

# Capítulo 8

## Enviando E-mails


O recurso mais utilizado na Internet é sem dúvida o e-mail. Neste capítulo iremos tratar deste assunto com extrema facilidade.

O Delphi 7 traz incorporado em sua biblioteca de componentes o famoso pacote INDY da empresa Nevrona ([www.nevrona.com](http://www.nevrona.com)). Na versão 5, o Delphi trazia o pacote da empresa NETMasters que utilizava API do Windows para executar suas funções, e desde a versão 6, foi incluído o pacote INDY. O pacote da NetMasters era muito instável, devido aos inúmeros bugs tanto da parte da NETMasters, como também do Windows. A Nevrona adaptou seu excelente pacote (INDY) para o Kylix, baseado na tecnologia CLX. Com isso temos um excelente desempenho devido à engenharia do pacote.


Neste capítulo iremos desenvolver um aplicativo para o envio de e-mails, utilizando os novos componentes da Nevrona. Este aplicativo está registrado na SourceForge com o nome de Mailing.NET em <http://sourceforge.net/projects/mailingnet>. A SourceForge é uma entidade responsável pelo gerenciamento de aplicações com código-fonte aberto, distribuídos sob a licença pública GNU. Um fator bastante importante deste projeto é que o mesmo poderá ser compilado em Kylix, pelo fato de ser baseado na tecnologia CLX.

### Início do Desenvolvimento


Vamos iniciar um novo projeto CLX, e no formulário principal alterar as seguintes propriedades.

OBJETO		
	TForm	
Objeto	Propriedade	Valor
fmeMail	BorderWidth	5
	Caption	Mailing.Net
	Name	fmeMail


Vamos gravar a unit com o nome *f\_principal.pas* e o projeto como *MailingNet.DPR*. Insira um objeto do tipo *TPanel* e altere as propriedades que seguem:

OBJETO		
	TPanel	
Objeto	Propriedade	Valor
PnConfigura	Align	alTop
	Caption	((deixe em branco))
	Height	180
	Name	PnConfigura

Agora com o foco no formulário (objeto fmEmail), insira um objeto do tipo *TSplitter* e altere as seguintes propriedades:


OBJETO		
	TSplitter	
Objeto	Propriedade	Valor
Splitter1	Align	alTop
	MinSize	30

O objeto *TSplitter* é utilizado para dividir seções de um formulário. Em nosso projeto, terá a função de separar as áreas de configuração e texto. Ainda com o foco no formulário insira outro objeto do tipo *TPanel* e altere as propriedades, como a seguir:

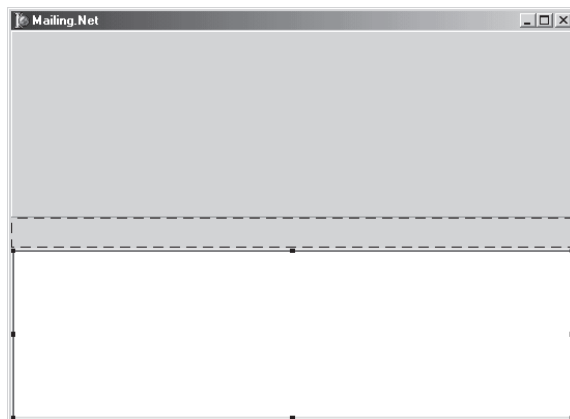
OBJETO		
	TPanel	
Objeto	Propriedade	Valor
PnTexto	Align	alClient
	Caption	((((deixe em branco))))
	Name	PnTexto

Repare que estamos na fase de desenvolvimento da interface do aplicativo. Claro que é uma tarefa um pouco cansativa devido ao grande número de objetos do formulário. Mas amigos, acreditem, vale a pena o pequeno esforço.

Agora com o foco no objeto *PnTexto*, insira um objeto do tipo *TMemo* e altere as propriedades que seguem:


OBJETO		
	TMemo	
Objeto	Propriedade	Valor
textodoemail	Align	alClient
	Lines	((((deixe em branco))))
	Name	TextodoEmail


Vamos dar uma pausa e observar como está ficando nosso aplicativo. A *figura 8.1* ilustra esse “grande” momento.





**Figura 8.1** Fase inicial do Mailing.Net


Dando continuidade ao nosso projeto, vamos inserir os objetos que seguem dentro do painel *PnConfigura*. Para facilitar, colocamos em destaque a seção em que se encontra o referido objeto. Exemplo: *[Standard]*.


OBJETO		
	TLabel <i>[Standard]</i>	
Objeto	Propriedade	Valor
	Caption	Texto
	Left	17
	Top	19

OBJETO		
	TLabel <i>[Standard]</i>	
Objeto	Propriedade	Valor
	Caption	Assunto
	Left	17
	Top	51


OBJETO		
	TEdit <i>[Standard]</i>	
Objeto	Propriedade	Valor
	Left	96
	Name	edTexto
	Text	(((deixe em branco)))
	Top	16
	Width	320

OBJETO		
	<b>TEdit [Standard]</b>	
Objeto	Propriedade	Valor
	Left	96
	Name	edAssunto
	Text	(((deixe em branco)))
	Top	46
	Width	320


OBJETO		
	<b>TSpeedButton [Additional]</b>	
Objeto	Propriedade	Valor
	Caption	>>
	Flat	True
	Left	420
	Name	btArquivo
	Top	16
	Width	23


OBJETO		
	<b>TButton [Standard]</b>	
Objeto	Propriedade	Valor
	Caption	Envia
	Left	468
	Name	btEnvia
	Top	85
	Width	75


Ainda com o foco no objeto *PnConfigura*, insira um objeto do tipo *TGroupBox* e altere as propriedades que seguem:


OBJETO		
	<b>TGroupBox [Standard]</b>	
Objeto	Propriedade	Valor
	Caption	Configuração Servidor SMTP
	Height	90
	Left	16
	Name	gbConfigura
	Top	80
	Width	425


Agora vamos suar a camisa e inserir alguns objetos dentro do *container GbConfigura*. Vamos ao batalhão de objetos do *container GbConfigura*.


OBJETO		
	TLabel [Standard]	
Objeto	Propriedade	Valor
	Caption	Host
	Left	16
	Top	20

OBJETO		
	TLabel [Standard]	
Objeto	Propriedade	Valor
	Caption	Usuário
	Left	16
	Top	44

OBJETO		
	TLabel [Standard]	
Objeto	Propriedade	Valor
	Caption	Conta
	Left	16
	Top	69


OBJETO		
	TLabel [Standard]	
Objeto	Propriedade	Valor
	Caption	Porta
	Left	327
	Top	20

OBJETO		
	TLabel [Standard]	
Objeto	Propriedade	Valor
	Caption	Senha
	Left	176
	Top	44


OBJETO		
	TEdit [Standard]	
Objeto	Propriedade	Valor
	Color	clAqua
	Left	63

	Name	edHost
	Text	coloque o endereço do seu servidor SMTP. Exemplo: smtp.provedor.com.br
	Top	16
	Width	250


**OBJETO**


	<b>TEdit [Standard]</b>	
<b>Objeto</b>	<b>Propriedade</b>	<b>Valor</b>
	Color	clAqua
	Left	63
	Name	edUsuario
	Text	coloque o usuário de sua conta SMTP. Exemplo: seunome
	Top	40
	Width	106

**OBJETO**

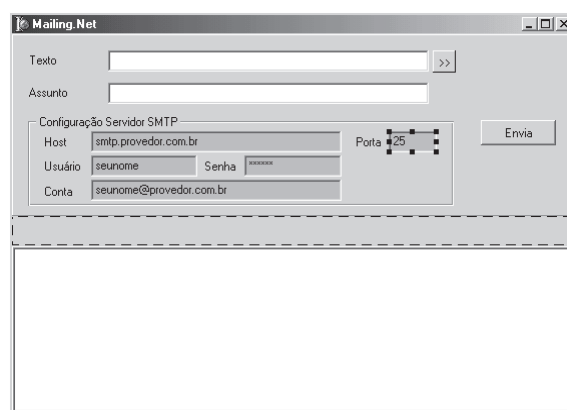
	<b>TMaskEdit [Additional]</b>	
<b>Objeto</b>	<b>Propriedade</b>	<b>Valor</b>
	Color	clAqua
	Left	216
	PassWordChar	*
	Name	edSenha
	Text	coloque a senha do usuário da conta de SMTP. Exemplo: suasenha
	Top	40
	Width	97

**OBJETO**

	<b>TEdit [Standard]</b>	
<b>Objeto</b>	<b>Propriedade</b>	<b>Valor</b>
	Color	clAqua
	Left	63
	Name	edConta
	Text	coloque o endereço da conta de SMTP. Exemplo: seunome@provedor.com.br
	Top	16
	Width	250

OBJETO		
	TMaskEdit [Additional]	
Objeto	Propriedade	Valor
	Color	clAqua
	EditMask	9999;1;
	Left	368
	Name	edPorta
	Text	25
	Top	16
	Width	48

Puxa, ainda bem que esta parte acabou. Acredito que todos se cansaram, mas temos um belo formulário para envio de e-mails. A *figura 8.2* ilustra o nosso formulário neste momento.




*Figura 8.2 Fase final do design do formulário*

## Detalhes do Projeto


Antes de prosseguirmos com o projeto, vamos conhecê-lo melhor. O Mailing.Net utiliza como base o mesmo banco de dados no padrão *Interbase* empregado no capítulo 7. Na realidade utilizamos apenas dois campos da tabela de clientes: *razao\_social* e *email*. O campo *razao\_social* será utilizado para substituir uma *tag* em nosso aplicativo e o campo *email* para compor o destinatário.

O mais interessante neste aplicativo é o uso do objeto *TDataSetPageProducer*, normalmente utilizado no desenvolvimento de aplicações Web, como vimos nos capítulos anteriores. Além disso, iremos utilizar um arquivo externo no padrão HTML para relacionar com o objeto *TMemo*.

Insira um objeto do tipo *TSQLConnection*, e crie uma nova conexão clicando no botão +. Altere as seguintes propriedades.

OBJETO		
	TSQLConnection	
Objeto	Propriedade	Valor
BancoDados	DriverName	Interbase
	CommitRetaining	True
	DataBase	localhost:C:/cursoweb/ /clientes.gdb
	SQLDialect	3
	LoginPrompt	False
	Name	BancoDados


Insira um objeto do tipo *TSQLDataSet* e altere as propriedades que seguem:

OBJETO		
	TSQLDataSet	
Objeto	Propriedade	Valor
tbClientes	Name	tbClientes
	SQLConnection	BancoDados
	CommandText	select * from tbclientes

### Codificando o botão *btArquivo*

O objeto *btArquivo* será responsável por carregar um arquivo externo (HTML) no objeto *TextodoEmail*. Para nos auxiliar nesta tarefa, precisamos de outro objeto: *TOpenDialog*.

Insira um objeto do tipo *TOpenDialog* e altere as propriedades que seguem:

OBJETO		
	TOpenDialog	
Objeto	Propriedade	Valor
AbrirArquivo	Name	AbrirArquivo

Agora vamos inserir o código do botão *btArquivo* no evento *OnClick* do mesmo.

```
procedure TfmEmail.btArquivoClick(Sender: TObject);
begin
    if AbrirArquivo.Execute then
    begin
        EdTexto:=AbrirArquivo.FileName;

        // Zera O TextodoEmail
        TextodoEmail.Clear;

        // Associa o Arquivo Texto/HTML com o
        // objeto TextodoEmail

        TextodoEmail.Lines.LoadFromFile(edTexto.Text);
    end;
```



```
end;
```

Vamos estudar o código em detalhe:

```
if AbrirArquivo.Execute then
```

Esta linha verifica se o método *Execute* do objeto *AbrirArquivo* foi executado com êxito, ou seja, indaga se o usuário selecionou algum arquivo.

```
EdTexto.Text:=AbrirArquivo.FileName;
```

Associa o nome do arquivo selecionado no objeto *AbrirArquivo* ao objeto *EdTexto*.

```
TextodoEmail.Clear;
```

O método *Clear* limpa o conteúdo do objeto *TextodoEmail*.

```
TextodoEmail.Lines.LoadFromFile(edTexto.Text);
```

Aqui estamos carregando (*LoadFromFile*) o arquivo (*edTexto.Text*) dentro do objeto *TextodoEmail*. Em resumo, poderemos digitar o texto ou então carregar um arquivo previamente criado.

## Falando mais um pouco dos objetos INDY


Um dos grandes problemas da biblioteca *NetMasters* (utilizada até a versão 5 do Delphi) era o congelamento (*Freeze*) da aplicação em diversos momentos: na conexão, desconexão, entre outros, invariavelmente. A biblioteca da *Nevrona* (INDY) traz um componente que trata especificamente deste problema: *IdAntiFreeze*. Sua principal função é priorizar as tarefas da aplicação, evitando o efeito *Freeze*. Na realidade ele cria um processo, otimizando as tarefas relacionadas a objetos INDY.

Vamos inserir um objeto do tipo *TIdAntiFreeze*, localizado na seção *Indy Misc*. Não é necessário alterar as propriedades deste objeto. Agora iremos inserir o objeto considerado coração da nossa aplicação: *TIdSMTP*, localizado na seção *Indy Clients*.

Insira um objeto do tipo *TIdSMTP* em nosso projeto. Iremos codificar as propriedades deste objeto em tempo de execução. O objeto *TIdSMTP* realiza a conexão da nossa aplicação *Cliente* com o servidor SMTP, responsável pela entrega dos e-mails.

Na realidade, nosso aplicativo apenas envia os e-mails para um servidor SMTP e o mesmo se encarrega de distribuí-los. Poderíamos criar um servidor SMTP próprio que enviaria diretamente os e-mails, mas devido à complexidade do caso, ficaremos com o exemplo mais simples. Vamos continuar com o nosso projeto.


Insira um objeto do tipo *TIdMessage*, localizado na seção *Indy Misc*, e altere as propriedades que seguem:

OBJETO		
 <b>TIdMessage</b>		
Objeto	Propriedade	Valor
Mensagem	ContentType	text/HTML
	Name	Mensagem

Repare que a propriedade *ContentType* refere-se ao tipo *MIME* da mensagem, ou seja, o tipo de seu conteúdo. Configuramos para text/HTML para que as mensagens sejam enviadas com o padrão HTML ou texto (o padrão HTML predomina por conter recursos de formatação).

Agora, amigos, iremos trabalhar com o nosso “curinga”: *TDataSetPageProducer*. Como visto nos capítulos anteriores, o objeto *TDataSetPageProducer* é responsável pela substituição automática de *Tags transparentes* relacionadas aos campos do *DataSet* vinculado.

Insira um componente do tipo *TDataSetPageProducer* e altere as propriedades que seguem:

OBJETO		
	TDataSetPageProducer	
Objeto	Propriedade	Valor
ppMensagem	DataSet	tbClientes
	Name	ppMensagem

Vejamos alguns exemplos da funcionalidade do nosso “curinga”.

```
Prezado(a) <#RAZAO_SOCIAL>,

Estamos enviando este boletim para o seu email<BR>
<#EMAIL>.

....
```

Em tempo de execução teremos:

```
Prezado(a) Emerson Facunte,

Estamos enviando este boletim para o seu email
emerson@facunte.com.br
```

O interessante é que podemos inserir quantas Tags forem necessárias, em qualquer posição. Vejamos outro exemplo:

```
Prezado(a) <#RAZAO_SOCIAL>,

Como cliente preferencial, você tem 10% de desconto em qualquer produto de nossa
loja.

Entretanto, Sr(a) <#RAZAO_SOCIAL>, esta oferta é por tempo limitado.
```

Amigos, como observamos nos capítulos anteriores, poderemos produzir documentos HTML com ferramentas de terceiros, como DreamWeaver da Macromedia. Com isso teremos um e-mail mais elegante.

Bem, finalmente vamos codificar o nosso glorioso botão *btEnvia*. Insira o código que segue no evento *OnClick* do objeto *btEnvia*.

```
procedure TfmeMail.btEnviaClick(Sender: TObject);
begin

// Configura Cliente
idSMTP1.UserId:=edUsuario.Text;
idSMTP1.Password:=edSenha.Text;
idSMTP1.Host:=edHost.Text;
idSMTP1.Port:=StrToInt(edPorta.Text);

// Atribui o Conteudo do objeto TextodoEmail
// ao objeto PPMensagem
ppMensagem.HTMLDoc:=TextodoEmail.Lines;
```

```

// Abre a tabela Clientes
tbClientes.Open;
tbClientes.First;

// Conecta ao Servidor SMTP
idSMTP1.Connect;

try

with Mensagem do
begin
// Atribui o conteúdo do edAssunto
// ao objeto Mensagem.Subject
Subject:=edAssunto.Text;

// Atribui o conteúdo do edConta
// ao objeto Mensagem.From.Text
From.Text:=edConta.Text;

// inicia o laço
while not(tbClientes.Eof) do
begin
// Atribui o conteúdo do campo EMAIL
// ao objeto Mensagem.Recipients.EmailAddresses
Recipients.EmailAddresses:=tbClientes.FieldByName('EMAIL').Value;
ReceiptRecipient.Address:=tbClientes.FieldByName('EMAIL').Value;

// Atribui o conteúdo do objeto ppMensagem
// ao objeto Mensagem.Body.Text
Body.Text:=ppMensagem.Content;

// Envia a mensagem
idSMTP1.Send(Mensagem);

// Próximo registro
tbClientes.Next;
end; // laço
end; // with Mensagem

finally
// Disconecta Servidor
idSMTP1.Disconnect;
end;

end;

```

Vamos analisar com cuidado este código. No primeiro bloco, estamos configurando o Cliente SMTP através do nosso objeto *idSMTP1*.

```

// Configura Cliente
idSMTP1.UserId:=edUsuario.Text;
idSMTP1.Password:=edSenha.Text;
idSMTP1.Host:=edHost.Text;
idSMTP1.Port:=StrToInt(edPorta.Text);

```

Repare que estamos configurando o usuário da conta *UserId*, a senha *Password*, o *HOST* e a porta de conexão (*Port*). O bloco seguinte atribui o conteúdo do objeto *TextodaMensagem* ao objeto *PPMensagem*.

```
// Atribui o Conteudo do objeto TextodoEmail
// ao objeto PPMensagem
ppMensagem.HTMLDoc:=TextodoEmail.Lines;
```

A propriedade HTMLDoc armazena um documento para ser processado pelo próprio objeto. O bloco que segue abre a tabela de Clientes e posiciona o ponteiro no primeiro registro:

```
// Abre a tabela Clientes
tbClientes.Open;
tbClientes.First;
```

O bloco a seguir conecta o objeto *idSMTP1* ao servidor de SMTP.

```
// Conecta ao Servidor SMTP
idSMTP1.Connect;
```

Agora estamos protegendo o código seguinte com o comando *Try/Except/Finally*. Em seguida estamos atribuindo o assunto da mensagem e o responsável pelo envio da mesma.

```
with Mensagem do
begin
// Atribui o conteudo do edAssunto
// ao objeto Mensagem.Subject
Subject:=edAssunto.Text;

// Atribui o conteudo do edConta
// ao objeto Mensagem.From.Text
From.Text:=edConta.Text;
```

Como as mensagens são enviadas uma a uma, fizemos um laço percorrendo toda a tabela de Clientes.

```
while not(tbClientes.Eof) do
```

Neste ponto estamos configurando o e-mail destino.

```
// Atribui o conteudo do campo EMAIL
// ao objeto Mensagem.Recipients.EmailAddresses
Recipients.EmailAddresses:=tbClientes.FieldName('EMAIL').Value;
ReceiptRecipient.Address:=tbClientes.FieldName('EMAIL').Value;
```

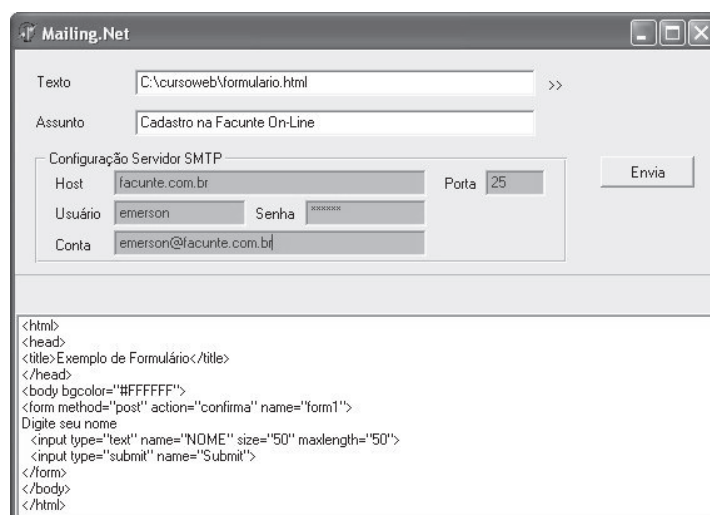
Chegamos ao ponto mais interessante do nosso projeto. Neste bloco estamos associando o resultado do *ppMensagem* ao corpo da mensagem do objeto *Mensagem*. Quando chamamos o método *Content*, ativamos o processamento das informações contidas na propriedade *HTMLDOC* do objeto.

```
// Atribui o conteúdo do objeto ppMensagem
// ao objeto Mensagem.Body.Text
Body.Text:=ppMensagem.Content;
```

E finalmente enviamos a mensagem.

```
// Envia a mensagem
idSMTP1.Send(Mensagem);
```

Bem, amigos, acredito que agora basta usar a imaginação para enviar os e-mails de maneira personalizada. Com isso concluímos nosso projeto de envio de e-mails, e de quebra refrescamos nossas idéias com um novo tipo de aplicação.



**Figura 8.3 MailingNet em ação**

### **Listagem 8.1 Código completo do Mailing Net**

```
unit f_principal;

interface

uses
  SysUtils, Types, Classes, QGraphics, QControls, QForms, QDialogs,
  QStdCtrls, QExtCtrls, QButtons, QMask, DBXpress, FMTBcd, DB, SqlExpr,
  IdBaseComponent, IdAntiFreezeBase, IdAntiFreeze, IdMessage, HTTPApp,
  HTTPProd, DSProd, IdComponent, IdTCPConnection, IdTCPClient,
  IdMessageClient, IdSMTP;

type
  TfmeMail = class(TForm)
    PnConfigura: TPanel;
    Splitter1: TSplitter;
    pnTexto: TPanel;
    TextodoEmail: TMemo;
    Label1: TLabel;
    Label2: TLabel;
    edTexto: TEdit;
    edAssunto: TEdit;
    btArquivo: TSpeedButton;
    btEnvia: TButton;
    gbConfigura: TGroupBox;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    edHost: TEdit;
    edUsuario: TEdit;
    edConta: TEdit;
    Label7: TLabel;
    edSenha: TMaskEdit;
    edPorta: TMaskEdit;
    BancoDados: TSQLConnection;
    tbClientes: TSQLDataSet;
    AbrirArquivo: TOpenDialog;
```

```

    IdAntiFreeze1: TIdAntiFreeze;
    Mensagem: TIdMessage;
    ppMensagem: TDataSetPageProducer;
    IdSMTP1: TIdSMTP;
    procedure btArquivoClick(Sender: TObject);
    procedure btEnviaClick(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    fmeMail: TfmeMail;

implementation

{$R *.xfm}

procedure TfmeMail.btArquivoClick(Sender: TObject);
begin
    if AbrirArquivo.Execute then
    begin
        EdTexto.Text:=AbrirArquivo.FileName;

        // Zera o TextodoEmail
        TextodoEmail.Clear;

        // Associa o Arquivo Texto/HTML com o objeto
        // TextodoEmail
        TextodoEmail.Lines.LoadFromFile(edTexto.Text);
    end;
end;

procedure TfmeMail.btEnviaClick(Sender: TObject);
begin
    // Configura Cliente
    idSMTP1.UserId:=edUsuario.Text;
    idSMTP1.Password:=edSenha.Text;
    idSMTP1.Host:=edHost.Text;
    idSMTP1.Port:=StrToInt(edPorta.Text);

    // Atribui o Conteudo do objeto TextodoEmail
    // ao objeto PPMensagem
    ppMensagem.HTMLDoc:=TextodoEmail.Lines;

    // Abre a tabela Clientes
    tbClientes.Open;
    tbClientes.First;

    // Conecta ao Servidor SMTP
    idSMTP1.Connect;

try
with Mensagem do

```

```
begin
// Atribui o conteudo do edAssunto
// ao objeto Mensagem.Subject
Subject:=edAssunto.Text;

// Atribui o conteudo do edConta
// ao objeto Mensagem.From.Text
From.Text:=edConta.Text;

// inicia o laço
while not(tbClientes.Eof) do
begin
  // Atribui o conteudo do campo EMAIL
  // ao objeto Mensagem.Recipients.EmailAddresses
  Recipients.EmailAddresses:=tbClientes.FieldName('EMAIL').Value;
  ReceiptRecipient.Address:=tbClientes.FieldName('EMAIL').Value;

  // Atribui o conteúdo do objeto ppMensagem
  // ao objeto Mensagem.Body.Text
  Body.Text:=ppMensagem.Content;

  // Envia a mensagem
  idSMTP1.Send(Mensagem);

  // Próximo registro
  tbClientes.Next;
end; // laço
end; // with Mensagem

finally
  // Disconecta Servidor
  idSMTP1.Disconnect;
end;

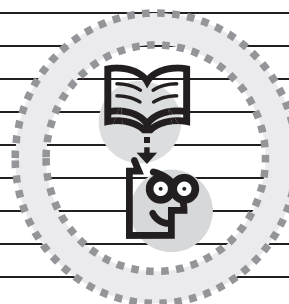
end;

end.
```

**Anotações de Dúvidas**



**Preciso Revisar**



**Anotações Gerais**

