Capítulo 10
Envio de Mensagens (E-mails)

Introdução

É muito comum todo e qualquer tipo de aplicação enviar e-mails para satisfazer um determinado processo ou notificar alguém de que uma condição foi alcançada. Felizmente, o .NET Framework fornece intrinsicamente, sem a necessidade de utilizar componentes de terceiros, um conjunto de classes que podem ser utilizadas para construir e enviar e-mails.

Nas versões 1.x do .NET Framework, as classes relacionado ao envio de e-mails estavam contidas em uma *namespace* chamado **System.Web.Mail**, dentro do *Assembly* **System.Web.dll**. Como envio de e-mails não é uma exclusividade de aplicações Web, isso ficou um pouco confuso e ainda, é necessário fazermos a referência ao *Assembly* **System.Web.dll** em uma aplicação Windows Forms se lá quisermos enviar e-mails. Definitivamente isso não faz sentido.

Na versão 2.0 do .NET Framework isso foi mudado e agora essas classes estão contidas dentro do *namespace* **System.Net.Mail**, prontas para serem utilizadas. Essas classes fornecem toda a infraestrutura para a criação de e-mails, possibilidade de anexar vários destinatários (inclusive em cópia carbono), anexar arquivos e embutir arquivos (imagens) no corpo da mensagem, utilizadas para compor a mensagem. Além da criação, ainda temos uma classe importante, chamada de **SmtpClient**, que encapsula todo o processo de envio da mensagem. Através deste capítulo, analisaremos as principais classes e como proceder para criar e enviar e-mail a partir de aplicações .NET.

Criando um Email

Para que possamos construir um e-mail precisamos utilizar a classe **MailMessage**. Como o próprio nome diz, essa classe representa um e-mail, contendo os arquivos em anexo, o remetente, destinatário, assunto, corpo, etc.. A instância desta classe pode ser passada para o método *Send* da classe **SmtpClient** para que possa definitivamente enviar o e-mail ao destinatário. Analisaremos essa classe com mais detalhes nas próximas seções.

Para familiarizarmos melhor com a classe **MailMessage**, a tabela abaixo mostra as principais propriedades que ela expõe e que podemos utilizar para configurá-la:

Propriedade	Descrição
AlternateViews	Esta coleção representa cópias de um mesmo e-mail em diferentes formatos, ou seja, você pode ter uma versão do e-mail em HTML e uma versão do e-mail em texto puro, para aqueles clientes que não conseguem visualizar o conteúdo da mensagem em formato HTML.
Attachments	Uma coleção de elementos do tipo **Attachment** que

	indica um determinado arquivo que será anexado à mensagem.
Bcc	Uma coleção de elementos do tipo **MailAddress** que indica os endereços que receberão uma cópia do e-mail, mas ficarão ocultos. Os destinatários colocados nesta seção não são visualizados pelos receptores do e-mail.
Body	Uma *string* contendo o corpo da mensagem. Essa *string* poderá conter tags HTML se o corpo do e-mail for criado baseando-se em HTML.
BodyEncoding	Define o *encoding* do corpo do e-mail.
CC	Uma coleção de elementos do tipo **MailAddress** que indica os endereços que estão copiados no e-mail e que, conseqüentemente, receberão uma cópia do e-mail.
DeliveryNotificationsOptions	Especifica se uma notificação deverá ser enviado ao remetente do e-mail. Essa propriedade é definida com uma das opções especificados pelo enumerador **DeliveryNotificationOptions**, que fornece os seguintes valores: • **Delay –** Notifica se a entrega está atrasada. • **Never –** Nunca notifica. • **None –** Sem notificação. • **OnFailure –** Notifica se a entrega falhou. • **OnSuccess –** Notifica se a entrega foi feita com sucesso.
From	Recebe um objeto do tipo **MailAddress** contendo as informações a respeito do remetente da mensagem.
Headers	Trata-se de uma coleção do tipo **NameValueCollection**, com as chaves do cabeçalho que são transmitidos com o e-mail.
IsBodyHtml	Especifica um valor booleano indicando se o corpo da mensagem está ou não em formato HTML.
Priority	Indica a prioridade da mensagem através do enumerador **MailPriority**. As opções que ele fornece são: • **High –** Prioridade alta. • **Low –** Prioridade baixa. • **Normal –** Prioriedade normal.
ReplyTo	Recebe um objeto do tipo **MailAddress** que é utilizado, ao invés da propriedade *From*, quando o usuário responder ao e-mail.
Sender	Recebe um objeto do tipo **MailAddress** que é utilizado

	como remetente do e-mail.
Subject	Uma *string* contendo o assunto do e-mail.
SubjectEncoding	Define o *encoding* do corpo do assunto.
To	Recebe um objeto do tipo **MailAddress** contendo as informações a respeito do destinatário da mensagem.

A classe **MailAddress** que mencionamos várias vezes na tabela acima, trata-se de um objeto que representa um endereço de correio eletrônico, independente se ele é remetente ou destinatário. Essa classe tem apenas quatro propriedades: *Address*, *DisplayName*, *Host* e *User*. A primeira delas, *Address*, recebe uma *string* contendo o endereço de e-mail; a seguir, temos a propriedade *DisplayName*, que também recebe uma *string* onde podemos definir o nome amigável a ser exibido que alguns leitores de e-mail utilizam para exibir; a terceira delas, a propriedade *Host*, trata-se de uma propriedade de somente leitura que retorna uma *string* contendo o host informado na propriedade *Address*; finalmente, a propriedade *User*, também retorna uma *string* contendo o nome do usuário (a primeira parte, antes do @) informado na propriedade *Address*. As propriedade To, CC e Bcc expõe uma coleção fortemente tipada do tipo **MailAddressCollection** que somente operam com objetos do tipo **MailAddress**.

O trecho de código abaixo exemplifica a utilização da classe **MailMessage** em conjunto com a classe **MailAddress**:

VB.NET
```
Imports System.Net.Mail

Dim de As New MailAddress("israel@projetando.net", "Israel Aéce - Via .NET")
Dim para As New MailAddress("israelaece@yahoo.com.br", "Israel Aéce")
Dim msg As New MailMessage(de, para)
msg.Attachments.Add(New Attachment("Teste.txt"))
msg.Subject = "Teste de envio no .NET"
msg.Body = "<b>E-mail enviado via .NET 2.0</b>"
msg.IsBodyHtml = True
```

C#
```
using System.Net.Mail;

MailAddress de = new MailAddress("israel@projetando.net", "Israel Aéce - Via .NET");
MailAddress para = new MailAddress("israelaece@yahoo.com.br", "Israel Aéce");
MailMessage msg = new MailMessage(de, para);
msg.Attachments.Add(new Attachment("Teste.txt"));
msg.Subject = "Teste de envio no .NET";
msg.Body = "<b>E-mail enviado via .NET 2.0</b>";
msg.IsBodyHtml = true;
```

Uma alternativa ao código acima para deixar o conteúdo do e-mail mais flexível a diversos leitores, é utilizar os *AlternateViews* para criar versões diferentes do mesmo corpo do e-mail, para que seja possível que usuários que suportam HTML e aqueles não suportam, consigam visualizar a mensagem. Neste caso, o código tem uma mudança um pouco radical na definição do corpo do e-mail, ou seja, não será mais necessário definir a propriedade *Body*, pois criaremos isso a partir da classe **AlternateView**. O código abaixo ilustra apenas a criação dos *AlternateViews*, mantendo o restando do código idêntico ao que temos acima:

```
VB.NET
Imports System.Net.Mail
Imports System.Net.Mime

msg.AlternateViews.Add( _
    AlternateView.CreateAlternateViewFromString( _
        "<b>E-mail enviado via .NET 2.0 - HTML</b>", _
        Nothing, _
        MediaTypeNames.Text.Html))

msg.AlternateViews.Add( _
    AlternateView.CreateAlternateViewFromString( _
        "E-mail enviado via .NET 2.0 - Plain Text", _
        Nothing, _
        MediaTypeNames.Text.Plain))

C#
using System.Net.Mail;
using System.Net.Mime;

msg.AlternateViews.Add( _
    AlternateView.CreateAlternateViewFromString(
        "<b>E-mail enviado via .NET 2.0 - HTML</b>",
        null, _
        MediaTypeNames.Text.Html));

msg.AlternateViews.Add(
    AlternateView.CreateAlternateViewFromString(
        "E-mail enviado via .NET 2.0 - Plain Text",
        null,
        MediaTypeNames.Text.Plain));
```


O método estático *CreateAlternateViewFromString* retorna um objeto do tipo AlternateView com o body pré-configurado. Para esse mesmo método, passamos como

último parâmetro, o tipo da visualização, indicando através da classe **MediaTypeNames**. A imagem abaixo ilustra o e-mail recebido dentro do Microsoft Outlook:

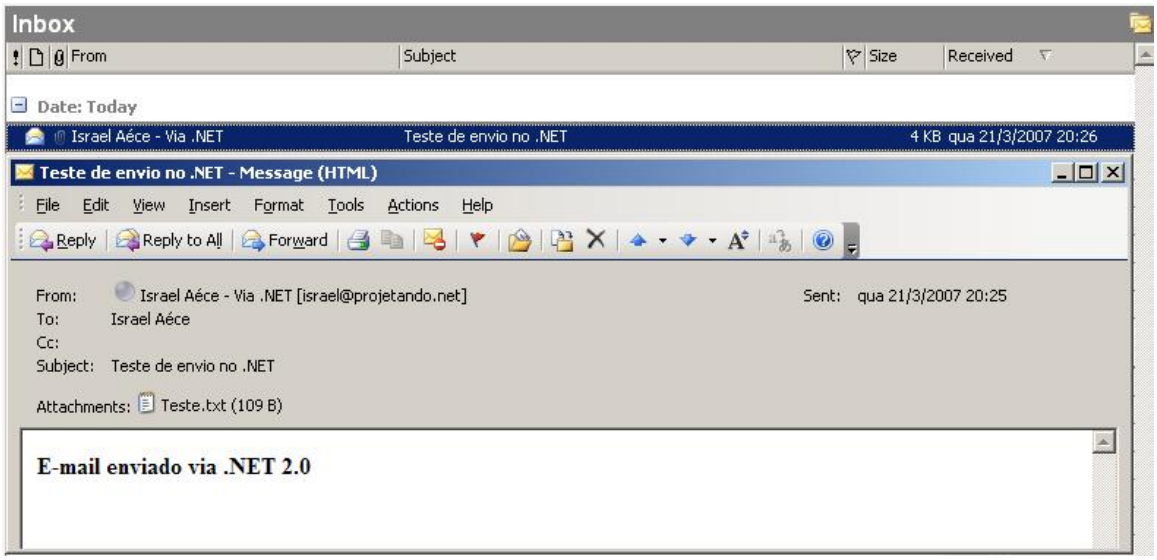

Imagem 11.1 – E-mail recebido no Microsoft Outlook.

Embutindo imagens como recursos

A versão 2.0 do .NET Framework já traz intrinsicamente um recurso que nas versões anteriores somente conseguíamos com a utilização de componentes de terceiros; trata-se da opção de agora podermos embutir dentro do e-mail imagens que farão parte do conteúdo do mesmo.

Nas versões anteriores, se não quiséssemos utilizar componentes de terceiros, tínhamos que disponibilizar em algum lugar público, geralmente imagens, que iriam fazer parte do conteúdo do email e, através do acesso via HTTP, a exibíamos no inteiro do corpo do e-mail. O ponto negativo disso é que o usuário que está lendo o e-mail depende de uma conexão ativa com a internet para que o consiga visualizar essas imagens.

Com a versão 2.0 do .NET Framework, temos duas principais classes para trabalharmos com isso. São elas: **AlternateView** e **LinkedResource**. A primeira especifica diferentes cópias do conteúdo do email, ou seja, você define o e-mail com o formato e tags HTML e, se o leitor de e-mails do destinatário não suportar HTML, você pode fornecer através desta classe, uma versão em *plain-text* do mesmo conteúdo. Já a segunda classe, representa um recurso externo que será embutido dentro do conteúdo do email que, na maioria dos casos, é uma imagem. Depois desta classe criada, o adicionamos na coleção de **LinkedResources** do objeto **AlternateView**.

O código abaixo mostra-nos como devemos proceder para conseguirmos enviar um e-mail com uma imagem embutida no corpo do mesmo:

VB.NET

```
Imports System.Net.Mail
Imports System.Net.Mime

Dim de As New MailAddress("israel@projetando.net", "Israel Aéce
")
Dim para As New MailAddress("israelaece@yahoo.com.br", "Israel
Aéce")

Dim msg As New MailMessage(de, para)
msg.Subject = "Teste de E-mail"
Dim body As String = _
    "<img src=""cid:Imagem1"" /><br><br><b>E-mail enviado via
.NET 2.0</b>"

Dim view As AlternateView = _
     AlternateView.CreateAlternateViewFromString(body, Nothing,
MediaTypeNames.Text.Html)

Dim resource As New LinkedResource("Logo.gif")
resource.ContentId = "Imagem1"
view.LinkedResources.Add(resource)
msg.AlternateViews.Add(view)
```

C#

```
using System.Net.Mail;
using System.Net.Mime;

MailAddress de =
    new MailAddress("israel@projetando.net", "Israel Aéce");
MailAddress para =
    new MailAddress("israelaece@yahoo.com.br", "Israel Aéce");

MailMessage msg = new MailMessage(de, para);
msg.Subject = "Teste de E-mail";
string body =
    @"<img src=""cid:Imagem1"" /><br><br><b>E-mail enviado via
.NET 2.0</b>";

AlternateView view =
     AlternateView.CreateAlternateViewFromString(body, null,
MediaTypeNames.Text.Html);

LinkedResource resource = new LinkedResource("Logo.gif");
resource.ContentId = "Imagem1";
view.LinkedResources.Add(resource);
msg.AlternateViews.Add(view);
```

Como podemos analisar no código acima, criamos uma classe do tipo **MailMessage**, como já fazíamos nas versões anteriores. Dentro do conteúdo do e-mail (*body*), definimos a tag img e o atributo *src* que corresponderá a imagem no local que desejarmos. Através do *cdi* especificamos que o conteúdo será "substituído" pelo conteúdo que mais tarde vamos vir a embutir. Através do método estático *CreateAlternateViewFromString*, onde passamos o corpo da mensagem e o tipo que ela irá ser (no caso HTML), devolvemos uma instancia da classe **AlternateView** baseada nesses mesmos parâmetros.

Depois disso, criamos um objeto do tipo **LinkedResource**, onde vamos definir a imagem (ou recurso) que vamos embutir. É importante dizer que a propriedade *ContentId* deve ter exatamente o mesmo ID que definimos no *cid* do corpo da mensagem. Agora basta adicionarmos o objeto na coleção de *LinkedResources* do objeto **AlternateView** e, este por sua vez, adicionarmos na coleção de *Views* do objeto **MailMessage**. A imagem abaixo ilustra o e-mail, dentro do Microsoft Outlook, já com a imagem embutida:

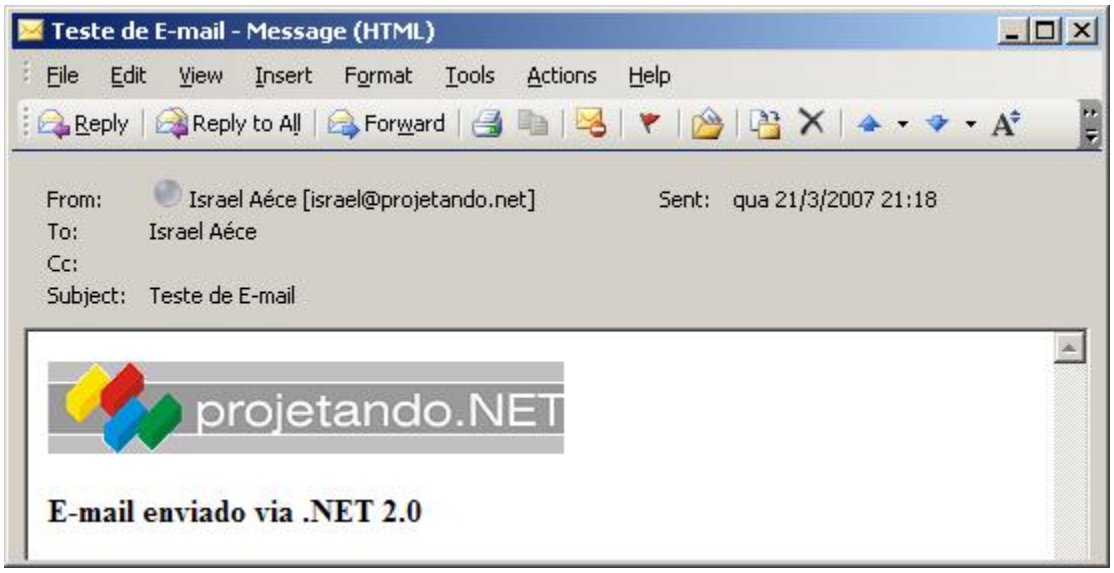

Imagem 11.2 – E-mail com imagem embutida.

A classe SmtpClient

Depois da mensagem criada, é necessário enviá-la para o seu destino. A classe **MailMessage** não tem funcionalidade para isso e, neste momento, utilizaremos a classe **SmtpClient**.

Essa classe permite enviar e-mails através do protocolo SMTP. Para que o envio seja possível, é necessário que você informe os seguintes dados:

- O *host* que é o servidor SMTP que você utilizará para enviar o e-mail. Essa informação pode ser definida no construtor da classe **SmtpClient** ou através da propriedade *Host*.

- Credenciais para autenticação, se requerida, podendo ser configurada através da propriedade *Credentials*, também da classe **SmtpClient**.
- Endereço do remetente, destinatário(s) e o conteúdo a ser enviado. Tudo isso é definido na classe **MailMessage**, que vimos acima como configurá-la.

A configuração dessa classe pode ser realizada de duas formas: via código ou declarativamente, através do arquivo *.config da aplicação. Basicamente, a diferença é que a segunda opção te fornece uma flexibilidade maior, já que as informações não ficam em *hard-code*. Para exemplificar, vamos analisar as duas formas, a começar pela configuração via código:

```
VB.NET
Imports System.Net.Mail

Dim msg As New MailMessage()
'configuração do MailMessage suprimido

Dim smtp As New SmtpClient("mail.servidor.com.br")
smtp.Send(msg)

C#
using System.Net.Mail;

MailMessage msg = new MailMessage();
//configuração do MailMessage suprimido

SmtpClient smtp = new SmtpClient("mail.servidor.com.br")
smtp.Send(msg);
```

Caso o servidor de SMTP necessite de autenticação, então é necessário criar uma instância da classe **NetworkCredential**, contida no *namespace* **System.Net**, informando o *userName* e o *password* e, em seguida, atribuir a instância desta classe na propriedade *Credentials* da classe **SmtpClient**.

A classe **SmtpClient** utiliza o método *Send*, passando uma instância de uma classe **MailMessage** para enviar. A classe **SmtpClient** ainda permite o envio assíncrono de e-mail, ou seja, ela fornece um método chamado *SendAsync* que, podemos trabalhar em conjuto com o evento *SendCompleted* que será disparado quando o envio do e-mail for completado. Esse evento utiliza o *delegate* **SendCompletedEventHandler** que define como argumento um objeto do tipo **AsyncCompletedEventArgs**, que retorna informações a respeito do processo de envio do e-mail. Ambas classes estão contidas dentro do *namespace* **System.ComponentModel**. O trecho de código abaixo ilustra como proceder para enviar o e-mail de forma assíncrona:

VB.NET
```
Imports System.ComponentModel
Imports System.Net.Mail

Dim msg As New MailMessage()
'configuração do MailMessage suprimido

Dim smtp As New SmtpClient("mail.servidor.com.br")
AddHandler smtp.SendCompleted, AddressOf Callback
smtp.SendAsync(msg, Nothing)

'...

Public Sub Callback(ByVal sender As Object, _
    ByVal e As AsyncCompletedEventArgs)

    If Not IsNothing(e.Error) Then
        Console.WriteLine(e.Error.Message)
    End If
End Sub
```

C#
```
using System.ComponentModel;
using System.Net.Mail;

MailMessage msg = new MailMessage();
//configuração do MailMessage suprimido

SmtpClient smtp = new SmtpClient("mail.servidor.com.br")
smtp.SendCompleted += new SendCompletedEventHandler(Callback);
smtp.SendAsync(msg, null);

//...

private void Callback(object sender, AsyncCompletedEventArgs e)
{
    if (e.Error != null)
    {
        Console.WriteLine(e.Error.Message);
    }
}
```

O segundo parâmetro (definido como *Nothing* no exemplo) que é passado para o método *SendAsync* é um objeto do tipo **System.Object** que será devolvido dentro do método de *Callback*, através da propriedade *UserState* do objeto **AsyncCompletedEventArgs**.

Tratamento de Erros

Quando a classe **SmtpClient** não consegue, por algum motivo, enviar o e-mail, algumas exceções específicas podem ser atiradas. Ainda dentro do *namespace* **System.Net.Mail**, temos algumas exceções que, como já sabemos, herdam direta ou indiretamente da classe Exception, quais são atiradas quando algum problema ocorrece. Para entendermos a hierarquia das exceções dentro deste namespace, vamos analisar a imagem abaixo:

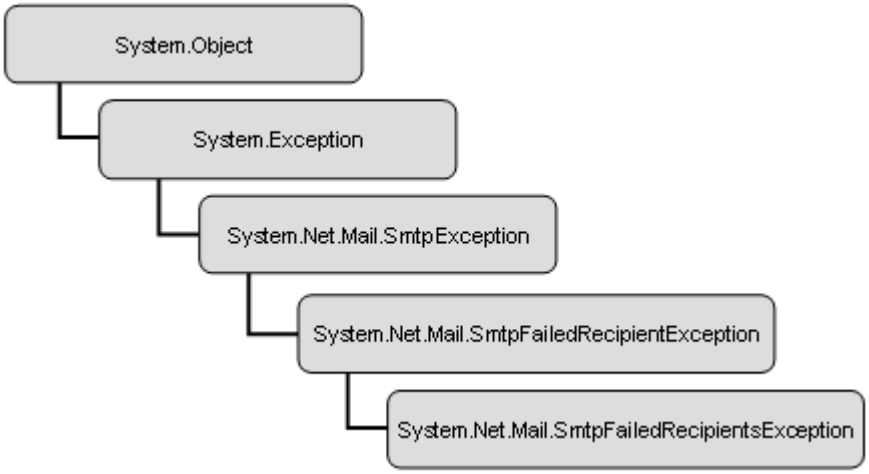

Imagem 11.3 – Hierarquia das exceções do *namespace* **System.Net.Mail**.

Abaixo está a descrição para cada uma das exceções:

System.Net.Mail.SmtpException: representa uma exceção que é atirada pela classe **SmtpClient** quando não é possível completar a operação de envio, invocado pelo método *Send* ou *SendAsync*. A propriedade *StatusCode* contém o código do status, retornado pelo servidor de SMTP.

System.Net.Mail.SmtpFailedRecipientException: representa uma exceção que é atirada pela classe **SmtpClient** quando não é possível completar a operação de envio para um destinatário específico, invocado pelo método *Send* ou *SendAsync*.

System.Net.Mail.SmtpFailedRecipientsException: representa uma exceção que é atirada pela classe **SmtpClient** quando não é possível entregar a mensagem para todos os destinatários.

Como pode ocorrer erros durante o envio de e-mails, é necessário envolver a chamada do método *Send* ou o método *SendAsync* em um bloco *Try/Catch* para capturar a falha e não corromper o seu código. É importante lembrar que a ordem dos blocos *"Catchs"* devem ser ordenados do mais específico para o mais genérico, que é exatamente a ordem de baixo para cima da imagem 11.3. Com isso conseguimos customizar a mensagem de erro para o usuário e tomar uma decisão mais compatível com o problema ocorrido. O código abaixo exemplifica o uso:

VB.NET

```
Imports System.Net.Mail

Try
    Dim msg As New MailMessage()
     'configuração do MailMessage suprimido

    Dim smtp As New SmtpClient("mail.servidor.com.br")
    smtp.Send(msg)
Catch e As SmtpFailedRecipientsException
    Console.WriteLine(e.ToString())
Catch e As SmtpFailedRecipientException
    Console.WriteLine(e.ToString())
Catch e As SmtpException
    Console.WriteLine(e.ToString())
Catch e As Exception
    Console.WriteLine(e.ToString())
End Try
```

C#

```
using System.Net.Mail;

try
{
    MailMessage msg = new MailMessage();
    //configuração do MailMessage suprimido

    SmtpClient smtp = new SmtpClient("mail.servidor.com.br")
    smtp.Send(msg);
}
catch(SmtpFailedRecipientsException e)
{
    Console.WriteLine(e.ToString());
}
catch(SmtpFailedRecipientException e)
{
    Console.WriteLine(e.ToString());
}
catch(SmtpException e)
{
    Console.WriteLine(e.ToString());
}
catch(Exception e)
{
    Console.WriteLine(e.ToString());
}
```