

Nesta seção você encontra artigos sobre técnicas que poderão aumentar a qualidade do desenvolvimento de software

Reaproveitamento de código

Organizando o código comum da aplicação



Nenhum programador gosta de refazer trabalho, e isso é bastante comum quando duplicamos código na aplicação. Imagine por exemplo, que temos um sistema que realiza baixa de contas a receber com lançamento no Caixa. O código para lançar no caixa está presente quando efetuamos a baixa na parcela e o mesmo pode estar quando do lançamento de uma conta, pois a mesma pode ter sido paga. Neste exemplo, teremos código duplicado na aplicação, onde o ideal é criar um método que recebe as informações do lançamento como parâmetro e executar apenas um código.

Neste artigo, vamos conhecer alguns métodos e funções que podemos organizar em uma *Lib* (biblioteca) para que possamos usar em todos nossos projetos.

Código Win32

Vamos mostrar nessa seção, códigos para aplicações *Win32*. Todos os exemplos mostrados aqui, podem ser coloca-

Resumo DevMan

O maior trauma de um desenvolvedor certamente é ter que desenvolver a mesma rotina mais de uma vez para a mesma aplicação ou para sistemas diferentes. Pensando nisso, diversos programadores optam por criar seus próprios componente e/ou bibliotecas de funções tais como: units personalizadas ou arquivos DLL. Nesse artigo veremos algumas boas práticas no desenvolvimento de sistemas evitando a redigitação de código-fonte.

Nesse artigo veremos

- Técnicas de reaproveitamento de código;
- Criação de funções para uso em diversas partes do sistema;

Qual a finalidade

- Mostrar algumas das principais técnicas para evitar a digitação repetitiva de código pela aplicação;

Quais situações utilizam esses recursos?

- Em praticamente todo tipo de aplicação/programa podemos aplicar essas técnicas que facilitam o trabalho na hora da manutenção;



Luciano Pimenta

(lucianoalmeidapimenta@gmail.com)

é Técnico em Processamento de Dados, desenvolvedor Delphi/C#. Palestrante da 4ª edição da Borland Conference (BorCon). Autor de mais de 50 artigos e de mais de 270 vídeo aulas publicadas em revistas e sites especializados. Atualmente é programador Web da FullSoft - Mobile Solutions em Caxias do Sul-RS. Blog: lucianopimenta.blogspot.com. Site: www.lucianopimenta.net. É consultor da FP2 Tecnologia (www.fp2.com.br).

dos em uma *unit* e usado em qualquer aplicação *Win32*.

Um exemplo simples que sintetiza bem o reaproveitamento de código é a herança visual de formulários, onde é criado um formulário base, com código comum e não precisamos redeclarar o mesmo em cada formulário. Temos vários exemplos de uso de herança visual, então não mostrarei aqui como fazer essa técnica.

O primeiro exemplo de reaproveitamento de código é bastante usado pelos desenvolvedores, a criação de formulários. O código mais comum de usar para criação de um formulário é o seguinte:

```
Application.CreateForm(TfrmClientes,
frmClientes);
try
  frmClientes.ShowModal;
finally
  frmClientes.Release;
end;
```

Uma aplicação pode ter vários formulários, imagine esse código duplicado para cada formulário, sendo que a diferença entre eles é somente o formulário que será aberto. Para criar um método comum de criação de formulário, use o código da **Listagem 1**.

Agora, para chamar qualquer formulário, vamos usar apenas uma linha de código:

```
AbreFormulario(TfrmCliente, frmCliente);
```

Simple e prático. Claro, se você não quiser abrir o formulário com o *ShowModal*, deverá adaptar o código.

Auto-incremento

Quem utiliza *InterBase/Firebird*, sabe que não temos campos auto-incremento no banco. Para simular isso, podemos usar *Generator* ou na versão mais nova, o *Sequence*. Muitos desenvolvedores, por diversas razões, necessitam do valor do código atual quando salvar o registro na aplicação Delphi.

Para ter esse valor atualizado, precisamos refazer a consulta ao banco e atualizar os dados do registro em tela. Nem sempre isso é a melhor maneira, portanto podemos ter um código que pega o valor do *Generator* no banco. Use o código da **Listagem 2** para isso.

A função que, retorna um inteiro, rece-

Listagem 1. Código para abrir o formulário

```
procedure AbreFormulario(aClasseForm: TComponentClass;
aForm: TForm);
begin
  Application.CreateForm(aClasseForm, aForm);
  try
    aForm.ShowModal;
  finally
    aForm.Release;
  end;
end;
```

Listagem 2. Pegando o valor do Generator

```
function GeneratorID (aName: string;
Connection: TSQLConnection;
Incrementa: Boolean): integer;
var
  Qry: TSQLQuery;
begin
  Qry := TSQLQuery.Create(nil);
  try
    Qry.SQLConnection := Connection;
    if Incrementa then
      Qry.SQL.Add(
        'SELECT GEN_ID('+aName+', 1) FROM RDB$DATABASE')
    else
      Qry.SQL.Add(
        'SELECT GEN_ID('+aName+', 0) FROM RDB$DATABASE');
    Qry.Open;
    Result := Qry.Fields[0].AsInteger;
  finally
    FreeAndNil(Qry);
  end;
end;
```

Listagem 3. Habilitando/desabilitando controles de tela

```
procedure HabilitaDesabilitaControles(Value: Boolean);
var
  i : integer;
begin
  for i := 0 to ComponentCount - 1 do
    if (Components[i] is TControl) then
      (Components[i] as TControl).Enabled := Value;
end;
```

be como parâmetro o nome do *Generator* e o *SQLConnection*, componente para conexão com o banco. Você pode alterar esse parâmetro para o componente que esteja usando. Por fim, temos um parâmetro que indica se vamos incrementar o *Generator* (1 ou 0).

Assim, podemos verificar o valor do *Generator* e incrementar o mesmo, ou apenas saber o valor atual do mesmo. Com esse código, basta usar o mesmo no evento *OnNewRecord* do *DataSet*, indicando o seu valor para o *TField*.

Habilitar/Desabilitar controles de tela

Outro código bastante usado em aplicações é para habilitar/desabilitar controles de tela. O código é simples conforme temos na **Listagem 3**.

Fizemos um laço pelos controles em tela e verificamos se o mesmo descende de *TControl*, assim repassamos para a

propriedade *Enabled* (usando *cast*) o valor passado como parâmetro (*True* ou *False*). Caso deseje verificar algum tipo específico, basta adicionar mais um *if*.

Nota: O código é mais utilizado em formulários, mas caso deseje implementar o código em um *unit* em separado e apenas chamar o mesmo em cada formulário, basta adicionar mais um parâmetro do tipo *TForm* para poder utilizar o *ComponentCount*.

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

Accesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Everson Borges Volcao que mostra como trabalhar com herança visual de formulários. <http://www.devmedia.com.br/articles/visualizacomponente2.asp?comp=250p>

www.devmedia.com.br/articles/viewcomp.asp?comp=1716&hl=*heran%E7a*

Trabalhando com strings

No dia-a-dia do desenvolvimento, precisamos trabalhar com *strings* para, por exemplo, remover um caractere específico, remover acentos de um texto entre outros. Vamos então criar métodos para esses exemplos e podermos usar o código em toda a aplicação.

Começaremos com o exemplo de remover um caractere de um texto. Esse exemplo é utilizado quando precisamos remover um “-” ou qualquer outro caractere do texto. Vamos usar o código, onde

passamos o caractere que queremos remover do texto (segundo parâmetro). O código retorna o texto sem a *string*, como podemos ver na **Listagem 4**.

No código, procuramos a posição do caractere no texto, usando o *Pos*. Com a posição, fizemos um laço e removemos o caractere do texto. Para testar, vamos adicionar dois *Edits* e um *Button* em um formulário. No primeiro vamos escrever o texto com o caractere que vamos remover.

No *Button* digite o seguinte código:

```
Edit2.Text := RemoveChar('-', Edit1.Text);
```

Listagem 4. Removendo um caractere do texto

```
function RemoveChar (const Ch: Char;
const S: string): string;
var
  Posicao: integer;
begin
  Result := S;
  Posicao := Pos(Ch, Result);
  while Posicao > 0 do
  begin
    Delete(Result, Posicao, 1);
    Posicao := Pos(Ch, Result);
  end;
end;
```

Listagem 5. Removendo letras acentuadas

```
function RemoveAcento(Str: string): string;
const
  ComAcento = 'ãäèðòãöäéíóúçüååêöôåöäéíóúçü';
  SemAcento = 'aeouaoaeioucuAAEOUAAOEIOUCU';
var
  x: Integer;
begin
  for x := 1 to Length(Str) do
  if Pos(Str[x], ComAcento) <> 0 then
    Str[x] := SemAcento[Pos(Str[x], ComAcento)];
  Result := Str;
end;
```

Listagem 6. Gerador de senhas aleatórias

```
function GeraSenha (aQuant: integer): string;
var
  i: integer;
const
  str = '1234567890ABCDEFHGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz';
begin
  for i:= 1 to aQuant do
  begin
    Randomize;
    Result := Result + str[Random(Length(str))+1];
  end;
end;
```

Listagem 7. Abrindo uma janela popup usando JavaScript

```
procedure OpenWindow (aPage: System.Web.UI.Page;
aRedirect, aWidth, aHeight: string);
var
  aScript: String;
begin
  { abre uma nova janela do browser }
  aScript := StringBuilder.Create;
  aScript.Append('<script language="JavaScript">');
  aScript.Append('window.open("'+ aRedirect +
  ", "", "resizable=no, menubar=no, scrollbars=yes,
  status=yes, left=350, top=150, width='+ aWidth +
  ', height='+ aHeight + '")');
  aScript.Append('</script>');
  if not aPage.IsClientScriptBlockRegistered(
  'client') then
    aPage.RegisterClientScriptBlock('client',
    aScript.ToString);
end;
```

Com o código, vamos remover todas as “-” que estiverem no texto. Note que estamos apenas removendo e não substituindo. Veja na **Figura 1** o exemplo em execução.

Removendo letras acentuadas

Neste exemplo, vamos criar uma função que retorna o texto passado como parâmetro sem letras acentuadas. O código é o da **Listagem 5**.

No código temos as letras acentuadas da língua portuguesa, como também as mesmas sem acento, sendo preenchidas como constantes. Assim, fizemos um laço pegando o tamanho do texto como parâmetro e verificamos se existe a letra acentuada no texto e substituímos pela sem acento.

Veja na **Figura 2** o exemplo em execução, onde usamos a mesma técnica do exemplo anterior.

Gerando senhas aleatórias

Em alguns sistemas quando criamos o usuário, precisamos indicar uma senha aleatória gerada pelo sistema. Para isso, use o código da **Listagem 6**.

No código anterior, usamos *Random* e *Randomize* para pegar aleatoriamente a letra da constante (com letras maiúsculas, minúsculas e números) para criar

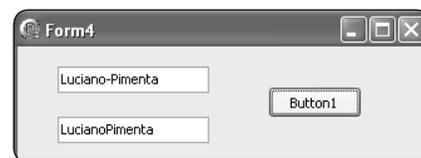


Figura 1. Removendo caractere de um texto

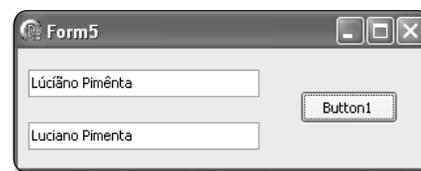


Figura 2. Substituindo letras acentuadas

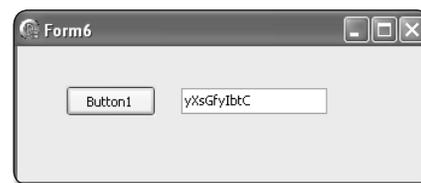


Figura 3. Gerador de senhas

a senha. Passamos como parâmetro a quantidade de caracteres que a senha deve ter.

Se quiser, você pode adicionar caracteres especiais para que a senha fique mais segura. Veja na **Figura 3** o exemplo em execução.

Trabalhando com Java Script

Para *ASP.NET* também podemos criar nosso códigos personalizados. Os mais comuns são códigos para trabalhar com *JavaScript* para abrir formulários, executar código e mostrar mensagens em tela, mas também podemos desabilitar e limpar controles como na aplicação Win32.

Podemos abrir páginas *pop-up* utilizando *JavaScript*, assim reaproveitando o código em uma *unit*, não precisamos duplicar o código em cada página. O código para chamar uma página *ASPX* com *popup* está na **Listagem 7**.

No código, passamos como parâmetro o objeto *Page*, o nome da página que queremos abrir e o tamanho da janela. Você pode criar mais parâmetros para parametrizar ainda mais o método. Criamos uma variável do tipo *StringBuilder*, que é mais otimizada para concatenar *strings*.

Adicionamos na *StringBuilder* a declaração de uma função *JavaScript* (*tags*) chamando o *open* do objeto *window*. Passamos como parâmetros as opções existentes, que você pode conhecer em: [http://msdn.microsoft.com/en-us/library/ms536651\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms536651(VS.85).aspx).

Listagem 8. Executando scripts na Web

```
procedure ExecuteScript(aPage: System.Web.UI.Page;
  aScript: string);
var
  Script: StringBuilder;
begin
  Script := StringBuilder.Create;
  Script.Append('<script language="JavaScript">');
  Script.Append(aScript);
  Script.Append('</script>');
  if not aPage.IsClientScriptBlockRegistered(
    'client') then
    aPage.RegisterClientScriptBlock('client',
      Script.ToString);
end;
```

Listagem 9. Caixas de Mensagens com JavaScript

```
procedure MessageScript(aPage: System.Web.UI.Page;
  aTexto: string);
var
  Script: StringBuilder;
begin
  Script := StringBuilder.Create;
  Script.Append('<script language="JavaScript">');
  Script.Append('alert('+ aTexto + ');');
  Script.Append('</script>');
  if not aPage.IsClientScriptBlockRegistered(
    'client') then
    aPage.RegisterClientScriptBlock('client',
      Script.ToString);
end;
```

Listagem 10. Habilitando/desabilitando controles ASP.NET

```
procedure EnableDisableControls (
  aControls: ControlCollection;
  aValue: Boolean);
var
  i, y: integer;
begin
  [ habilita e desabilita controles ]
  for i := 0 to aControls.Count - 1 do
    begin
      [ controles de tela esta dentro de HTMLForm ]

      if (aControls.Item[i] is HtmlForm) then
        begin
          for y := 0 to aControls.Item[i].Controls.Count - 1 do
            if (aControls.Item[i].Controls.Item[y] is WebControl) then
              (aControls.Item[i].Controls.Item[y] as WebControl).Enabled := aValue;
          end;
        end;
      end;
    end;
end;
```

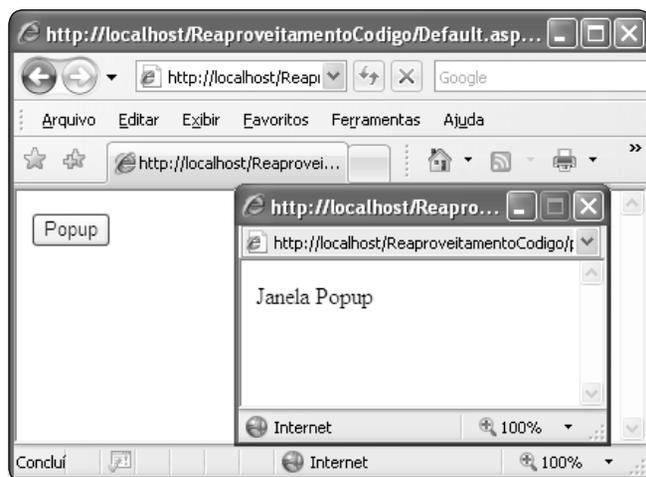


Figura 4. Abrindo janelas popup no ASP.NET

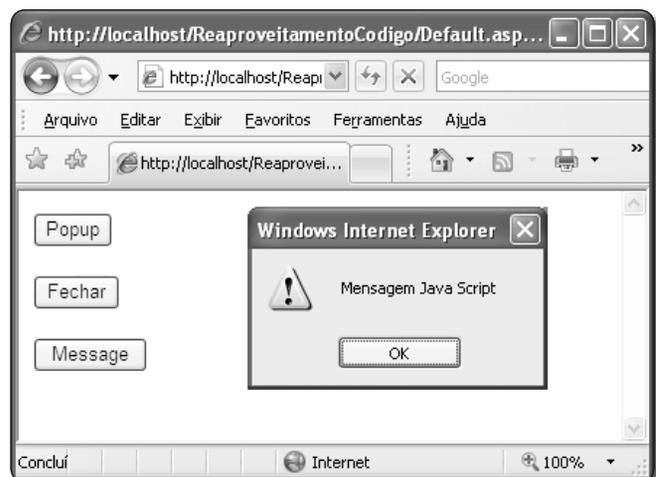


Figura 5. Caixas de mensagens no ASP.NET com JavaScript

Listagem 11. Limpando controles de tela no ASP.NET

```
procedure ClearControls (aControls: ControlCollection);
var
  i, y: integer;
begin
  { limpa controles de tela }
  for i := 0 to aControls.Count - 1 do
  begin
    if (aControls.Item[i] is HtmlForm) then
    begin
      for y := 0 to aControls.Item[i].Controls.Count - 1 do
      begin
        { controles mais comuns }
        if (aControls.Item[i].Controls.Item[y] is TextBox) then
          (aControls.Item[i].Controls.Item[y] as TextBox).Text := '';
        if (aControls.Item[i].Controls.Item[y] is RadioButtonList) then
          (aControls.Item[i].Controls.Item[y] as RadioButtonList).SelectedIndex := -1;
        if (aControls.Item[i].Controls.Item[y] is DropDownList) then
          (aControls.Item[i].Controls.Item[y] as DropDownList).SelectedIndex := -1;
        if (aControls.Item[i].Controls.Item[y] is ListBox) then
          (aControls.Item[i].Controls.Item[y] as ListBox).SelectedIndex := -1;
      end;
    end;
  end;
end;
```

Listagem 12. Criptografando textos

```
uses System.Web.Security;
...
function EncryptaSenha (aSenha: string): string;
begin
  Result := FormsAuthentication.HashPasswordForStoringInConfigFile(
    aSenha, 'MD5');
end;
```

Por fim, registramos o bloco de código chamando *RegisterClientScriptBlock* do objeto *Page*. Em uma aplicação ASP.NET use o código e teremos o exemplo da Figura 4.

Executando JavaScript

E se quisermos, por exemplo, executar determinado código *JavaScript* como fechar um formulário (browser)? Podemos criar um novo método, adaptado do exemplo anterior, usando o código da Listagem 8.

Veja que o código é bastante semelhante ao anterior, com a diferença que passamos o parâmetro *aScript* no meio da tag *<script>*. Para executar o código, precisamos apenas usar:

```
ExecuteScript(Page, 'window.close()');
```

Mensagens com JavaScript

Podemos usar o método anterior para usar o *alert* do *JavaScript*, mas também podemos criar um método específico para a caixa de mensagem. Veja o código da Listagem 9.

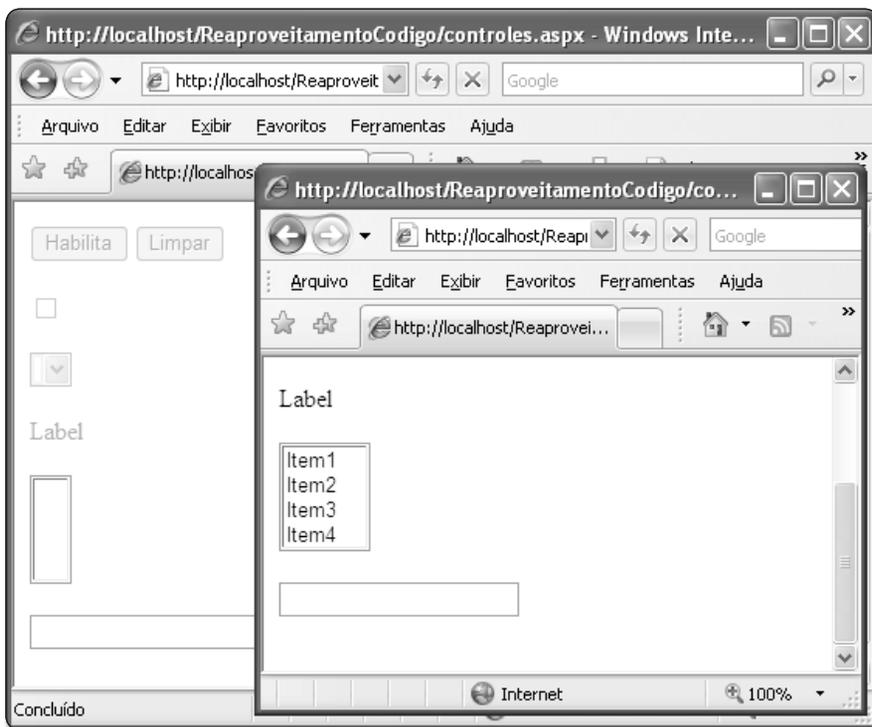
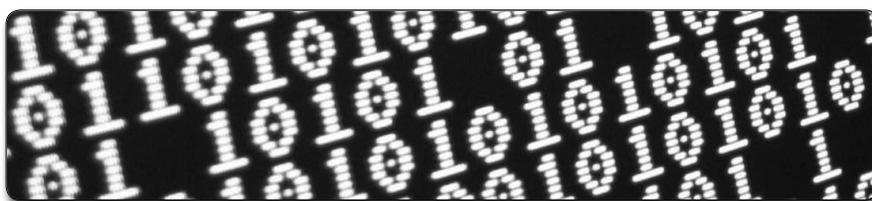


Figura 6. Trabalhando com controles de tela



Nota do DevMan

O MD5 (Message-Digest algorithm 5) é um algoritmo de *hash* de 128 bits unidirecional desenvolvido pela *RSA Data Security, Inc.*, descrito na *RFC 1321*, e muito utilizado por softwares com protocolo *ponto-a-ponto* (*P2P*, ou *Peer-to-Peer*, em inglês), verificação de integridade e logins.

Foi desenvolvido em 1991 por Ronald Rivest para suceder ao MD4 que tinha alguns problemas de segurança. Por ser um algoritmo unidirecional, uma *hash md5* não pode ser transformada novamente no texto que lhe deu origem. O método de verificação é, então, feito pela comparação das duas *hash* (uma da base de dados, e a outra da tentativa de login). O MD5 também é usado para verificar a integridade de um arquivo através, por exemplo, do programa *md5sum*, que cria a *hash* de um arquivo. Isso pode se tornar muito útil para *downloads* de arquivos grandes, para programas *P2P* que constroem arquivos através de porções e estão sujeitos à corrupção dos mesmos. Como autenticação de login é utilizada em vários sistemas operacionais *unix* e em muitos sites com autenticação.

Para testar, basta usar:

```
MessageScript(Page, 'Mensagem Java Script');
```

Veja o exemplo em execução na **Figura 5**.

Trabalhando com controles de tela

Podemos criar métodos para, por exemplo, habilitar/desabilitar e limpar controles de tela. Para habilitar/desabilitar use o código da **Listagem 10**.

Veja que precisamos percorrer a coleção de controles, onde passaremos a coleção de controles do objeto *Page*. Para poder verificar os *WebControls* precisamos achar o tipo *HtmlForm* e depois verificar os *WebControls*.

Para limpar controles de tela (*TextBoxs*, *RadioButtonList*, *DropDownList*, *ListBox* etc.), usamos um código bastante semelhante, conforme a **Listagem 11**.

Para usar os métodos, use o seguinte código:

```
{ limpar controles de telas }
ClearControls(Page.Controls);
{ habilitar/desabilitar controles }
EnableDisableControls(Page.Controls, false);
```

Veja na **Figura 6** o exemplo em execução.

Criptografia

Para criptografar textos no *ASP.NET* o framework possui métodos prontos para usar, portanto podemos usar esse método para nossa biblioteca. Para isso, basta usarmos o código da **Listagem 12**.

O *HashPasswordForStoringInConfigFile* retorna uma *string* criptografada com o

formato passado como parâmetro (MD5), onde podemos ainda ter o formato SHA1. Uma característica interessante é que não podemos “criptografar” a *string*.

O método é indicado para armazenar dados criptografados em arquivos de configuração, mas podemos usá-los para criptografar senhas por exemplo. Veja na **Figura 7** o texto criptografado.

Conclusão

Vimos neste artigo, como reaproveitar código *Win32* e *ASP.NET* usando uma biblioteca para que os métodos possam ser usados em toda a aplicação ou até em várias aplicações. Alguns exemplos *Win32* foram desenvolvidos por mim e outros encontrados na internet.

Os exemplos não continham a indicação do autor, por isso não pudemos mencionar o mesmo nos exemplos. Os exemplos em *ASP.NET* foram desenvolvidos por mim e podem ser usados em suas aplicações Web.

Um grande abraço a todos e sucesso em seus projetos! ●

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

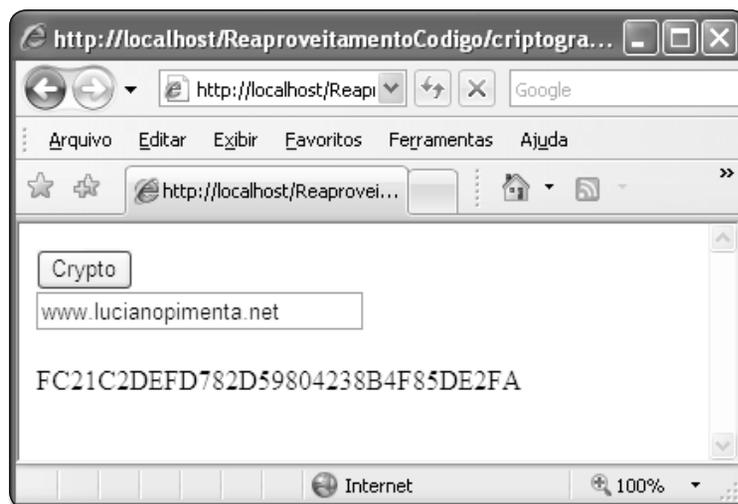


Figura 7. Criptografando textos com o formato MD5

