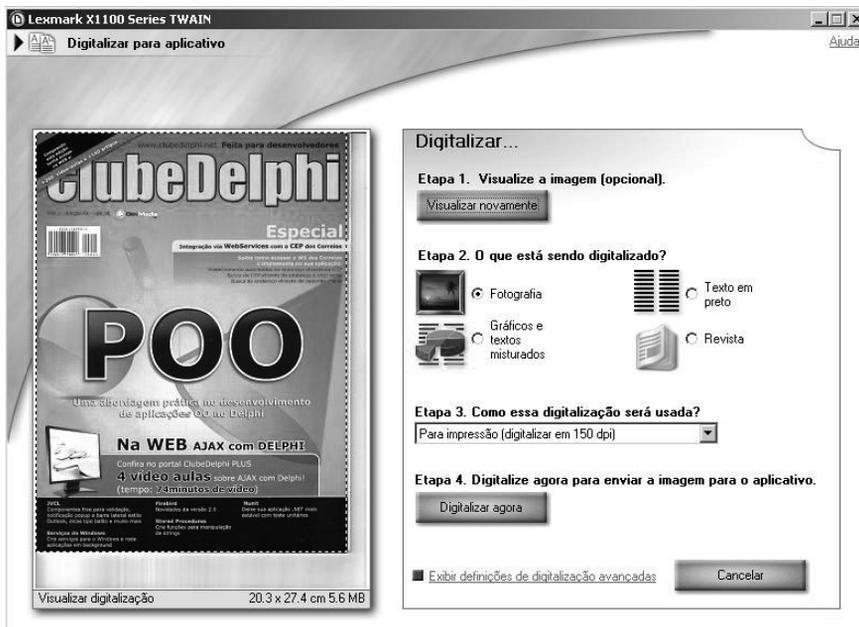


Figura 4. Gerenciador de digitalização do scanner

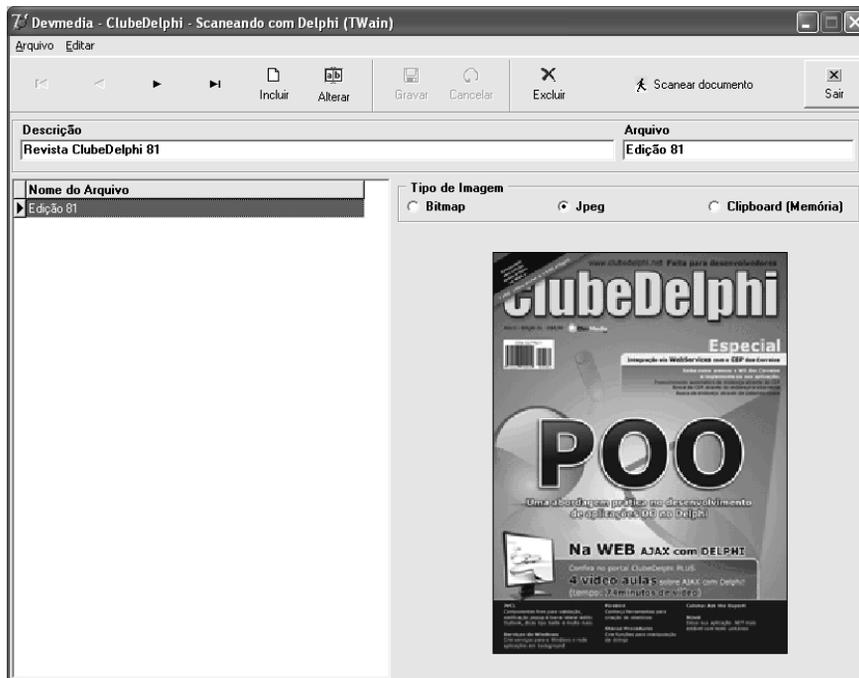


Figura 5. Imagem escaneada e salva no banco de dados do aplicativo

adTWainModule que retorna *True* se conseguiu carregar e *False* se ocorreu algum erro durante o processo. Os erros mais prováveis ocorrem quando a DLL já está em uso por outro processo ou mesmo se ela não existir. Caso ocorra algum problema enviamos uma mensagem ao usuário.

Logo em seguida abrimos o *SourceManager* usando o *OpenSourceManager*. O método se encarrega de verificar quais são as “fontes” que estão disponíveis no sistema operacional. Essas fontes são os aplicativos de digitalização que podem ser utilizados (Figura 3).

Logo em seguida chamamos a função *GetSource*, que abre a janela de dispositivos e aguarda até que o usuário escolha um na lista. Feito isso invocamos a função *OpenSource* para preparar a abertura do aplicativo, e em seguida definimos para onde será enviada a imagem capturada: *Bitmap*, *Jpeg* ou *Clipboard* (memória).

Nesta última fase, usamos o *RadioGroup* para definir e chamar a função necessária, de acordo com o tipo de imagem desejada. Conforme a seleção, chamamos: *AcquireBmp(Bmp)* para adquirir em formato *Bitmap*, *AcquireJpg(Jpg, 50)* para *Jpeg* ou *AcquireToClipboard* para enviar a imagem a memória. São essas três funções responsáveis por chamar efetivamente o gerenciador de digitalização do scanner (Figura 4).

Por fim marcamos a variável *Flag* para *True* pra sinalizar que a digitalização correu bem. Não podemos esquecer de fechar a sessão do módulo DLL (*CloseTWainSession*) e descarregar a mesma da memória (*UnloadTWainModule*).

O último passo é verificar se a *Flag* é verdadeira, indicando que a digitalização aconteceu, e atualizar o *DBImage*. Na Figura 5 podemos ver a imagem escaneada e salva no banco de dados.

Conclusão

O armazenamento de documentos no sistema pode ser uma grande vantagem para quem quer economizar espaço no escritório e sumir com boa parte do papel que é gerado na empresa. Neste artigo, aprendemos a desenvolver uma aplicação capaz de digitalizar e armazenar documentos escaneados. ■