



Edição 110 :: Ano X :: R\$ 14,90



DEVMEDIA

Integrando sistemas com Integration Services
Construindo soluções com SQL Server
Integration Services 2012

Desenvolvendo um cliente de VNC
Como criar uma aplicação cliente em C#
que se conecta a um servidor VNC remoto

Realizando Data Binding em WPF
Como utilizar Data Binding, LINQ to SQL
e MVVM em aplicações reais

NOTIFICAÇÕES NO WINDOWS PHONE

Uma forma de manter usuário atualizado



ISSN 1980393-1



SEU CARRO TEM SEGURO, SUA SAÚDE TEM SEGURO, MAS E O SEU EMPREGO... TÁ SEGURO??



NÃO DEIXE JUSTAMENTE A SUA CARREIRA FICAR EM RISCO!

Manter-se atualizado com todas as novidades do mercado de desenvolvimento é obrigação de todo bom programador. Faça agora mesmo um seguro para a sua carreira. Seja um assinante MVP!

Saia do risco! ↗

TENHA ACESSO A:



+DE 260 CURSOS ONLINE



09 REVISTAS MENSais



7.850 VÍDEO-AULAS

POR APENAS **59,90** MENSais



QUEM TEM ESTÁ TRANQUILO.



DEVMEDIA

Acesse: www.devmedia.com.br/mvp

Corpo Editorial

Editor Geral

Joel Rodrigues (joelrlneto@gmail.com)

Jornalista Responsável

Kaline Dolabella - JP24185

Capa e Diagramação

Romulo Araújo

Na Web

www.devmedia.com.br/dotnet

Distribuição

FC Comercial e Distribuidora S.A
Rua Teodoro da Silva, 907 | Grajaú - RJ - 206563-900

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

www.devmedia.com.br/central
(21) 3382-5038

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

Cristiany Queiroz
publicidade@devmedia.com.br

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique à vontade para entrar em contato com os editores e dar a sua sugestão!

Se você estiver interessado em publicar um artigo na revista ou no site .NET Magazine, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Joel Rodrigues - Editor da Revista
joelrlneto@gmail.com



Joel Rodrigues

Editor Chefe da .net Magazine e Easy .net Magazine

MVP

Assine agora e tenha acesso a todo o conteúdo da DevMedia:
www.devmedia.com.br/mvp

Conteúdo sobre Novidades

04 – Notificações Push no Windows Phone 8

[Felipe Farias Ferrari]

Conteúdo sobre Boas Práticas

24 – Data Binding em WPF: Interagindo com o BD

[Henrique Machado Gasparotto]

Artigo no estilo Curso

36 – Integration Services2012: implementando soluções de ETL - Parte 1

[Renato José Groffe]

Artigo no estilo Solução Completa

58 – Criando uma aplicação para conexões remotas

[Mauro Pichiliani]

Sumário



Dê seu feedback sobre esta edição!

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

Dê seu voto sobre esta edição, artigo por artigo, através do link:

www.devmedia.com.br/netmagazine/feedback

Notificações Push no Windows Phone 8

Enviando e recebendo notificações a partir de um serviço externo

No Windows Phone, com o número crescente de Apps, temos que sempre estar dispostos a nos diferenciar dos concorrentes, com isso, um dos diferenciais pode ser notificar o usuário sobre algum evento importante em tempo real, ou mesmo de tempos em tempos dependendo do propósito da App. Além disso, podemos utilizar as notificações Push não apenas como diferencial, mas também como "core" da App, por exemplo, em jogos por turno.

Desse modo, esse artigo se propõe a abordar como podemos trabalhar com notificações Push no Windows Phone 8.

Push Notifications

Podemos descrever as notificações Push como mensagens que contêm informações e podem ser recebidas pelas Apps, mesmo quando esta não se encontra em execução, dependendo do tipo de notificação enviada.

O serviço Microsoft Push Notification para o Windows Phone oferece aos desenvolvedores um canal para envio de dados a uma App do Windows Phone, de forma assíncrona, a partir de um serviço web ou na nuvem que resulte em baixo consumo de energia pelo telefone.

Conforme a **Figura 1**, podemos entender como funciona o serviço.

1. A App solicita ao cliente Push do telefone uma URI para notificação.
2. O cliente Push negocia com o Microsoft Push Notification Service (MPNS), e o MPNS retorna uma URI única de notificação ao cliente Push do telefone.
3. O cliente Push retorna a URI de notificação para a App.
4. A App pode enviar a URI recebida a um serviço web ou na nuvem para que esse serviço web possa enviar mensagens à App quando necessário.
5. Quando o serviço web tem alguma mensagem para ser enviada à App, este irá enviar uma requisição

Resumo DevMan

Porque esse artigo é útil:

Este artigo será útil para os desenvolvedores que desejam utilizar novas formas de comunicação e interação com suas Apps por parte dos usuários, por exemplo, notificar sobre novas notícias, exibir novas imagens nos Tiles, a partir de um mecanismo que funciona mesmo com a App fechada, e que consome pouco recurso do telefone, como bateria.

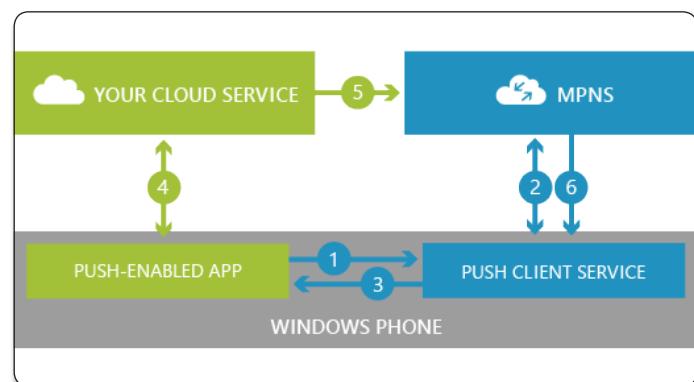


Figura 1. Diagrama sobre como funciona o envio das notificações. Fonte: MSDN

utilizando a URI de notificação recebida anteriormente para o MPNS.

6. O MPNS encaminha a notificação Push para o telefone.

Nesse processo de envio, o MPNS responderá com um código de resposta ao solicitante do Push, indicando se o mesmo foi encaminhado ou será encaminhado na próxima possibilidade ao telefone. Também, o MPNS pode retornar indicando que o Push não será entregue e com isso, dependendo da situação, o Push pode ser reenviado ou não. Apesar disso, o MPNS não oferece uma confirmação de ponta a ponta de que o Push foi realmente entregue.

Para o uso desse serviço a Microsoft exige e/ou recomenda que algumas ações sejam tratadas, ou implementadas para garantir melhor o funcionamento do sistema, tais como:

- Para o uso do Push o desenvolvedor deve incluir no seu arquivo WMAppManifest.xml a indicação do uso do Push através da capability “ID_CAP_PUSH_NOTIFICATION”, conforme **Figura 2**.

```
<Capabilities>
  <Capability Name="ID_CAP_PUSH_NOTIFICATION" />
<Capabilities>
```

Figura 2. Demonstração da funcionalidade que deve ser adicionada no arquivo de Manifesto

- Como parte da política de certificação da App, a mesma deve descrever ao usuário qual será a finalidade para a qual o Push será usado, e que o mesmo deve confirmar o uso. Também, a App deve oferecer um meio para que o usuário possa cancelar o recebimento das notificações push;
- A App sempre deve verificar se um canal de push já existe antes de tentar abri-lo, para evitar que o mesmo gere uma exceção durante a execução. O canal, por sua vez, persiste mesmo que a App seja atualizada, contudo, isso exige que o mesmo esteja atrelado a uma notificação do tipo Tile ou Toast;
- A App deve registrar-se no evento ChannelUriUpdated no caso de uma mudança da Uri de notificação, mesmo que o canal já esteja aberto. A URI expira de tempos em tempos, necessitando que a App solicite uma nova URI;
- A App deve registrar-se no evento ErrorOccurred para tratar qualquer erro. Alguns dos possíveis erros estão na **Tabela 1**;
- Sempre que a App for aberta, ela deve enviar a URI ao servidor web para que este tenha a URI sempre atualizada. Também é recomendado que a App envie um identificador único do telefone para que seja possível rastrear uma mudança de URI por parte do servidor web. Como exemplo para obter um identificador de cada telefone, na **Listagem 1** temos o exemplo de uma solução para Windows Phone 7.1, já na **Listagem 2**, para uma solução Windows Phone 8. Em uma solução Windows Phone 8, o identificador será único por telefone e publicador, sendo assim, caso em outra App

de outro publicador tenha esse mesmo trecho de código para recuperar o identificador do telefone, o identificador retornado será diferente entre uma App e outra. Contudo, para que seja possível obter esse identificador é necessário adicionar ao arquivo de Manifesto a identificação ID_CAP_IDENTITY_DEVICE;

- A quantidade de aplicativos com notificações Push por aparelho é limitada em 15 no Windows Phone 7 pré Mango (7.5), 30 para Windows Phone 7.5 e ilimitada para Windows Phone 8. Caso o limite seja atingido, ao abrir o canal com cliente MPNS será lançada uma exceção e não será possível habilitar a notificação para o aparelho até que o usuário remova as notificações de outra App.

Listagem 1. Propriedade para recuperar o identificador único em uma solução Windows Phone 7.1

```
01 private string m_DeviceUniqueId;
02 public string DeviceId
03 {
04   get
05   {
06     if (m_DeviceUniqueId == null)
07     {
08       object val;
09       if (Microsoft.Phone.Info.DeviceExtendedProperties.TryGetValue
("DeviceUniqueId", out val))
10         m_DeviceUniqueId = Convert.ToString(val as byte[]);
11     }
12   return m_DeviceUniqueId;
13 }
14 }
```

Listagem 2. Propriedade para recuperar o identificador único em uma solução Windows Phone 8

```
01 private string m_DeviceUniqueId;
02 public string DeviceId
03 {
04   get
05   {
06     if (m_DeviceUniqueId == null)
07     {
08       m_DeviceUniqueId =
Windows.Phone.System.Analytics.HostInformation.PublisherHostId;
09     }
10   return m_DeviceUniqueId;
11 }
12 }
```

| Erro | Descrição |
|--------------------------------------|---|
| PushErrorTypeChannelOpenFailed | O canal não existe, portanto não pode ser aberto. A App deve tentar abrir novamente. |
| PushErrorTypeMessageBadContent | Para uma mensagem do tipo Tile, a mesma não se enquadrou nas condições: 1) A URL não é http://, 2) O nome do servidor excedeu 256 caracteres, 3) A URL excedeu 2055 caracteres, 4) O domínio não foi especificado dentro dos domínios permitidos. Para uma mensagem do tipo Toast, o valor da tag Param excede o limite de 256 caracteres. |
| PushErrorTypeNotificationRateTooHigh | A taxa de notificações recebidas está muito alta. Diminuir a taxa de notificações recebidas para evitar mensagens descartadas. |
| PushErrorTypePayloadFormatInvalid | O XML da mensagem está mal formatado ou com erros, ou o tipo de notificação não foi definido no cabeçalho da requisição, ou não se enquadra no tipo da requisição efetuada, com isso, o canal será fechado. Portanto, devemos verificar o XML da requisição e reabrir o canal para obter uma nova URI de notificação. |
| PushErrorTypeUnknown | Ocorreu um erro interno. O telefone talvez necessite ser reiniciado. |

Tabela 1. Possíveis erros que podem ser retornados no evento ErrorOccurred

Notificações Push no Windows Phone 8

- A App deve se autenticar ao serviço web;
- A App deve criptografar a sua URI de notificação antes de enviar ao serviço web;
- Se o usuário desligar o recebimento das notificações dentro da App, esta deve fechar o canal através do método Close;
- O envio das notificações Push pode ser feito de modo autenticado ou não. No modo não autenticado, o número e frequência das notificações enviadas ao MPNS são controlados, no caso, o limite atual é de 500 por canal (telefone), por dia. Para utilizar o modo autenticado, o qual não apresenta restrições, o desenvolvedor primeiro precisa registrar um certificado com o serviço de notificações Push através da loja do Windows Phone. O certificado precisa ser expedido por uma unidade certificadora confiável por parte da Microsoft, e então esse certificado é usado para estabelecer uma conexão segura através de SSL entre o serviço web e o serviço de notificações push.

Após a explicação sobre o que é e como funciona o envio de notificações push, iremos falar dos três tipos de notificações, que são Tile, Toast e Raw.

Tile Notification

Como o próprio nome diz, a notificação para Tile deve ser utilizada para atualizar o conteúdo de um Tile, seja ele o principal ou não. Caso seja o principal, não existe a possibilidade de trocar seu template para o outro sem submeter outra versão da App, já para o secundário é possível criá-lo e alterá-lo em tempo de execução. Esse tipo de notificação funciona mesmo quando a App não está aberta.

O Windows Phone 7.8 (através de reflection, documentação disponibilizada na seção [Links](#)) e Windows Phone 8 suportam três tipos de templates para Tiles: Flip, Iconic e Cycle. A escolha do template irá depender da finalidade da App, já que cada um tem uma característica própria. Os Tiles têm os tamanhos descritos na [Tabela 2](#). No caso de telas WVGA, para os tamanhos 720p e 1080p (a partir do Update 3 do Windows Phone 8) as imagens são automaticamente ajustadas.

| | Flip e Cycle | Iconic |
|---------|------------------|------------------|
| Pequeno | 159 x 159 pixels | 110 x 110 pixels |
| Médio | 336 x 336 pixels | 202 x 202 pixels |
| Largo | 691 x 336 pixels | N/A |

Tabela 2. Tamanho dos Tiles para telas no tamanho WXGA

O Flip template é único tipo de notificação para Tiles anterior ao Windows Phone 7.8, que provê duas superfícies no tamanho médio e largo, na qual é possível inserir uma quantidade de texto considerável, conforme [Figura 3](#).

O padrão Iconic exibe uma imagem no centro do Tile, juntamente com um número, que é opcional, conforme [Figura 4](#). Esse tipo de padrão é recomendado se deseja exibir um contador.



Figura 3. Imagem demonstrando o funcionamento para um Tile no padrão Flip para Windows Phone 7.8 e 8 - Fonte: MSDN



Figura 4. Imagem demonstrando o funcionamento para um Tile no padrão Iconic - Fonte: MSDN

O padrão Cycle exibe uma série de até nove imagens em sequência, conforme [Figura 5](#). O uso desse template é recomendado para exibição de imagens.

Toast Notification

A notificação por Toast exibe uma mensagem por cerca de dez segundos. Se o usuário clicar sobre o Toast, a App será aberta na página inicial ou, caso seja definido no parâmetro “Param” do Toast, a página especificada.

Em telefones com Windows Phone 8 sem o Update 3, as notificações Toast não são exibidas se a App estiver em execução em primeiro plano, já nos aparelhos com essa atualização os Toasts são exibidos, mas ficam obscuros por outra atividade como uma chamada telefônica ou a tela de bloqueio. Outra funcionalidade que também será possível com a atualização é customizar o som da notificação quando o Toast for exibido.

Em um Toast, é exibida uma versão miniaturizada do ícone da App, um título e uma mensagem, conforme [Figura 6](#). No caso do título e da mensagem, ambos são truncados, caso o tamanho do texto seja maior que o espaço disponível para exibição.

Raw Notification

Esse tipo de notificação não oferece nenhuma interação direta com o usuário, ele é utilizado principalmente para recebimento

de dados quando a App está aberta, já que quando encerrada, o canal de comunicação com cliente Push é finalizado. A utilização de uma notificação Push, ao invés fazer a App consumir serviços ou APIs, faz com que o uso de recursos do telefone, principalmente a bateria, seja minimizado. Adiante no artigo veremos na prática como o esse tipo de notificação é bastante útil.

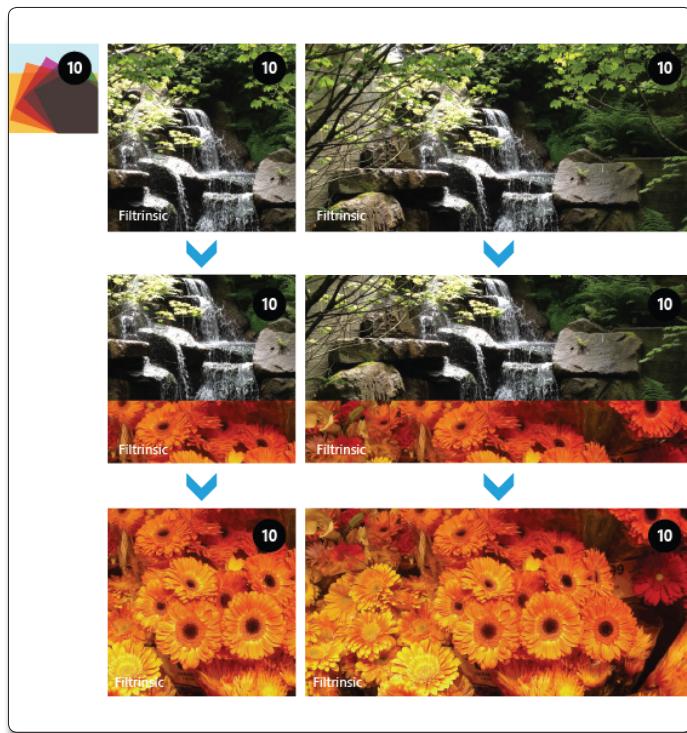


Figura 5. Imagem demonstrando o funcionamento para um Tile no padrão Cycle - Fonte: MSDN

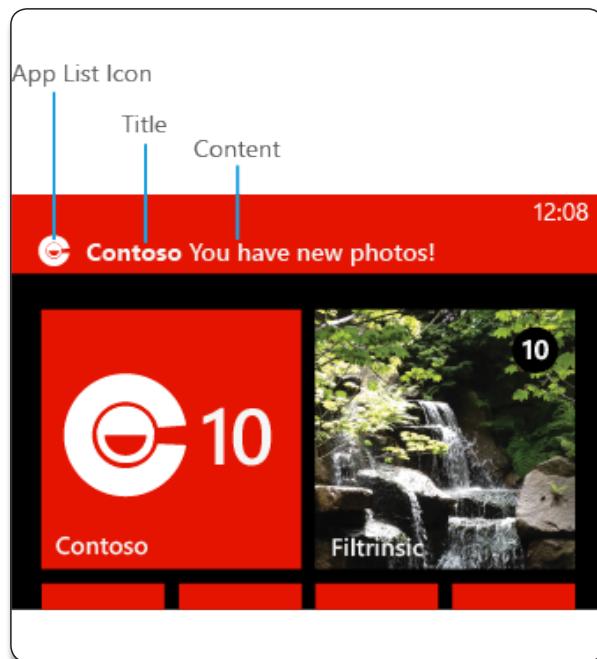


Figura 6. Imagem demonstrando como a notificação Toast é exibida - Fonte: MSDN

O projeto

A fim de exemplificar como funcionam os tipos de notificação já explicados, criaremos um projeto Web que fará as requisições ao MPNS enviando um XML de acordo com o template de cada tipo de notificação especificado pela Microsoft. Nesse caso, o projeto tem o objetivo de mostrar como funciona a comunicação entre o servidor Web e o MPNS, e entre o Windows Phone e o cliente Push. Portanto, em um ambiente real teríamos outra comunicação, entre o servidor Web e Windows Phone, para que os telefones pudessem enviar os canais de Push ao servidor através de um serviço ou API.

Windows Phone 8

Para os projetos, estaremos utilizando o Visual Studio 2012. Dessa forma, abriremos esse IDE em modo administrador e criaremos o primeiro projeto, Windows Phone 8.

Aproveitando a própria página MainPage.xaml que é criada por padrão na solução, alteraremos o grid LayoutRoot conforme a **Listagem 3**. No caso, criaremos um elemento do tipo LongList-Selector, que irá listar os lances do jogo de futebol e um TextBlock que deverá informar quais são os times que estão se enfrentando. Também será necessário adicionar algumas imagens que serão utilizadas no Tile do tipo Cycle, o qual não permite o uso de imagens externas ao telefone, conforme a **Figura 7**.

Listagem 3. Conteúdo do grid que mostrará as informações ao usuário

```

01 <!--LayoutRoot is the root grid where all page content is placed-->
02 <Grid x:Name="LayoutRoot" Background="Transparent">
03   <Grid.RowDefinitions>
04     <RowDefinition Height="Auto"/>
05     <RowDefinition Height="*"/>
06   </Grid.RowDefinitions>
07
08   <!--TitlePanel contains the name of the application and page title-->
09   <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
10     <TextBlock Text="EXEMPLO PUSH"
11       Style="{StaticResource PhoneTextNormalStyle}" Margin="12,0"/>
12     <TextBlock Text="{Binding TituloJogo}" Margin="9,-7,0,0"
13       Style="{StaticResource PhoneTextTitle1Style}"/>
14   </StackPanel>
15
16   <!--ContentPanel - place additional content here-->
17   <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
18     <ScrollViewer>
19       <StackPanel Orientation="Vertical">
20         <TextBlock Text="Informações do jogo" Margin="0,0,0,10"
21           Style="{StaticResource PhoneTextSubtleStyle}"/>
22         <phone:LongListSelector Grid.Row="1"
23           ItemsSource="{Binding InformacoesJogo}" LayoutMode="List"/>
24       </StackPanel>
25     </ScrollViewer>
26   </Grid>
27 </Grid>
```

Após definido o layout da tela e seus elementos, implementaremos a lógica para abertura do canal de comunicação das notificações push e o mecanismo de exibição das informações recebidas nas notificações para a tela do usuário, como também as configurações dos Tiles. Para isso, criaremos uma variável na

Notificações Push no Windows Phone 8

classe que conterá no nome do canal a ser utilizado pela App, a do endereço da URI do nosso projeto Web que criaremos em seguida e um objeto do tipo `HttpNotificationChannel`, que concentrará todas as informações das notificações do App, conforme a **Listagem 4**.

Listagem 4. Váriaveis globais da classe

```
01 // Nome do canal push do App
02 private readonly string channelName = "WP8PushChannel";
03
04 //Uri do Projeto WebServer
05 private readonly string uriBaseImagensTile = "http://169.254.80.2704";
06
07 // Objeto que contem as informações do canal Push.
08 HttpNotificationChannel pushChannel;
```

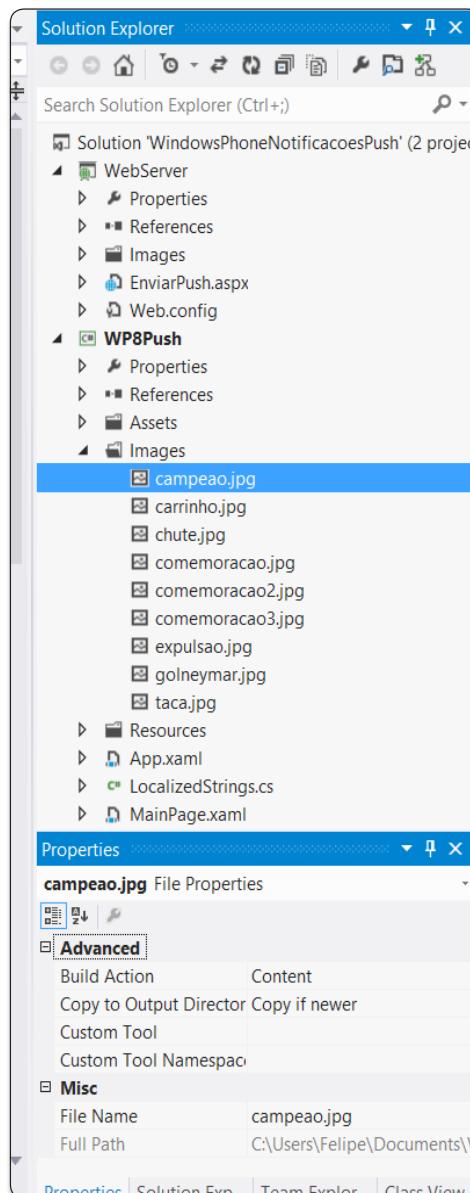


Figura 7. Adicione imagens configurando para serem copiadas quando compilar o projeto

Depois disso, adicionamos os bindings das propriedades `InformacoesJogo` e `TituloJogo`, que atualizarão os elementos da página da App, conforme a **Listagem 5**.

Listagem 5. Propriedades, método e evento utilizado no binding

```
01 private string _tituloJogo;
02 public string TituloJogo
03 {
04     get
05     {
06         return _tituloJogo;
07     }
08     set
09     {
10         if (value != _tituloJogo)
11         {
12             _tituloJogo = value;
13             NotifyPropertyChanged("TituloJogo");
14         }
15     }
16 }
17
18 private ObservableCollection<string> _informacoesJogo;
19 public ObservableCollection<string> InformacoesJogo
20 {
21     get
22     {
23         if (_informacoesJogo == null)
24             _informacoesJogo = new ObservableCollection<string>();
25
26         return _informacoesJogo;
27     }
28     set
29     {
30         if (value != _informacoesJogo)
31         {
32             _informacoesJogo = value;
33             NotifyPropertyChanged("InformacoesJogo");
34         }
35     }
36 }
37 // Evento para notificação de atualização dos elementos da
38 // pagina que estão atrelados às propriedades
39 public event PropertyChangedEventHandler PropertyChanged;
40 private void NotifyPropertyChanged(String propertyName)
41 {
42     PropertyChangedEventHandler handler = PropertyChanged;
43     if (null != handler)
44     {
45         handler(this, new PropertyChangedEventArgs(propertyName));
46     }
47 }
```

Pelo fato da App ser um exemplo para todos os tipos de notificação hoje existentes, ao abrir a App pela primeira vez esta irá criar três Tiles secundários, um de cada tipo, que serão atualizados pelas notificações. Para isso, adicionamos no evento de abertura da página, `OnNavigatedTo`, a criação de todos eles. Também nesse evento é onde atualizamos os times que estão se enfrentando. Quando esses times forem enviados por um parâmetro na URL, esse parâmetro será utilizado nas notificações Tile e Toast, assim, quando o usuário clicar sobre Tile ou Toast, ele será redirecionado para essa página com os seus devidos parâmetros, conforme a **Listagem 6**.

Listagem 6. Rotina executada na abertura da página

```
01 protected override void OnNavigatedTo(NavigationEventArgs e)
02 {
03     if (NavigationContext.QueryString.Keys.Contains("TituloJogo"))
04     {
05         this.TituloJogo = NavigationContext.QueryString["TituloJogo"];
06
07         if (NavigationService.CanGoBack)
08             NavigationService.RemoveBackEntry();
09     }
10
11 //Cria todos os Tiles secundarios que serão usados: Flip, Iconic e Cycle
12 CriarTilesSecundarios();
13
14 base.OnNavigatedTo(e);
15}
16 private void CriarTilesSecundarios()
17{
18     var tile = ShellTile.ActiveTiles.Where
19     (t => t.NavigationUri.OriginalString.Contains
20     ("Tile=iconic")).FirstOrDefault();
21
22     IconicTileData iconicTile = new IconicTileData();
23     iconicTile.Title = "Iconic";
24     ShellTile.Create(new Uri("// MainPage.xaml?TituloJogo=BRA x ESP&Tile=iconic",
25     UriKind.Relative), iconicTile, true);
26
27     return;
28 }
29
30     tile = ShellTile.ActiveTiles.Where
31     (t => t.NavigationUri.OriginalString.Contains("Tile=flip")).FirstOrDefault();
32
33     if (tile == null)
34     {
35         FlipTileData flipTile = new FlipTileData();
36         flipTile.Title = "Flip";
37         ShellTile.Create(new Uri("// MainPage.xaml?TituloJogo=BRA x ESP&Tile=flip",
38         UriKind.Relative), flipTile, true);
39
40     return;
41 }
42
43     tile = ShellTile.ActiveTiles.Where
44     (t => t.NavigationUri.OriginalString.Contains("Tile=cycle"))
45     .FirstOrDefault();
46
47     if (tile == null)
48     {
49         CycleTileData cycleTile = new CycleTileData();
50         cycleTile.Title = "Cycle";
51         ShellTile.Create(new Uri("// MainPage.xaml?TituloJogo=BRA x ESP&Tile=cycle",
52         UriKind.Relative), cycleTile, true);
53
54     return;
55 }
56 }
```

Além das criações dos Tiles, se executarem o projeto em modo Debug, na aba output do Visual Studio deve ser exibida a URI do canal criado com o serviço de Push, o que é ilustrado na **Figura 8**. Essa URI será utilizada na requisição das notificações.

Por fim, criaremos os métodos que irão se registrar junto ao cliente Push do telefone todas as notificações para a App. Alguns passos são comuns para todos os tipos de notificação, como a verificação da existência do canal antes de abri-lo, o registro no evento ChannelUriUpdated, que irá informar quando houver alteração no endereço do canal. Nesse momento, em um cenário real, precisaríamos enviar essa nova URI ao serviço Web. Outro evento comum é o ErrorOccurred, que deve informar a ocorrência de algum dos erros da **Tabela 1**.

Iniciando pela notificação do tipo Tile, podemos dizer que essa notificação funciona mesmo quando a App não está em execução, e pode ser usado para baixar recursos, como imagens externas ao telefone. Para isso, precisamos informar todas as URIs absolutas confiáveis no

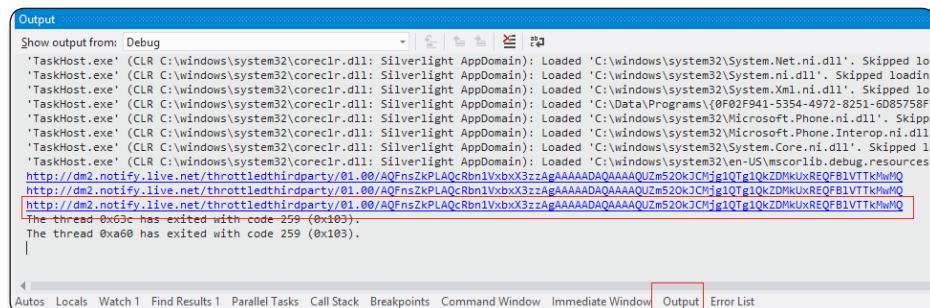


Figura 8. URI a ser utilizada nas requisições das notificações

momento do binding do canal, BindToShellTile, através do parâmetro do tipo Collection<Uri>, caso contrário, se for utilizada a sobrescrita do tipo "void", a notificação acessará somente recursos locais da App, de acordo com a **Listagem 7**. Apesar disso, existe uma exceção para a notificação de Tiles do tipo Cycle que não permite o uso de recursos externos, nesse caso, para utilizar recursos externos é preciso utilizar um Background Agent para baixar qualquer recurso de um servidor, e assim salvar localmente para depois ser utilizado no momento do Push.

A notificação do tipo Toast também funciona com a App fechada, a diferença é que ao receber a notificação, pode existir uma interação direta com o usuário temporariamente. Caso o mesmo clique sobre a área do Toast, a App irá se abrir em uma URI determinada no XML da notificação, caso contrário, a notificação desaparece após alguns segundos. Essa situação ocorrerá quando a App não estiver em execução em primeiro plano, senão, será preciso registrar-se no evento ShellToastNotificationReceived para que a App possa identificar que uma notificação Toast foi recebida. Nesse caso, a decisão sobre o que

Notificações Push no Windows Phone 8

fazer fica a cargo do desenvolvedor. No exemplo da **Listagem 8** apenas será exibida uma mensagem com os parâmetros recebidos na notificação.

Alem disso, para registrar no canal a utilização do Toast, é necessário efetuar o bind através do método BindToShellToast.

Além do tipo de notificação Tile e Toast, também é possível utilizar no Windows Phone a notificação do tipo Raw, que se diferencia das demais pelo fato do canal de comunicação para o recebimento das notificações permanecerem operacional apenas enquanto a App está em primeiro plano. Além disso, para o recebimento das notificações é preciso registrar-se no evento HttpNotificationReceived, conforme a **Listagem 9**. Os XMLs enviados através das

requisições não requerem um modelo especificado pela Microsoft e podem ser utilizados pelos desenvolvedores livremente, nesse caso, o XML trafegado contém apenas dois elementos que serão utilizados, mostrados no projeto Web.

Projeto Web

O projeto Web consiste em demonstrar parte da funcionalidade que um serviço web faria em um ambiente real para enviar as notificações ao MPNS. Desse modo, montaremos uma página que servirá para parametrizar o XML a ser enviado na requisição ao MPNS. Neste exemplo, esse projeto está configurado para ser executado no IIS Express, na porta 2704, conforme **Figura 9**.

Listagem 7. Inicialização para as notificações do tipo Tile

```
01 // Construtor
02 public MainPage()
03 {
04     InitializeComponent();
05
06     DataContext = this;
07
08     Loaded += MainPage_Loaded;
09 }
10 void MainPage_Loaded(object sender, RoutedEventArgs e)
11 {
12     InicializarRaw();
13
14     InicializarTile();
15
16     InicializarToast();
17 }
18 private void InicializarTile()
19 {
20     // Tenta encontrar o canal
21     pushChannel = HttpNotificationChannel.Find(channelName);
22
23     // Se não encontrado tenta criar uma nova conexão com o cliente Push.
24     if (pushChannel == null)
25     {
26         pushChannel = new HttpNotificationChannel(channelName);
27
28         // Registra em todos os eventos antes de abrir o canal
29         pushChannel.ChannelUriUpdated +=
30             new EventHandler<NotificationChannelUriEventArgs>(
31                 (PushChannel_ChannelUriUpdated);
32             pushChannel.ErrorOccurred +=
33                 new EventHandler<NotificationChannelErrorEventArgs>(
34                     (PushChannel_ErrorOccurred);
35
36             pushChannel.Open();
37
38             // Adiciona o endereço do nosso Web Server como confiável
39             Collection<Uri> remoteUris = new Collection<Uri>();
40             remoteUris.Add(new Uri(uriBaseImagensTile));
41
42             // Fixa ao canal os eventos de tile.
43             if (!pushChannel.IsShellTileBound)
44             {
45                 // Adiciona o endereço do nosso Web Server como confiável
46
47             Collection<Uri> remoteUris = new Collection<Uri>();
48             remoteUris.Add(new Uri(uriBaseImagensTile));
49
50             // Fixa ao canal os eventos de tile.
51             pushChannel.BindToShellTile(remoteUris);
52         }
53
54         // Se o canal já existir, apenas se registra aos eventos
55         pushChannel.ChannelUriUpdated +=
56             new EventHandler<NotificationChannelUriEventArgs>(
57                 (PushChannel_ChannelUriUpdated);
58             pushChannel.ErrorOccurred +=
59                 new EventHandler<NotificationChannelErrorEventArgs>(
60                     (PushChannel_ErrorOccurred);
61
62             if (pushChannel.ChannelUri != null)
63             {
64                 // Exibe a nova URI, apenas para exemplo. Em um ambiente real,
65                 // a nova URI deverá ser enviada ao servidor Web.
66                 System.Diagnostics.Debug.WriteLine(pushChannel.ChannelUri.ToString());
67                 MessageBox.Show(String.Format("Channel Uri is {0}",
68                     pushChannel.ChannelUri.ToString()));
69             }
70
71         void PushChannel_ErrorOccurred(object sender, NotificationChannelError
72             EventArgs e)
73     {
74         // Passível para tratamento de erro por parte da aplicação.
75         Dispatcher.BeginInvoke(() =>
76             MessageBox.Show(String.Format("Um erro de notificação
77             {0} ocorreu. {1} ({2}) {3}",
78                 e.ErrorType, e.Message, e.ErrorCode, e.ErrorAdditionalData));
79     }
80
81         void PushChannel_ChannelUriUpdated(object sender, NotificationChannelUri
82             EventArgs e)
83     {
84         Dispatcher.BeginInvoke(() =>
85             {
86                 // Exibe a nova URI, apenas para exemplo. Em um ambiente real,
87                 // a nova URI deverá ser enviada ao servidor Web.
88                 System.Diagnostics.Debug.WriteLine(e.ChannelUri.ToString());
89                 MessageBox.Show(String.Format("A nova Uri é {0}",
90                     e.ChannelUri.ToString()));
91             });
92     }
93 }
```

Listagem 8. Inicialização para as notificações do tipo Toast

```
01 private void InicializarToast()
02 {
03     // Tenta encontrar o canal
04     pushChannel = HttpNotificationChannel.Find(channelName);
05
06     // Se não encontrado tenta criar uma nova conexão com o cliente Push.
07     if (pushChannel == null)
08     {
09         pushChannel = new HttpNotificationChannel(channelName);
10
11        // Registra em todos os eventos antes de abrir o canal
12        pushChannel.ChannelUriUpdated += 
13            new EventHandler<NotificationChannelUriEventArgs>
14                (PushChannel_ChannelUriUpdated);
15        pushChannel.ErrorOccurred += 
16            new EventHandler<NotificationChannelErrorEventArgs>
17                (PushChannel_ErrorOccurred);
18
19        // Registra esse evento, caso seja necessário receber
12        // uma notificação Toast enquanto o App estiver aberto.
16        pushChannel.ShellToastNotificationReceived += 
17            new EventHandler<NotificationEventArgs>
18                (PushChannel_ShellToastNotificationReceived);
19
20        pushChannel.Open();
21
22        // Fixa ao canal os eventos toast.
23        pushChannel.BindToShellToast();
24    }
25    else
26    {
27        // Fixa ao canal os eventos toast.
28        if(!pushChannel.IsShellToastBound)
29            pushChannel.BindToShellToast();
30
31        // Se o canal já existir, apenas se registra aos eventos
32        pushChannel.ChannelUriUpdated += 
33            new EventHandler<NotificationChannelUriEventArgs>
34                (PushChannel_ChannelUriUpdated);
35        pushChannel.ErrorOccurred += 
36            new EventHandler<NotificationChannelErrorEventArgs>
37                (PushChannel_ErrorOccurred);
38
39        // Registra esse evento, caso seja necessário receber
40        // uma notificação Toast enquanto o App estiver aberto.
41        pushChannel.ShellToastNotificationReceived += 
42            new EventHandler<NotificationEventArgs>
43                (PushChannel_ShellToastNotificationReceived);
44
45    }
46
47}
48
49 private void PushChannel_ShellToastNotificationReceived
50     (object sender, NotificationEventArgs e)
51 {
52     String message = new StringBuilder();
53     string relativeUri = string.Empty;
54
55     // Criação de mensagem apenas para demonstrar o recebimento
56     // do toast enquanto o App está aberto
57     message.AppendFormat("Toast recebido {0}:\n", DateTime.Now.ToString());
58
59     // Recupera todas as informações que estão na mensagem.
60     foreach (string key in e.Collection.Keys)
61     {
62         message.AppendFormat("{0}: {1}\n", key, e.Collection[key]);
63
64         if (string.Compare(
65             key,
66             "wp:Param",
67             System.Globalization.CultureInfo.InvariantCulture,
68             System.Globalization.CompareOptions.IgnoreCase) == 0)
69         {
70             relativeUri = e.Collection[key];
71         }
72
73         // Exibe todos os campos da mensagem do toast
74         Dispatcher.BeginInvoke(() => MessageBox.Show(message.ToString()));
75
76         // Navega para a página recebida no toast
77         Dispatcher.BeginInvoke(() => NavigationService.Navigate(
78             new Uri(e.Collection["wp:Param"], UriKind.Relative)));
79     }
80 }
```

Sendo assim, temos que liberar o acesso do emulador do Windows Phone, o qual roda em um ambiente virtualizado, através da execução da linha de comando exibida na **Figura 10**. Apenas substitua pela porta especificada no projeto, caso seja diferente, em um prompt de comando (cmd.exe) em modo administrador para liberar o acesso através do firewall do Windows Vista, 7 e 8. Caso você tenha outro programa de firewall, provavelmente precisará liberar acesso a essa porta também.

Então, por último, temos que configurar o acesso ao site do IIS Express ao endereço IP em que o emulador do Windows Phone está trabalhando. Para isso, podemos utilizar esse mesmo prompt de comando e executar “ipconfig”, e assim localizar o endereço IP do adaptador cujo nome aparece “Windows Phone Emulator Internal Switch”, como se vê na **Figura 11**.

Depois de recuperar o IP, iremos utilizá-lo para adicionar o acesso através do arquivo de configuração do IIS Express, localizado na pasta IISExpress dentro da pasta “Meus Documentos”, que pode ser facilmente acessada pelo caminho: %userprofile%\documents\IISExpress\config\applicationhost.config.

Abrindo o arquivo, temos que localizar o site configurado no nosso projeto, que deverá ser compilado e executado pelo menos uma vez para que o Visual Studio possa criá-lo dentro do arquivo applicationhost.config. Assim, devemos copiar a linha já existente, colar e editar, conforme a linha selecionada na **Figura 12**, onde “2704” representa a porta em que o IISExpress irá executar e “169.254.80.80” o IP referente ao emulador do Windows Phone 8.

Agora que temos o projeto devidamente configurado para funcionar junto com o Windows Phone, criaremos uma única pagina

Notificações Push no Windows Phone 8

ASPX que servirá como entrada para parametrizar e enviar o XML ao MPNS. Assim, criaremos a página “EnviarPush.aspx”, responsável por enviar informações e fotos de um jogo de futebol.

Partindo do design da página, criaremos apenas elementos básicos, sem se preocupar com estilo e apresentação, com objetivo

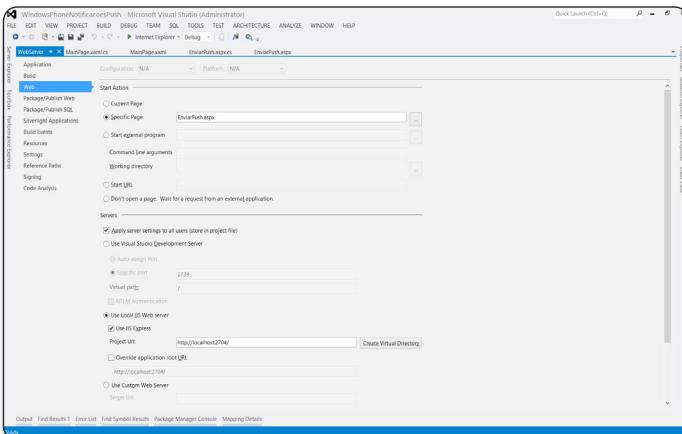


Figura 9. Imagem com a configuração do projeto Web Forms

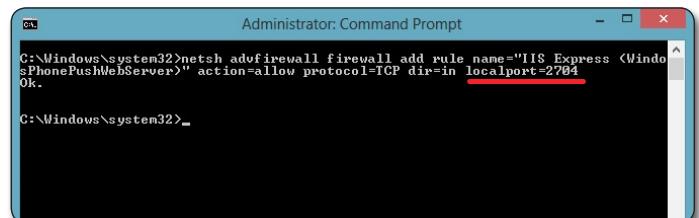


Figura 10. Linha de comando para adicionar uma exceção ao firewall do Windows

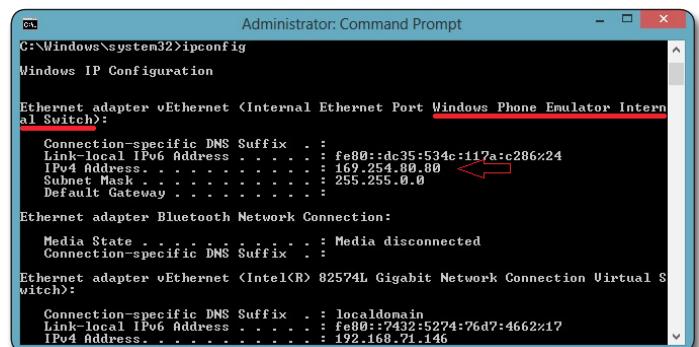


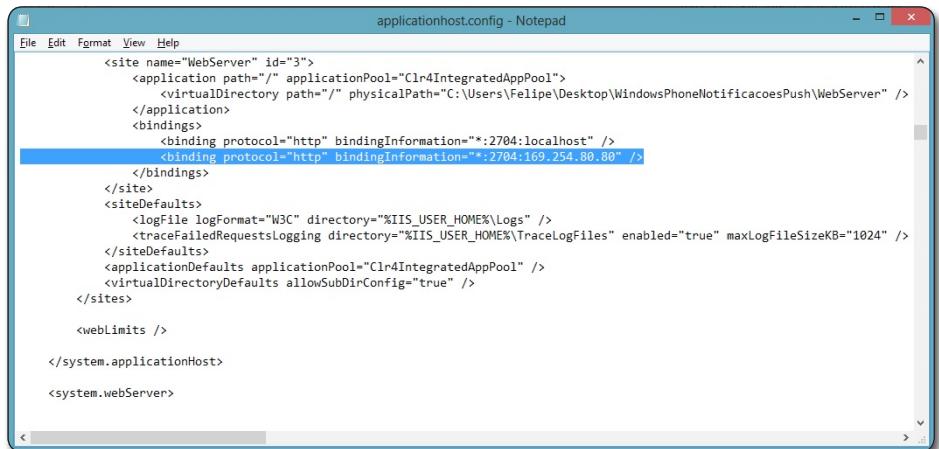
Figura 11. Recuperando o endereço IP do emulador do Windows Phone 8

Listagem 9. Inicialização para as notificações do tipo Raw

```
01 private void InicializarRaw()
02 {
03     // Tenta encontrar o canal
04     pushChannel = HttpNotificationChannel.Find(channelName);
05
06     // Se não encontrado tenta criar uma nova conexão com o cliente Push.
07     if (pushChannel == null)
08     {
09         pushChannel = new HttpNotificationChannel(channelName);
10
11        // Registra em todos os eventos antes de abrir o canal
12        pushChannel.ChannelUriUpdated +=
13            new EventHandler<NotificationChannelUriEventArgs>
14                (PushChannel_ChannelUriUpdated);
15
16        pushChannel.ErrorOccurred +=
17            new EventHandler<NotificationChannelErrorEventArgs>
18                (PushChannel_ErrorOccurred);
19
20        pushChannel.HttpNotificationReceived +=
21            new EventHandler<HttpNotificationEventArgs>
22                (PushChannel_HttpNotificationReceived);
23
24        pushChannel.Open();
25
26    }
27
28    // Exibe a nova URI, apenas para exemplo. Em um ambiente
29    // real, a nova URI deverá ser enviada ao servidor Web.
30    System.Diagnostics.Debug.WriteLine(pushChannel.ChannelUri.ToString());
31
32    MessageBox.Show(String.Format("A Uri do canal é {0}",
33        pushChannel.ChannelUri.ToString()));
34
35 private void PushChannel_HttpNotificationReceived
36     (object sender, HttpNotificationEventArgs e)
37 {
38     string message;
39
40     // Lê o conteúdo do corpo da notificação
41     using (System.IO.StreamReader reader = new System.IO.StreamReader
42         (e.Notification.Body))
43     {
44         message = reader.ReadToEnd();
45
46        // Uso de Linq to XML para recuperar o valor dos elementos do XML
47        XDocument doc = XDocument.Parse(message);
48        Dispatcher.BeginInvoke(() =>
49        {
50            TituloJogo = doc.Descendants().Where
51                (n => n.Name == "TituloJogo").First().Value;
52            InformacoesJogo.Add(doc.Descendants().Where
53                (n => n.Name == "InformacaoJogo").First().Value);
54        });
55
56        // Exibe uma mensagem, apenas para mostrar o conteúdo recebido
57        Dispatcher.BeginInvoke(() =>
58            MessageBox.Show(String.Format("Received Notification {0}\n{1}",
59                DateTime.Now.ToShortTimeString(), message)));
60    };
61 }
```

apenas de mostrar como funciona o envio de notificações push. Desse modo, na **Listagem 10** criamos um primeiro menu que terá a divisão entre os tipos Toast, Tile e Raw de notificação, e um segundo menu interno ao menu de Tile com os três tipos de notificação para Tile que existem para o Windows Phone 8, que são: Flip, Iconic e Cycle. Na **Listagem 10** está sendo exibido apenas o conteúdo do corpo da página, ou seja, que se encontra dentro da tag body.

Após o design da página, implementaremos a lógica para seu funcionamento, nesse caso, entre os eventos dá pagina utilizaremos o clique do botão “Enviar Notificação” e o clique sobre os menus de troca do tipo de Tile e notificação, de acordo com a **Listagem 11**. A partir desse momento conseguimos testar se o emulador do Windows Phone está conseguindo acessar o nosso projeto Web, e assim garantir que quando as notificações para Tile forem enviadas, o mesmo conseguirá baixar as imagens para exibição, como



```

<site name="WebServer" id="3">
    <application path="/" applicationPool="Clr4IntegratedAppPool"
        <virtualDirectory path="/" physicalPath="C:\Users\Felipe\Desktop\WindowsPhoneNotificacoesPush\WebServer" />
    </application>
    <bindings>
        <binding protocol="http" bindingInformation="*:2704:localhost" />
        <binding protocol="http" bindingInformation="*:2704:169.254.80.80" />
    </bindings>
</site>
<siteDefaults>
    <logFile logFormat="W3C" directory="%IIS_USER_HOME%\Logs" />
    <traceFailedRequestsLogging directory="%IIS_USER_HOME%\TraceLogFiles" enabled="true" maxLogFileSizeKB="1024" />
</siteDefaults>
<applicationDefaults applicationPool="Clr4IntegratedAppPool" />
<virtualDirectoryDefaults allowSubDirConfig="true" />
</sites>

<webLimits />

</system.applicationHost>
</system.webServer>

```

Figura 12. Editando o arquivo de configuração do IISExpress para permitir o acesso do emulador ao site do projeto

demonstrado na **Figura 13**. Caso não tenha sucesso no teste, verifique as configurações explicadas anteriormente.

Para o envio das notificações, chamaremos um método a partir do clique do botão. Para cada tipo de notificação existe um modelo de XML diferente a ser seguido, apesar disso, todos partem do mesmo princípio. Nesse

caso, é preciso utilizar uma URI do canal Push, enviar apenas requisições do tipo POST, complementar o modelo do XML respeitando o uso de caracteres especiais conforme a **Tabela 3**, adicionar ao cabeçalho da requisição o tipo da notificação e a prioridade para entrega e, por fim, enviar a requisição e recuperar do status do envio.



Estas soluções são monitoradas pela tecnologia FairCom todos os dias!



Confiabilidade no espaço exterior



Segurança quando você mais precisa



Segurança Financeira em cada transação



O Poder por trás da logística



Notificações Push no Windows Phone 8

Listagem 10. Design da página que enviará as notificações

```
01 <form id="form1" runat="server">
02 URI Canal:<br />
03 <asp:TextBox ID="TextBoxUri" runat="server" Width="50%"></asp:TextBox><br />
04 Tipo:
05 <asp:Menu ID="TipoPush" Width="100%" runat="server" Orientation="Horizontal"
06 StaticEnableDefaultPopOutImage="False" OnMenuItemClick=
    "TipoPush_MenuItemClick">
07 <Items>
08 <asp:MenuItem Text="Toast" Value="0" Selected="true"></asp:MenuItem>
09 <asp:MenuItem Text="Title" Value="1"></asp:MenuItem>
10 <asp:MenuItem Text="Raw" Value="2"></asp:MenuItem>
11 </Items>
12 </asp:Menu>
13 <div style="border: 1px solid black; width: 75%; padding: 10px">
14 <asp:MultiView ID="MultiViewPush" runat="server" ActiveViewIndex="0">
15 <asp:View ID="Tab1" runat="server">
16     Title:<br />
17     <asp:TextBox ID="TextBoxTitle" runat="server"
        Width="50%"></asp:TextBox><br />
18     Subtitle:<br />
19     <asp:TextBox ID="TextBoxSubTitle" runat="server"
        Width="50%"></asp:TextBox><br />
20     Titulo Jogo:<br />
21
22     <asp:TextBox ID="TextBoxTituloJogo" runat="server"
        Width="50%"></asp:TextBox><br />
23 </asp:View>
24 <asp:View ID="Tab2" runat="server">
25     <asp:Menu ID="TipoTile" Width="100%" runat="server"
        Orientation="Horizontal"
26     StaticEnableDefaultPopOutImage="False"
        OnMenuItemClick="TipoTile_MenuItemClick">
27     <Items>
28         <asp:MenuItem Text="Flip" Value="0"
            Selected="true"></asp:MenuItem>
29         <asp:MenuItem Text="Iconic" Value="1"></asp:MenuItem>
30         <asp:MenuItem Text="Cycle" Value="2"></asp:MenuItem>
31     </Items>
32 </asp:Menu>
33 <div style="border: 1px solid black; width: 50%; padding: 10px">
34     <asp:MultiView ID="MultiViewTile" runat="server"
        ActiveViewIndex="0">
35         <asp:View ID="View1" runat="server">
36             <br />
37             Flip<br />
38             Front Title:<br />
39             <asp:TextBox ID="TextBoxFrontTitle" runat="server">
40                 </asp:TextBox><br />
41             Front Background Image:<br />
42             <asp:TextBox ID="TextBoxBackgroundImage" runat="server">
43                 </asp:TextBox><br />
44             Front Count:<br />
45             <asp:TextBox ID="TextBoxCount" runat="server"
                MaxLength="2"></asp:TextBox><br />
46             Back Title:<br />
47             <asp:TextBox ID="TextBoxBackTitle" runat="server">
48                 </asp:TextBox><br />
49             Back Background Image:<br />
50             <asp:View ID="View2" runat="server">
51                 <br />
52                 Iconic<br />
53                 Title:<br />
54                 <asp:TextBox ID="TextBoxIconicTitle" runat="server">
55                     </asp:TextBox><br />
56                 Icon Image:<br />
57                 <asp:TextBox ID="TextBoxIconImage" runat="server">
58                     </asp:TextBox><br />
59                 Content1:<br />
60                 <asp:TextBox ID="TextBoxContent1" runat="server">
61                     </asp:TextBox><br />
62                 Content2:<br />
63                 <asp:TextBox ID="TextBoxContent2" runat="server">
64                     </asp:TextBox><br />
65                 Content3:<br />
66                 <asp:TextBox ID="TextBoxContent3" runat="server">
67                     </asp:TextBox><br />
68                 Count:<br />
69                 <asp:TextBox ID="TextBoxIconicCount" runat="server"
                    MaxLength="2"></asp:TextBox><br />
70                 Background Color:<br />
71                 <asp:TextBox ID="TextBoxBackgroundColor" runat="server">
72                     </asp:TextBox><br />
73                 Cycle:<br />
74                 Title:<br />
75                 <asp:TextBox ID="TextBoxCycleTitle" runat="server">
76                     </asp:TextBox><br />
77                 Count:<br />
78                 <asp:View ID="View3" runat="server">
79                     <br />
80                     Titulo Jogo:<br />
81                     <asp:TextBox ID="TextBoxTituloJogoRaw" runat="server"
                        Width="50%"></asp:TextBox><br />
82                     Informação Jogo:<br />
83                     <asp:TextBox Rows="5" ID="TextBoxInformacaoJogo" runat="server"
                        TextMode="MultiLine" Width="50%"></asp:TextBox><br />
84 </asp:View>
85 </asp:MultiView>
86 </div>
87 <asp:Button ID="ButtonSend" runat="server" OnClick="ButtonSend_Click"
    Text="Enviar Notificação"/><br />
88 Resposta:<br />
89 <asp:Label ID="LabelRespostaMPNS" runat="server" Width="50%"></asp:Label>
90 </form>
```

Listagem 11. Métodos acionados através dos eventos

```

01 //Uri do Projeto WebServer
02 protected readonly string uriBaselImagesTile = "http://169.254.80.80:2704";
03
04 protected void TipoPush_MenuItemClick(object sender, MenuEventArgs e)
05 {
06     //Troca o menu de Push ativo
07     MultiViewPush.ActiveViewIndex = Int32.Parse(e.Item.Value);
08 }
09
10 protected void TipoTile_MenuItemClick(object sender, MenuEventArgs e)
11 {
12     //Troca o menu de Push de Tile ativo
13     MultiViewTile.ActiveViewIndex = Int32.Parse(e.Item.Value);
14 }
15
16 protected void ButtonSend_Click(object sender, EventArgs e)

```

```

17 {
18     //Faz o envio da Notificação de acordo com o menu selecionado
19     switch (Convert.ToInt16(TipoPush.SelectedItem.Value))
20     {
21         case 0:
22             EnviarToast();
23             break;
24         case 1:
25             EnviarTile();
26             break;
27         case 2:
28             EnviarRaw();
29             break;
30     }
31 }

```

| Caráter | Codificação XML |
|---------|-----------------|
| < | < |
| > | > |
| & | & |
| ' | ' |
| " | " |

Tabela 3. Caracteres que devem ser tratados quando inseridos dentro das tags do XML da notificação.



Figura 13. O emulador conseguiu abrir a página que criamos no projeto Web, a comunicação no momento do Push terá sucesso

A partir do status da resposta do envio da requisição é possível identificar alguns cenários que dizem a respeito da entrega das notificações e estado do canal, de acordo com a **Tabela 4**.

Os estados de conexão com o telefone podem ser definidos entre Connected, Temp Disconnected e Disconnected, os quais possuem as transições demonstradas na **Figura 14**. No caso, pelo

estado Connected pode ser entendido que o telefone tem uma conexão ativa com o servidor de notificação Push e pode receber notificações em tempo real.

O estado Temp Disconnected é um estado temporário onde a conexão com o servidor de notificação Push foi perdida e será tentado restabelecer a comunicação assim que o telefone tiver uma conexão válida com a Internet. Nesse estado, o aparelho pode permanecer por 24 horas e todas as notificações são enfileiradas

Não perca tempo reinventando a roda!

COBREBEMX

Componente completo para sua Cobrança por Boleto Bancário e Débito em Conta Corrente

Mais de 40 exemplos em diversas linguagens de programação

Geração e leitura de arquivos (remessa e retorno) nos padrões FEBRABAN e CNAB

Testes e Downloads gratuitos em nosso site

ACESSE E CONHEÇA O COMPONENTE EM:
WWW.COBBREM.COM

Notificações Push no Windows Phone 8

até que a fila não esteja cheia. Algumas razões podem levar o telefone entrar nesse estado, como: o telefone pode estar em área de roaming e por isso a conexão pode ser interrompida; o sinal de celular onde o usuário está sofre instabilidades; alguns proxys de acesso à Internet podem interromper essa conexão direta; o modo

economia de bateria se encontra ligado e com isso as notificações Push são desligadas temporariamente.

Já no estado Disconnected o telefone está sem conexão com o servidor de notificação por mais de 24 horas. Quando o telefone entra nesse estado, todas as notificações que foram enfileiradas

| Código de Resposta | Status da Notificação | Status da Conexão com o telefone | Status do Canal | Descrição |
|-------------------------|-----------------------|----------------------------------|-----------------|---|
| 200 OK | Received | Connected | Active | A requisição da notificação foi aceita e incluída na fila para entrega. O MPNS pode responder com esse status mesmo que o telefone esteja alternando para o estado Temp Disconnected. Isso significa que a notificação não será entregue até que o telefone retorne do estado Temp Disconnected. |
| 200 OK | Received | Temp Disconnected | Active | A requisição foi aceita e incluída na fila para entrega. No entanto, o telefone está temporariamente desconectado. |
| 200 OK | QueueFull | Connected | Active | O MPNS mantém no máximo 30 notificações de cada canal para entrega. Uma vez que esse limite é atingido, todas as mensagens são descartadas até que o telefone se conecte e a fila seja esvaziada. |
| 200 OK | QueueFull | Temp Disconnected | Active | O MPNS mantém no máximo 30 notificações para entrega de cada canal. Uma vez que esse limite é atingido, todas as mensagens são descartadas até que o telefone se conecte e a fila seja esvaziada. |
| 200 OK | Suppressed | Connected | Active | A notificação foi recebida e foi descartada pelo serviço de notificação push. O status de descartada pode ocorrer se o tipo de notificação não foi ativado pela chamada BindToShellTile ou BindToShellToast pelo App, ou no caso de uma notificação do tipo Raw, a requisição foi enviada enquanto o App não estava em execução em primeiro plano, ou se foi uma notificação do tipo Tile, nenhum Tile está fixado. |
| 200 OK | Suppressed | Temp Disconnected | Active | A notificação foi recebida e foi descartada pelo serviço de notificação push. O status de descartada pode ocorrer se o tipo de notificação não foi ativado pela chamada BindToShellTile ou BindToShellToast pelo App, ou no caso de uma notificação do tipo Raw, a requisição foi enviada enquanto o App não estava em execução em primeiro plano, ou se foi uma notificação do tipo Tile, nenhum Tile está fixado. |
| 400 BadRequest | N/A | N/A | N/A | Esse erro ocorre quando o serviço Web enviou uma requisição de notificação com um XML ou URI que contém erros. |
| 401 Unauthorized | N/A | N/A | N/A | Uma notificação não é autorizada se existe alguma divergência entre o nome da pessoa no certificado do serviço Web com o nome da pessoa no certificado do serviço de notificação push, ou mesmo se o token não é válido para o canal ou foi modificado. |
| 404 Not Found | Dropped | Connected | Expired | O canal é inválido e não está presente no serviço de notificação push. O serviço Web deve parar de enviar novas notificações para esse canal. |
| 404 Not Found | Dropped | Temp Disconnected | Expired | O canal é inválido e não está presente no serviço de notificação push. O serviço Web deve parar de enviar novas notificações para esse canal. |
| 404 Not Found | Dropped | Disconnected | Expired | O canal é inválido e não está presente no serviço de notificação push. O serviço Web deve parar de enviar novas notificações para esse canal. |
| 405 Method Not Allowed | N/A | N/A | N/A | Apenas o método POST é aceito no envio das requisições das notificações. |
| 406 Not Acceptable | Dropped | Connected | Active | Esse erro pode acontecer quando um serviço Web está utilizando o modo sem autenticação de envio de notificação e esse atingiu o limite por dia de envio para determinado canal. Quando ocorrer esse erro, o serviço Web pode tentar enviar uma notificação a cada hora, no entanto, pode ser necessário esperar até 24 horas para que o envio de notificações seja normalizado. |
| 406 Not Acceptable | Dropped | Temp Disconnected | Active | Esse erro pode acontecer quando um serviço Web está utilizando o modo sem autenticação de envio de notificação e esse atingiu o limite por dia de envio para determinado canal. Quando ocorrer esse erro, o serviço Web pode tentar enviar uma notificação a cada hora, no entanto, pode ser necessário esperar até 24 horas para que o envio de notificações seja normalizado. |
| 412 Precondition Failed | Dropped | Disconnected | N/A | O telefone está em um estado desconectado. O serviço Web deve continuar enviando as notificações normalmente. |
| 503 Service Unavailable | N/A | N/A | N/A | O serviço de notificação Push não foi capaz de processar a requisição, desse modo, a requisição pode ser reenviada mais tarde. |

Tabela 4. Tabela com os códigos de retorno do MPNS

para esse telefone são descartadas e não serão entregues quando o mesmo restabelecer a comunicação.

Um dos requisitos para o envio da notificação do tipo Toast é o preenchimento do cabeçalho da requisição, de acordo com os cabeçalhos da **Tabela 5**. Também existe um modelo de XML a ser seguido, que deve ser devidamente preenchido de acordo com a necessidade, nesse caso, podemos ver na **Listagem 12** que o XML é composto pelos elementos wp:Text1, wp:Text2 e wp:Param, demonstrados na **Figura 15**, onde o elemento wp:Param corresponde à página na qual a App será aberta, caso o usuário clique sobre o Toast. Caso o conteúdo desse elemento seja vazio, a App será

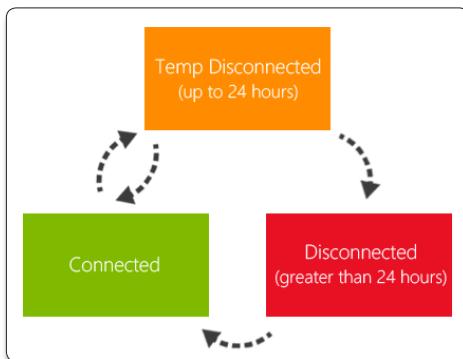


Figura 14. Transição entre os estados de conexão dos telefones com o servidor de notificação Push, fonte: MSDN

| Cabeçalho da Requisição | Valor | Descrição |
|-------------------------|-------|---------------------------------|
| X-WindowsPhone-Target | toast | Identificação da notificação |
| | 2 | Entrega imediata. |
| X-NotificationClass | 12 | Entrega dentro de 450 segundos. |
| | 22 | Entrega dentro de 900 segundos. |

Tabela 5. Cabeçalhos que devem conter no envio da requisição para notificações do tipo Toast

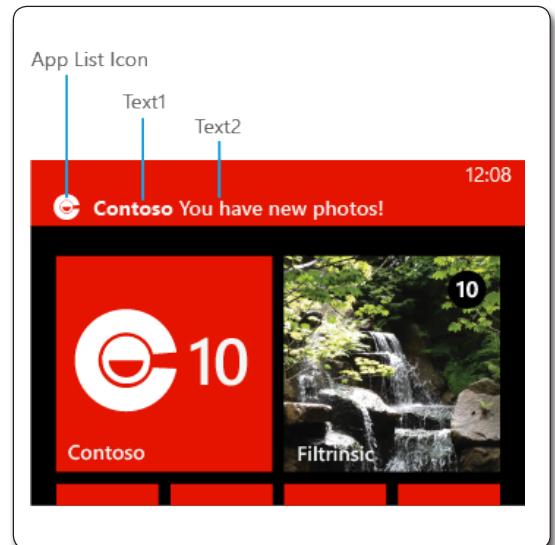


Figura 15. Visualização de cada elemento da notificação Toast - Fonte: MSDN

Listagem 12. Rotina que faz o envio da notificação do tipo Toast para o MPNS

```

01 private void EnviarToast()
02 {
03     try
04     {
05         // URI retornada pelo cliente Push quando criado o canal para notificação
06         string subscriptionUri = TextBoxUri.Text.ToString();
07
08         HttpWebRequest sendNotificationRequest =
09             (HttpWebRequest)WebRequest.Create(subscriptionUri);
10
11         // HTTP POST é o único método aceito para as notificações
12         sendNotificationRequest.Method = "POST";
13
14         // O cabeçalho X-MessagelD é opcional e pode ser usado
15         // para identificar uma mensagem de notificação.
16         // Se estiver presente, o mesmo valor é retornado na resposta da requisição.
17         // sendNotificationRequest.Headers.Add("X-MessagelD", "<UUID>");
18
19         // Criando uma mensagem para a notificação.
20         string toastMessage = "<?xml version=\"1.0\" encoding=\"utf-8\"?>" +
21             "<wp:Notification xmlns:wp=\"WPNotification\">" +
22                 "<wp:Text1>" + TextBoxTitle.Text + "</wp:Text1>" +
23                 "<wp:Text2>" + TextBoxSubTitle.Text + "</wp:Text2>" +
24                 "<wp:Param>/MainPage.xaml?TituloJogo=" +
25                     TextBoxTituloJogo.Text + "</wp:Param>" +
26                 "</wp:Text>" +
27             "</wp:Notification>";
28
29         // Convertendo para byte[].
30         byte[] notificationMessage = Encoding.UTF8.GetBytes(toastMessage);
31
32         sendNotificationRequest.ContentLength = notificationMessage.Length;
33         sendNotificationRequest.ContentType = "text/xml";
34
35         // A configuração 2 do cabeçalho X-NotificationClass
36         // serve para solicitar uma entrega imediata.
37         sendNotificationRequest.Headers.Add("X-WindowsPhone-Target", "toast");
38         sendNotificationRequest.Headers.Add("X-NotificationClass", "2");
39
40         using (Stream requestStream = sendNotificationRequest.GetRequestStream())
41         {
42             requestStream.Write(notificationMessage, 0, notificationMessage.Length);
43         }
44
45         // Recuperando a resposta do MPNS.
46         HttpWebResponse response = (HttpWebResponse)sendNotificationRequest.
47             GetResponse();
48         string notificationStatus = response.Headers["X-NotificationStatus"];
49         string notificationChannelStatus = response.Headers["X-SubscriptionStatus"];
50         string deviceConnectionStatus = response.Headers["X-DeviceConnection
51             Status"];
52
53         // Exibindo resposta recebida. Recomendado tratamento de erros,
54         // até porque as notificações podem não ser enviadas ao telefone
55         LabelRespostaMPNS.Text = notificationStatus + " | " +
56             deviceConnectionStatus + " | " + notificationChannelStatus;
57     }
58     catch (Exception ex)
59     {
60         LabelRespostaMPNS.Text = "Ocorreu uma exceção ao enviar a requisição:" +
61             ex.ToString();
62     }
63 }
```

Notificações Push no Windows Phone 8

aberta na página inicial padrão. Na **Figura 16** vemos a aplicação sendo testada, com a notificação sendo enviada pela página web e recebida pelo telefone.

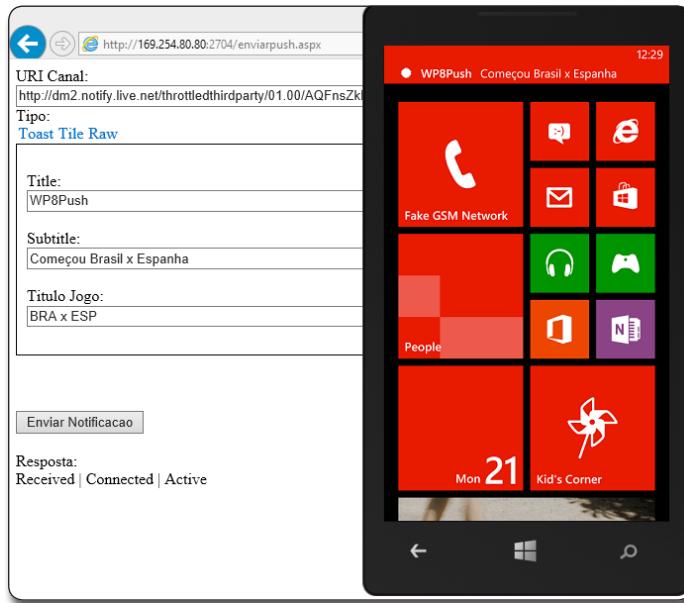


Figura 16. Notificação do tipo Toast recebida

Assim como a notificação do tipo Toast, as notificações do tipo Tile também apresentam o mesmo requisito, tanto para o preenchimento do modelo do XML, exemplificado na **Listagem 13**, como o preenchimento do cabeçalho da requisição de acordo com a **Tabela 6**. Além dos requisitos, existem outros parâmetros do XML, todos visualmente demonstrados nas **Figuras 17, 18 e 19**, que são utilizados nesse tipo de notificação, como o elemento ID, que deve ser utilizado para indicar o Tile que será atualizado. Sendo assim, o valor desse elemento deve conter a URI exata do Tile que deverá ser atualizado, caso esse atributo não seja informado, o Tile principal da App será atualizado. Cabe observar que este deve estar fixado na tela inicial para funcionar.

Outro atributo exclusivo para esse tipo de notificação é o atributo/valor Action="Clear", que quando utilizado irá limpar as informações contidas no Tile. No caso de deixar um dos elementos a ser preenchido, como wp:Count para a notificação do tipo Iconic, e não explicitar o uso do atributo Action="Clear", o número exibido no tile não será limpo, nem sobreescrito, assim o valor que estiver no momento da atualização irá permanecer. Outra observação com relação ao tipo Iconic é que o valor para o elemento BackgroundColor tem que iniciar em #FF, como #FFAB4434, senão ao invés de exibir essa cor, será exibida a cor do tema padrão do telefone.

Durante a parametrização do XML, temos que lembrar que para os tipos Flip e Iconic, as imagens podem ser carregadas do servidor Web, já para o tipo Cycle, apesar de poder exibir até nove imagens, é permitido apenas a exibição de imagens que estão no gravadas localmente telefone. Nas **Figuras 20, 21 e 22** é exibido o exemplo em execução, com as novas notificações recebidas.

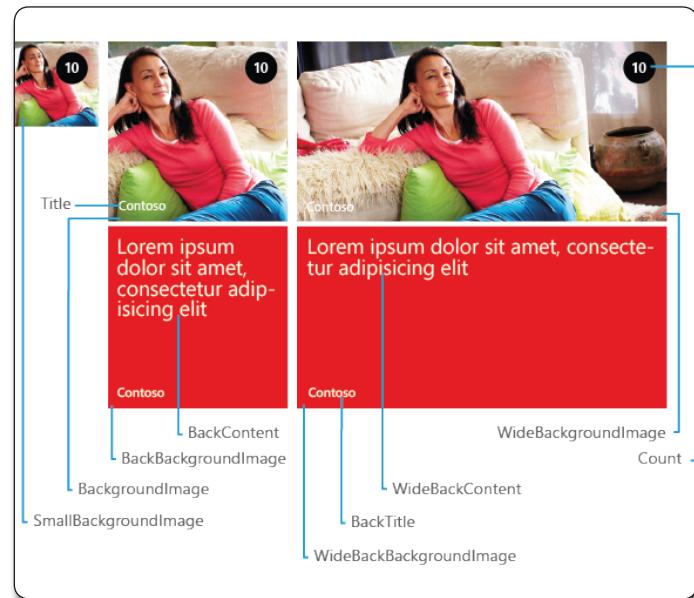


Figura 17. Visualização de cada elemento da notificação para Tiles do tipo Flip - Fonte: MSDN

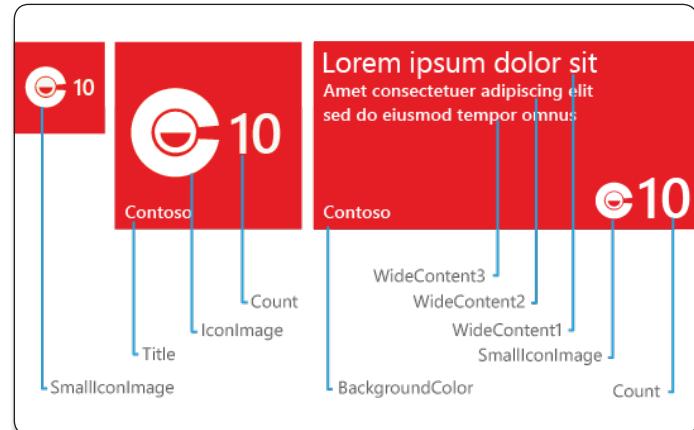


Figura 18. Visualização de cada elemento da notificação para Tiles do tipo Iconic - Fonte: MSDN

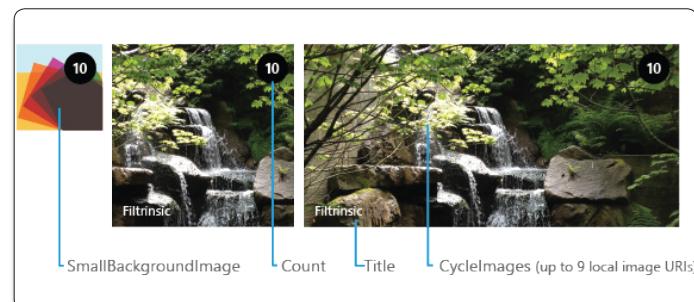


Figura 19. Visualização de cada elemento da notificação para Tiles do tipo Cycle, fonte: MSDN

| Cabeçalho da Requisição | Valor | Descrição |
|-------------------------|-------|---------------------------------|
| X-WindowsPhone-Target | token | Identificação da notificação |
| | 1 | Entrega imediata. |
| X-NotificationClass | 11 | Entrega dentro de 450 segundos. |
| | 21 | Entrega dentro de 900 segundos. |

Tabela 6. Cabeçalhos que devem conter no envio da requisição para notificações do tipo Tile.

Listagem 13. Rotina que faz o envio da notificação do tipo Tile para o MPNS

```

01 private void EnviarTile()
02 {
03     try
04     {
05         // URI retornada pelo cliente Push quando criado o canal para notificação
06         string subscriptionUri = TextBoxUri.Text.ToString();
07         HttpWebRequest sendNotificationRequest =
08             (HttpWebRequest)WebRequest.Create(subscriptionUri);
09         // HTTP POST é o unico method aceito para as notificações
10         sendNotificationRequest.Method = "POST";
11         // O cabeçalho X-MessagelD é opcional e pode ser usado
12         // para identificar uma mensagem de notificação.
13         // Se estiver presente, o mesmo valor é retornado na resposta da requisição.
14         sendNotificationRequest.Headers.Add("X-MessagelD", "<UUID>");
15         // Criando uma mensagem para a notificação.
16         string tileMessage;
17         switch (Convert.ToInt16(TipoTile.SelectedItem.Value))
18         {
19             // Iconic
20             case 1:
21                 tileMessage = "<?xml version='1.0' encoding='utf-8'?>" +
22                     "<wp:Notification xmlns:wp='WPNotification' Version='2.0'>" +
23                     "<wp:Tile Id='/MainPage.xaml?TituloJogo=BRA x" +
24                         ESP&#amp;Tile=iconic&Template='IconicTile'>" +
25                         "<wp:SmallIconImage>" + (string.IsNullOrEmpty(
26                             TextBoxIconImage.Text) ? " Action='Clear'"" : "") +
27                             + uriBaselImagesTile + TextBoxIconImage.Text +
28                             "</wp:SmallIconImage>" +
29                             "<wp:IconImage>" + (string.IsNullOrEmpty(
30                             TextBoxIconImage.Text) ? " Action='Clear'"" : "") +
31                             + ">" + uriBaselImagesTile + TextBoxIconImage.Text +
32                             "</wp:IconImage>" +
33                             "<wp:WideContent1>" + (string.IsNullOrEmpty(
34                             TextBoxContent1.Text) ? " Action='Clear'"" : "") +
35                             + ">" + TextBoxContent1.Text + "</wp:WideContent1>" +
36                             "<wp:WideContent2>" + (string.IsNullOrEmpty(
37                             TextBoxContent2.Text) ? " Action='Clear'"" : "") +
38                             + ">" + TextBoxContent2.Text + "</wp:WideContent2>" +
39                             "<wp:WideContent3>" + (string.IsNullOrEmpty(
40                             TextBoxContent3.Text) ? " Action='Clear'"" : "") +
41                             + ">" + TextBoxContent3.Text + "</wp:WideContent3>" +
42                             "<wp:Count>" + (string.IsNullOrEmpty(
43                             TextBoxIconicCount.Text) ? " Action='Clear'"" : "") +
44                             + ">" + TextBoxIconicCount.Text + "</wp:Count>" +
45                             "<wp>Title>" + (string.IsNullOrEmpty(
46                             TextBoxIconicTitle.Text) ? " Action='Clear'"" : "") +
47                             + ">" + TextBoxIconicTitle.Text + "</wp>Title>" +
48                             "<wp:BackgroundColor>" + (string.IsNullOrEmpty(
49                             TextBoxBackgroundColor.Text) ? " Action='Clear'"" : "") +
50                             + ">" + TextBoxBackgroundColor.Text + "</wp:BackgroundColor>" +
51                             "</wp:Notification>";
52             break;
53         // Flip
54         default:
55             tileMessage = "<?xml version='1.0' encoding='utf-8'?>" +
56                     "<wp:Notification xmlns:wp='WPNotification' Version='2.0'>" +
57                     "<wp:Tile Id='/MainPage.xaml?TituloJogo=BRA x &Tile=flip'>" +
58                         Template='FlipTile'" +
59                         "<wp:SmallBackgroundImage>" +
60                         (string.IsNullOrEmpty(TextBoxBackgroundImage.Text) ?
61                             " Action='Clear'"" : "") +
62                             + uriBaselImagesTile +
63                             TextBoxBackgroundImage.Text + "</wp:SmallBackgroundImage>" +
64                             "<wp:WideBackgroundImage>" +
65                         (string.IsNullOrEmpty(TextBoxBackgroundImage.Text) ?
66                             " Action='Clear'"" : "") +
67                             + uriBaselImagesTile +
68                             TextBoxBackgroundImage.Text + "</wp:WideBackgroundImage>" +
69                             "<wp:WideBackgroundImage>" +
70                         (string.IsNullOrEmpty(TextBoxBackBackgroundImage.Text) ?
71                             " Action='Clear'"" : "") +
72                             + uriBaselImagesTile +
73                             TextBoxBackBackgroundImage.Text + "</wp:WideBackgroundImage>" +
74                             "<wp:BackgroundImage>" +
75                         (string.IsNullOrEmpty(TextBoxBackgroundImage.Text) ?
76                             " Action='Clear'"" : "") +
77                             + uriBaselImagesTile +
78                             TextBoxBackgroundImage.Text + "</wp:BackgroundImage>" +
79                             "<wp:Count>" +
80                         (string.IsNullOrEmpty(TextBoxCount.Text) ?
81                             " Action='Clear'"" : "") +
82                             + ">" + TextBoxCount.Text + "</wp:Count>" +
83                             "<wp:Title>" +
84                         (string.IsNullOrEmpty(TextBoxFrontTitle.Text) ?
85                             " Action='Clear'"" : "") +
86                             + ">" + TextBoxFrontTitle.Text + "</wp:Title>" +
87                             "<wp:BackBackgroundImage>" +
88                         (string.IsNullOrEmpty(TextBoxBackBackgroundImage.Text) ?
89                             " Action='Clear'"" : "") +
90                             + uriBaselImagesTile +
91                             TextBoxBackBackgroundImage.Text + "</wp:BackBackgroundImage>" +
92                             "<wp:BackTitle>" +
93                         (string.IsNullOrEmpty(TextBoxBackTitle.Text) ?
94                             " Action='Clear'"" : "") +
95                             + ">" + TextBoxBackTitle.Text + "</wp:BackTitle>" +
96                             "<wp:BackContent>" +
97                         (string.IsNullOrEmpty(TextBoxBackContent.Text) ?
98                             " Action='Clear'"" : "") +
99                             + ">" + TextBoxBackContent.Text + "</wp:BackContent>" +
100                             "</wp:Notification>";
101             }
102         // Convertendo para byte[].
103         byte[] notificationMessage = Encoding.UTF8.GetBytes(tileMessage);
104         // Apontando o tamanho e tipo do corpo da requisição.
105         sendNotificationRequest.ContentLength = notificationMessage.Length;
106         sendNotificationRequest.ContentType = "text/xml";
107         // A configuração 1 do cabeçalho X-NotificationClass serve
108         // para solicitar uma entrega imediata.
109         sendNotificationRequest.Headers.Add("X-WindowsPhone-Target", "token");
110         sendNotificationRequest.Headers.Add("X-NotificationClass", "1");
111         using (Stream requestStream = sendNotificationRequest.GetRequestStream())
112         {

```

Notificações Push no Windows Phone 8

Continuação: Listagem 13. Rotina que faz o envio da notificação do tipo Tile para o MPNS

```
83     requestStream.Write(notificationMessage, 0, notificationMessage.Length);
84 }
85 // Recuperando a resposta do MPNS.
86 HttpWebResponse response = (HttpWebResponse)sendNotificationRequest.
GetResponse();
87 string notificationStatus = response.Headers["X-NotificationStatus"];
88 string notificationChannelStatus = response.Headers["X-SubscriptionStatus"];
89 string deviceConnectionStatus = response.Headers["X-DeviceConnectionStatus"];
90 // Exibindo resposta recebida. Recomendado tratamento de erros,
// até porque as notificações podem não ser enviadas ao telefone
91     LabelRespostaMPNS.Text = notificationStatus + " | " + deviceConnectionStatus
+ " | " + notificationChannelStatus;
92 }
93 catch (Exception ex)
94 {
95     LabelRespostaMPNS.Text = "Ocorreu uma exceção ao enviar a requisição:" +
ex.ToString();
96 }
97 }
```

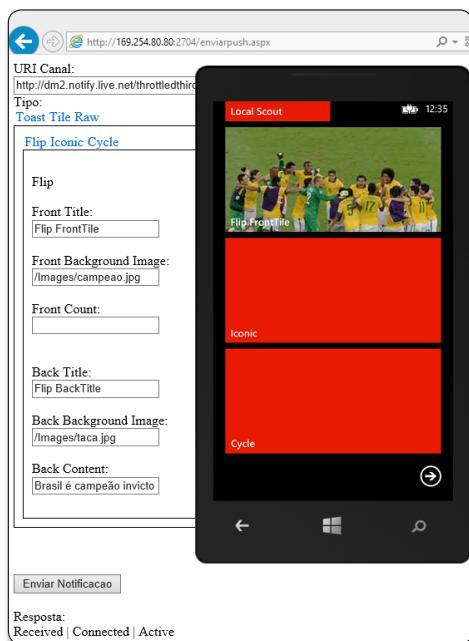


Figura 20. Notificação do tipo Tile - Flip recebida

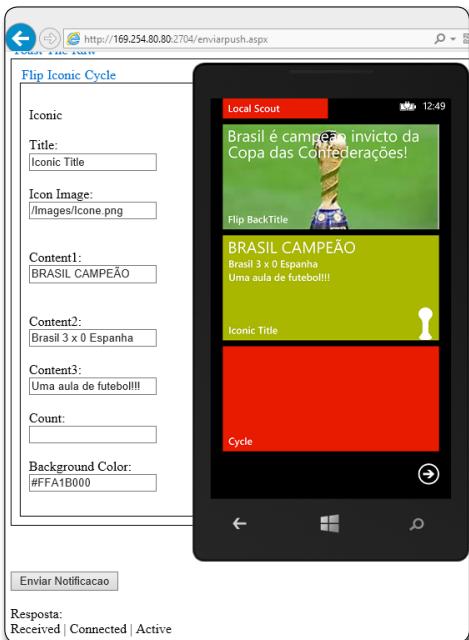


Figura 21. Notificação do tipo Tile - Iconic recebida

O último tipo demonstrado será a notificação do tipo Raw, que será recebida somente quando a App estiver aberta em primeiro plano. Por esse motivo, a sua utilização é destinada para notificações regulares que necessitem ser visualizadas apenas com a App em execução, que nesse caso, conforme a **Listagem 14** está sendo utilizada para atualizar a App com as informações do minuto a minuto do jogo de futebol. Conforme explicado anteriormente, o XML utilizado não possui formatação determinada e deve ser utilizado livremente pelo desenvolvedor. Apesar disso, a utilização do cabeçalho da requisição é obrigatória, conforme a **Tabela 7**.

| Cabeçalho da Requisição | Valor | Descrição |
|-------------------------|-------|---------------------------------|
| | 3 | Entrega imediata. |
| X-NotificationClass | 13 | Entrega dentro de 450 segundos. |
| | 23 | Entrega dentro de 900 segundos. |

Tabela 7. Cabeçalho que devem conter no envio da requisição para notificações do tipo Raw

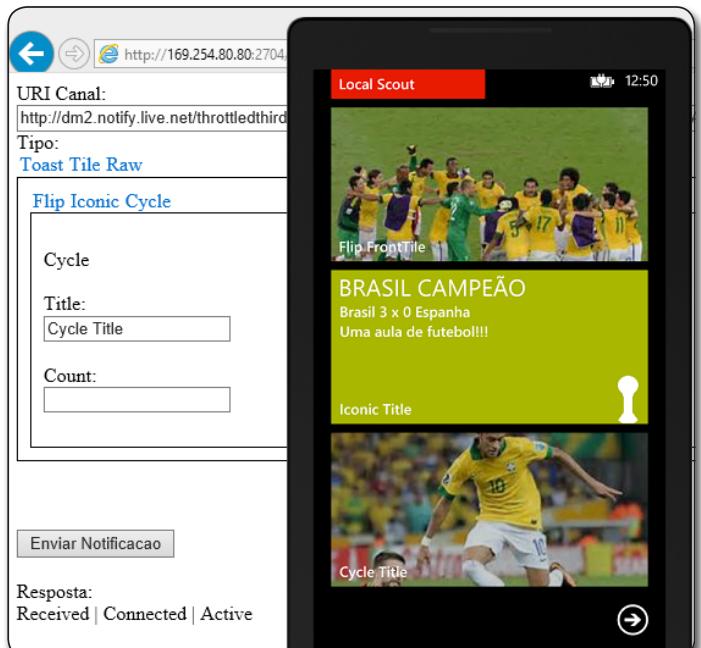


Figura 22. Notificação do tipo Tile - Cycle recebida

Listagem 14. Rotina que faz o envio da notificação do tipo Raw para o MPNS

```
01 private void EnviarRaw()
02 {
03     try
04     {
05         // URI retornada pelo cliente Push quando criado o canal para notificação
06         string subscriptionUri = TextBoxUri.Text.ToString();
07
08         HttpWebRequest sendNotificationRequest =
09             (HttpWebRequest)WebRequest.Create(subscriptionUri);
10
11         // HTTP POST é o unico method aceito para as notificações
12         sendNotificationRequest.Method = "POST";
13
14         // Criando um XML. Esse XML não tem template, o App terá que trata-lo.
15         string rawMessage = "<?xml version=\"1.0\" encoding=\"utf-8\"?>" +
16             "<WP8Push>" +
17             "<TituloJogo>" + TextBoxTituloJogoRaw.Text + "</TituloJogo>" +
18             "<InformacaoJogo>" + TextBoxInformacaoJogo.Text +
19             "</InformacaoJogo>" +
20             "</WP8Push>";
21
22         // Convertendo para byte[].
23         byte[] notificationMessage = Encoding.UTF8.GetBytes(rawMessage);
24
25         // Apontando o tamanho e tipo do corpo da requisição.
26         sendNotificationRequest.ContentLength = notificationMessage.Length;
27         sendNotificationRequest.ContentType = "text/xml";
28
29         // A configuração 3 do cabeçalho X-NotificationClass serve
30         // para solicitar uma entrega imediata.
31         sendNotificationRequest.Headers.Add("X-NotificationClass", "3");
32
33         // Enviando conteudo ao MPNS
34         using (Stream requestStream = sendNotificationRequest.
35             GetRequestStream())
36         {
37             requestStream.Write(notificationMessage, 0,
38                 notificationMessage.Length);
39         }
40
41         // Recuperando a resposta do MPNS.
42         HttpWebResponse response = (HttpWebResponse)sendNotification
43             Request.GetResponse();
44         string notificationStatus = response.Headers["X-NotificationStatus"];
45         string notificationChannelStatus = response.Headers["X-Subscription
46             Status"];
47         string deviceConnectionStatus = response.Headers["X-DeviceConnection
48             Status"];
49
50         // Exibindo resposta recebida. Recomendado tratamento de erros,
51         // até porque as notificações podem não ser enviadas ao telefone
52         LabelRespostaMPNS.Text = notificationStatus + " | " + deviceConnection
53             Status
54             + " | " + notificationChannelStatus;
55     }
56     catch (Exception ex)
57     {
58         LabelRespostaMPNS.Text = "Ocorreu uma exceção ao enviar a requisição:" +
59             ex.ToString();
60     }
61 }
```

CURSOS ONLINE

A Revista Clube Delphi oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.



CONHEÇA ALGUNS DOS CURSOS:

- **Curso de Multicamadas com Delphi e DataSnap**
- **Delphi para Iniciantes**
- **Criando componente Boleto em Delphi**
- **Loja Virtual em Delphi Prism**

Para mais informações :
www.devmedia.com.br/cursos/delphi
(21) 3382-5038

Notificações Push no Windows Phone 8

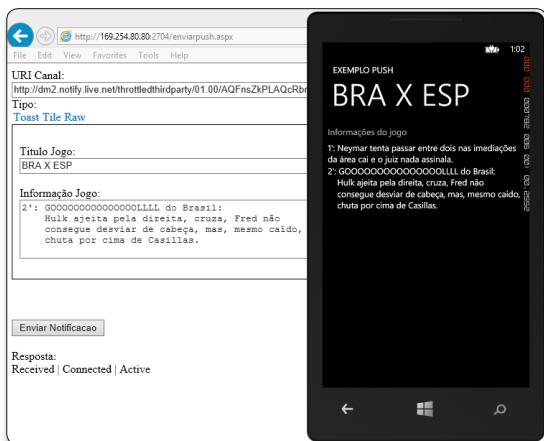


Figura 23. Notificações do tipo Raw recebidas

O resultado desse exemplo é apresentado na Figura 23.

Conclusão

A utilização das notificações Push pode fazer com que sua App tenha maior aceitação e, por consequência, alcance maior popularidade, visto que é uma funcionalidade que permite ao usuário estar sempre atualizado ou em contato mais vezes com a App, e que dá a sensação para o usuário de que ele está recebendo um conteúdo exclusivo para ele. Para isso, é necessário que seja feita uma análise em relação a escolha do tipo de notificação, levando em conta objetivo da notificação e a melhor maneira de apresentação ao usuário.

Autor



Felipe Farias Ferrari

Especialista em Desenvolvimento de Software, graduado em Ciéncia da Computação pelo Centro Universitário da FEI. Atua como Desenvolvedor .NET pela Hewlett-Packard Company, e Consultor Especializado em Arquitetura OO, Web, Mobile e Plataforma .Net.



Links:

Documentação oficial referente o Microsoft Push Notification Service

[http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff402558\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff402558(v=vs.105).aspx)

Adicionando os tipos de Notificação para Tiles do Windows Phone 8, no Windows Phone 7.8

[http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj720574\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj720574(v=vs.105).aspx)

Documentação oficial sobre notificação Push no Windows Phone

[http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh202945\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh202945(v=vs.105).aspx)

Você gostou deste artigo?

Dê seu voto em www.devmedia.com.br/netmagazine/feedback

Ajude-nos a manter a qualidade da revista!



FÓRUM DEVMEDIA

O lugar perfeito para você ficar por dentro de tudo
o que acontece nas tecnologias do mercado atual



No fórum da DevMedia você irá encontrar uma equipe disponível e altamente qualificada com consultores e colaboradores prontos para te ajudar a qualquer hora e sobre qualquer assunto. Temos as salas de Java, .NET, Delphi, Banco de Dados, Engenharia de Software, PHP, Java Script, Web Design, Automação comercial, Ruby on Rails e muito mais!

ACESSE AGORA
www.devmedia.com.br/forum

Data Binding em WPF: interagindo com o banco de dados

Como utilizar Data Binding, LINQ to SQL e MVVM em aplicações reais

O WPF fornece ao desenvolvedor uma forma muito poderosa e de fácil utilização para criar interfaces de usuário. Porém, como tem se enxergado ao longo dos últimos anos, as aplicações precisam trabalhar com dados para serem consideradas completas. Uma aplicação comercial, por exemplo, não pode simplesmente se esquivar de utilizar dados, geralmente organizados em bancos de dados relacionais. E, em tempo de execução, esses dados serão atualizados, provavelmente. A partir disso, como os controles irão saber que os dados que estão exibindo são os mais recentes?

Para essa questão, o WPF fornece uma tecnologia chamada de *Data Binding*, ou ligação de dados, em uma tradução literal. Com essa tecnologia, é possível realizar a ligação entre as propriedades de controles, uma fonte e um alvo, no caso. No cenário mais comum, isso será utilizado para realizar a conexão entre os controles que mostram dados com a fonte de dados que faz a população do mesmo. Então, quando a propriedade do objeto fonte (no caso, os dados) é alterada, a propriedade do outro objeto (no caso, o controle) é atualizada. Isso se dá de uma forma bem simplificada para o desenvolvedor, e quem cuida de todas as atividades que existem por trás dessa atualização é o WPF.

Outro ponto importante é que as aplicações necessitam de uma organização para serem mantidas. O padrão MVVM fornece ideias interessantes a esse respeito, trazendo um padrão de projetos que permite uma clara separação entre a interface de usuário e sua lógica de negócios. Isso é muito útil nesse caso, porque, obviamente, os dados devem ser buscados em algum lugar, fora dos

Resumo DevMan

Porque esse artigo é útil:

Em geral, todas as aplicações reais precisam trabalhar com dados que, normalmente, vêm na forma de um banco. Esse artigo é útil porque traz informações sobre como utilizar dados em aplicações WPF de forma segura e rápida. O Data Binding do WPF é um recurso extremamente interessante e útil para o desenvolvedor, que consegue realizar a ligação de dados sem precisar criar um código muito extenso. Além disso, o uso de LINQ se dá pelo fato de ser uma tecnologia muito útil e fácil de utilizar para acesso a bancos de dados relacionais, e o padrão MVVM torna a separação entre a lógica de negócios, os dados e a interface de usuário muito clara, criando uma maior organização do código.

arquivos XAML que compõem a interface básica do aplicativo. E o MVVM fornece as ferramentas necessárias para que isso seja realizado de forma organizada e totalmente independente. Por exemplo, os *Models* serão as classes do LINQ que representarão o banco de dados, e os *ViewModels* serão responsáveis pela ligação entre os *Models* e as *Views*, criando todas as rotinas que envolvem o banco de dados.

WPF Data Binding

O Data Binding é uma parte fundamental da plataforma do WPF. Com os poderes dessa tecnologia, é possível criar uma conexão entre as propriedades de dois objetos, permitindo que, quando a propriedade do objeto fonte é alterada, a propriedade do objeto alvo é atualizada. Um excelente exemplo é a utilização de um controle Slider para modificar a propriedade de outro controle. A mudança é refletida em tempo de execução.

Com isso, Data Binding também é um conceito chave na criação de animações em WPF. É importante notar que, em WPF, é possível criar uma ligação entre objeto e fonte de dados através de forma procedural, através de código, ou de forma declarativa, via XAML.

Basicamente, o Data Binding no WPF possui quatro peças básicas:

- Target: o objeto que irá utilizar o resultado da ligação;
- Target Property: a propriedade do objeto alvo que irá utilizar o resultado;
- Source: o objeto que fornece o valor;
- Path: o caminho que localiza o valor dentro do objeto fonte.

Dessas quatro, as duas primeiras são as mais fáceis de serem entendidas. O Target é o controle que contém a ligação, ou seja, o controle que receberá o valor que o Source irá enviar. Já a Target Property é a propriedade desse objeto alvo que irá receber o valor da ligação. É importante notar que a Target Property de um Binding deve ser uma dependency property (**BOX 1**). Como a maioria das propriedades cujo Data Binding traria alguma vantagem são dependency properties, isso acaba não sendo um grande problema. A fonte do objeto (Source) é o objeto do qual a ligação busca o valor. Esse objeto pode ser um controle, dados ou um objeto definido em code-behind.

BOX 1. Dependency Properties

O propósito de dependency properties é fornecer um meio de basear o valor de uma propriedade no valor de outras entradas. Essas outras entradas incluem propriedades de sistema, mecanismos de determinação de propriedades em tempo de execução, templates, estilos, etc. Além disso, elas podem ser implementadas para fornecer a validação dos dados que são passados a ela, valores padrão, monitoramento de mudanças em outras propriedades, entre outros. Em resumo, dependency properties são propriedades que podem ser definidas através de métodos como estilização, data binding, animação e herança.

É interessante observar que existem três tipos de propriedades que podem ser definidas para a especificação da fonte, que são ElementName, Source e RelativeSource (**BOX 2**). Só é possível especificar uma delas por vez, ou seja, o desenvolvedor precisa escolher uma, dependendo da aplicação em questão. Por fim, mas não menos importante, vem o caminho da ligação de dados (Path). Na maior parte dos casos, ele é definido pelo nome da propriedade fonte que o desenvolvedor deseja utilizar. No caso da utilização de dados de um banco relacional, ele irá representar o campo a ser mostrado.

Em WPF, alguns controles, como botões e caixas de texto, são considerados controles de conteúdo, ou content controls. Eles foram criados para mostrar apenas um dado, geralmente um texto. Outros controles, como listBoxes e comboBoxes, foram criados para mostrar uma sequência de valores de um mesmo tipo, e como tal, são considerados controles de itens, ou item controls. Como é possível perceber facilmente, realizar a ligação de dados a controles de conteúdo é muito simples e direto. Basta setar os

quatro elementos básicos (Target, Target Property, Source e Path) e está concluído o serviço. No caso de controles de itens, a coisa fica um pouco mais complexa. Para determinar o que mostrar para cada item, o controle chama o método `ToString()` da classe base. No caso de números ou strings, classes que existem por padrão, esse método retorna a representação textual dos itens. Porém, no caso de objetos mais complexos, como as classes de dados que queremos definir, ele simplesmente retorna o nome da classe. A solução mais simples é sobreescriver o método `ToString()` na classe de dados que foi definida para que ele retorne algo mais útil para o usuário. Uma solução mais elaborada, e melhor do ponto de vista da qualidade da aplicação, é criar um template que mostre os dados de outra forma. Será nesse template que será realizada a ligação com os dados da classe. Essa abordagem permite que o listBox mostre mais que um tipo de dados por linha, transformando a experiência do usuário em algo muito mais interessante.

BOX 2. Binding Source

Como foi dito anteriormente, a fonte da ligação de dados pode ser definida de três formas: através de ElementName, Source ou RelativeSource. Desses três, a primeira é a mais simples. A propriedade ElementName simplesmente recebe o nome de um controle, o controle fonte, no caso. Já a propriedade Source é um pouco mais complexa, e geralmente é definida através de um recurso estático (StaticResource). Ela identifica um objeto que deve ser utilizado como a fonte da ligação de dados, ou seja, é utilizada, geralmente, para realizar a ligação com dados oriundos de uma classe. Por fim, o RelativeSource permite que o objeto fonte seja especificado pela sua relação com o controle alvo. Normalmente é utilizado quando se desejam duas propriedades no mesmo controle. Além desses três, existe um meio adicional de definir a fonte de ligação em um Data Binding: definindo a propriedade `DataContext` do controle. Uma vez que controles herdam essa propriedade de seus ancestrais, esse método permite que vários controles dividam a mesma fonte de dados.

Outro recurso muito importante que o WPF traz para o Data Binding é a possibilidade de mudança de modos. O WPF possui quatro modos de ligação de dados que permitem ao desenvolvedor especificar a relação entre os objetos alvo e fonte. Com o modo OneWay, como o próprio nome sugere, apenas mudanças no objeto fonte são importantes, e atualizarão o objeto alvo. No modo Two-Way, a abordagem é um pouco diferente, permitindo atualização em qualquer um dos objetos para refletir as mudanças em qualquer dos dois. Já o modo OneWayToSource é o completo inverso do modo OneWay, e permite que mudanças no objeto alvo sejam refletidas no objeto fonte. Por fim, o modo OneTime permite que o objeto alvo seja alterado apenas uma vez, e as demais mudanças tanto na fonte como no alvo não são refletidas no outro.

LINQ to SQL

LINQ, ou Language Integrated Query, é uma linguagem criada pela Microsoft que permite ao desenvolvedor escrever consultas estruturadas seguras em torno de coleções locais de objetos e fontes remotas de dados. Basicamente, a LINQ permite a realização de consultas em qualquer coleção, seja um array, uma lista ou uma fonte remota de dados, como uma tabela em

um banco relacional. Os tipos fundamentais que dão suporte à LINQ estão definidos nos namespaces System.Linq, System.Linq.Expressions e System.Core.

Já a LINQ to SQL é uma das subtecnologias que permeiam a LINQ. Ela permite que o desenvolvedor utilize qualquer classe para representar dados, desde que utilize alguns atributos simples. Por exemplo, o atributo **[Table]** diz à LINQ to SQL que um objeto deste tipo é totalmente equivalente a uma linha em uma tabela do banco de dados. Ou seja, a classe em si representa uma das tabelas do banco.

Como em um diagrama E-R (Entidade-Relacionamento), em LINQ to SQL também existe o conceito de entidades. Uma classe com o atributo que define a mesma como uma tabela é chamada de entidade. É importante ter em mente que esse “mapeamento” com relação a um banco de dados, para ser útil, deve ser no mínimo aproximadamente igual à tabela do banco de dados. A linguagem traz outros atributos, como **[Column]**, responsável por definir colunas de uma tabela. Além disso, cada um dos atributos possui propriedades, que dizem se a mesma é uma chave primária, entre outras informações importantes para bancos de dados. Além disso, como o que está sendo definido é um banco de dados, também é importante definir o relacionamento entre as tabelas, através de referências a chaves estrangeiras. Isso é realizado através do atributo **[Association]**.

Com as entidades já criadas, é interessante partir para as consultas aos dados. A LINQ possui diversas formas de consulta aos dados, como consultas Lambda (**BOX 3**), consultas de compreensão e subconsultas. As consultas Lambda são mais flexíveis e fundamentais, funcionando melhor para expressões com um único operador, enquanto consultas de compreensão são muito mais simples para expressões mais complexas, e mais complexas para expressões mais simples. Depende muito do desenvolvedor qual utilizar em quais situações. É preciso analisar cada caso para ter uma noção do que é melhor.

BOX 3. Consultas Lambda

Expressões Lambda são expressões e métodos anônimos com uma sintaxe no sentido de tornar mais agradável e elegante a leitura do código. As expressões Lambda surgiram dos delegates, que nada mais são do que ponteiros para funções. Logo, expressões Lambda são funções, onde a parte à esquerda do símbolo “=>” (operador lambda) são os argumentos da função e a parte à direita da mesma contém o corpo do método. Elas são utilizadas para expressões simples de consulta a dados, por serem extremamente flexíveis e elegantes, com pouco código para realizar muita coisa.

O designer LINQ to SQL no Visual Studio traz uma ferramenta de geração automática de entidades que é muito simples e interessante. No caso de uma aplicação que possui um banco de dados existente, é muito mais interessante pedir a uma ferramenta para gerar o código automaticamente do que simplesmente construir todas as tabelas e colunas em código. Além disso, para muitos bancos de dados reais, isso é impraticável, devido à grande extensão dos mesmos. Além desse designer, existe uma ferramenta de linha de comando chamada SqlMetal que faz o mesmo serviço.

Ambas geram entidades como classes parciais, de forma que lógica adicional possa ser incorporada em arquivos separados. Mais importante, as duas ferramentas também são responsáveis pela geração das associações entre as tabelas.

Padrão MVVM (Model-View-View Model)

Para todo desenvolvedor, é importante considerar que a interface de usuário, além de expor o sistema abaixo dela, deve satisfazer os imprevisíveis requerimentos estilísticos dos usuários. Logo, ela deve ser a parte mais volátil da aplicação. Existem muitos padrões de design que auxiliam o desenvolvedor na tarefa de tornar a interface de usuário mais independente do código, entre eles o MVP (*Model-View-Presenter*), o MVC (*Model-View-Controller*) e o próprio MVVM. Com várias possibilidades, é importante entender bem todas elas, para que a melhor escolha seja feita.

Porém, antes de tudo, é interessante notar que padrões de design nem sempre são necessários. Em aplicações simples, pequenas, é totalmente desnecessário e contra produtivo utilizá-los, uma vez que qualquer desenvolvedor é capaz de entender algumas poucas linhas de código. Mas, conforme a aplicação começa a crescer, é importante, essencial até, que haja um padrão de design por trás da mesma. Isso irá auxiliar na manutenção da aplicação, irá permitir que outros desenvolvedores possam dar continuidade à mesma, além de que a organização estará lá, permitindo que a complexidade da aplicação continue a aumentar através de novas funcionalidades adicionadas. Para cada caso, é interessante analisar os possíveis padrões a serem utilizados, e escolher o que melhor se encaixa no escopo do aplicativo.

O MVP é um dos mais populares padrões de design do mercado. Ele contém três componentes: *View*, *Model* e *Presenter*. A primeira é a interface com o usuário, sendo que os dados que ela mostra compõem o *Model*, e o *Presenter* é a interface entre os dois. Já o MVC também é muito popular e vem sendo usado há décadas. MVP e MVVM nada mais são do que adaptações do MVC a alterações de tecnologia. O MVC também possui um modelo e uma *view*, porém o elemento que une os dois é o *Controller*. Existem algumas diferenças entre o *Controller* e o *Presenter*, entre as quais o fato de que o controlador é responsável por determinar qual *View* será mostrada em resposta a qualquer ação, incluindo quando a aplicação é iniciada, enquanto o *Presenter* é utilizado pela *View* para trabalhar com dados, reagir a entradas do usuário e prover validação das entradas do mesmo.

Já o padrão MVVM foi criado em 2005 por John Goosman, um dos arquitetos do WPF e Silverlight na Microsoft. Conceitualmente, o MVVM é muito parecido com o MVP, diferenciando-se pelo fato de que o MVVM é típico para WPF e Silverlight (e, posteriormente, Windows Phone e Windows Store Apps), enquanto o MVP é completamente independente de plataforma. Ou seja, no MVVM as *Views* são definidas através de arquivos XAML que contêm todos os elementos de interface com o usuário.

Em termos de utilidade, o MVVM visa estabelecer uma clara separação entre o modelo de objetos e a interface com o usuário, definida em XAML. Ele traz uma clara separação de camadas

(Figura 1), em que a *View Model* é utilizada para realizar a ligação entre a *View* e o *Model*, que não se conhecem. A comunicação entre a *View* e a *View Model* se dá através do mecanismo de *binding* do WPF, além de *routed events* e *routed commands* (BOX 4). São esses três últimos elementos que, juntos, fazem do MVVM um padrão extremamente poderoso para construção de aplicações WPF, Silverlight, Windows Phone e Windows Store.

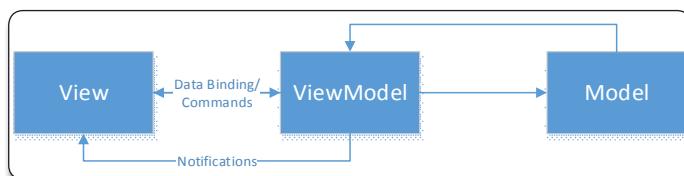


Figura 1. Interação entre camadas MVVM

BOX 4. Routed Events e Routed Commands

Eventos roteados são eventos que navegam para cima ou para baixo na árvore de visual da aplicação, dependendo da estratégia de roteamento. Isso significa que ele pode invocar manipuladores em múltiplos ouvintes ao longo da árvore de elementos, ao invés de apenas no próprio elemento que levantou o evento. As estratégias de roteamento são Tunneling, Bubbling ou Direct. Normalmente, eventos roteados aparecem em pares, sendo o primeiro um evento Tunneling e o segundo um Bubbling, sempre nessa ordem.

Comandos roteados são comandos que implementam a interface `ICommand`, ao invés de apenas herdarem da mesma, como fazem os comandos comuns. Assim como eventos roteados, eles podem navegar pela hierarquia de elementos do WPF. Basicamente, pelo fato de implementarem a interface `ICommand`, Routed Commands são comandos que não sabem o que fazem. Quando recebem a ordem de executar, eles simplesmente delegam essa responsabilidade a outro, navegando para cima ou para baixo (dependendo da estratégia de roteamento) na árvore de visual da aplicação, procurando alguém que seja capaz de executar o comando.

Esses dois elementos – Routed Events e Routed Commands – estão relacionados um ao outro. Eles formam a base para a comunicação ao longo das partes da interface de usuário, sejam essas partes controles ou mesmo toda a janela.

Criando e organizando a aplicação no padrão MVVM

Antes mesmo de se falar de forma mais aprofundada em MVVM, é preciso ter uma noção do que significa, e porque é feita, a separação entre Model e View. O modelo é a representação via software da solução a um problema, enquanto a vista é utilizada para permitir que o usuário interaja com o modelo para resolver o mesmo problema. A separação entre eles exige uma grande quantidade de trabalho extra, portanto, é preciso entender porque eles precisam ser separados. Geralmente, em software, é aceito que um design modular é melhor do que um monolítico. E designs modulares bons possuem duas características em comum: *loose coupling* e *high cohesion*. A primeira diz respeito ao nível de dependência entre os dois módulos, ou seja, quanto menos dependente os módulos são uns dos outros, melhor. Já o segundo é uma questão de coesão: o módulo deve fazer apenas aquilo a que se propõe, nem mais, nem menos. Um módulo com baixa coesão normalmente possui várias funções irrelevantes para a resolução do problema.

Os programadores vêm enxergando o valor de dividir o modelo e as views por um longo tempo. Separando esses dois elementos,

é possível obter a primeira fase na diferenciação das responsabilidades de subsistemas dentro do produto. Porém, é preciso que haja uma ligação entre eles. Implementações de padrões de projetos mais recentes, dão conta de uma arquitetura *Model-View-X*, onde X é o elemento responsável por realizar a integração entre o Model e a View. No caso do MVVM, a integração é realizada através da *ViewModel*.

A partir daqui, é interessante focar o desenvolvimento da aplicação no modelo MVVM. Como é possível notar a partir do que foi visto, a *ViewModel* é o elemento mais importante dessa trinca que compõe o MVVM. Mas, ao mesmo tempo, é a menos familiar, e a mais complicada de entender. Aqui, será criada a estrutura completa de uma aplicação com o padrão MVVM, bem como uma classe *ViewModelBase*, utilizada para trazer operações básicas que dizem respeito à *ViewModel* e serão posteriormente sobreescritas para mais especificidade.

Antes de tudo, é preciso definir o que serão os Models, as *ViewModels* e as Views. As Views são as mais simples de serem definidas. Como se trata de uma aplicação WPF, as vistas serão os arquivos XAML contendo as interfaces com o usuário. Além disso, eles serão responsáveis por realizar o Data Binding com as *ViewModels*. Já os modelos são um pouco mais complexos. Aqui, será criada apenas uma pasta no projeto, chamada *Models*, que conterá todos os modelos de dados. Essas classes modelo serão criadas posteriormente, quando o banco de dados for criado e mapeado como classes do LINQ. Por fim, e mais importante, é preciso que se criem as *ViewModels*. Elas serão colocadas em uma pasta separada no projeto, com esse mesmo nome, e essa pasta, inicialmente, conterá apenas a classe *ViewModelBase*, que será criada a seguir. Vale ressaltar que esse modelo de pastas em um projeto só não é o único possível para aplicações com o padrão MVVM. Models, Views e *ViewModels* poderiam ser criados em projetos separados, referenciando uns aos outros. A Figura 2 mostra a organização do projeto no Visual Studio.

Nota

É interessante ressaltar que, no caso de projetos diferentes para Models, Views e *ViewModels*, é preciso haver referências entre eles. Porém, essas referências devem ser cuidadas, pois é sabido que, pelo padrão MVVM, *ViewModels* referenciam Models e Views referenciam *ViewModels*. Qualquer referência direta entre Views e Models pode levar a problemas no futuro, seja ela implícita ou explícita, além de quebrar o padrão MVVM.

A partir daí, é preciso que haja alguma coisa na classe básica das *ViewModels*, ou seja, a classe será utilizada como uma base para as demais. Ela irá simplesmente implementar a interface `INotifyPropertyChanged`. Isso é necessário para que os *ViewModels* sejam capazes de entender quando as propriedades são alteradas pela View. Isso é fundamental, pois, essencialmente, não haverá nenhum tipo de código administrável na View, de acordo com o padrão MVVM. A questão aqui é que a interface em questão contém apenas um evento, chamado `PropertyChanged`, que deve ser implementado. Esse evento deve ser público, pois pode

Data Binding em WPF: interagindo com o banco de dados

ser acessado por qualquer classe. Para as classes que implementam esse evento, faz sentido escrever um ajudante simples, que simplifica a chamada do evento. Esse método é aqui chamado de OnPropertyChanged(string prop). A **Listagem 1** mostra o resultado da classe base de ViewModels. Posteriormente, será visto como esse método é utilizado para controlar a mudança de propriedades na View.

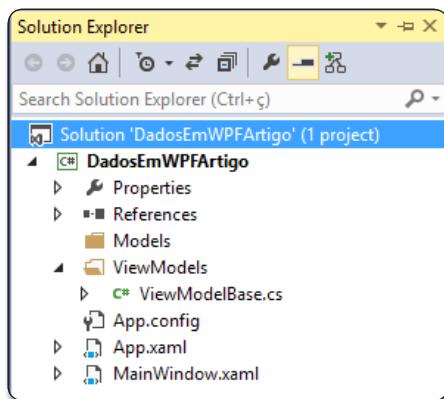


Figura 2. Organização do projeto

Listagem 1. Classe ViewModelBase

```
01 namespace DadosEmWPFArtigo.ViewModels
02 {
03     class ViewModelBase : INotifyPropertyChanged
04     {
05         public event PropertyChangedEventHandler PropertyChanged;
06         protected virtual void OnPropertyChanged(string prop)
07         {
08             if (PropertyChanged != null)
09             {
10                 PropertyChanged(this, new PropertyChangedEventArgs(prop));
11             }
12         }
13     }
14 }
```

A partir daí, a classe básica está definida. Porém, com essa classe base, a comunicação entre ViewModel e View será realizada somente através de eventos. Mas essa comunicação também pode ser realizada através de comandos. Para isso, será criada uma classe básica, chamada RelayCommand, ou comando de transmissão, em uma tradução literal. É a classe base de todos os comandos que poderão ser chamados na ViewModel que será criada. Essa classe é vista em muitos projetos MVVM, e se tornou quase um padrão. Embora varie de projeto para projeto, em essência ela é sempre muito parecida, servindo como uma base a qualquer comando. Porém, é preciso entender que existem alguns requisitos que a classe RelayCommand deve atender. Ela deve manter a separação entre os controles e o método que irá ser chamado, a implementação deve ser tão artificial quanto for necessário, nem mais, nem menos, fazendo com que a mesma seja extremamente intuitiva para usar, e também que dependências extras devem ser mantidas ao mínimo, sempre lembrando que dependências

em excesso podem quebrar o padrão MVVM, gerando problemas futuros.

Com essas técnicas de design em mente, a criação da classe RelayCommand é muito simples. Ela implementará a interface ICommand e, como tal, terá à disposição os métodos CanExecute(), Execute() e o evento CanExecuteChanged, que como o nome sugere, será disparado quando a possibilidade de execução passar de true para false ou vice-versa. Como é visível na **Listagem 2**, trata-se de uma implementação bastante simples, genérica, mas que contém todos os elementos necessários a um comando, sendo intuitiva para usar. Como vemos na linha 27, a tarefa de executar o comando é delegada ao delegate Action<object>, portanto não há sobrecarga de dependências extras, o que aconteceria se fosse utilizado um CommandExecuteHandler, por exemplo. Outro delegate utilizado com o mesmo propósito foi o Predicate<object>, que evita uma chamada ao CanCommandExecuteHandler. Como a presença do canExecute é opcional, foram criados dois construtores (linhas 03 e 05). Já o método CanExecute() simplesmente diz que caso o campo _canExecute não tenha sido especificado no construtor, o comando pode ser executado sim, delegando a responsabilidade da execução para o Predicate<object>. Por fim, o evento criado diz respeito a quando o método CanExecute() muda seu valor de retorno. Como é possível ver nas linhas 20 e 21, o CommandManager (**BOX 5**) fornece um evento chamado RequerySuggested, que é chamado quando ele acredita que houve mudança no valor de retorno.

Listagem 2. Classe RelayCommand

```
01 class RelayCommand : ICommand
02 {
03     public RelayCommand(Action<object> execute) : this(execute, null)
04     {}
05     public RelayCommand(Action<object> execute,
06                         Predicate<object> canExecute)
07     {
08         if (execute == null)
09         {
10             throw new ArgumentNullException("execute");
11         }
12         _execute = execute;
13         _canExecute = canExecute;
14     }
15     public bool CanExecute(object parameter)
16     {
17         return _canExecute == null ? true : _canExecute(parameter);
18     }
19     public event EventHandler CanExecuteChanged
20     {
21         add { CommandManager.RequerySuggested += value; }
22         remove { CommandManager.RequerySuggested -= value; }
23     }
24     public void Execute(object parameter)
25     {
26         _execute(parameter);
27     }
28     private readonly Action<object> _execute;
29     private readonly Predicate<object> _canExecute;
29 }
```

Seu conhecimento tem visibilidade na DevMedia

Publique artigos e vídeos no site e ganhe mais visibilidade em seu trabalho. Somos o maior site do Brasil para desenvolvedores.

Para publicar é bem simples: mande um email para nosso editor **Wesley Yamazack (yamazack@gmail.com)**.



Para mais informações acesse:
[http://www.devmedia.com.br/
seja-um-autor-devmedia/28103](http://www.devmedia.com.br/seja-um-autor-devmedia/28103)

Data Binding em WPF: interagindo com o banco de dados

BOX 5. CommandManager

É uma classe selada que provê métodos úteis para utilização em comandos. Além disso, a classe fornece serviços que permitem descobrir o status de um comando. Os métodos úteis variam desde métodos de registro de objetos de binding até handlers de eventos de comandos. Como visto há pouco, o único evento da classe é o `RequerySuggested`, que ocorre quando ele detecta condições que podem alterar a capacidade do comando de executar.

Criando o banco de dados com LINQ

Como foi visto, a LINQ to SQL oferece uma série de ferramentas que permitem ao desenvolvedor representar dados através de classes. Para isso, basta que estas sejam decoradas com alguns atributos especiais, como `[Table]` ou `[Column]`. Cada atributo possui um propósito e várias propriedades, todas elas dizendo respeito a bancos de dados. Por exemplo, o atributo `[Column]` possui uma propriedade chamada `IsPrimaryKey`, do tipo `boolean`, para indicar se a coluna corresponde à chave primária da tabela. Esse é um conceito unicamente de bancos de dados relacionais, o que reforça a ideia de que a LINQ foi feita para que as classes sejam transformadas em tabelas de bancos de dados.

Para este exemplo, será criada uma tabela de consumidores, chamada `Customer`. Essa tabela conterá as colunas correspondentes ao Nome do consumidor, seu CPF e seu Endereço, apenas, como uma forma de simplificação da mesma, além, obviamente, de seu identificador único, `Id`. Como é possível observar na **Listagem 3**, a classe `Customer` possui o atributo `[Table]` decorando-a, indicando que se trata de uma tabela. Além disso, também se vê que todas as propriedades da classe foram transformadas em colunas, através do atributo `[Column]`. É possível notar uma propriedade comum a todos eles, chamada `Storage`. Essa propriedade é responsável por indicar ao banco de dados que o local de onde o dado deve ser buscado, e onde ele deve ser armazenado, não é aquela propriedade pública ali, e sim o campo privado que foi declarado no começo da classe.

Listagem 3. Classe de dados Customer

```
01 [Table]
02 public class Customer
03 {
04     private int id;
05     private string name;
06     private string cpf;
07     private string endereco;
08     [Column (IsPrimaryKey=true, Storage="id")]
09     public int Id {...}
10     [Column (Storage="name")]
11     public string Name {...}
12     [Column (Storage="cpf")]
13     public string Cpf {...}
14     [Column (Storage="endereco")]
15     public string Endereco {...}
16 }
```

Porém, como é facilmente perceptível, essa classe é apenas uma representação do banco de dados; o banco em si ainda não foi criado. Para isso, será criada uma segunda classe, que representará

o `DataContext` do banco de dados. Na LINQ, é essa classe que efetua a comunicação com a fonte de dados. A classe que será criada deve derivar da classe base em `System.Data.Linq.DataContext`. Ela contém a informação e os métodos para efetuar a conexão com o banco de dados, além de manipular esses dados. Como é visível na **Listagem 4**, essa classe simples, `DBDataContext`, apenas possui um campo, `cust`, para a tabela de consumidores do banco de dados. Essa será a única tabela do banco, mas, caso existissem outras, elas também seriam definidas aqui. Também é possível ver que o construtor da classe simplesmente chama o construtor da classe base `DataContext`, com a string de conexão ao banco. Essa abordagem de classes parciais é dita como fortemente tipada, e é muito mais elegante e recomendada que a abordagem comum, onde se instancia um `DataContext` genérico e passa-se as tabelas a ele.

Listagem 4. Classe DBDataContext

```
public partial class DBDataContext : DataContext
{
    public Table<Customer> cust;
    public DBDataContext(String connectionString) : base(connectionString)
    {...}
}
```

Mas um problema persiste. Embora a classe de conexão ao banco de dados tenha sido criada, o banco em si ainda não existe. Para isso, será criada uma terceira classe, chamada `CustomerData`. Essa classe será responsável pela criação e por toda a manipulação dos dados na aplicação. Ou seja, ela será responsável por adicionar, atualizar, excluir e buscar dados no banco. E, para isso, ela utilizará o `DBDataContext` criado anteriormente. Inicialmente, conforme mostrado na **Listagem 5**, a classe simplesmente irá, em seu construtor, realizar uma chamada ao `DataContext`. Com isso, irá ser realizada uma simples verificação: se o banco de dados existe ele continua, se não existe ele cria o banco. Isso é uma função que a LINQ traz que é muito importante, pois ele garante que o banco de dados existirá antes de começar a realizar qualquer operação nele.

Listagem 5. Construtor da Classe CustomerData

```
01 public CustomerData()
02 {
03     using (var c = new DBDataContext(conn))
04     {
05         if (!c.DatabaseExists())
06         {
07             c.CreateDatabase();
08         }
09     }
10 }
```

A seguir, temos a criação dos métodos responsáveis pela adição, atualização e exclusão de linhas do banco de dados, além do método responsável por criar uma lista de consumidores. Esses

métodos foram criados também dentro da classe CustomerData, e utilizam novamente o mesmo DBDataContext que é responsável pela conexão ao banco de dados. O método Add(Customer c), basicamente, realiza a inserção do consumidor passado como parâmetro na tabela do banco de dados. Isso é realizado conforme a **Listagem 6**, que mostra que primeiramente é realizado um InsertOnSubmit(c), na tabela de consumidores do banco, e posteriormente essas mudanças são enviadas ao banco de dados pelo DataContext. Os demais métodos, Update(Customer c) e Delete(Customer c) seguem o mesmo padrão, recuperando, entretanto, o consumidor do banco de dados através de uma consulta, para realizar a alteração e a exclusão dos dados. Isso é necessário, pois o método não sabe que o valor passado por parâmetro corresponde a um dado do banco de dados. A **Listagem 7** mostra em detalhes esse comportamento. Por fim, o método List() retorna uma lista que contém todos os elementos da tabela Customer do banco de dados.

Listagem 6. Método Add da classe CustomerData

```
01 public void Add(Customer c)
02 {
03     using (var ctx = new DBDataContext(conn))
04     {
05         ctx.cust.InsertOnSubmit(c);
06         ctx.SubmitChanges();
07     }
08 }
```

Listagem 7. Método Delete da classe CustomerData

```
01 public void Delete(Customer c)
02 {
03     Customer aux = new Customer();
04     using(var ctx = new DBDataContext(conn))
05     {
06         var query = from cust in ctx.cust
07                 where cust.Id == c.Id select cust;
08         foreach (Customer cus in query)
09         {
10             aux = cus;
11         }
12         ctx.cust.DeleteOnSubmit(aux);
13     }
14 }
```

Como vimos, essas três classes são os modelos da aplicação. Elas contêm a representação dos dados, bem como o acesso aos mesmos. Porém, esses dados ainda estão completamente sem uso, isolados nos Models. Para que eles possam ser acessados, é necessário que ViewModels sejam criadas para buscar esses dados e enviá-los, de alguma forma, para a View exibi-los e para que o usuário tenha alguma liberdade para realizar alguma atividade com esses dados. Isso será feito a seguir.

Criando a ViewModel

Agora, chegou a hora da criação das ViewModels. Como sabemos, pelo padrão MVVM, cada uma das Views deve possuir uma

ViewModel, que irá realizar toda a interface entre o Model e a própria View. Ou seja, como a aplicação exemplo possui apenas uma View (MainWindow), só existirá uma ViewModel, que será chamada de MainWindowViewModel. É importante notar que, como a declaração da classe base de ViewModels tinha apenas um evento, é ele que será utilizado para saber quando a propriedade será alterada pelo usuário.

Como é sabido, a comunicação entre as ViewModels e as Views é realizada através de eventos e comandos. Não é necessário declarar ambos, mas pelo menos um deles é sempre preciso; do contrário não haverá meio de comunicação entre a ViewModel e a View e a aplicação ficará completamente sem propósito. Neste caso, a comunicação entre ambos consistirá da verificação de se a propriedade do objeto do Binding foi alterada ou não, como já foi comentado. Além disso, como também foi criada uma classe base de comandos, a ViewModel irá utilizar essa classe base para realizar comandos de adição, remoção e atualização no banco de dados.

Antes da criação das ViewModels, é preciso ter em mente que as ViewModels são modelos das Views. Embora óbvio, a maior parte dos desenvolvedores não dá atenção a esse detalhe, e acaba não sabendo exatamente onde colocar o código: na ViewModel ou no code-behind da View. Pode ser tentador colocar o código diretamente na View, por uma simples questão de conveniência. Porém, isso quase sempre acarreta problemas futuros e, como foi explicada anteriormente, uma boa aplicação MVVM precisa ser loosely coupled, ou seja, quanto menos dependente a View for da ViewModel, e vice-versa, melhor. É importante limitar as Views a apenas o código que é específico aos eventos, elementos, recursos, entre outros. Essa separação é um objetivo que vale a pena, e deve ser muito claro quando da criação das aplicações.

Nessa criação de ViewModel específica, toda a lógica será feita através da ViewModel, como será visto. Ou seja, a View não terá responsabilidade alguma, em termos de lógica de negócios, confiando na ViewModel para isso, o que é totalmente condizente com o padrão MVVM. É importante ter em mente que, como a ViewModel irá referenciar o Model, ela poderá utilizar algumas ações que foram colocadas no modelo. Em outras palavras, as ações que o Model realiza e que devem ser expostas à View devem ser utilizadas pela ViewModel. Aqui, vale um adendo: o modelo apenas define funções. Já a ViewModel deve definir comandos ou eventos que sirvam para utilizar essas funções na View, através do Data Binding.

Com esses conceitos em mente, é hora da construção da ViewModel em si. Como foi criada anteriormente uma classe base de ViewModels, chamada ViewModelBase, que implementa INotifyPropertyChanged, não é necessário fazer isso agora. Basta que a classe que será criada herde dessa classe básica. Como vimos, ela possui apenas um evento e um método que trata esse evento. Ou seja, todo o resto terá que ser implementado pela MainWindowViewModel, a ViewModel específica nesse caso.

Primeiramente: a classe deve referenciar o modelo. Ou seja, é necessário trazer um objeto do tipo CustomerData para que ele

Data Binding em WPF: interagindo com o banco de dados

possa ter acesso ao banco de dados. Além disso, a classe CustomerData traz uma série de métodos, como vimos anteriormente, para adição, remoção e atualização de dados no banco. Agora, na ViewModel, é preciso realizar a ponte entre o modelo e a vista. Basicamente, o que será feito aqui é a “transformação” desses métodos em comandos que podem ser entendidos pela View. Repare que poderiam ser utilizados eventos, sem problema nenhum. Porém, a abordagem por comandos se justifica devido à sua maior simplicidade de operação e codificação.

Em um segundo momento, é preciso estabelecer os requisitos. É importante lembrar que antes de tudo, a ViewModel é uma classe e, como tal, ela terá atributos, propriedades e métodos. Portanto, em termos de atributos, é necessário que haja um do tipo CustomerData, para criação do banco de dados. Esse objeto será inicializado no construtor da classe e somente lá. Por outro lado, também é necessária a criação de três objetos do tipo RelayCommand, que serão os comandos de adição, remoção e atualização do banco. Também são necessários dois atributos do tipo Customer, uma vez que a View precisa indicar quem será adicionado, atualizado ou removido do banco de dados. Um deles será o consumidor atual, que pode ser excluído ou atualizado, enquanto o outro será o consumidor a ser adicionado. Tudo isso é mostrado na **Listagem 8**. Todos esses atributos possuirão suas respectivas propriedades, para que possam ser acessados pelo arquivo XAML.

Listagem 8. Atributos e Construtor da classe *MainWindowsViewModel*

```
01 class MainWindowViewModel : ViewModelBase
02 {
03     private CustomerData cData;
04     private RelayCommand addCommand;
05     private RelayCommand updateCommand;
06     private RelayCommand delCommand;
07     private Customer currentCustomer;
08     private Customer newCustomer;
09
10    public MainWindowViewModel()
11    {
12        cData = new CustomerData();
13    }
14    ...
15 }
```

A questão dos comandos é importante de ser ressaltada, uma vez que não é muito trivial. Caso não seja a primeira vez que eles estão sendo chamados, ou seja, o comando é diferente de nulo, o código irá criar uma nova instância de RelayCommand, para tratar daquela chamada de comando. Cada um dos comandos define uma chamada de função como parâmetro, sendo que essas funções são simplesmente chamadas às aquelas definidas no Model CustomerData. Além disso, eles só possuem o getter; o setter é desnecessário, uma vez que eles não receberão atribuições – será somente leitura. A **Listagem 9** mostra o resultado para o comando de adição. Os demais comandos (atualização e remoção) utilizam o consumidor atual (currentCustomer) para realizar a operação, conforme dito anteriormente.

Listagem 9. Comando Add

```
01 public RelayCommand AddCommand
02 {
03     get
04     {
05         if (addCommand == null)
06         {
07             addCommand = new RelayCommand(param => this.Add());
08         }
09         return addCommand;
10     }
11 }
12 public void Add()
13 {
14     cData.Add(newCustomer);
15     ListData = cData.List();
16 }
```

Por fim, é preciso que a ViewModel utilize a lista que foi criada na classe CustomerData. Para isso, foi criada uma propriedade chamada ListData, que receberá a lista no construtor da ViewModel. Ou seja, quando for criada uma instância da ViewModel, será criada a lista que corresponde a todos os dados do banco. Esses dados serão atualizados a cada comando, como foi visto na **Listagem 9**. É importante ressaltar a necessidade de se referenciar a propriedade ListData, e não o atributo listData. Isso é essencial porque, do contrário, não haveria chamada ao método OnPropertyChanged() (pois não seria a propriedade que mudaria) e o listBox da View não seria atualizado.

Realizando o Data Binding com a View do WPF

Finalmente, é a hora de realizar o tão falado Data Binding entre a MainWindowViewModel e a View. Para tanto, são necessários alguns elementos. O primeiro e principal deles: é preciso referenciar o namespace dos ViewModels no código XAML. Com isso, é possível referenciar qualquer uma das classes (no caso, as ViewModels) dentro do código da janela. A partir disso, é preciso definir um recurso que será utilizado por toda a janela da aplicação. Ou seja, esse recurso será definido localmente, e será simplesmente a classe MainWindowViewModel. Ele é declarado como recurso porque contém uma série de elementos que foram criados especialmente para a View ter acesso aos dados dos modelos. Por fim, é preciso definir os controles que receberão esses dados, e farão também a sua atualização, mas vamos por etapas. A **Listagem 10** mostra a importação do namespace e definição do recurso MainWindowViewModel. O namespace foi rotulado como *vm*, e o recurso, *vmMainWindow*.

Listagem 10. Código inicial da View

```
01 <Window x:Class="DadosEmWPFArtigo.MainWindow"
02     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
03     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
04     xmlns:vm="clr-namespace:DadosEmWPFArtigo.ViewModels"
05     Title="MainWindow" Height="350" Width="525">
06     <Window.Resources>
07         <vm:MainWindowViewModel x:Key="vmMainWindow"/>
08     </Window.Resources>
09     <Grid DataContext="{StaticResource vmMainWindow}">
```

Agora que há uma definição com relação ao recurso a ser utilizado, é interessante preparar a interface. Trata-se de uma interface bastante simples, dividida, basicamente, em três partes: uma com os campos para adição de novos consumidores, uma com os campos para atualização e exclusão dos mesmos e outra com o listBox que contém todos os consumidores do banco de dados. A **Figura 3** mostra como ficou a interface, ainda sem nenhum dado. Repare que, como não há nenhum dado, o listBox à direita se encontra totalmente vazio. É interessante ressaltar um detalhe com relação à **Listagem 10**. Na última linha, é definido um DataContext para o Grid. Os controles WPF possuem a capacidade de herdar propriedades de seus “pais”. Com isso, todos os controles “filhos” do Grid que não possuírem uma definição de DataContext irão procurar, e encontrar essa definição, no próprio GridView.

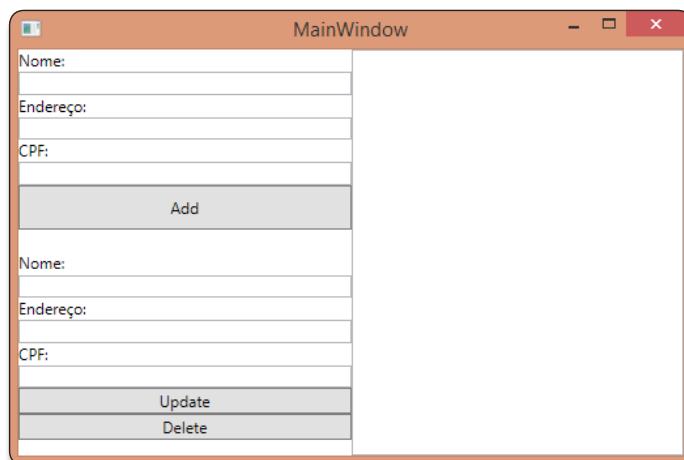


Figura 3. UI sem Data Binding

A primeira ação a ser realizada é garantir que o listBox irá receber os dados referentes do banco. Ou seja, realizar um DataBinding com a propriedade ListData, da MainWindow-ViewModel. Como se trata de um listBox, a propriedade que deve ser setada com esse binding é a propriedade que contém a fonte dos itens, ou ItemsSource. Essa propriedade foi definida, conforme a **Listagem 11**. Como é observável nessa mesma listagem, a propriedade DisplayMemberPath foi definida para mostrar o nome do consumidor no listBox. Caso essa propriedade não fosse definida, ele chamaría o método ToString() da classe Customer, o que não é aconselhável, pois mostraria uma saída mais ou menos como mostra a **Figura 4**. O método ToString() poderia ser sobreescrito, porém essa não é uma prática recomendada de programação. Caso o interessante seja mostrar algo mais elaborado no listBox, é recomendado o uso de DataTemplates (**BOX 6**).

Listagem 11. Código do ListBox

```
<ListBox Grid.Column="1" Grid.Row="0" Grid.RowSpan="18"
    ItemsSource="{Binding Path=ListData, Mode=OneWay}"
    DisplayMemberPath="Name" Name="IstDados"/>
```

BOX 6. DataTemplates

Templates de dados são utilizados para mostrar dados de um jeito particular. Basicamente, eles são utilizados juntamente com objetos complexos, ou seja, objetos que possuem várias propriedades básicas, como os Customers que foram criados nesse artigo. No caso, é possível mostrar em listBoxes, comboBoxes e controles similares vários tipos de itens de uma maneira particular, definida no próprio template. Como um exemplo, com o advento desses templates, é possível para o desenvolvedor mostrar Nome, Endereço e Telefone dos consumidores de uma forma organizada e rica. É um dos principais diferenciais do WPF para criação de interfaces de usuário ricas.

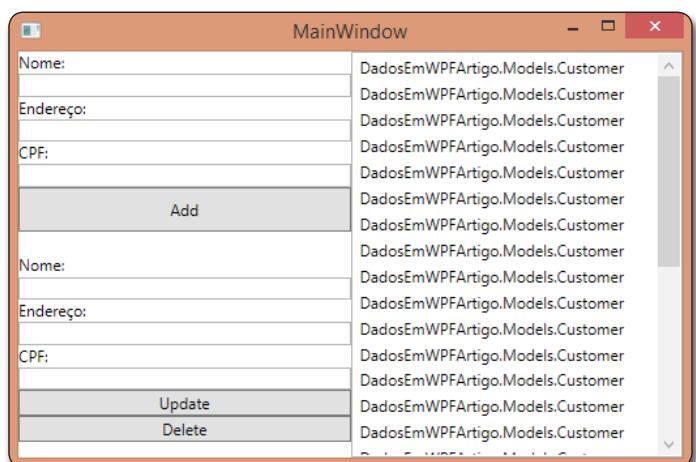


Figura 4. Saída sem definição da propriedade a ser mostrada

Agora que o listBox já está habilitado para mostrar os objetos Customer, é necessário criar meios de adição, atualização e exclusão desses mesmos objetos através do XAML. Para isso, serão utilizados os vários textBoxes criados à esquerda da interface de usuário. Para tanto, é necessária a realização de uma DataBinding. Para a adição, essa ligação é realizada com a propriedade NewCustomer, que é enviado juntamente com o comando de adição para ser adicionado ao banco de dados. Como é visível na **Listagem 12**, trata-se de um binding de modo OneWayToSource. Isso é realizado dessa forma, pois não há nenhuma necessidade de os textBoxes receberem de volta os dados que estão sendo enviados para o banco de dados. É interessante ressaltar também o binding adicionado ao comando do botão, pois ele irá simplesmente realizar uma chamada e executará o comando de adição definido na ViewModel.

Porém, com isso ainda existem alguns textBoxes sem utilização. Eles serão utilizados para realizar a atualização de dados do banco. Os três receberão os dados do item selecionado no listBox, sendo responsáveis pela atualização do mesmo quando houver mudanças. O código se assemelha muito àquele verificado para a adição de um novo consumidor, com algumas diferenças básicas. Ao invés de realizar a ligação com a propriedade NewCustomer, ele liga os dados com a propriedade CurrentCustomer. Além disso, esse novo binding utiliza uma propriedade diferente, chamada UpdateSourceTrigger, nesse caso o evento PropertyChanged. Esse gatilho é utilizado cada vez que a propriedade é alterada, e funciona para garantir que o valor será atualizado quando alterado.

Data Binding em WPF: interagindo com o banco de dados

Já o comando de exclusão é chamado através do botão Delete, que é responsável por chamar o comando através do mecanismo de binding e excluir a linha correspondente ao CurrentCustomer no banco. Por fim, a Figura 5 mostra o resultado desses DataBindings na interface com o usuário. Por questões de simplificação, o código XAML desses textBoxes foram omitidos desse artigo, mas o leitor pode ter acesso ao código completo da aplicação, na página de download da revista, no portal DevMedia.

Listagem 12. DataBinding para adição de Customers

```
01 <TextBlock Text="Nome:" Grid.Row="0"  
      Grid.Column="0"/>  
02 <TextBox Text="{Binding Path=NewCustomer.Name,  
      Mode=OneWayToSource}"  
03 Grid.Row="1" Grid.Column="0"/>  
04 <TextBlock Text="Endereço:" Grid.Row="2"  
      Grid.Column="0"/>  
05 <TextBox Text="{Binding Path=NewCustomer.Endereco,  
      Mode=OneWayToSource}"  
06 Grid.Row="3" Grid.Column="0"/>  
07 <TextBlock Text="CPF:" Grid.Row="4" Grid.Column="0"/>  
08 <TextBox Text="{Binding Path=NewCustomer.Cpf,  
      Mode=OneWayToSource}"  
09 Grid.Row="5" Grid.Column="0"/>  
10 <Button Content="Add" Grid.Column="0" Grid.Row="6"  
     Grid.RowSpan="2"  
11 Command="{Binding Path=AddCommand}"/>
```

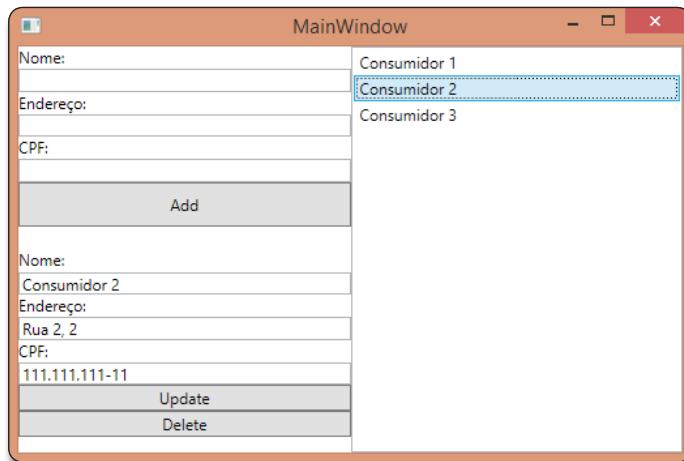


Figura 5. UI concluída

Conclusão

Esse artigo trouxe uma nova abordagem a respeito de Data Binding em WPF. Além disso, trouxe uma série de conceitos, teóricos e práticos, a respeito de MVVM, que não possui muita literatura a respeito, e a respeito de LINQ to SQL, com a criação de um banco de dados puramente em classes do C#. São ferramentas muito poderosas, que podem ser utilizadas juntas ou separadamente, trazendo uma excelente forma de trabalhar com dados em aplicações WPF.

O MVVM é um padrão de design que foi criado principalmente para criação de aplicações WPF e Silverlight. Isso porque, na época, eram as duas tecnologias permeadas pelo XAML. Hoje em dia, há outras tecnologias que utilizam o poder dessa linguagem de marcação e, por isso, o MVVM se aplica a elas também. Realizando uma citação, aplicações para Windows Phone e Windows Store utilizam essa linguagem para criação de interfaces com o usuário. Além do MVVM, outros padrões podem ser utilizados tranquilamente para criação de aplicações empresariais, como MVP ou MVC. Todos eles possuem suas particularidades, vantagens e desvantagens. A principal vantagem na abordagem com MVVM é que ele foi criado para ser utilizado com o WPF, sendo, portanto, o melhor meio de tirar vantagem dos poderes dessa tecnologia.

Por fim, a questão do acesso a dados também poderia ter sido realizada de outra forma. A escolha por LINQ to SQL se deu pelo fato de ser uma tecnologia do .NET framework que simplificou muito as coisas para o acesso e consulta a dados. O principal poder da tecnologia LINQ são suas consultas a dados, que podem ser feitas de várias formas. Além disso, a LINQ oferece uma forma interessante, simples e intuitiva de representar dados no formato de classes de dados, o que encaixa perfeitamente no modelo MVVM para aplicações WPF, como foi visto ao longo do artigo.

Autor



Henrique Machado Gasparotto

hmgasparotto@hotmail.com

Estudante de Engenharia de Computação na Universidade Federal de Santa Maria – UFSM e Técnico em Informática pelo SENAC Santa Maria. Experiência em programação C# .NET e Java. Atualmente, trabalha como bolsista no Laboratório de Computação para Clima Espacial no Centro Regional Sul de Pesquisas Espaciais (LCCE/CRS/INPE), com atividades voltadas à computação de alto desempenho.



Links e Referências:

LINQ to SQL: .NET Language-Integrated Query for Relational Data

<http://msdn.microsoft.com/en-us/library/bb425822.aspx>

Entendendo o pattern Model-View-ViewModel (MVVM)

<http://imasters.com.br/artigo/18900/desenvolvimento/entendendo-o-pattern-model-view-viewmodel-mvvm/>

Data Binding (WPF)

[http://msdn.microsoft.com/en-us/library/ms750612\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/ms750612(v=vs.110).aspx)

Você gostou deste artigo?

Dê seu voto em www.devmedia.com.br/netmagazine/feedback

Ajude-nos a manter a qualidade da revista!





DEVMEDIA

DÊ UM SALTO EM CONHECIMENTO!

Acesse o maior
portal para
desenvolvedores
da América
Latina!



20
mil
posts

430
mil
cadastrados

10
milhões de
page-views
por mês

Integration Services 2012: implementando soluções de ETL - Parte 1

Construindo soluções para a integração entre sistemas



ESTE ARTIGO FAZ PARTE DE UM CURSO

A necessidade de implementação de projetos envolvendo a integração de diferentes sistemas é algo extremamente comum nos dias atuais. As soluções resultantes podem se basear tanto no compartilhamento de dados em tempo real através do uso de Web Services, quanto na transferência em lotes e num horário específico de informações que partam de uma origem para um determinado repositório de destino.

Embora Web Services representem um meio rápido e instantâneo para a troca de dados entre aplicações, o comum é que a utilização destes componentes esteja restrita a contextos caracterizados por um tráfego menor de informações. Por outro lado, grandes quantidades de dados são geralmente manipuladas por softwares executados dentro de intervalos de tempo pré-definidos, de forma a não comprometer com isto a performance na utilização de um ou mais sistemas.

No caso específico desta segunda alternativa, as rotinas responsáveis por essas tarefas de integração costumam ser implementadas empregando as mais variadas tecnologias. É bastante comum que tais soluções possuam um código extenso e complexo, dificultando não apenas a sua manutenção, como também o entendimento do processo como um todo. A inexistência de uma documentação que detalhe o fluxo das diferentes atividades realizadas durante o processamento só vem agravar esta situação, não sendo raro que modificações em aplicações deste tipo produzam efeitos indesejáveis e exijam esforços posteriores na correção de problemas.

Resumo DevMan

Porque esse artigo é útil:

Este artigo tem por finalidade apresentar o SQL Server Integration Services 2012 e de que maneira este serviço (que é parte integrante do SQL Server) pode ser útil na integração entre diferentes aplicações. Combinando recursos do .NET Framework a funcionalidades para a manipulação de bases relacionais, XML e outros formatos para a representação de informações, esta ferramenta permite a implementação dos mais variados tipos de soluções voltadas ao gerenciamento de dados. A implementação de projetos baseados no Integration Services acontece através do Visual Studio, sendo que demonstraremos isto através de dois exemplos (o primeiro envolvendo a exportação de informações, ao passo que um segundo exemplo descreve de que maneira informações podem ser carregadas em uma base de dados).

O SQL Server Integration Services (SSIS) representa a solução oferecida pela Microsoft para atender a esses cenários de integração. Combinando a utilização de tecnologias heterogêneas para a manipulação de dados (tais como bancos de dados relacionais, XML, arquivos texto, planilhas do Excel, dentre outras alternativas) a recursos do .NET Framework, esta ferramenta possibilita a construção de aplicações capazes de se comportar de uma forma robusta e escalável (**BOX 1**) diante de grandes volumes de informações.

Outro ponto que pesa a favor do uso do Integration Services está na construção de projetos a partir da modelagem gráfica de fluxos divididos em tarefas. Isto contribui para tornar mais simples o desenvolvimento de novas aplicações, evitando assim a implementação de extensos trechos de código que coordenem as interações entre atividades. O entendimento de como uma solução do Integration Services funciona também se torna mais intuitivo,

característica esta que facilita o aprendizado desta ferramenta por parte de desenvolvedores que não estejam familiarizados com a mesma.

A meta da primeira parte deste artigo é apresentar as principais características do Integration Services e de que maneira esta ferramenta pode ser útil na integração entre sistemas. Para isto será apresentada uma aplicação que aborda a exportação de dados de uma base relacional, gerando como resultado arquivos no formato .csv. Numa próxima edição demonstraremos o processo inverso, com a importação de informações para um banco de dados a partir de arquivos em diferentes formatos.

BOX 1. Escalabilidade

O conceito de escalabilidade refere-se à capacidade de um sistema se adequar a uma demanda crescente no seu uso, sem que com isto deixe de corresponder ao que se esperava inicialmente para o mesmo.

SQL Server Integration Services: uma visão geral

O SQL Server Integration Services pode ser considerado uma evolução de outra ferramenta de ETL (**BOX 2**) da Microsoft: trata-se do Data Transformation Services (DTS), que foi disponibilizado quando do lançamento da versão 7 do SQL Server. Já a primeira versão daquilo que atualmente se conhece como Integration Services surgiu com o SQL Server 2005.

Desde então, cada novo release do SQL Server vem acompanhado por avanços no que se refere aos recursos oferecidos pelo Integration Services. Na versão 2012 isto não foi diferente, com melhorias em questões como a configuração e o deployment de soluções, monitoramento de aplicações e resolução de problemas, mecanismos que possibilitam uma maior qualidade em transformações de dados, dentre outros aspectos. Uma relação completa dos novos recursos do Integration Services pode ser encontrada num dos endereços listados na seção de [Links](#) deste artigo.

BOX 2. ETL

O processo conhecido como ETL (sigla do inglês “Extraction, Transformation and Load”) funciona, em termos gerais, da seguinte forma: a partir de um repositório de origem serão extraídos dados, com os mesmos sendo transformados seguindo critérios específicos a um determinado contexto. Por fim, tais dados serão carregados numa base de destino. Soluções próprias de ETL podem vir a ser desenvolvidas, muito embora existam ferramentas práticas e bastante flexíveis como o SQL Server Integration Services.

Do ponto de vista arquitetural, uma aplicação do Integration Services é constituída por um ou mais packages (pacotes). Já um package é formado por tasks (tarefas) que seguem geralmente um fluxo de execução contínuo. A implementação de um pacote é feita por meio do Visual Studio, através da modelagem gráfica de um fluxo com as tarefas que irão compor o mesmo; essa última característica torna a construções de soluções para o Integration Services um processo intuitivo e, portanto, mais fácil de ser as-

similado por desenvolvedores que começam a ter os primeiros contatos com esta ferramenta.

Na prática, um package nada mais é do que um arquivo XML com a extensão .dtsx, podendo combinar instruções para o acesso a bancos de dados, arquivos e outros tipos de repositórios a trechos de código baseados em recursos da plataforma .NET. No caso específico da versão 2012, o SQL Server conta com o suporte para bibliotecas geradas sob do .NET Framework 4.0, sendo que essa característica se estende ao Integration Services.

Packages podem acessar repositórios de dados dos mais variados tipos:

- Bases de dados relacionais que suportem drivers ODBC, OLE DB e ADO.NET;
- Bases multidimensionais mantidas através da ferramenta Analysis Services (**BOX 3**);
- Arquivos de texto;
- Planilhas do Excel;
- Web Services.

BOX 3. Analysis Services

O Analysis Services é um serviço que integra o SQL Server, sendo utilizado na construção de aplicações baseadas em conceitos de Business Intelligence. Esta ferramenta viabiliza a realização de sofisticadas análises, as quais normalmente servem de suporte para a tomada de decisões no meio corporativo. Isto acontece através da implementação de bases históricas alimentadas a partir de bancos de dados transacionais, sendo que tais repositórios costumam ser modelados de uma forma multidimensional (empregando para isto estruturas como cubos e dimensões).

Quanto à sua forma de execução, packages podem ser acionados das seguintes maneiras:

- A partir de uma aplicação .NET que tenha acesso ao diretório em que se encontra um determinado package;
- Remotamente, através de instruções .NET para se acessar um package previamente instalado num servidor SQL Server.

Maiores detalhes a respeito do funcionamento e da implementação de packages do Integration Services serão abordados mais adiante nos exemplos práticos deste artigo.

Possíveis cenários de utilização do Integration Services

Graças à sua flexibilidade e ao suporte no acesso a diferentes repositórios de dados, diversos são os cenários em que o Integration Services poderá ser empregado. A seguir estão listadas algumas das possibilidades de uso desta ferramenta:

- Na replicação de informações entre diferentes bases de dados;
- Na integração entre sistemas fiscais/contábeis e órgãos governamentais, sendo possível destacar iniciativas como o SPED (sigla de “Sistema Público de Escrituração Digital”). Este último é um projeto do governo federal que procura automatizar o recebimento de informações relativas a Livros Fiscais, Contábeis e de PIS / COFINS mantidos por organizações dos mais variados segmentos;

- Em processos de ETL, os quais normalmente estão associados à produção de dados para a geração de complexas análises em soluções de BI (**BOX 4**);
- Na exportação de dados sob a forma de arquivos, em situações que geralmente envolvem a integração entre um sistema de origem e outras aplicações. Exemplos disso é a geração de listas de preço disponibilizadas por um fornecedor ou fabricante de um conjunto de produtos, o envio de informações para cobrança bancária utilizando o padrão conhecido como CNAB, dentre outros processos relacionados ao compartilhamento de dados.

BOX 4. Business Intelligence

BI (sigla em inglês para “Business Intelligence”; termo também conhecido como “Inteligência Empresarial”) é um conjunto de técnicas que envolvem a coleta, o processamento e a geração de informações a partir de dados gerados em operações cotidianas de uma organização. Este tipo de procedimento busca, a partir de tais práticas, prover análises que auxiliem profissionais de gestão em atividades relacionadas à tomada de decisões.

Exemplos de soluções utilizando o Integration Services

A fim de demonstrar como o SQL Server Integration Services 2012 pode ser utilizado, serão criadas quatro soluções no Microsoft Visual Studio 2012 Professional:

- TesteIntegration01: solution em que constará um projeto do Integration Services para a exportação de informações de produtos sob a forma de arquivos .csv. Existirão ainda nesta solução três projetos do tipo Console Application, sendo que os mesmos apresentam diferentes maneiras de se executar um package do Integration Services por meio de recursos da plataforma .NET;
- TesteIntegration02: nesta solução será demonstrada a importação de arquivos .csv, .xlsx e .txt a partir de aplicações do Integration Services. Será criado para isto um projeto em que serão carregadas informações sobre notas fiscais que envolvam o pagamento de serviços contratados por uma empresa hipotética.

Antes de iniciar a implementação destas Solutions, será necessário instalar o pacote “Microsoft SQL Server Data Tools - Business Intelligence for Visual Studio 2012” (caso este procedimento ainda não tenha sido realizado anteriormente). Essa extensão é responsável por ativar dentro do Visual Studio 2012 templates que permitem a construções de soluções baseadas no Integration Services, Analysis Services e Reporting Services (**BOX 5**).

Implementando o primeiro exemplo

Conforme já discutido na parte inicial deste artigo, arquivos ainda representam um dos principais meios para o compartilhamento de informações entre sistemas. Muitas organizações investem pesado na automação de processos que envolvem a transferência de informações, construindo assim soluções que permitem a manipulação de grandes volumes de informações em formatos padronizados.

Neste primeiro exemplo estaremos implementando uma solução chamada TesteIntegration01, a qual será responsável pela geração

de um arquivo CSV (**BOX 6**) contendo o catálogo de uma distribuidora de produtos alimentícios fictícia. O arquivo em questão não possuirá cabeçalho, sendo formado pelos seguintes campos (estes já na sequência esperada, devendo ainda estar separados por ponto-e-vírgula):

- Código do produto;
- Nome do produto;
- Categoria;
- Fornecedor;
- Preço Unitário.

BOX 5. Reporting Services

O Reporting Services é uma solução da Microsoft para a criação e o gerenciamento de relatórios no ambiente corporativo. Trata-se de uma ferramenta extremamente flexível, que possibilita inclusive a exportação de relatórios para formatos como planilhas do Excel e documentos PDF. Muito embora corresponda a mais um dos serviços que compõem o SQL Server, o Reporting Services também pode ser empregado na produção de relatórios que acessem outras bases relacionais (como Oracle, por exemplo), bancos multidimensionais do Analysis Services ou, até mesmo, fontes de dados como XML e Web Services.

BOX 6. Arquivos CSV

CSV (sigla do inglês “Comma-separated values”) é um padrão para a representação de dados em um formato tabular. Arquivos deste tipo possuem a extensão .csv e, em termos práticos, nada mais são do que sequências de texto separadas por um caractere especial (o mais comum é que se utilize vírgula ou ponto-e-vírgula em tais representações). O uso de arquivos .csv para a integração entre diferentes sistemas corresponde a um tipo de prática bastante comum, sendo que o próprio pacote Office (por meio do Excel) oferece suporte a este tipo de formato.

Nota

Na seção de [Links](#) desse artigo é possível encontrar o endereço para download da extensão “Microsoft SQL Server Data Tools - Business Intelligence for Visual Studio 2012”.

No caso específico dos campos “Nome do Produto”, “Categoria” e “Fornecedor”, as informações associadas aos mesmos serão sequências de texto com todos os caracteres em maiúsculo. Já o campo “Preço Unitário” será um valor numérico com duas casas decimais (estas últimas separadas por vírgula), ao passo que a coluna “Código do Produto” corresponderá a um número inteiro.

Eventuais clientes desta companhia receberiam tal catálogo, com o preço de comercialização sugerido para cada item. A base que fornecerá essas informações é o banco de dados Northwind, com que pode ser obtido a partir de um endereço listado na seção de [Links](#) deste artigo.

A Solution TesteIntegration01 será formada pelos seguintes projetos:

- TesteIntegration01.SSIS: aplicação baseada no Integration Services e na qual acontecerá a geração do arquivo CSV com os dados do catálogo de produtos;
- TesteIntegration01.ExecucaoLocalPackage: Console Application em que será demonstrada a execução de um package localizado em um determinado diretório;

- TesteIntegration01.ExecucaoRemotaPackage: outro projeto do tipo Console Application, sendo que esta aplicação apresentará como executar via .NET Framework um package previamente instalado num servidor SQL Server;
- TesteIntegration01.ExecucaoRemotaJob: neste último projeto (também uma Console Application) será demonstrado como acionar um Job do SQL Server que executa uma aplicação do Integration Services.

O projeto TesteIntegration01.SSIS será gerado a partir do template “Integration Services Project” (**Figura 1**), o qual é habilitado após a instalação da extensão “Microsoft SQL Server Data Tools - Business Intelligence for Visual Studio 2012”.

O próximo passo agora consistirá na criação do package GeracaoCSVProdutos.dtsx (**Figura 2**), removendo ainda o arquivo Package.dtsx (que foi gerado automaticamente ao se executar a etapa anterior).

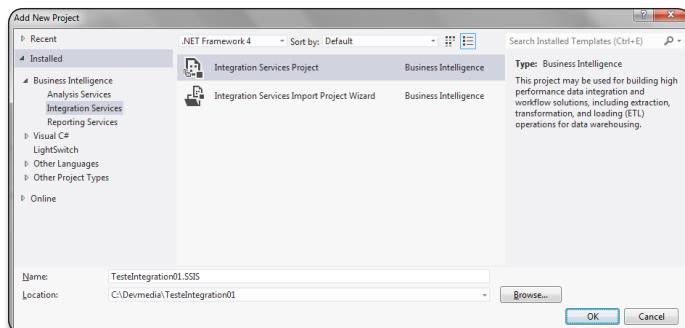


Figura 1. Criando o projeto TesteIntegration01.SSIS

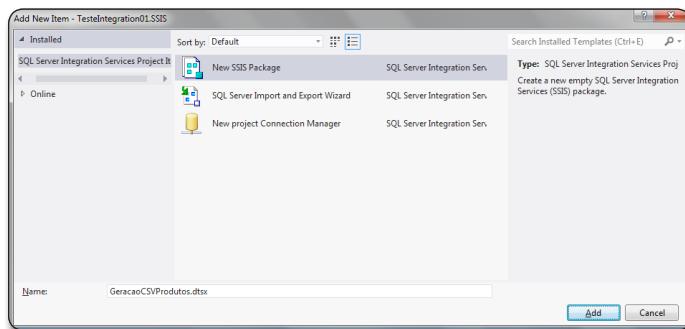


Figura 2. Criando o package GeracaoCSVProdutos.dtsx

Neste momento já será possível iniciar a implementação da aplicação para a exportação de informações sobre produtos. Conforme é possível observar na **Figura 3**, o package GeracaoCSVProdutos.dtsx conta com as seguintes seções:

- Control Flow: corresponde ao “centro nervoso” de um pacote no Integration Services. É neste local em que acontece a orquestração das diferentes tarefas (tasks) definidas para um package, seguindo para isto um fluxo de execução que obedece a uma sequência lógica;
- Data Flow: o principal tipo de funcionalidade oferecido pelo Integration Services consiste na extração de dados de um local

de origem, alguma forma de processamento produzindo um tipo específico de informação e, finalmente, a transferência disto para um ponto de destino que pode ser um banco de dados relacional ou, mesmo, um arquivo que faça uso de um padrão específico de representação (XML, planilhas do Excel, CSV, texto com posições delimitadas, etc.). Conjuntos de instruções definidos nesta seção são executados através de tarefas acionadas dentro do fluxo principal (Control Flow) de um package;

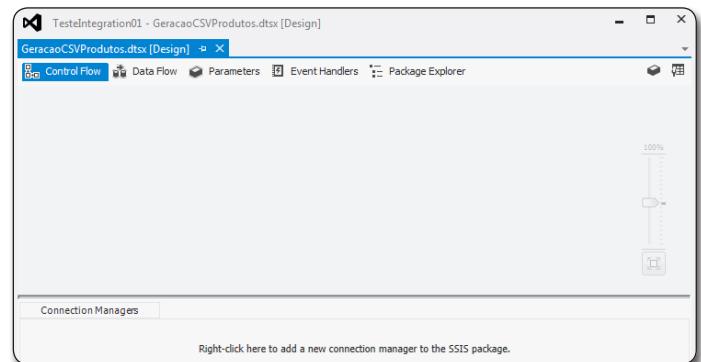


Figura 3. O package GeracaoCSVProdutos.dtsx ainda não implementado

- Parameters: aqui são declarados itens de configuração (**BOX 7**) que servem de base para a execução de um package;
- Event Handlers: permite a implementação de rotinas (Event Handlers) para o tratamento de eventos que ocorram ao longo do processamento de um package. É possível também a definição de rotinas em resposta a eventos que associados a elementos específicos de um package. Quanto à utilização de estruturas desse tipo, é comum que Event Handlers sejam implementados em tarefas que envolvam tarefas como logging e tratamento de erros;
- Package Explorer: neste local é possível a visualização e o acesso a todos os elementos definidos dentro de um package;
- Connection Managers: itens declarados nesta seção possibilitam a conexão de elementos de um package a repositórios de dados como bases relacionais e/ou arquivos dos mais variados tipos.

BOX 7. Parâmetros no Integration Services 2012

As estruturas conhecidas como Parameters (parâmetros) passaram a fazer parte do Integration Services com o lançamento do SQL Server 2012. Em termos práticos, um parâmetro nada mais é do que uma configuração formada por um identificador, além de um valor previamente definido. Importante destacar que tal valor não sofre alteração durante o processamento da aplicação que se está considerando.

Também merece destaque o fato de que este recurso surgiu com o intuito de simplificar o gerenciamento de configurações para aplicações do Integration Services, algo que em versões anteriores era feito por meio do uso de arquivos no formato XML.

Todo parâmetro é criado com base em um tipo de dado específico. Atualmente, esses itens podem ser declarados com base nos tipos: Boolean, Byte, DateTime, Decimal, Double, Int16, Int32, Int64, SByte, Single, String, UInt32 e UInt64.

Quanto ao escopo de execução, parâmetros podem ser definidos de forma global (acessíveis, portanto, a todos os packages de um projeto), ou ainda, dentro de um package específico (sendo utilizados neste último caso apenas por elementos que constem nessa estrutura).

No caso do package GeracaoCSVProdutos.dtsx, estaremos definindo um parâmetro do tipo String chamado “pCaminhoGeracaoArquivosCSV” (**Figura 4**). Esta configuração indicará o caminho no qual serão gerados os arquivos .csv contendo o catálogo de produtos (num primeiro momento, partiu-se da premissa que existirá o caminho “C:\Devmedia\TesteVisualStudio\” na máquina em que a aplicação de exemplo for executada).

A interação entre as diferentes tarefas definidas em um package é possível graças ao uso de variáveis (**BOX 8**). Para o package GeracaoCSVProdutos.dtsx, estará sendo criada a variável “vNomeArquivoCSV” (**Figura 5**). Embora preenchida com o valor “C:\Devmedia\TesteVisualStudio\Produtos.csv” (para efeitos de configuração dentro do Visual Studio), esta referência do tipo String terá o seu valor gerado dinamicamente, armazenando o nome completo para o arquivo de produtos que será criado durante a execução do pacote GeracaoCSVProdutos.dtsx. Além disso, possuirá um escopo que a torna acessível a todos os elementos deste package.

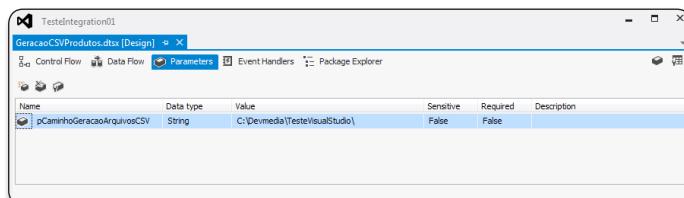


Figura 4. Criando o parâmetro pCaminhoGeracaoArquivosCSV

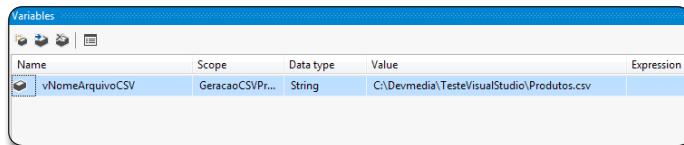


Figura 5. Criando a variável vNomeArquivoCSV

Nota

Dentro do Visual Studio, a janela para definição de variáveis pode ser acessada através do menu SSIS, opção Variables.

BOX 8. Variáveis no Integration Services

Variáveis são elementos que auxiliam na manipulação de dados e na comunicação entre as diferentes tasks que compõem um package do Integration Services.

Assim como acontece nas linguagens de programação convencionais, uma variável possui um identificador e armazena um valor baseado em um tipo de dado específico. Para este tipo de estrutura, o Integration Services disponibiliza os seguintes tipos de dados: Boolean, Byte, Char, DateTime, DBNull, Decimal, Double, Int16, Int32, Int64, Object, SByte, Single, String, UInt32 e UInt64. Diferentemente daquilo que acontece com parâmetros, os valores associados variáveis podem ser modificados em tempo de execução. Graças a esta característica, variáveis representam a alternativa ideal para a manipulação de dados em cenários com um comportamento mais dinâmico, tais como loopings sobre coleções de valores, desvios condicionais e construções de código .NET que envolvam a produção de novos valores a partir de dados pré-existentes.

Variáveis também possuem um escopo. Ao criar um novo item deste tipo, é possível definir que tal elemento estará disponível a todos os elementos de um package ou, até mesmo, apenas para conjuntos de elementos específicos.

Duas conexões também deverão ser criadas a partir da seção Connection Managers.

A primeira destas conexões (chamada “connNorthwind”) faz uso do ADO.NET como mecanismo de acesso a dados, permitindo a execução de consultas à base Northwind (**Figura 6**). É justamente este o repositório de onde serão obtidas as informações para a geração do arquivo com a listagem de produtos.

Já a conexão “connArquivoCSVProdutos” será empregada na gravação de novos arquivos .csv contendo o catálogo de produtos.

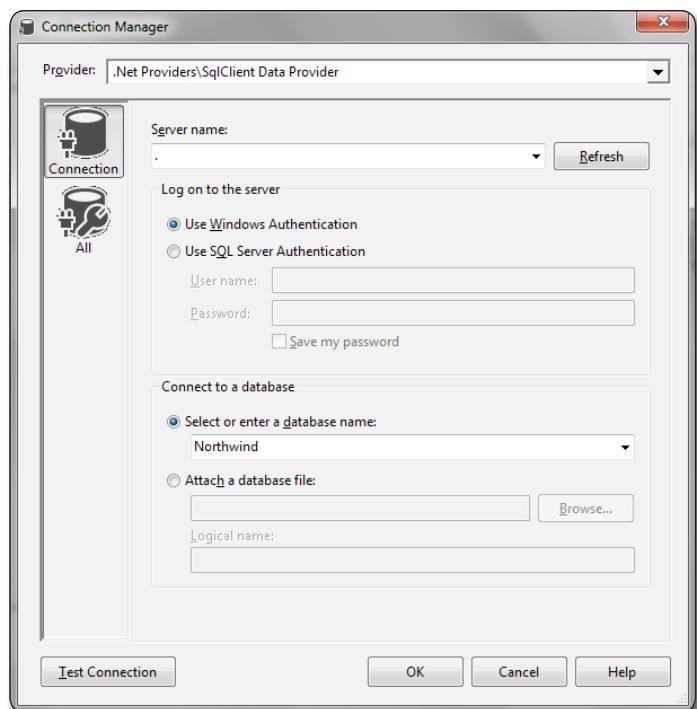


Figura 6. Configurações da conexão connNorthwind

Nota

A conexão connNorthwind foi criada a partir da opção “New ADO.NET Connection...”, a qual está disponível através do menu de atalho da seção Connection Managers.

Nota

A conexão connArquivoCSVProdutos foi gerada através da opção “New Flat File Connection...”, que também está localizada no menu de atalho da seção Connection Managers.

Na **Figura 7** estão algumas das configurações necessárias para a geração do arquivo .csv. Esses ajustes foram efetuados dentro da seção “General” na janela “Flat File Connection Manager Editor”:

- No campo “Connection manager name” foi informado o valor “connArquivoCSVProdutos”, ao passo que em Description está um comentário que descreve a finalidade desta nova conexão;
- O campo “File name” foi preenchido com um caminho que aponta para um novo arquivo .csv a ser criado durante o processamento do package GeracaoCSVProdutos.dtsx (apenas para efeito de

configuração da conexão “connArquivoCSVProdutos”. Embora com um conteúdo fixo, este valor será gerado posteriormente de forma dinâmica (por meio da variável “vNomeArquivoCSV”);

- O item “Locale” foi configurado de maneira que a criação do arquivo .csv aconteça em conformidade com os padrões de formatação regionais adotados no Brasil;
- Já no campo Format foi marcada a opção “Delimited”, que indica a geração de um arquivo delimitado por um separador (característica típica do formato CSV);
- O elemento “Header row delimiter” determina o conjunto de caracteres empregado na quebra de linhas em um arquivo;
- Por fim, a opção “Column names in the first row” desmarcada fará com que o arquivo .csv seja criado sem um cabeçalho.

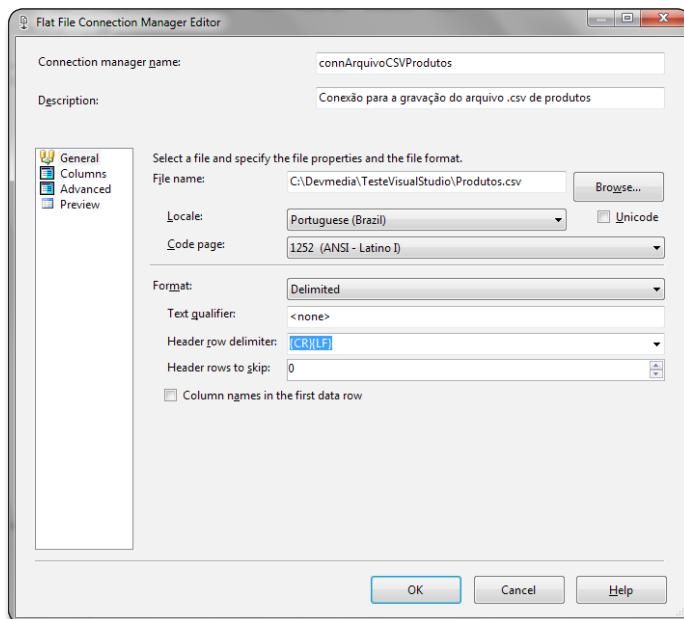


Figura 7. Configurando a conexão connArquivoCSVProdutos

Na seção “Columns” foi selecionada a opção “Semicolon {;}” para o campo “Column delimiter” (Figura 8). Com essa configuração determinamos que todos os valores do arquivo .csv estarão separados por ponto e vírgula.

Na Tabela 1 estão as configurações para cada uma das colunas que farão parte do arquivo .csv contendo o catálogo de produtos. Estes itens deverão ser preenchidos na seção “Advanced” da janela “Flat File Connection Manager Editor” (Figura 9).

| Name | DataType | OutputColumnWidth | DataScale |
|---------------|----------------------------------|-------------------|-----------|
| CodProduto | four-byte signed integer [DT_I4] | - | - |
| NomeProduto | string [DT_STR] | 40 | - |
| Categoria | string [DT_STR] | 15 | - |
| Fornecedor | string [DT_STR] | 40 | - |
| PrecoUnitario | decimal [DT_DECIMAL] | - | 2 |

Tabela 1. Colunas a serem configuradas para a conexão connArquivoCSVProduto

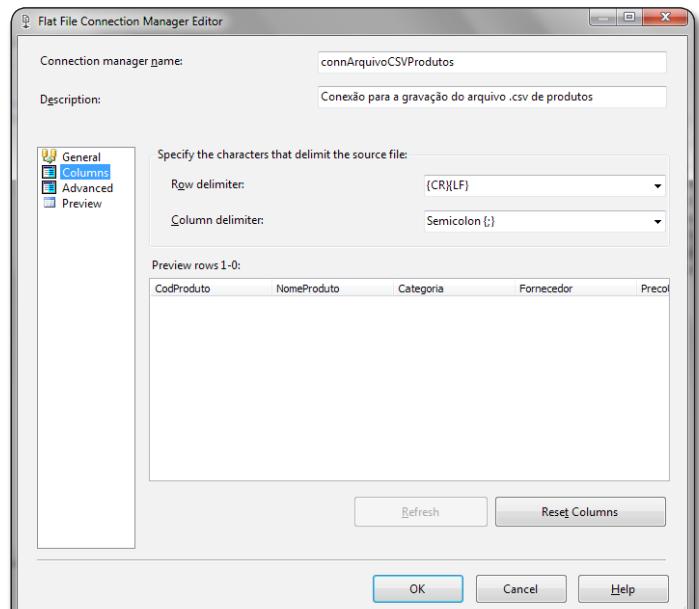


Figura 8. Definindo o separador para as colunas na conexão connArquivoCSVProdutos

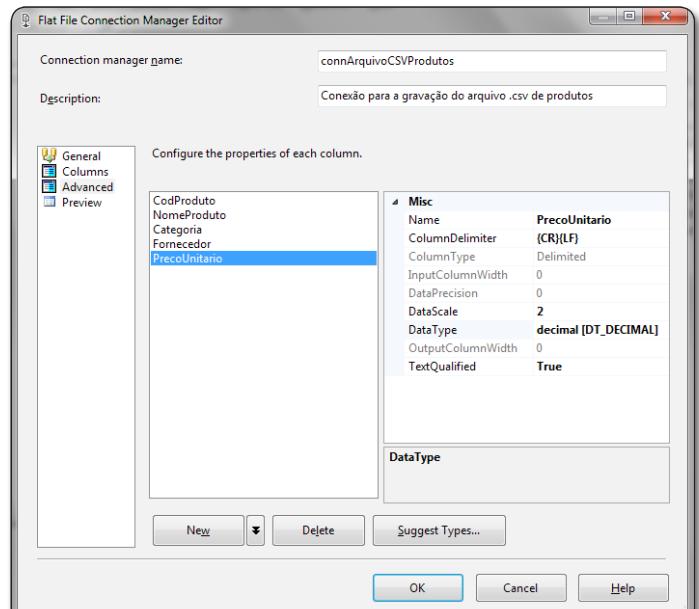


Figura 9. Configurando as colunas para a conexão connArquivoCSVProdutos

Confirmadas as definições do arquivo .csv de produtos, será necessário ainda um último ajuste. Como a geração do nome de tal arquivo ocorre de forma automática, a conexão connArquivoCSVProdutos precisará receber este valor (por meio da propriedade ConnectionString), a fim de que o processo de gravação dos dados aconteça de maneira correta.

O Integration Services permite que os valores de expressões baseadas em variáveis e parâmetros sejam atribuídos ao conteúdo de uma propriedade, com este processo acontecendo dinamicamente durante a execução da aplicação. Essa característica é possível graças a uma propriedade chamada Expressions, a qual

pode conter uma coleção de expressões que vinculam os valores correspondentes a outras propriedades de um componente. Praticamente todos os controles disponíveis para a implementação de um package contam com a propriedade Expressions.

Na **Figura 10** é apresentada a janela a partir da qual acontece o preenchimento da propriedade Expressions (nota-se que foi selecionada a propriedade ConnectionString da conexão connArquivoCSVProdutos). Um assistente (no caso, a janela “Expression Builder”) pode ser utilizado para a geração de novas expressões, combinando variáveis, parâmetros e funções para manipulação de valores, conforme indicado na **Figura 11**.

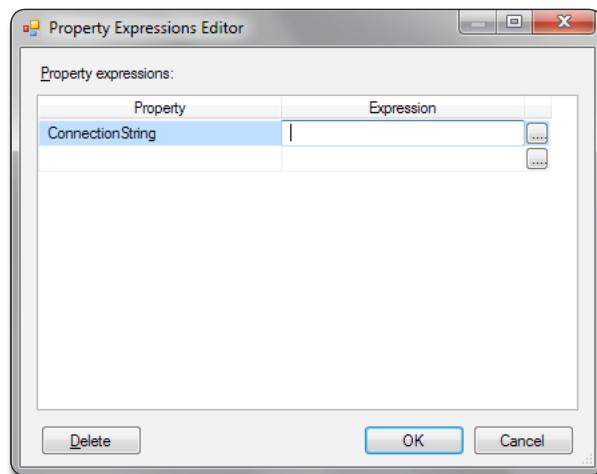


Figura 10. Preenchendo a propriedade Expressions para a conexão connArquivoCSVProdutos

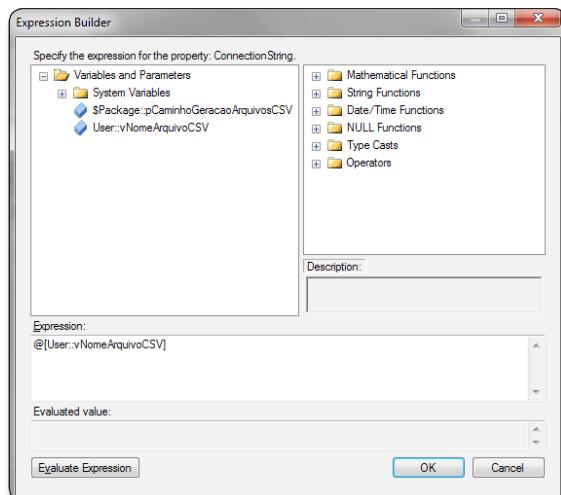


Figura 11. A janela Expression Builder

Já na **Figura 12** está a expressão responsável pelo preenchimento automático da propriedade ConnectionString da conexão connArquivoCSVProdutos (através da variável vNomeArquivoCSV).

Concluído o processo de geração das conexões, iniciaremos agora a configuração das diferentes tarefas que farão parte do package GeracaoCSVProdutos.dtsx. Este procedimento acontecerá a partir da seção Control Flow.

Nota

Para acessar a propriedade Expressions de um controle, utilize a janela Properties do Visual Studio.

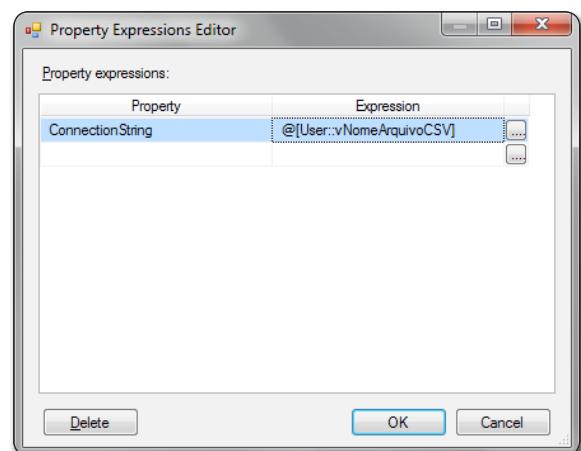


Figura 12. Expressão que preencherá a propriedade ConnectionString

Conforme mencionado anteriormente, o fluxo principal que define um package é formado por uma sucessão de tarefas (tasks). Outros elementos conhecidos como containers permitem o agrupamento lógico de uma série de tarefas, além de disponibilizar informações para a realização de ações específicas.

Na **Figura 13** é apresentada a janela Toolbox com alguns dos controles que podem vir a ser empregados na construção do fluxo principal de um pacote. Outros tipos de tasks (agrupadas na seção “Other Tasks”) também estão disponíveis para uso, sendo possível se obter um maior detalhamento sobre os mesmos a partir da documentação online do Integration Services.

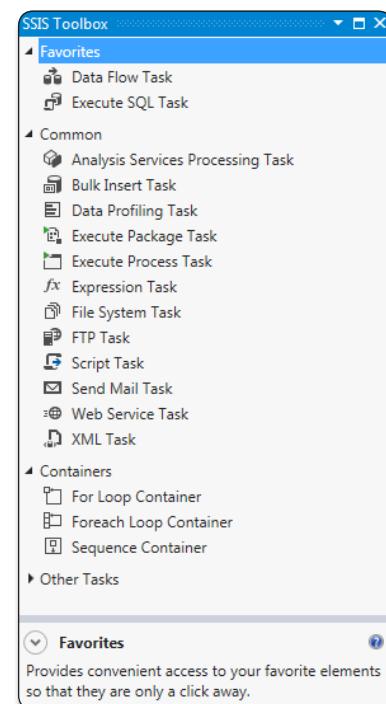


Figura 13. Principais tasks e containers para a construção de um package

Quantos às tasks que podem ser empregadas em packages do Integration Services, merecem destaque os seguintes controles:

- Data Flow Task: permite a obtenção de dados de uma fonte de origem, o seu processamento e, consequentemente, a gravação das informações produzidas em um repositório de destino;
- Execute SQL Task: possibilita a execução de instruções SQL e stored procedures a partir de uma base de dados relacional;
- Analysis Services Processing Task: realiza o processamento de alguma estrutura (cubos, dimensões, modelos de data mining) definida no Analysis Services;
- Bulk Insert Task: efetua o carregamento de dados em uma tabela empregando para isto o comando BULK INSERT do SQL Server (**BOX 9**);

BOX 9. Bulk Copy

Bulk Copy é um tipo de operação empregada na transferência de grandes volumes de informações para uma base relacional. O comum é que ações desse gênero envolvam a carga de arquivos no formato texto para tabelas de um banco de dados específico. O SQL Server oferece total suporte a operações de Bulk Copy, disponibilizando para isto o comando BULK INSERT; no caso do ADO.NET, este mecanismo de acesso a dados disponibiliza a classe `SqlBulkCopy` (namespace `System.Data.SqlClient`) para a implementação deste tipo de funcionalidade em aplicações .NET.

- Data Profiling Task: permite a execução de análises, a fim de identificar potenciais problemas relativos à qualidade dos dados;
- Execute Package Task: torna possível a execução de outro package, a partir do pacote em que esta task for utilizada;
- Execute Process Task: permite a execução de programas externos a um package, como acionar um utilitário para compactação de arquivos, por exemplo;
- File System Task: possibilita a realização de operações envolvendo diretórios (criar, renomear ou excluir uma pasta) e arquivos (mover, copiar ou, até mesmo, excluir um arquivo);
- FTP Task: tarefa utilizada no envio e/ou recebimento de arquivos via FTP;
- Script Task: graças a esta task é possível a codificação de scripts em .NET para a realização de uma ou mais ações específicas;
- Send Mail Task: permite o envio de uma mensagem de e-mail, fazendo uso para isto do protocolo SMTP;
- Web Service Task: aciona um método definido em um Web Service;
- XML Task: tarefa utilizada no processamento de documentos XML.

Dos componentes citados, as tasks Data Flow Task, Execute SQL Task e Script Task correspondem, basicamente, aos controles cuja utilização é mais comum em soluções baseadas no Integration Services.

Dentre os containers disponibilizados pelo Integration Services, podem ser mencionados os seguintes componentes:

- For Loop Container: permite a repetição de um conjunto de tasks dentro de um package. As sucessivas iterações dentro

deste controle são controladas por expressões atribuídas a uma propriedade chamada `EvalExpression`;

- Foreach Loop Container: assim como o controle For Loop Container, este container permite a repetição de uma série de tarefas por meio de iterações. A diferença está no fato de que o componente Foreach Loop Container funciona de uma maneira similar à instrução foreach em .NET, efetuando iterações sobre um conjunto de registros retornados de uma base de dados ou, até mesmo, em uma lista de arquivos localizados em um diretório específico;
- Sequence Container: permite o agrupamento lógico de um conjunto de tarefas relacionadas.

Para o package `GeracaoCSVProdutos.dtsx` serão adicionadas duas tasks, conforme listado na **Tabela 2**. Alterar a propriedade `Name` destes controles com um valor sugestivo é uma boa prática: eventuais erros durante a execução do package `GeracaoCSVProdutos.dtsx` a partir do SQL Server produzirão informações em log; levando tal fato em consideração, uma clara identificação do ponto em que ocorreu um problema facilitará na posterior resolução do mesmo.

Na **Figura 14** é possível observar o fluxo principal do pacote `GeracaoCSVProdutos.dtsx`, com as tasks que farão parte do mesmo já adicionadas.

| Tipo da Task | Valor da propriedade Name |
|----------------|---|
| Script Task | Script Task - Gerar nome do arquivo CSV |
| Data Flow Task | Data Flow Task - Gerar arquivo CSV |

Tabela 2. Tasks a serem adicionadas no package `GeracaoCSVProdutos.dtsx`

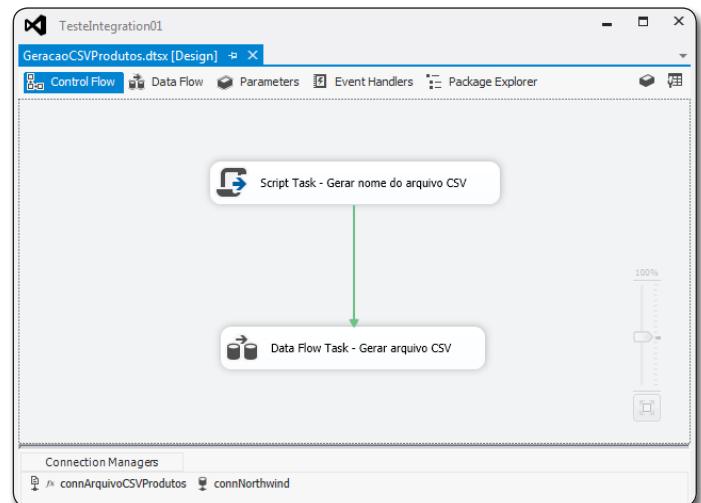


Figura 14. Fluxo principal do package `GeracaoCSVProdutos.dtsx`

O Script Task que foi adicionado ao package `GeracaoCSVProdutos.dtsx` irá receber o caminho associado ao parâmetro `pCaminhoGeracaoArquivosCSV`. A partir disso será gerado um novo nome para o arquivo .csv que conterá o catálogo de

produtos, com este valor sendo finalmente associado à variável vNomeArquivoCSV.

A **Figura 15** apresenta o componente Script Task devidamente configurado. Sobre os itens preenchidos na janela “Script Task Editor”, é possível destacar:

- **ScriptLanguage:** indica a linguagem que será empregada para a implementação do código associado ao controle Script Task. Além de C# (que estaremos utilizando neste exemplo), este componente aceita ainda o uso de instruções escritas em VB.NET;
- **EntryPoint:** indica o método em .NET que será executado quando a task em questão for acionada. Por default é criada uma operação de nome Main, sendo este o ponto em que serão executadas as instruções definidas para um controle Script Task;
- **ReadOnlyVariables:** variáveis ou parâmetros que serão utilizados como parâmetros de leitura. Caso se tente atribuir um valor a algum dos itens informados nesta configuração, um erro será gerado em tempo de execução;
- **ReadWriteVariables:** neste local normalmente são definidas variáveis nas quais também será possível a atribuição de valores. Os itens aqui definidos normalmente armazenam o resultado das ações desencadeadas dentro do método Main.

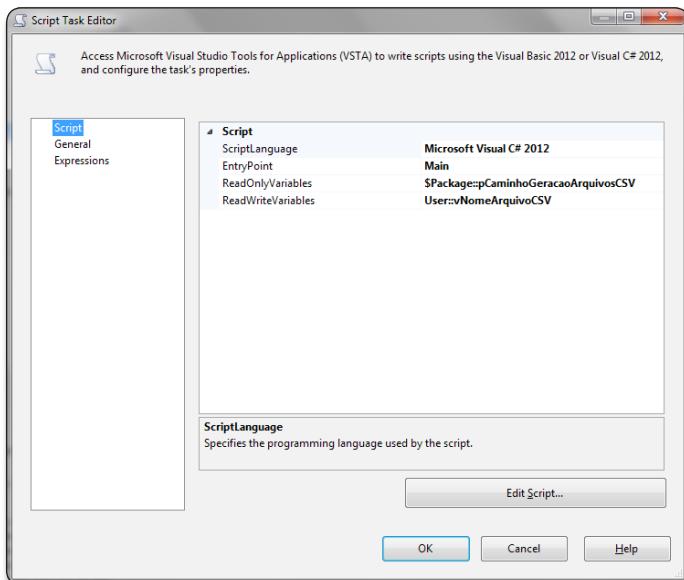


Figura 15. Configurando o controle Script Task

Nota

É recomendável o uso de prefixos como “p” e “v” para identificar, respectivamente, parâmetros e variáveis. Essa prática ajuda a evitar equívocos durante a manipulação destes recursos em controles do tipo Script Task.

Nota

Para se configurar uma task dentro do Visual Studio é necessário apenas clicar duas vezes com o mouse sobre este controle. Uma janela então aparecerá, com os principais elementos que poderão ser preenchidos para este componente (como no caso do Script Task adicionado ao package GeracaoCSVProdutos.dtsx).

Para a codificação das ações que farão parte do controle Script Task, acione a opção “Edit Script”. Uma tela como a que consta na **Figura 16** aparecerá então.

O código que constará num componente Script fará parte de uma classe chamada ScriptMain, a qual é baseada no tipo básico VSTARTScriptObjectModelBase (namespace Microsoft.SqlServer.Dts.Tasks.ScriptTask). Essa última estrutura faz parte de um mecanismo do .NET Framework conhecido como Visual Studio Tools for Applications ou, simplesmente, VSTA (**BOX 10**). Dentro do tipo ScriptMain existirá, por sua vez, um método Main, sendo este o local em que acontecerá as ações definidas para o controle Script Task adicionado a um package.

```

/// <summary>
/// ScriptMain is the entry point class of the script. Do not change the name, attributes,
/// or parent of this class.
/// </summary>
[Microsoft.SqlServer.Dts.Tasks.ScriptTask.SSIScriptTaskEntryPointAttribute]
public partial class ScriptMain : Microsoft.SqlServer.Dts.Tasks.ScriptTask.VSTARTScriptObjectModelBase
{
    // Help: Using Integration Services variables and parameters in a script
    // Help: Firing Integration Services events from a script
    // Help: Using Integration Services connection managers in a script

    /// <summary>
    /// This method is called when this script task executes in the control flow.
    /// Before returning from this method, set the value of Dts.TaskResult to indicate success or failure.
    /// </summary>
    public Main()
    {
        // TODO: Add your code here
        Dts.TaskResult = (int)ScriptResults.Success;
    }
}

ScriptResults declaration

```

Figura 16. Editando o código de um controle Script Task

BOX 10. Visual Studio Tools for Applications

O Visual Studio Tools for Applications (VSTA) é um conjunto de ferramentas da plataforma .NET que pode ser utilizado em soluções dos mais variados tipos, de forma a permitir a customização de funcionalidades pré-existentes ou, até mesmo, possibilitar a estensão das capacidades oferecidas por tais recursos. Surgido ainda com o Visual Studio 2005, é justamente este o mecanismo que torna possível o uso de instruções em .NET dentro de soluções do Integration Services.

Na **Listagem 1** está o código esperado para o método Main do controle Script Task. Quanto à implementação desta operação, é necessário destacar:

- O uso do objeto associado à propriedade Dts (linhas 4, 6 e 10), o qual representa uma instância do tipo ScriptObjectModel (namespace Microsoft.SqlServer.Dts.Tasks.ScriptTask). Esta referência serve de base para a manipulação de variáveis e parâmetros, com isso acontecendo através da propriedade Variables, além de indicar se a execução do método Main aconteceu com sucesso ou não (por meio da propriedade TaskResult);
- Na linha 3 o conteúdo definido para o parâmetro pCaminhoGeracaoArquivosCSV é acessado, de forma a se determinar o diretório em que ocorrerá a gravação do arquivo .csv com o catálogo de produtos;

- Já na linha 6 a variável vNomeArquivoCSV está sendo preenchida com o nome completo (incluindo diretório) do arquivo com informações de produtos;
- Por fim, o valor de enumeration ScriptResults.Success é atribuído à propriedade TaskResult do objeto Dts, indicando assim que a execução do código definido para o controle Script Task teve sucesso (no caso de uma eventual falha, seria utilizado o valor ScriptResults.Failure).

Listagem 1. Método Main com as instruções para geração de um novo nome de arquivo

```

01 public void Main()
02 {
03     string nomeArquivoCSV =
04         Dts.Variables["pCaminhoGeracaoArquivosCSV"].Value.ToString() +
05         "Produtos_{0}.csv";
06     Dts.Variables["vNomeArquivoCSV"].Value =
07         String.Format(nomeArquivoCSV,
08             DateTime.Now.ToString("yyyy-MM-dd_HH:mm:ss"));
09
10    Dts.TaskResult = (int)ScriptResults.Success;
11}

```

Finalizada a configuração do componente Script Task, o último passo antes de se concluir a implementação do package Geracao-CSVProdutos.dtsx será a definição do fluxo associado ao controle Data Flow Task (dentro da seção Data Flow). Será a partir da sequência de ações configuradas nesta tarefa que acontecerá a leitura dos dados de produtos armazenados na base Northwind e, finalmente, a gravação do catálogo sob a forma de um arquivo com a extensão .csv.

O fluxo de dados definido para uma tarefa do tipo Data Flow Task costuma envolver:

- Uma fonte de origem (Source), de onde serão obtidos dados que podem estar sujeitos a alguma forma de processamento;
- O processamento de tais informações, com prováveis transformações (reformatação de valores, concatenação de dois ou mais dados etc.);
- Uma estrutura de destino (Destination) que receberá os dados já transformados. Bancos de dados relacionais e arquivos são algumas das possibilidades para este caso específico.

Para o exemplo que envolve a construção do package Geracao-CSVProdutos.dtsx, estarão sendo utilizados apenas controles para a manipulação de dados na origem e gravação num repositório de destino. Na próxima edição abordaremos alguns dos controles que poderão ser utilizados na transformação de dados.

Na **Figura 17** é possível observar alguns dos controles disponíveis para a configuração de um Data Flow Task. Já a **Tabela 3** apresenta alguns desses elementos, assim como a tecnologia empregada pelos mesmos no processamento de informações.

Será preciso adicionar ao fluxo de dados do controle Data Flow Task os controles listados na **Tabela 4**, modificando ainda o valor da propriedade Name para esses itens.

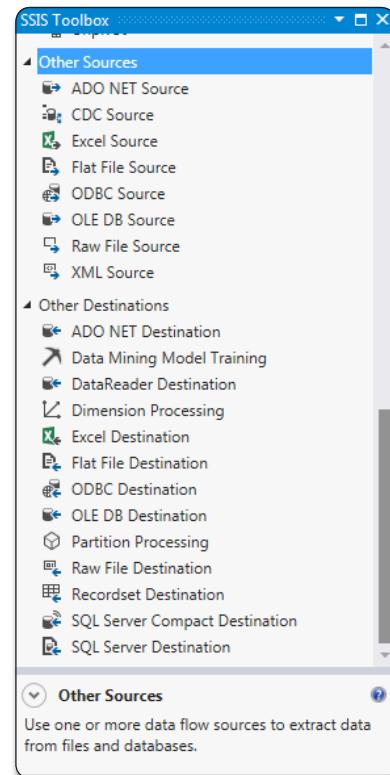


Figura 17. Alguns dos controles utilizados na definição de um Data Flow

| Origem | Destino | Tecnologia empregada |
|------------------|-----------------------|--|
| ADO.NET Source | ADO.NET Destination | Drivers ADO.NET |
| Excel Source | Excel Destination | Planilhas do Microsoft Excel |
| Flat File Source | Flat File Destination | Arquivos de texto (delimitados por caracteres, posições fixas) |
| ODBC Source | ODBC Destination | Drivers ODBC |
| OLE DB Source | OLE DB Destination | Driver OLE DB |
| Raw File Source | Raw File Destination | Arquivos binários |

Tabela 3. Alguns dos controles disponíveis para a implementação de um Data Flow

| Tipo do Controle | Valor da propriedade Name |
|-----------------------|-------------------------------------|
| ADO.NET Source | ADO.NET Source - Northwind |
| Flat File Destination | Flat File Destination - Arquivo CSV |

Tabela 4. Controles a serem adicionados para o componente Data Flow Task

Na **Figura 18** está a representação para o fluxo de dados que resultará na geração de um novo arquivo .csv contendo o catálogo de produtos.

Já as definições esperadas para o controle ADO.NET Source são apresentadas na **Figura 19**. A configuração da fonte de dados requer o preenchimento dos seguintes itens na seção “Connection Manager” da janela “ADO.NET Source Editor”:

- ADO.NET connection manager: conexão do tipo ADO.NET empregada no acesso a uma base de dados relacional (para este exemplo será a conexão connNorthwind criada anteriormente);

Integration Services 2012: implementando soluções de ETL - Parte 1

- Data access mode: indica se um comando SQL será executado para a obtenção dos dados de origem (opção “SQL Command”) ou ainda, se tais informações virão de uma tabela ou view que pertence à base para a qual o controle ADO.NET Source está apontando (opção “Table or view”). Para o package GeracaoCSVProdutos.dtsx será utilizada uma query (**Listagem 2**) que relaciona as tabelas Products (produtos), Categories (categorias) e Suppliers (fornecedores) do banco Northwind.

Na seção “Columns” da janela “ADO.NET Source Editor” é possível observar as colunas que estarão disponíveis para a geração do arquivo .csv de produtos (**Figura 20**).

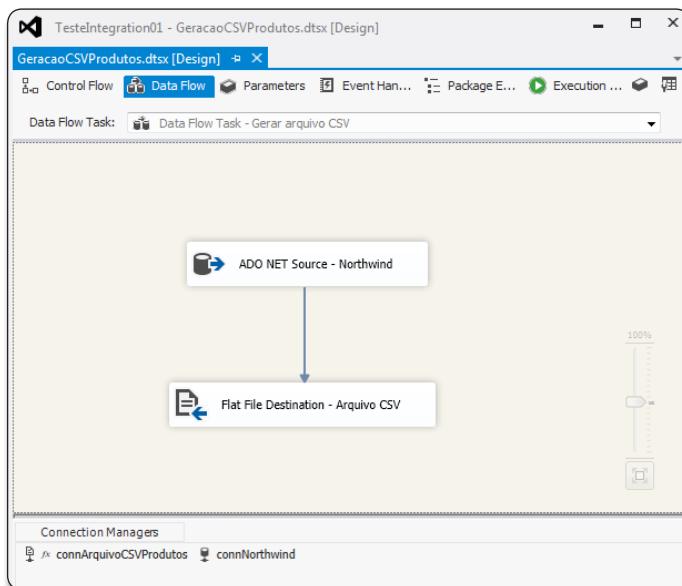


Figura 18. Fluxo do componente Data Flow Task

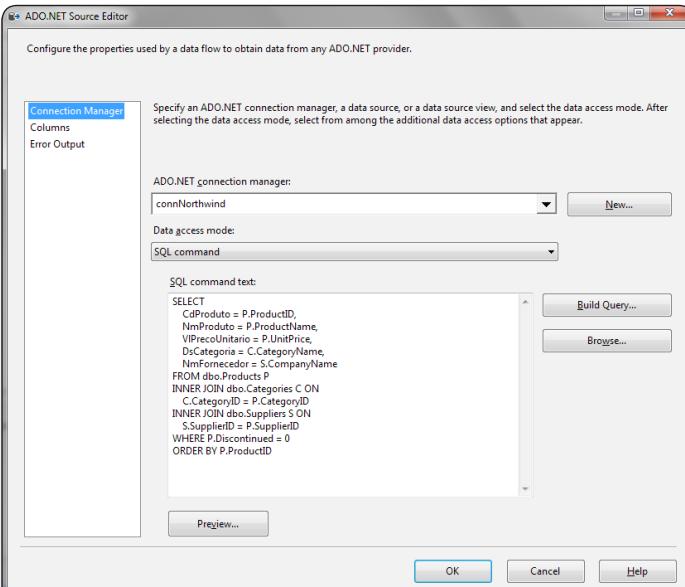


Figura 19. Configurando o controle ADO.NET Source

Listagem 2. Query para a obtenção de informações sobre o catálogo de produtos

```
01 SELECT
02 CdProduto = P.ProductID,
03 NmProduto = P.ProductName,
04 VIPrecoUnitario = P.UnitPrice,
05 DsCategoria = C.CategoryName,
06 NmFornecedor = S.CompanyName
07 FROM dbo.Products P
08 INNER JOIN dbo.Categories C ON
09 C.CategoryID = P.CategoryID
10 INNER JOIN dbo.Suppliers S ON
11 S.SupplierID = P.SupplierID
12 WHERE P.Discontinued = 0
13 ORDER BY P.ProductID
```

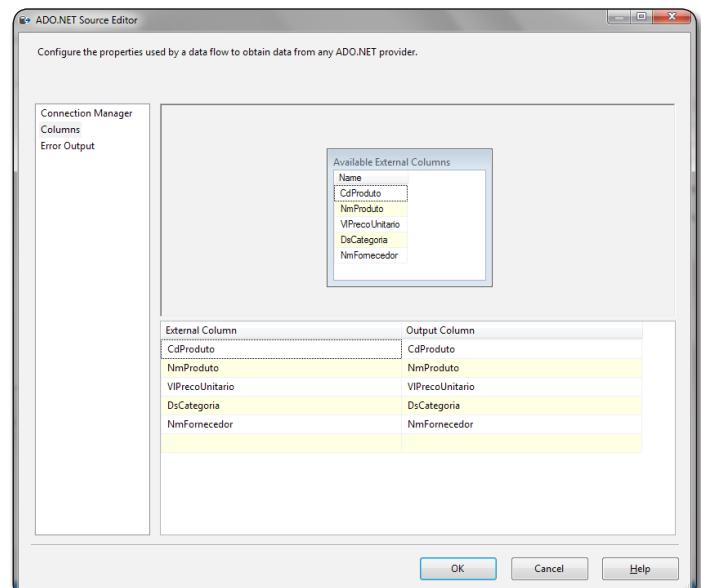


Figura 20. Colunas geradas a partir do controle ADO.NET Source

Concluindo a implementação do package GeracaoCSVProdutos.dtsx, estaremos configurando agora o controle Flat File Destination. Na seção “Connection Manager” da janela “Flat File Destination Editor” efetuar os seguintes ajustes (**Figura 21**):

- Selecionar a conexão connArquivoCSVProdutos em “Flat File connection manager”;
- Manter a opção “Overwrite data in the file” desmarcada, uma vez que a cada execução do pacote GeracaoCSVProdutos.dtsx um novo arquivo .csv será gerado;
- Opcionalmente um cabeçalho para o arquivo .csv poderá ser definido ao se preencher o elemento “Header” (o que não é o caso para o exemplo aqui abordado).

Na seção “Mappings” da janela “Flat File Destination Editor” (**Figura 22**) serão realizados os mapeamentos entre as colunas da instrução SQL definida no componente ADO.NET Source e os campos correspondentes do arquivo .csv de destino (os quais foram configurados na conexão connArquivoCSVProdutos).

Finalizada a construção do package GeracaoCSVProdutos.dtsx, estaremos executando este arquivo para efeitos de testes a

partir do Visual Studio 2012 (da mesma maneira que uma aplicação .NET convencional). Tão logo o IDE tenha concluído este procedimento, marcações em verde aparecerão em cada task da seção Control Flow, indicando que a geração do arquivo .csv teve sucesso (**Figura 23**).

Na seção “Execution Results” (ainda dentro do Visual Studio) será ainda visualizar detalhes da execução do pacote GeracaoCSVProdutos.dtsx (**Figura 24**). Eventuais erros estarão indicados nesse local, servindo assim para que desenvolvedores possam diagnosticar e corrigir problemas existentes em um package.

Um arquivo .csv deverá constar então no diretório C:\Devmedia\TesteVisualStudio\, como indicado na **Figura 25**.

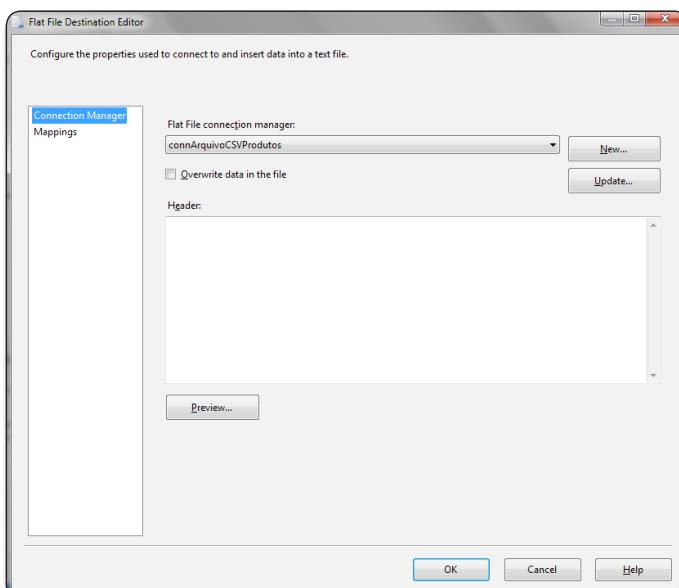


Figura 21. Configurando o controle Flat File Destination

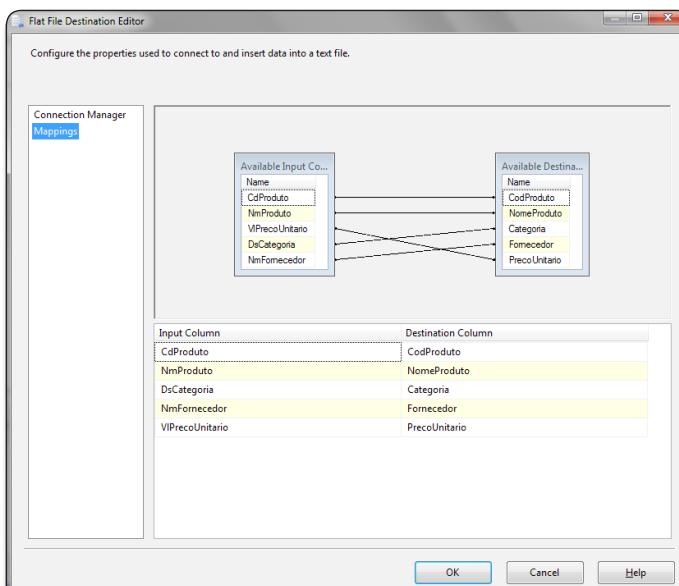


Figura 22. Mapeamentos das colunas empregadas na geração do arquivo .csv

Na **Figura 26** é possível visualizar o conteúdo do arquivo .csv dentro do Bloco de Notas do Windows. Já a **Figura 27** exibe os dados deste arquivo a partir do Microsoft Excel.

Executando um package localmente

Packages do Integration Services podem ser acionados a partir de soluções .NET dos mais variados tipos. Isto é possível graças

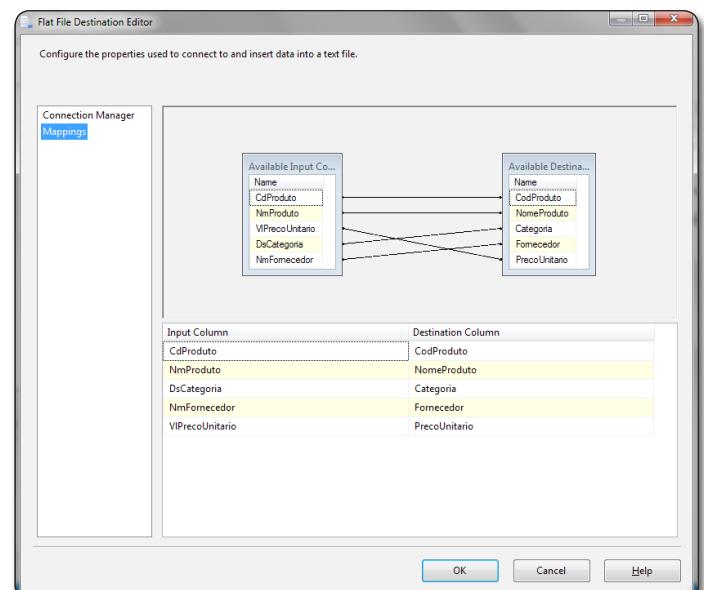


Figura 23. Package GeracaoCSVProdutos.dtsx executado com sucesso

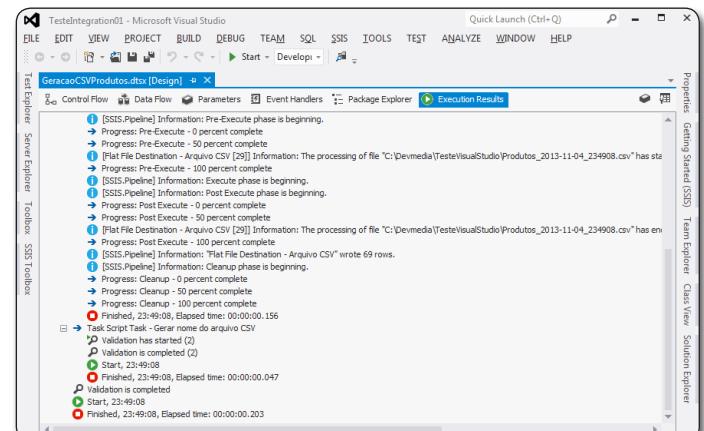


Figura 24. Visualizando detalhes da execução do package GeracaoCSVProdutos.dtsx

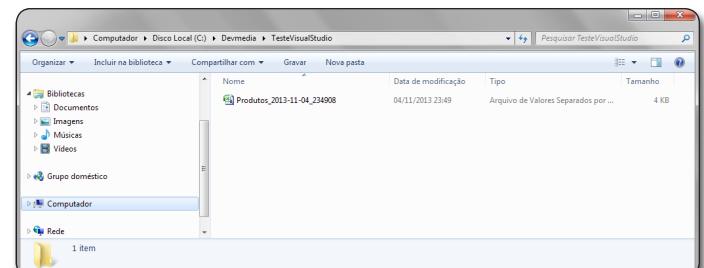


Figura 25. Arquivo .csv gerado após a execução do package GeracaoCSVProdutos.dtsx

Integration Services 2012: implementando soluções de ETL - Parte 1

aos recursos disponibilizados pela biblioteca Microsoft.SqlServer.ManagedDTS.dll, a qual permite que aplicações Windows Forms, Windows Service, ASP.NET e, até mesmo, Web Services executem arquivos .dtsx sem maiores dificuldades.

A fim de demonstrar como este procedimento pode ser implementado em projetos .NET, estaremos criando uma Console Application chamada TesteIntegration01.ExecucaoLocalPackage.

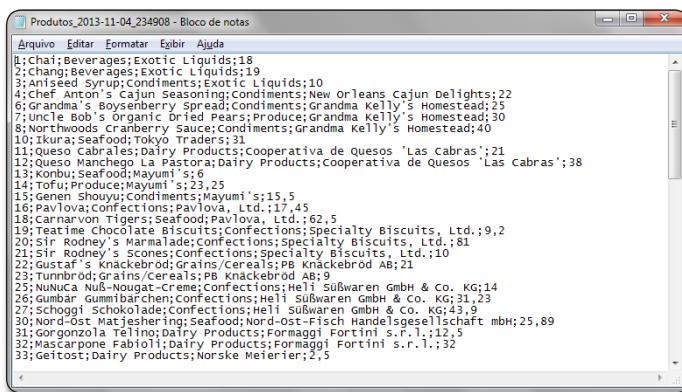


Figura 26. Arquivo .csv sendo visualizado a partir do Bloco de Notas

| Produtos_2013-11-04_234908 - Microsoft Excel | | | | | |
|--|----------------|------------------------------------|-------|---|---|
| A1 | B | C | D | E | F |
| 1 Chai | Beverages | Exotic Liquids | 18 | | |
| 2 Chang | Beverages | Exotic Liquids | 19 | | |
| 3 Aniseed Syrup | Condiments | Exotic Liquids | 10 | | |
| 4 Chef Anton's Cajun Seasoning | Condiments | New Orleans Cajun Delights | 22 | | |
| 5 Grandma's Boysenberry Spread | Condiments | Grandma Kelly's Homestead | 25 | | |
| 6 Uncle Bob's Organic Dried Pears | Produce | Grandma Kelly's Homestead | 30 | | |
| 7 Northwoods Cranberry Sauce | Condiments | Grandma Kelly's Homestead | 40 | | |
| 8 Ikuра | Seafood | Tokyo Traders | 31 | | |
| 9 Queso Manchego La Pastora | Dairy Products | Cooperativa de Quesos 'Las Cabras' | 21 | | |
| 10 Queso Manchego La Pastora | Dairy Products | Cooperativa de Quesos 'Las Cabras' | 38 | | |
| 11 Konbu | Seafood | | | | |
| 12 Tofu | Produce | Mayumi's | 23,25 | | |
| 13 Genen Shouyu | Condiments | Mayumi's | 15,5 | | |
| 14 Pavlova | Confections | Pavlova, Ltd. | 17,45 | | |
| 15 Caranarow Tigers | Seafood | Pavlova, Ltd. | 62,5 | | |

Figura 27. Arquivo .csv sendo visualizado a partir do Microsoft Excel

Nota

A biblioteca Microsoft.SQLServer.ManagedDTS.dll pode ser encontrada no diretório C:\Program Files (x86)\Microsoft SQL Server\110\SDK\Assemblies\.

Na Listagem 3 está o arquivo app.config com as configurações esperadas para este projeto:

- No item CaminhoPackageGeracaoCSVProdutos foi definido o caminho em que se encontra o pacote a ser executado (no caso o package GeracaoCSVProdutos.dtsx definido na seção anterior);
- Já para o item CaminhoGeracaoArquivosCSV, informe o diretório em que serão criados os arquivos .csv de produtos quando a aplicação aqui descrita for executada.

Listagem 3. Arquivo app.config do projeto TesteIntegration01.ExecucaoLocalPackage

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <configuration>
03   <appSettings>
04     <add key="CaminhoPackageGeracaoCSVProdutos"
05       value="..." />
06     <add key="CaminhoGeracaoArquivosCSV"
07       value="C:\Devmedia\TesteAplicacaoDotNet\" />
08   </appSettings>
09 </configuration>
```

Na Listagem 4 está a definição da classe Program. Quanto à estrutura deste tipo é possível observar:

- Inicialmente é criada uma instância do tipo Application (namespace Microsoft.SqlServer.Dts.Runtime), sendo que esta referência servirá de base para o acesso a packages do Integration Services (linha 16);
- O método LoadPackage é acionado a partir do objeto associado à variável de nome "appSSIS" (linha 17), devolvendo como resultado uma instância da classe Package (namespace Microsoft.SqlServer.Dts.Runtime). Esta operação recebe como parâmetros o caminho em que se encontra o package GeracaoCSVProdutos.dtsx e, opcionalmente, uma referência baseada na interface IDTSEvents (namespace Microsoft.SqlServer.Dts.Runtime);
- O parâmetro pCaminhoGeracaoArquivosCSV do package GeracaoCSVProdutos.dtsx está sendo preenchido através da propriedade Parameters (linha 21), a qual é parte integrante do objeto representado pela variável "package". Esta ação substitui o valor definido anteriormente no package pelo caminho indicado no arquivo app.config;
- A execução do package GeracaoCSVProdutos.dtsx é iniciada ao se invocar o método Execute pertencente à instância do tipo Package (linha 28). O resultado disto será um dos valores previstos para o enumeration (namespace Microsoft.SqlServer.Dts.Runtime). Em caso de sucesso, a operação Execute retornará o valor DTSEExecResult.Success, ao passo que o valor DTSEExecResult.Failure corresponderá a um erro durante o processamento.

Se a execução da aplicação TesteIntegration01.ExecucaoLocalPackage ocorrer sem falhas (Figura 28), um arquivo .csv será gerado no diretório especificado na seção appSettings do arquivo app.config (Figura 29).

Efetuando o deploy de um projeto do Integration Services

Desenvolver soluções para o Integration Services envolverá, quase que invariavelmente, a publicação de packages em um servidor SQL Server. Dentre os motivos que contribuem para a escolha por esse ambiente estão a possibilidade de se executar de maneira programática Jobs que acionarão um ou mais packages, a própria facilidade em se implementar esse tipo de alternativa (a partir de uma interface gráfica e sem a necessidade de construção de aplicações), além um mecanismo de log que permite a análise de erros ocorridos durante o processamento de tais Jobs.

Listagem 4. Classe Program (projeto TesteIntegration01.ExecucaoLocalPackage)

```

01 using System;
02 using System.Collections.Generic;
03 using System.Linq;
04 using System.Text;
05 using System.Configuration;
06 using Microsoft.SqlServer.Dts.Runtime;
07
08 namespace TesteIntegration01.ExecucaoLocalPackage
09{
10    class Program
11    {
12        static void Main(string[] args)
13        {
14            Console.WriteLine(
15                "Carregando o package GeracaoCSVProdutos.dtsx...");
16            Application appSSIS = new Application();
17            Package package = appSSIS.LoadPackage(
18                ConfigurationManager
19                    .AppSettings["CaminhoPackageGeracaoCSVProdutos"],
20                    null);
21
22            package.Parameters["pCaminhoGeracaoArquivosCSV"].Value =
23                ConfigurationManager
24                    .AppSettings["CaminhoGeracaoArquivosCSV"];
25
26            Console.WriteLine(
27                "Executando localmente o package " +
28                "GeracaoCSVProdutos.dtsx...");
29            DTSExecResult resultado = package.Execute();
30            Console.WriteLine(
31                "Resultado da execução do package " +
32                "GeracaoCSVProdutos.dtsx: " +
33                resultado.ToString());
34
35            Console.Write(
36                "Pressione qualquer tecla para " +
37                "encerrar esta aplicação...");
38            Console.ReadKey();
39        }
40    }

```

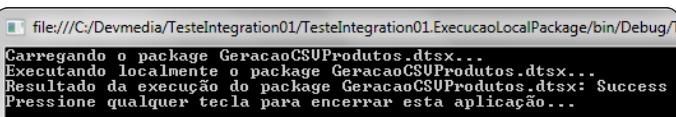


Figura 28. Executando a aplicação TesteIntegration01.ExecucaoLocalPackage

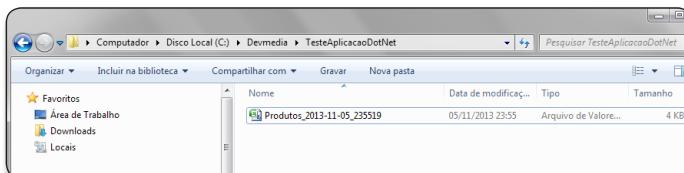


Figura 29. Arquivo .csv gerado após a execução do projeto TesteIntegration01.ExecucaoLocalPackage

Antes do lançamento da versão 2012 do SQL Server, o deployment de aplicações do Integration Services costumava ser uma tarefa árdua para desenvolvedores familiarizados com esta tecnologia:

- O Visual Studio não dispunha de nenhuma funcionalidade que permitisse de forma direta o deployment de arquivos .dtsx em um servidor SQL Server;
- Um arquivo de manifesto precisava então ser criado através do Visual Studio e executado posteriormente, promovendo com isto a instalação dos packages num determinado servidor;
- A única forma de modificar as configurações esperadas para um ou mais packages era através do uso de documentos XML criados especificamente para esse fim. Outra possibilidade (bem mais trabalhosa) seria alterar manualmente os diferentes elementos em cada pacote antes de se proceder com a publicação dos mesmos. Diante desse cenário, não é difícil de imaginar que todo este processo estava sujeito a falhas, sobretudo em soluções que envolvessem o uso de inúmeros packages;
- Os packages poderiam ser publicados no banco de dados MSDB (**BOX 11**) ou ainda, em um diretório acessível para o servidor SQL Server responsável pela execução dos mesmos.

BOX 11. O banco de dados MSDB

A base MSDB é um dos bancos de dados de sistema do SQL Server, sendo utilizada por este SGDB em operações que envolvam o backup/restore de outras bases, envio de e-mail, agendamento e execução de Jobs, armazenamento de packages do Integration Services (sobretudo em releases anteriores à versão 2012), dentre outras funcionalidades.

Este método para a implantação de soluções baseadas no Integration Services é conhecido como Package Deployment Model e, embora ainda seja possível a utilização desta alternativa, não se recomenda mais que isso seja feito em conjunto com a versão 2012. Neste novo release do SQL Server foi disponibilizado um modelo mais simplificado e flexível para o deployment de aplicações do Integration Services: trata-se do mecanismo conhecido como Project Deployment Model.

Quanto ao funcionamento do modelo Project Deployment Model, é possível destacar:

- O deployment de packages acontece agora a partir do Visual Studio, através de uma funcionalidade que dispensa assim o uso dos arquivos de manifesto requeridos pelo método anterior;
- Parâmetros substituíram os arquivos XML utilizados na manipulação de configurações, procurando com isto facilitar o gerenciamento das informações necessárias ao funcionamento de aplicações do Integration Services;
- Será num repositório chamado de catálogo que acontecerá a publicação de projetos do Integration Services. Para cada instância de um servidor SQL Server poderá ser criado somente um catálogo, com este recurso fazendo uso do mecanismo conhecido como CLR Integration (**BOX 12**) para a execução de packages.

Nesta seção demonstraremos como realizar o deploy de uma aplicação do Integration Services a partir do Visual Studio. Além deste IDE, estaremos fazendo uso ainda do Microsoft SQL Server Management Studio 2012.

Integration Services 2012: implementando soluções de ETL - Parte 1

BOX 12. CLR Integration

O recurso conhecido como CLR Integration foi introduzido ainda na versão 2005 do SQL Server, permitindo a integração de estruturas deste SGBD ao Common Language Runtime (CLR) do .NET Framework. Este mecanismo permite, dentre outras coisas, que objetos típicos de bases de dados relacionais como stored procedures, functions e triggers sejam implementados em .NET.

Antes de proceder com a implantação de uma solução do Integration Services, será necessário configurar o ambiente para a execução deste serviço (caso isto ainda não tenha sido feito).

O primeiro passo consiste em habilitar o uso do Common Language Runtime da plataforma .NET. Na **Listagem 5** é apresentado um script que ativa este mecanismo, utilizando para isto as rotinas SP_CONFIGURE e RECONFIGURE.

Com a integração junto ao CLR ativada, estaremos prosseguindo agora com a criação do catálogo que armazenará os projetos do Integration Services. Selecione dentro do SQL Server Management Studio a opção “Create Catalog...”, que estará disponível a partir do menu de contexto para o item “Integration Services Catalogs” (**Figura 30**).

Listagem 5. Habilitando o uso do CLR num servidor SQL Server

```
01 sp_configure 'clr enabled', 1
02 go
03 reconfigure
04 go
```

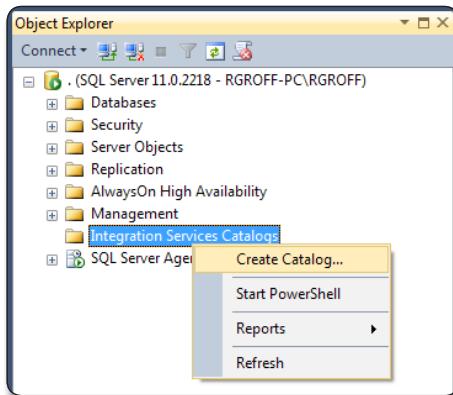


Figura 30. Acionando a opção para criação de um novo catálogo

Aparecerá então a janela “Create Catalog” (**Figura 31**) em que constarão algumas opções referentes à criação do catálogo “SSISDB”:

- Quando marcado, o item “Enable automatic execution of Integration Services stored procedure at SQL Server startup” determinará, logo após a reinicialização do SGBD, a execução automática de uma rotina responsável por corrigir o status de packages que estavam em processamento quando o servidor em questão deixou de operar;
- Uma senha que serve de base para a proteção da chave de criptografia empregada pelo catálogo SSISDB.

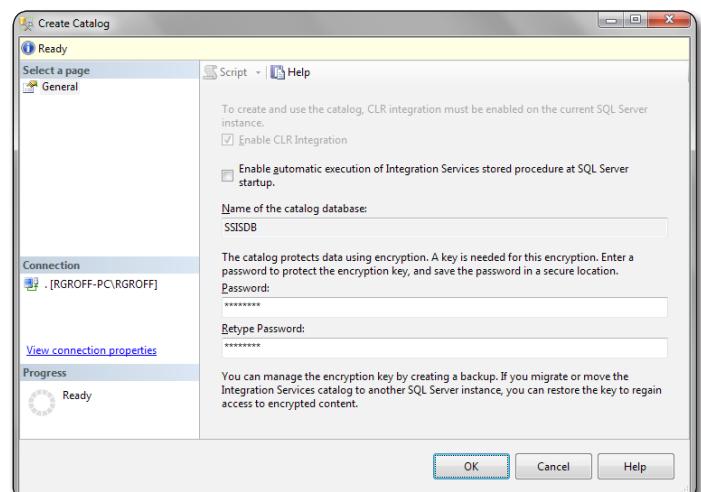


Figura 31. Criando o catálogo SSISDB

Importante ressaltar que um banco de dados (também chamado “SSISDB”) será gerado como resultado da criação de um catálogo em uma instância do SQL Server.

A fim de possibilitar uma melhor organização dos diferentes projetos que farão parte do catálogo SSISDB, pastas poderão ser criadas dentro deste repositório. Quando utilizada (**Figura 32**), a funcionalidade “Create Folder...” (acionada a partir do menu de contexto do catálogo SSISDB) permite a separação de soluções do Integration Services em conformidade com algum critério específico. Para os exemplos abordados nesta série de artigos, estaremos criando uma pasta chamada “Testes” (**Figura 33**).

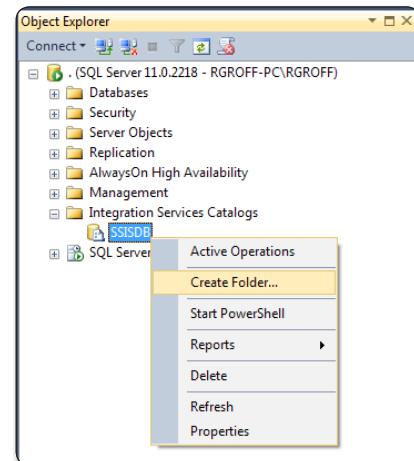


Figura 32. Acionando a opção para a criação de pastas a partir do catálog SSISDB

Neste momento já estaremos com o ambiente devidamente configurado para a publicação da aplicação TesteIntegration01.SSIS, conforme pode ser observado na **Figura 34**.

Para iniciar o processo de publicação do projeto TesteIntegration01.SSIS, ação a opção “Deploy” a partir do menu de contexto desta aplicação no Visual Studio (**Figura 35**). Será exibida neste instante a janela “Integration Services Deployment Wizard” (**Figura 36**).

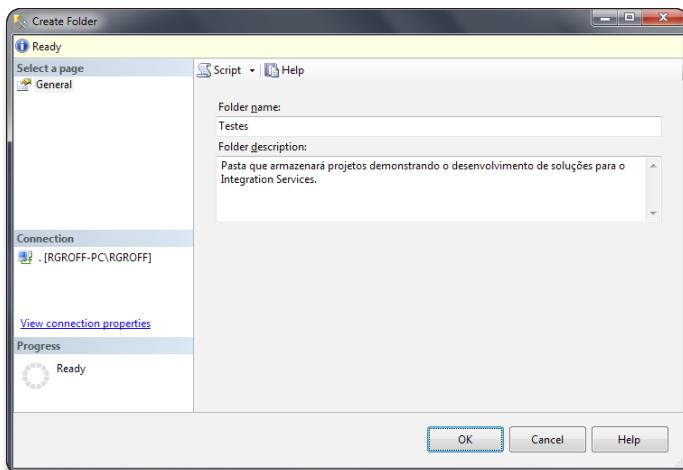


Figura 33. Criando a pasta Testes para o catálogo SSISDB

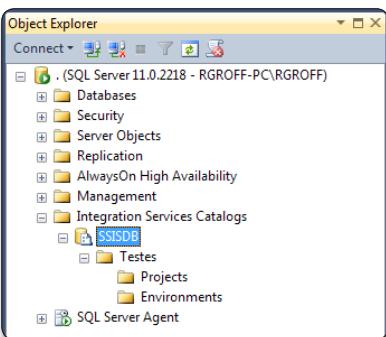


Figura 34. O catálogo SSISDB já configurado para a publicação de aplicações

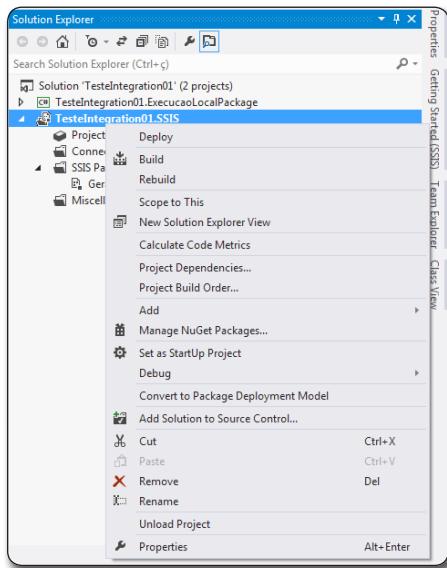


Figura 35. Acionando a opção para o deploy da aplicação TesteIntegration01.SSIS

Na seção “Select Destination”, informe o servidor e o caminho configurado anteriormente para a publicação da aplicação TesteIntegration01.SSIS (**Figura 37**).

Já na seção “Review” aparecerão detalhes referentes ao projeto que será publicado, assim como o caminho em que este processo

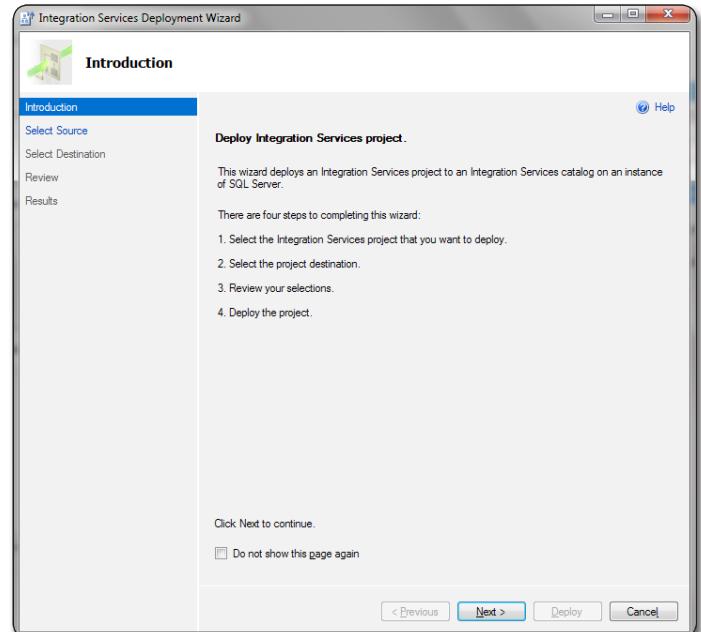


Figura 36. A janela Integration Services Deployment Wizard

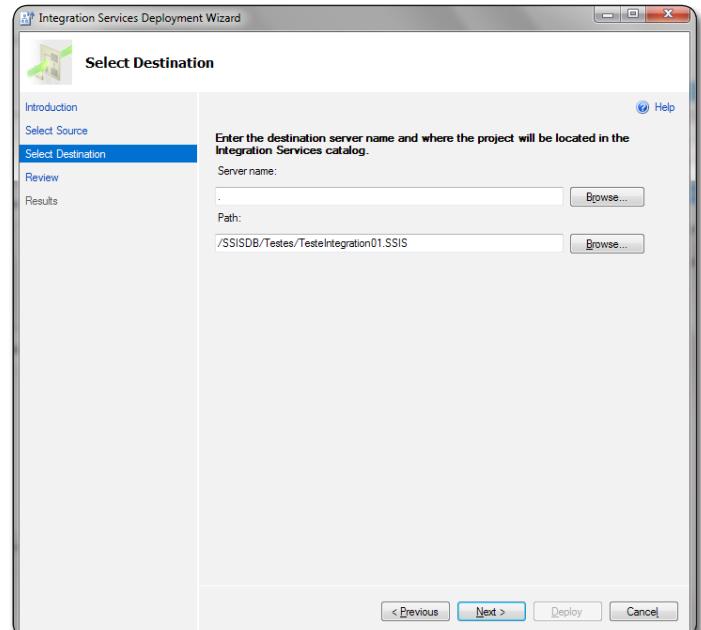


Figura 37. Selecionando o local em que o projeto TesteIntegration01.SSIS será publicado

acontecerá (**Figura 38**). Acionar a opção “Deploy”, para com isto concluir esse processo.

Caso não ocorram problemas, será exibida uma confirmação de que o projeto TesteIntegration01.SSIS foi publicado no catálogo SSISDB abaixo da pasta “Testes” (**Figura 39**).

Retornando ao SQL Server Management Studio, será possível observar o projeto TesteIntegration01.SSIS (incluindo nisto o package GeracaoCSVProdutos.dtsx) logo abaixo da pasta “Testes” do catálogo SSISDB (**Figura 40**).

Integration Services 2012: implementando soluções de ETL - Parte 1

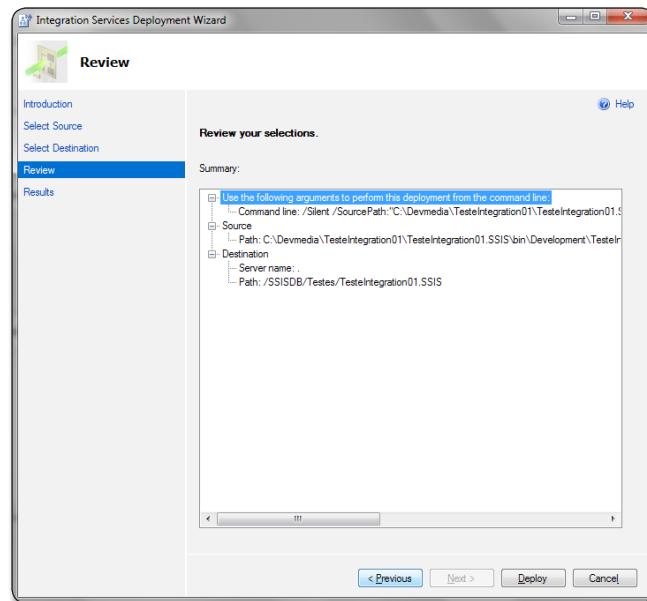


Figura 38. Selecionando o local em que o projeto TesteIntegration01.SSIS será publicado

processo de implantação desta solução tenha se encerrado. Uma das formas mais comuns para a execução de soluções do Integration Services consiste na criação de Jobs em uma instância do SQL Server, com tais estruturas sendo acionadas de forma automática e periodicamente através do mecanismo conhecido como serviço SQL Server Agent (**BOX 13**).

BOX 13. SQL Server Agent

O SQL Server Agent é mais um dos serviços que fazem parte do SQL Server, fornecendo a estrutura necessária para a criação e o agendamento da execução de tarefas conhecidas como Jobs. Quanto à forma como os Jobs são definidos, este tipo de estrutura é formado de um ou mais passos (steps), com estes últimos podendo executar uma instrução SQL, um stored procedure ou, até mesmo, acionar um package do Integration Services.

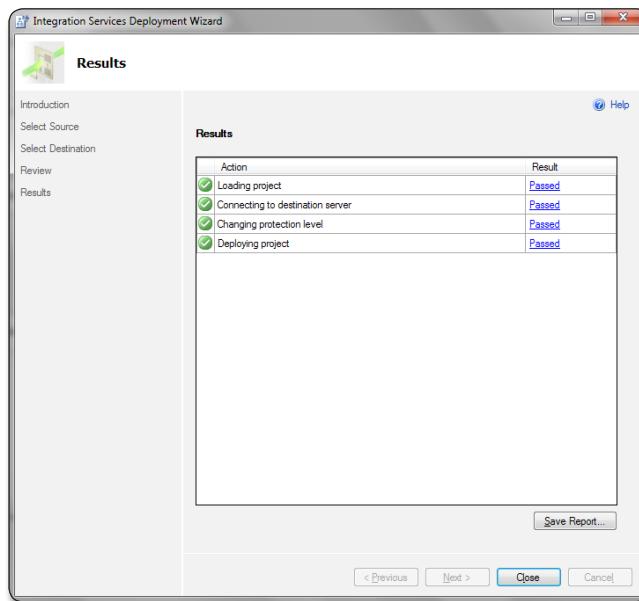


Figura 39. Publicação com sucesso do projeto TesteIntegration01.SSIS

Os valores dos parâmetros e conexões utilizados pela aplicação podem ser modificados acionando a opção “Configure...”, a partir do menu de contexto da aplicação TesteIntegration01.SSIS. Altere o valor do parâmetro pCaminhoGeracaoArquivosCSV para “C:\Devmedia\testeSQLServer\” (Figura 41); este caminho deverá existir fisicamente, a fim de que possamos executar com sucesso os outros testes descritos mais adiante.

Criando um Job para a Execução de um projeto do Integration Services

A simples publicação da aplicação TesteIntegration01.SSIS num servidor SQL Server não significa, necessariamente, que o

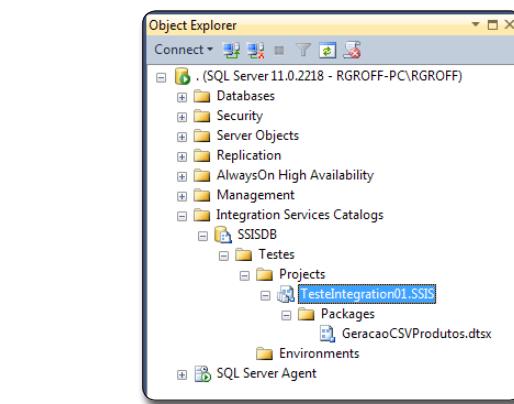


Figura 40. Projeto TesteIntegration01.SSIS já publicado no servidor SQL Server

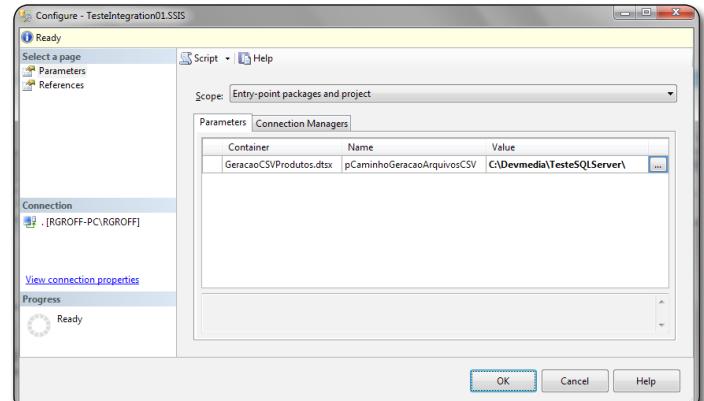


Figura 41. Configurando a aplicação TesteIntegration01.SSIS a partir do servidor SQL Server

Nesta seção será demonstrado de que maneira um Job pode ser criado para a execução diária do projeto TesteIntegration01.SSIS. O primeiro passo para isto será acessar (Figura 42), a partir do elemento SQL Server Agent no utilitário Management Studio, a opção “New Job...” (acionando para isto o menu de contexto para o item “Jobs”).

Dentro da janela “New Job” acontecerá a configuração de um Job chamado “Job TesteIntegration01.SSIS”. Na seção “General” serão preenchidas informações para a identificação deste novo Job (Figura 43).

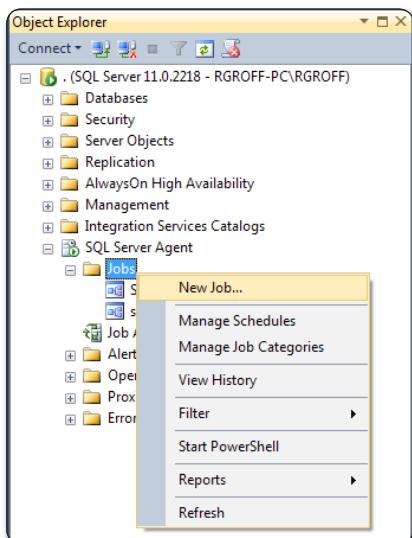


Figura 42. Acionando a opção para a criação de um novo Job

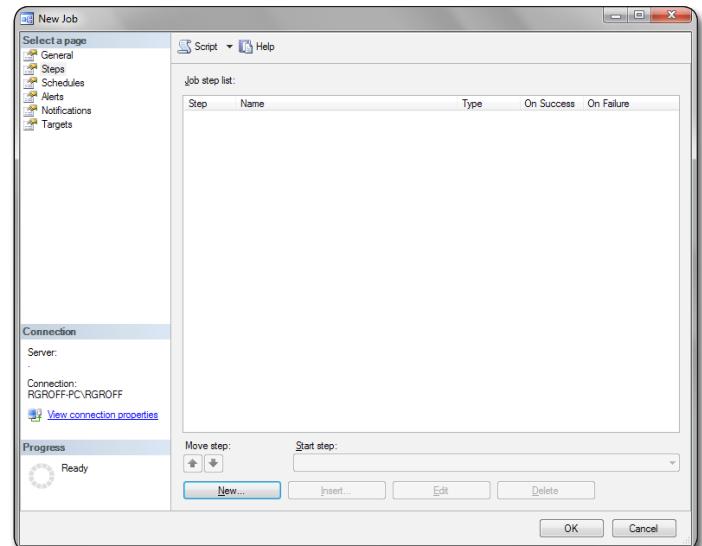


Figura 44. Configurando os steps que farão parte de um Job

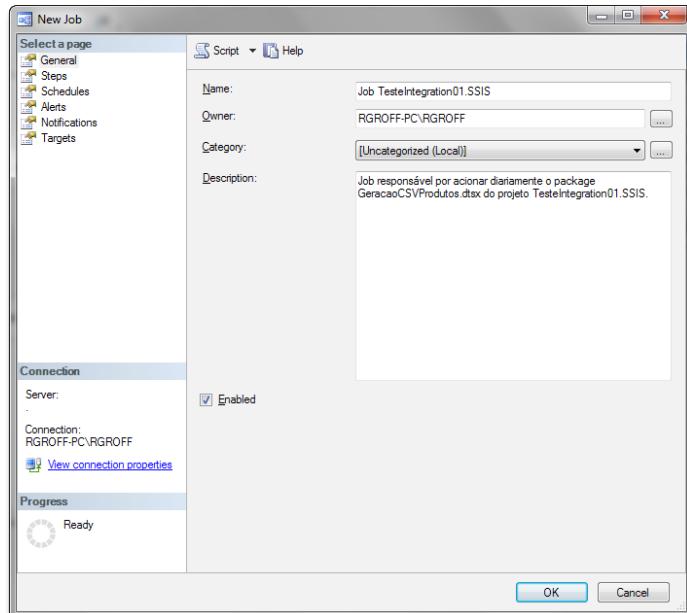


Figura 43. Criando o job TesteIntegration01.SSIS

Crie dentro da seção “Steps” (Figura 44) o passo que irá acionar o package GeracaoCSVProdutos.dtsx. Na janela “New Job Step” (Figura 45), preencha as seguintes configurações:

- Step name: identificação de um passo que fará parte de um Job. Informe o valor “Acionar package GeracaoCSVProdutos.dtsx”;
- Type: tipo de ação a ser executada. O valor “SQL Server Integration Services Packages” permite invocar um pacote .dtsx (que pode ser ter sido previamente publicado num servidor SQL Server ou ainda, estar localizado em algum diretório do servidor);
- Seção Package: informar o servidor em que se encontra instalado o package GeracaoCSVProdutos.dtsx, assim como o caminho deste dentro do catálogo SSISDB.

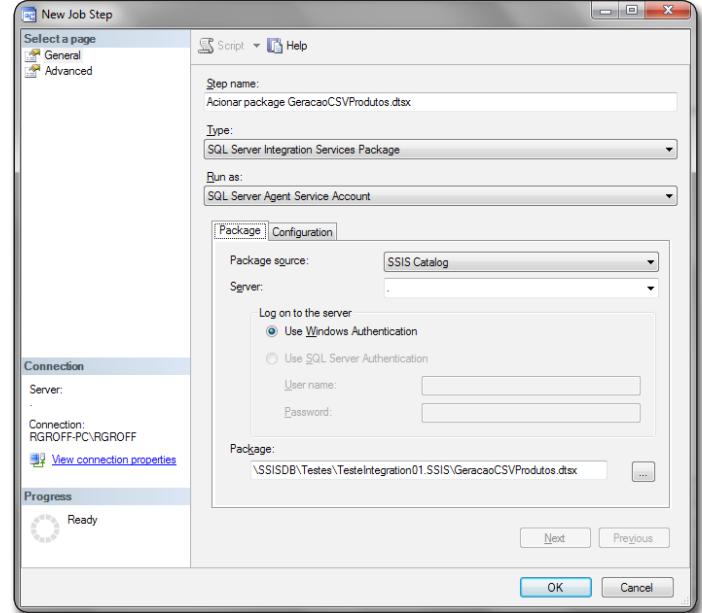


Figura 45. Configurando o step que acionará o package GeracaoCSVProdutos.dtsx

Já na seção “Schedules” (Figura 46), crie o agendamento para a execução do pacote GeracaoCSVProdutos.dtsx. A partir da janela “New Job Schedule” (Figura 47), realize o preenchimento das seguintes configurações:

- Name: texto que irá identificar o agendamento. Informe o valor “Agendamento - package GeracaoCSVProdutos.dtsx”;
- Schedule type: selecione a opção “Recurring”, indicando com isto que o Job será executado continuamente;
- Frequency: selecione a opção “Daily” (execução diária) para o item “Occurs” e o valor “1” para “Recurs every” (intervalos de um dia);
- Daily frequency: defina em “Occurs every” que a execução acontecerá a cada 24 horas, sendo que isso será feito entre 11:45:00 e 12:00:00;
- Duration: certificar-se de que foi marcada a opção “No end date”.

Integration Services 2012: implementando soluções de ETL - Parte 1

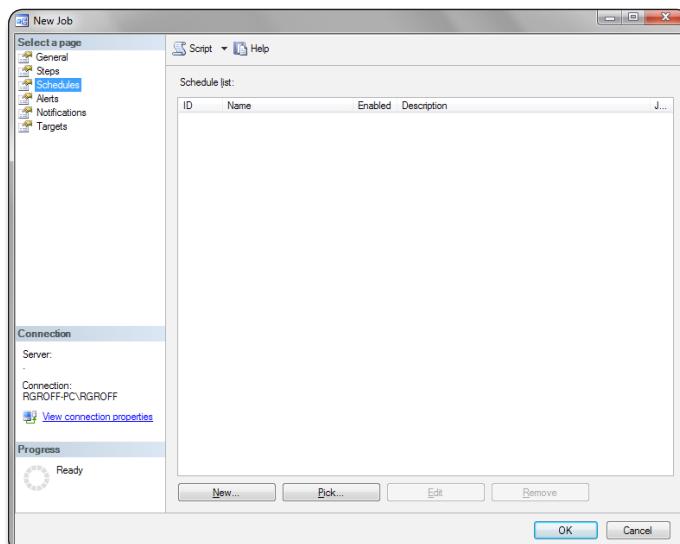


Figura 46. Configurando o agendamento de um Job

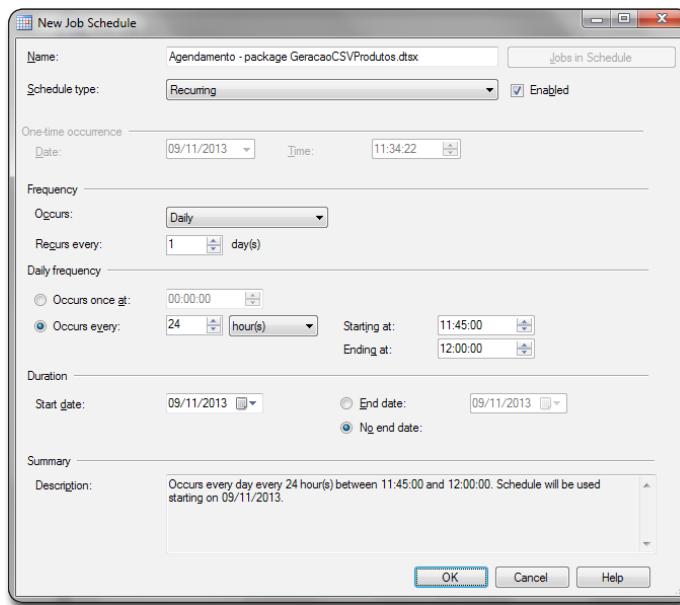


Figura 47. Preenchendo as configurações de agendamento de um Job

Na Figura 48 é possível visualizar o Job que acionará o package GeracaoCSVProdutos.dtsx após a criação do mesmo dentro do SQL Server Agent. Após a primeira execução desta rotina, um arquivo .csv terá sido gerado no diretório "C:\Devmedia\TesteSQLServer" (Figura 49).

Todo job executado a partir do SQL Server Agent gera informações de log que poderão vir a ser consultadas posteriormente. Na Figura 50 é apresentando um exemplo disto, em que é possível observar a execução com sucesso desta rotina.

Executando um package remotamente

Packages publicados em um catálogo do Integration Services também podem ser acessados por aplicações .NET de forma remota. Nesta seção apresentaremos de que forma isto é feito,

criando para isto a Console Application TesteIntegration01.Exe cucaRemotaPackage.

Este projeto fará uso das seguintes bibliotecas:

- Microsoft.SqlServer.Management.IntegrationServices.dll;
- Microsoft.SqlServer.ConnectionInfo.dll;
- Microsoft.SqlServer.Management.Sdk.Sfc.dll;
- Microsoft.SqlServer.Smo.dll.

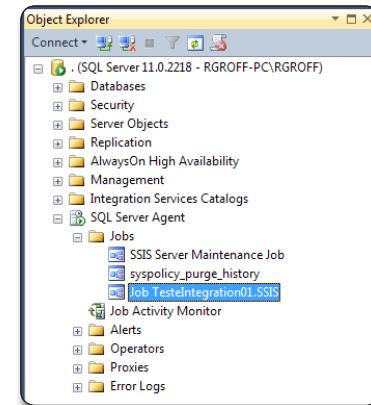


Figura 48. Job já criado no SQL Server Agent

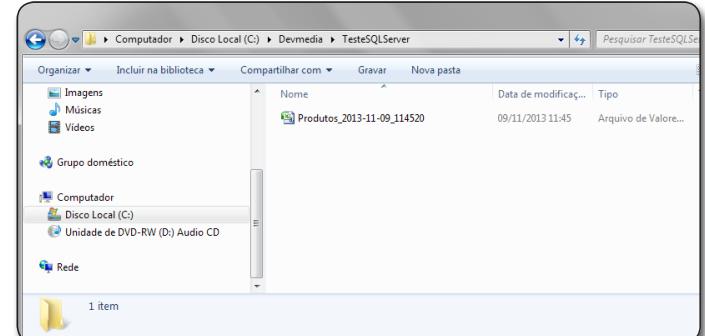


Figura 49. Arquivo .csv gerado após a execução do Job

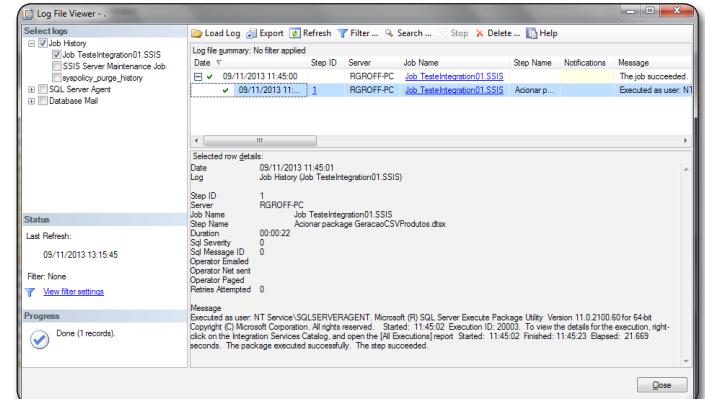


Figura 50. Consultando o log de um Job

A execução remota de um package tem como pré-requisito a realização de uma conexão junto ao servidor em que essa estrutura foi publicada. Por esse motivo, será preciso incluir no arquivo app.config uma Connection String que aponte para a base SSISDB

(gerada juntamente com o catálogo de mesmo nome). A **Listagem 6** apresenta o conteúdo do arquivo app.config com as configurações esperadas para este projeto de testes.

Já na **Listagem 7** está a implementação da classe Program. Quanto às instruções para se acessar remotamente um package previamente publicado em um servidor, merece ser destacado:

- As classes IntegrationServices, Catalog, CatalogFolder, ProjectInfo e PackageInfo estão sendo utilizadas ao longo do método Main, sendo que todas essas estruturas pertencem ao namespace Microsoft.SqlServer.Management.IntegrationServices.

- Uma instância da classe IntegrationServices é gerada (linha 17), recebendo como parâmetro uma referência baseada no tipo SqlConnection (namespace System.Data.SqlClient). Será o objeto associado à variável de nome “integration” que servirá de base para o acesso aos packages e projetos em uma determinada instância do SQL Server;
- Utilizando a referência da classe IntegrationServices obtida anteriormente, uma instância da classe Catalog será atribuída à variável “catalogo” (linha 21);
- O próximo passo agora será a obtenção de uma instância da classe CatalogFolder, a qual permitirá o acesso a informações da pasta “Testes” dentro do catálogo SSISDB (linha 23);
- A partir da variável pasta será retornada uma instância do tipo ProjectInfo (linha 25);
- Por fim, através da instância obtida no passo anterior será associado um objeto do tipo PackageInfo à variável “package”. Será com esta última referência que iniciaremos a execução do package, invocando para isto o método Execute (linha 33);
- A operação Execute (definida na classe PackageInfo) está recebendo dois parâmetros. O primeiro deles é um flag que indica se o package será processado como se estivesse num ambiente de 32 bits (mesmo se o servidor SQL se basear numa arquitetura de 64 bits); já o segundo parâmetro é uma instância do tipo EnvironmentReference (namespace Microsoft.SqlServer.Management.IntegrationServices), sendo que a passagem de tal referência é opcional.

Nota

Com exceção da biblioteca Microsoft.SqlServer.Management.IntegrationServices (normalmente localizada em C:\Windows\assembly\GAC_MSIL\), todas as demais DLLs que contam com recursos do SQL Server utilizados pelo projeto TesteIntegration01.ExecucaoRemotaPackage podem ser encontradas a partir do diretório c:\Program Files (x86)\Microsoft SQL Server\110\SDK\Assemblies.

Caso a aplicação TesteIntegration01.ExecucaoRemotaPackage seja executada sem falhas (**Figura 51**), um novo arquivo .csv constará no diretório “C:\Devmedia\TesteSQLServer\” (**Figura 52**).

Executando remotamente um package a partir de um Job

Como último exemplo de execução remota de um package, estaremos criando o projeto TesteIntegration01.ExecucaoRemotaJob. Isso acontecerá através de uma chamada à stored procedure SP_START_JOB, que faz parte do banco de dados de sistema MSDB.

Listagem 6. Arquivo app.config do projeto TesteIntegration01.ExecucaoRemotaPackage

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <configuration>
03   <connectionStrings>
04     <add name="SSISDB"
05       connectionString=
06         "Data Source=.;Initial Catalog=SSISDB;Integrated Security=SSPI;"
07       providerName="System.Data.SqlClient"/>
08   </connectionStrings>
09 </configuration>
```

Listagem 7. Classe Program (projeto TesteIntegration01.ExecucaoRemotaPackage)

```
01 using System;
02 using System.Collections.Generic;
03 using System.Linq;
04 using System.Text;
05 using System.Configuration;
06 using System.Data.SqlClient;
07 using Microsoft.SqlServer.Management.IntegrationServices;
08
09 namespace TesteIntegration01.ExecucaoRemotaPackage
10 {
11   class Program
12   {
13     static void Main(string[] args)
14     {
15       Console.WriteLine(
16         "Carregando o package GeracaoCSVProdutos.dtsx...");
17       IntegrationServices integration =
18         new IntegrationServices(new SqlConnection(
19           ConfigurationManager.ConnectionStrings["SSISDB"]
20             .ConnectionString));
21       Catalog catalogo =
22         integration.Catalogs("SSISDB");
23       CatalogFolder pasta =
24         catalogo.Folders["Testes"];
25       ProjectInfo projeto =
26         pasta.Projects["TesteIntegration01.SSIS"];
27       PackageInfo package =
28         projeto.Packages["GeracaoCSVProdutos.dtsx"];
29
30       Console.WriteLine(
31         "Executando remotamente o package " +
32         "GeracaoCSVProdutos.dtsx...");
33       package.Execute(true, null);
34
35       Console.Write(
36         "Pressione qualquer tecla para " +
37         "encerrar esta aplicação...");
38       Console.ReadKey();
39     }
40   }
41 }
```

Na **Listagem 8** está o arquivo app.config para esta aplicação de testes, já constando no mesmo a string de conexão para acesso à base MSDB.

Já na **Listagem 9** está a definição da classe Program. Quanto à forma como o método Main foi implementado, é possível observar:

- A partir da variável “conexao” (instância do tipo System.Data.SqlClient.SqlConnection) é estabelecida uma conexão com a base de dados MSDB (linha 15);

Integration Services 2012: implementando soluções de ETL - Parte 1

- A referência “cmd” (objeto do tipo System.Data.SqlClient.SqlCommand) invocará o stored procedure “SP_START_JOB” através de uma chamada ao método ExecuteNonQuery (linha 34), iniciando assim a execução da rotina “Job TesteIntegration01.SSIS”.

Se não ocorrerem falhas durante a execução do projeto (Figura 53), um novo arquivo .csv terá sido gerado no diretório “C:\Devmedia\TesteSQLServer” (Figura 54).

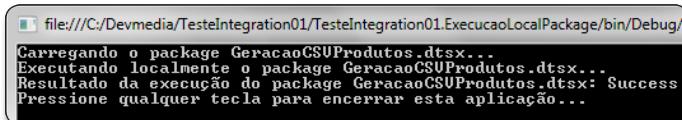


Figura 51. Executando a aplicação TesteIntegration01.ExecucaoRemotaPackage

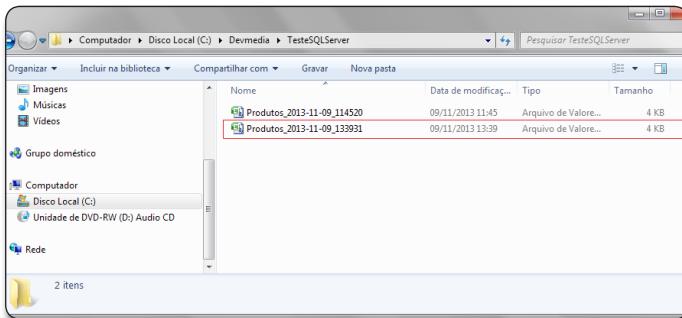


Figura 52. Arquivo .csv gerado após a execução do projeto TesteIntegration01.ExecucaoRemotaPackage

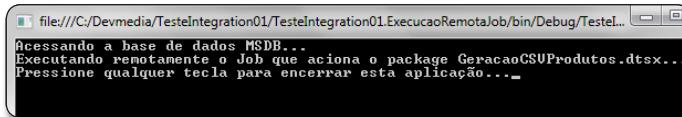


Figura 53. Executando a aplicação TesteIntegration01.ExecucaoRemotaJob

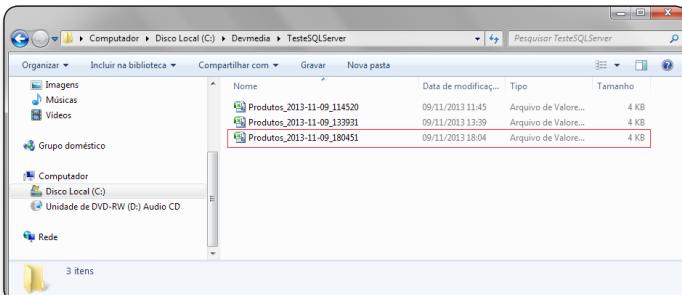


Figura 54. Arquivo .csv gerado após a execução do projeto TesteIntegration01.ExecucaoRemotaJob

Listagem 9. Classe Program (projeto TesteIntegration01.ExecucaoRemotaJob)

```
01 using System;
02 using System.Collections.Generic;
03 using System.Linq;
04 using System.Text;
05 using System.Configuration;
06 using System.Data;
07 using System.Data.SqlClient;
08
09 namespace TesteIntegration01.ExecucaoRemotaJob
10 {
11     class Program
12     {
13         static void Main(string[] args)
14         {
15             using (SqlConnection conexao =
16                 new SqlConnection(ConfigurationManager
17                     ..ConnectionStrings["MSDB"]
18                     .ConnectionString))
19             {
20                 SqlCommand cmd = conexao.CreateCommand();
21                 cmd.CommandText = "SP_START_JOB";
22                 cmd.CommandType = CommandType.StoredProcedure;
23
24                 cmd.Parameters.Add("@JOB_NAME", SqlDbType.VarChar)
25                     .Value = "Job TesteIntegration01.SSIS";
26
27                 Console.WriteLine(
28                     "Acessando a base de dados MSDB...");
29                 conexao.Open();
30
31                 Console.WriteLine(
32                     "Executando remotamente o Job que " +
33                     "aciona o package GeracaoCSVProdutos.dtsx...");
34                 cmd.ExecuteNonQuery();
35
36                 conexao.Close();
37             }
38
39             Console.Write(
40                 "Pressione qualquer tecla para " +
41                 "encerrar esta aplicação...");
42             Console.ReadKey();
43         }
44     }
45 }
```

Conclusão

Este artigo procurou fornecer uma visão geral a respeito do Integration Services, enfatizando como tal serviço pode ser empregado na integração entre diferentes sistemas. Por contar com um escopo bastante abrangente no que se refere à manipulação de informações, esta ferramenta pode se revelar como um instrumento de grande valia e flexibilidade nos mais variados contextos.

Dentre os cenários mais comuns de uso do Integration Services, é possível destacar situações nas quais o processamento de um grande volume de informações requer um bom nível de desempenho. É importante ressaltar que esta ênfase em questões envolvendo performance é uma preocupação central em grandes

Listagem 8. Arquivo app.config do projeto TesteIntegration01.ExecucaoRemotaJob

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <configuration>
03   <connectionStrings>
04     <add name="MSDB"
05       connectionString=
06         "Data Source=.;Initial Catalog=msdb;Integrated Security=SSPI;"
07       providerName="System.Data.SqlClient"/>
08   </connectionStrings>
09 </configuration>
```

organizações, representando um fator crítico para o sucesso em muitos ramos de negócio.

Embora parte integrante do SQL Server, o Integration Services pode ser utilizado em conjunto com outros tipos de bancos relacionais (como Oracle, DB2, Postgre, Firebird), já que o mesmo conta com suporte a mecanismos de acesso a dados como OLE DB, ODBC e ADO.NET. Outras fontes como arquivos texto, planilhas do Excel, documentos XML e Web Services também podem ser utilizadas em projetos do Integration Services.

Autor



Renato José Groffe

renato.groffe@yahoo.com.br - <http://www.devmedia.com.br/renatogroffe>



Atua como consultor em atividades voltadas ao desenvolvimento de softwares há mais de 10 anos. Bacharel em Sistemas de Informação, com especialização em Engenharia de Software. Microsoft Certified Technology Specialist (Web, WCF, Distributed Applications, ADO.NET, Windows Forms), Microsoft Specialist (HTML5 with JavaScript and CSS3, Developing ASP.NET MVC 4 Web Applications), Oracle Certified Associate (PL/SQL), Sun Certified (SCJP, SCWCD), ITIL Foundation V2, Cobit 4.1 Foundation.

Links:

How to: Install Sample Databases

<http://msdn.microsoft.com/en-us/library/vstudio/8b6y4c7s.aspx>

Integration Services Programming Overview

<http://technet.microsoft.com/en-us/library/ms403344.aspx>

Microsoft SQL Server Data Tools - Business Intelligence for Visual Studio 2012

<http://www.microsoft.com/en-us/download/details.aspx?id=36843>

SQL Server Integration Services

<http://technet.microsoft.com/en-us/library/ms141026.aspx>

What's New (Integration Services - SQL Server 2012)

<http://technet.microsoft.com/en-us/library/bb522534.aspx>

Você gostou deste artigo?

Dê seu voto em www.devmedia.com.br/netmagazine/feedback

Ajude-nos a manter a qualidade da revista!



CURSOS ONLINE

A Revista .net Magazine oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.

.net
magazine



CONHEÇA OS CURSOS MAIS RECENTES:

- [Curso Padrões de Projeto com C#](#)
- [Curso básico de ASP .NET](#)
- [Curso de Introdução ao .NET Framework](#)
- [Curso Básico de C#](#)
- [C# 5 e suas novidades](#)
- [ASP.NET MVC – Sistema de Vestibular](#)

Para mais informações :

www.devmedia.com.br/curso/netmagazine

(21) 3382-5038



DEVMEDIA

Criando uma aplicação para conexões remotas

Como criar uma aplicação cliente em C# que se conecta a um servidor VNC remoto

A utilização de vários computadores conectados em rede acabou atraindo diversos novos serviços que auxiliam o compartilhamento e acesso a recursos. Um dos serviços mais populares é representado pela categoria de softwares que permite o compartilhamento da tela (screen sharing) e que possibilita o acesso e controle remoto da área de trabalho (desktop) de um computador.

As principais tecnologias para permitir o compartilhamento e acesso remoto à interface gráfica de um sistema operacional trabalham com o conceito de cliente/servidor, onde um computador servidor (o host) executa um serviço que permitirá clientes (clients) remotos acessarem o desktop. O compartilhamento e o acesso remoto são realizados através da transferência da imagem (tela) do host para o cliente e do envio das interações do mouse e teclado no cliente para o servidor. Dentre os softwares existentes para trabalhar com estas tecnologias podemos destacar o VNC, que é a sigla utilizada para representar as implementações de código livre que se baseiam no uso do protocolo RFB.

A utilização de ferramentas de compartilhamento de desktop permite diversas aplicações, como a possibilidade de suporte remoto e resolução de problemas com o acompanhamento do usuário, facilitar o processo de deploy de componentes em servidores remotos, treinamento de usuários e monitoria de ações. Contudo, tais usos da tecnologia de compartilhamento de tela possuem implicações relevantes para a privacidade e segurança dos usuários.

Baseado nesse contexto, este artigo apresentará como construir uma aplicação console em C# que acessa remotamente um servidor VNC com a biblioteca VNCSharp. Esta aplicação pode ser utilizada para automatizar a execução de tarefas administrativas (aplicação de service pack, atualização de configurações, etc), treinamento de

Resumo DevMan

Porque esse artigo é útil:

A utilização de serviços de compartilhamento de tela e controle remoto é útil em situações onde é preciso automatizar uma tarefa que deve ser realizada diretamente na interface gráfica do sistema operacional. Esta automação pode ser realizada através do envio de comandos (teclas) e eventos do mouse para diversos computadores remotos que tiverem um servidor VNC instalado. Este artigo mostra exemplos de como enviar caracteres e eventos do mouse para um servidor VNC remoto através de uma aplicação cliente desenvolvida em linguagem C#.

usuários, colaboração, captura de tela e outras. A partir do projeto de exemplo descrito neste artigo o leitor poderá integrar as bibliotecas de acesso a um servidor VNC e montar suas próprias soluções que envolvam o acesso a computadores remotos através de um cliente VNC que permite o envio de teclas e eventos do mouse.

O VNC e tecnologias de compartilhamento de tela

As tecnologias para o compartilhamento de tela e o acesso remoto já existem há um bom tempo. Contudo, apenas recentemente estas ferramentas começaram a serem utilizadas em larga escala, principalmente devido ao aumento da largura de banda da conexão entre computadores.

Possuir uma largura de banda adequada é necessário para a utilização de tecnologias de compartilhamento de tela, uma vez que é necessário transmitir frequentemente uma grande quantidade de bytes associada à tela do servidor remoto para os clientes. Apesar de existirem técnicas de compactação e redução de informações necessárias para o compartilhamento de tela, ainda hoje esta tecnologia requer uma conexão confiável, constante e adequada entre o cliente e o servidor.

A definição de como a tela do servidor será transmitida para os clientes e como os eventos de mouse e teclado do cliente devem

ser transmitidos para o servidor é detalhada no protocolo utilizado. Atualmente existem diversos protocolos empregados por servidores de compartilhamento de tela, como o protocolo RDP (Remote Desktop Connection), utilizado pelo serviço Terminal Services da Microsoft, e o protocolo RFB (Remote Framebuffer), que é empregado nos servidores de código livre representados pela sigla VNC (Virtual Network Connection). Este artigo se concentrará na utilização do VNC devido à facilidade de download, instalação e execução. Também se destaca a possibilidade de criação de clientes VNC graças às bibliotecas de código aberto em C# existentes.

Antes de começarmos a detalhar como vamos criar uma aplicação cliente que acessa remotamente um servidor VNC, é preciso fazer a instalação do servidor e testar a conexão. Dentre as opções gratuitas disponíveis recomenda-se o RealVNC ou o TightVNC (veja na seção [Links](#)). Estes dois softwares possuem pacotes para download que contêm tanto o servidor como ferramentas cliente para as plataformas Windows, Mac OS e Linux.

Os leitores que desejarem executar os códigos fontes apresentados neste artigo devem possuir um servidor VNC instalado e configurado. Executar esta instalação é simples e não requer muitos passos, apesar de serem necessários privilégios administrativos no computador e a eventual liberação de portas de comunicação em um firewall. Uma vez que o servidor já esteja instalado é preciso saber qual é o endereço IP (ou nome do host), a porta utilizada (geralmente é 5900) e definir uma senha para controlar o acesso remoto. O protocolo VNC utiliza o tipo de conexão TCP e, por isso, é necessária uma conexão constante e com uma boa qualidade para que o modo de interação remoto e visualização sejam adequados.

A partir do momento em que o servidor VNC está instalado e com estas informações em mãos (endereço do host, porta e senha), recomenda-se utilizar um segundo computador, ou seja, um computador diferente daquele que possui o servidor VNC, e testar a conexão remota com a ferramenta cliente disponibilizada junto com o RealVNC ou o TightVNC. Uma vez que a conexão esteja estabelecida, pode-se controlar remotamente o computador que possui o servidor VNC instalado e interagir com o sistema operacional como se o usuário estivesse fisicamente em frente ao computador.

Um servidor VNC pode ser executado como um serviço que é executado em background, isto é, sem exigir interação com o usuário. Desta maneira, quando o sistema operacional for iniciado o serviço VNC é iniciado junto e, a partir desse momento, o servidor pode receber conexões remotas. Em geral, os serviços que trabalham desta forma possuem suas permissões limitadas ao usuário do sistema operacional associado ao serviço. Contudo, um usuário remoto que se conecta no servidor pode realizar um logon com qualquer usuário (fornecendo um login e senha do sistema operacional) e obter o acesso desejado. Esta característica não é uma falha de segurança, pois o que o serviço VNC proporciona é um acesso ao computador: se já houver algum usuário logado, o serviço VNC vai fornecer acesso ao computador como

se fosse o usuário. Se o computador estiver bloqueado ou pedindo credenciais pra realizar o logon, o usuário remoto deve fornecer o login e senha como se estivesse em frente ao computador.

É importante destacar que o serviço de compartilhamento de tela proporcionado pelo VNC não requer um login. Contudo, opcionalmente é possível configurar apenas senha de acesso ao servidor VNC. Uma vez que se tenha acessado o computador remoto, geralmente é necessário fazer um login no sistema operacional fornecendo as credenciais completas, ou seja, um usuário e senha válidos para o sistema operacional.

Quando um cliente se conecta remotamente a um servidor VNC, ele assume o controle do mouse e do teclado, ou seja, os eventos do teclado e do mouse são replicados para o servidor remoto. Porém, isso não quer dizer que o usuário que está em frente ao mouse e teclado do servidor remoto não possa utilizá-lo. O que acontece é que tanto os eventos de mouse/teclado remoto e local são aceitos pelo sistema operacional. Deste modo, é preciso coordenar como será a interação, uma vez que ambos os usuários (o remoto e o local) podem controlar o computador.

Apesar desta situação de controle mútuo parecer estranha, existem diversas possibilidades para colaboração neste cenário. Há, inclusive, ferramentas de áudio e vídeo conferência que exploram esta possibilidade para permitir que, durante uma reunião, mais de um usuário possa utilizar de forma síncrona uma aplicação de desenho, por exemplo. Estes cenários são comuns onde há uma adoção de ferramentas de colaboração síncrona que são agregadas aos canais de comunicação em tempo real com áudio e vídeo.

O maior benefício do compartilhamento de aplicativos é que um usuário remoto pode executar o software sem instalação no seu computador, mesmo que o software não seja compatível com o seu sistema operacional ou que exija muito mais poder de processamento do que o seu computador normalmente pode possuir. Isso ocorre porque o usuário remoto não está executando o software em seu computador, ele está apenas visualizando e controlando o ambiente de trabalho (e, portanto, o software) do computador host.

Apesar dos servidores e clientes VNC existentes possuírem algoritmos e técnicas adequadas para reduzir e compactar a quantidade de bytes necessários para se utilizar remotamente um computador como se ele estivesse local, a tecnologia e o protocolo RFB não são adequados para a transmissão de vídeo e áudio. Sendo assim, a tecnologia é recomendada para situações onde não há uma atualização constante da tela e não há a necessidade de transmissão de áudio.

Contudo, certas empresas exploraram o conceito de controle remoto e compartilhamento de tela para proporcionar serviços baseados em tecnologias armazenadas na nuvem (cloud computing). Um exemplo claro é a empresa OnLive (ver seção [Links](#)) que possui um serviço onde é possível jogar diversos games sob demanda diretamente no computador. O que a plataforma da OnLive permite é um serviço de alta fidelidade e baixa latência para transferir a imagem do game, que está rodando em seus servidores. A plataforma da OnLive também capta e envia para

Criando uma aplicação para conexões remotas

o servidor as interações do mouse/teclado/controle/Kinect/Wiimote ou qualquer outro dispositivo utilizado para interagir como game. Em resumo, o serviço proporciona a um usuário jogar um game sem ter que comprar e instalar o console e o jogo. Há, inclusive, a possibilidade de se conectar nas redes Xbox Live, PSN Network e Nintendo network para jogar com colegas, comprar filmes, visualizar estatísticas e troféus e interagir com a comunidade de jogadores.

Outro exemplo de empresa que se especializou na utilização de tecnologias para o compartilhamento de tela é a Citrix. O foco é na utilização de sistemas gerenciais do tipo ERP, evitando que seja necessário instalar e manter softwares adicionais nos computadores dos clientes. Além de proporcionar o acesso remoto a servidores que contém o software, evitando o processo de instalação, atualização, manutenção e suporte, as tecnologias da Citrix também possuem uma integração com servidores virtuais, chegando ao ponto de permitir a virtualização e o acesso remoto completo do desktop do usuário que pode acessar o software de qualquer PC, Mac, Linux, Tablet ou smartphone.

Conhecendo as alternativas

O desenvolvedor da plataforma .NET que trabalha com C# possui algumas alternativas para criar sua própria aplicação que age como cliente VNC. Tanto o RealVNC como o TightVNC possuem versões com código livre, porém elas são baseadas na linguagem de programação C/C++. Os projetos mais populares no .NET que disponibilizam classes e outros recursos para acessar um servidor VNC em C# são o VNC-Client for .NET, o .NET VNC Viewer e o VNCSharp.

O projeto VNC-Client for .NET contém apenas um exemplo de como criar uma aplicação que se conecta a um servidor VNC remoto. Ele faz uso do arquivo binário das bibliotecas NzipLib e DirectX e possui uma documentação simples das classes e métodos. Contudo, este projeto não foi criado na forma de uma biblioteca autocontida que possa ser utilizada em qualquer tipo de aplicação .NET (console, Windows Forms, WPF, ASP .NET), o que requer do desenvolvedor um esforço grande para trabalhar com seus métodos e classes.

O projeto .NET VNC Viewer é disponibilizado como uma aplicação cliente criada a partir do tipo de projeto Windows Forms. Apesar de ser completo, este projeto não disponibiliza uma biblioteca e nem um controle. Além disso, há não documentação do projeto e ele é disponibilizado como um conjunto de classes que deve ser compilado diretamente no console, ou seja, não há uma solução ou projeto do Visual Studio. Este projeto não possui dependências externas e nos testes realizados apresentou boa performance de atualização da tela do computador remoto.

Por fim, o projeto VNCSharp segue o modelo de licença GPL e também apresentou boa performance de atualização da tela do computador remoto. Este projeto é disponibilizado como uma biblioteca .NET (assembly) que contém classes e um controle do tipo Windows Forms. Apesar do projeto não estar mais em

desenvolvimento, não possuir uma boa documentação e não precisar de dependências externas, o código do projeto é bem comentado. Este projeto conta ainda com um exemplo simples que mostra como utilizar a biblioteca e o controle em uma aplicação do tipo Windows Forms.

Devido às características dos projetos descritos e à facilidade de uso, este artigo se concentrará na utilização da biblioteca VNCSharp para montar um exemplo de aplicação console em C# que se conecta a um servidor VNC remoto e envia um conjunto de letras como se elas tivessem sido digitadas pelo usuário no teclado. O exemplo mostra também como enviar eventos do mouse para movê-lo remotamente ou simular um clique.

O exemplo apresentado pode ser modificado para realizar tarefas de manutenção automaticamente em diversos computadores de usuários finais, mesmo realizar um treinamento remoto ou compartilhar uma aplicação existente entre vários usuários em uma aplicação de áudio/vídeo conferência.

Download, instalação e exemplo do VNCSharp

O projeto VNCSharp disponibiliza seus recursos em diferentes arquivos. Neste artigo utilizou-se a versão 1.0 e o leitor pode fazer o download do arquivo vncsharp-1.0-bin.zip, que contém a biblioteca compilada no assembly VncSharp.dll, ou o download do arquivo vncsharp-1.0-src.zip, que contém uma solução do Visual Studio com os códigos fontes deste projeto. Há também o projeto de exemplo no formato Windows Form compactado no arquivo vncsharpxexample-csharp-1.0-src.zip. Veremos como compilar o projeto de exemplo e iniciar conexões remotas para testar a biblioteca e o componente Windows Form disponibilizados.

Após o download e extração dos arquivos contidos em vncsharpxexample-csharp-1.0-src.zip, abra a solução disponibilizada no arquivo VncSharpExampleCS.sln no Visual Studio (a partir da versão 2005) e compile o projeto. O leitor deve notar através da lista de referências (References) que a única dependência externa deste projeto é o assembly VncSharp.dll que contém as classes, componentes e outros recursos organizados no namespace VncSharp.

O projeto de exemplo contém um Form chamado VncSharpExampleForm, que é composto de menus e de um controle especial da classe VncSharp.RemoteDesktop chamado rd. Este controle está dentro do assembly VncSharp.dll e ele é responsável pela apresentação da tela e captura/envio de eventos do mouse e teclado. Existem outros detalhes importantes deste exemplo, porém fica a cargo do leitor explorar o projeto para compreendê-lo melhor.

Após compilar e executar este projeto, podemos clicar no menu File e escolher a opção Connect. Em seguida devemos informar o endereço IP ou nome do servidor VNC e a senha. Por padrão, este cliente assume que o servidor está utilizando a porta 5900, mas caso seja empregada uma porta diferente, basta indicar no nome do servidor o número da porta precedido do caractere : (dois pontos), como 10.0.0.100:5901 para indicar que o computador 10.0.0.100 possui um servidor VNC na porta 5901.

A **Figura 1** mostra a conexão remota com um servidor executando o Windows XP (note que o ícone do VNC no canto esquerdo da barra de tarefas do Windows mudou de cor para indicar a conexão), enquanto a **Figura 2** mostra uma conexão remota com um computador executando o Mac OS Lion, que possui um servidor VNC nativo, isto é, disponibilizado junto com o sistema operacional. Já a **Figura 3** mostra um exemplo de conexão com um dispositivo celular rodando o sistema operacional Android 4.0.4 (Ice Cream Sandwich) com um servidor VNC. Qualquer sistema operacional que possua um servidor VNC que siga o padrão do protocolo RFB pode receber uma conexão do VncSharp, inclusive sistemas operacionais das plataformas móveis (iOS, Android e Windows 8).

Uma vez que o teste de conexão tenha sido realizado com sucesso, é preciso utilizar a opção Disconnect do menu File para finalizar a conexão e liberar os recursos utilizados do lado do cliente.

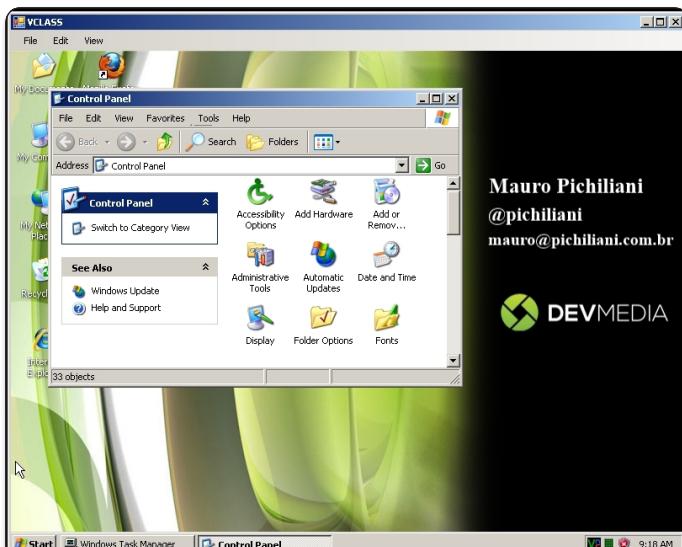


Figura 1. Cliente VncSharpExampleForm conectado a um servidor VNC no Windows XP



Figura 2. Cliente VncSharpExampleForm conectado a um servidor VNC no Mac OS Lion



Figura 3. Cliente VncSharpExampleForm conectado a um servidor VNC no Android 4.0.4 (Ice Cream Sandwich)

Enviando caracteres

Para auxiliar o leitor a compreender como trabalhar com a biblioteca do projeto VNCSharp, este artigo apresentará um projeto do tipo Console Application escrito em C# que utiliza apenas as classes básicas do VNCSharp. O objetivo é mostrar como é possível abrir uma conexão com um servidor VNC remoto, enviar uma sequência de caracteres como se fosse algo digitado pelo usuário e fechar a conexão. Apesar de este exemplo ser simples, ele ilustra bem como é possível utilizar o C# para se conectar a um servidor VNC independente da plataforma e sistema operacional na qual este servidor esteja sendo executado.

Apesar do exemplo apresentado neste artigo utilizar o tipo de projeto Console Application, as classes do VNCSharp podem ser utilizadas em qualquer tipo de aplicação .NET. Há, inclusive, a possibilidade de criar um cliente VNC inteiro em HTML5 e ASP.NET para que o usuário possa acessar remotamente outros computadores diretamente pelo browser sem a necessidade de instalação de nenhum software adicional.

Um cliente VNC basicamente faz três tarefas principais: ele controla a conexão (abrir/fechar), recebe a atualização de telas e envia teclas ou eventos do mouse. Também é possível realizar tarefas mais complexas, como copiar o conteúdo do que foi colocado na área de transferência do servidor remoto com o comando cópia (ou Control+C) e colar (Control+V) este conteúdo no computador que executa o cliente. Porém, como foi explicado na primeira sessão deste artigo, veremos um exemplo que detalhará como

Criando uma aplicação para conexões remotas

abrir a conexão, enviar caracteres, enviar eventos do mouse e fechar a conexão. O leitor que se interessar por recursos avançados proporcionados pela tecnologia VNC pode analisar o código fonte do projeto VNCSharp, pois ele possui diversas classes que implementam o protocolo RFB, interagem com os dados enviados e recebidos, compactam e descompactam os dados e permitem a utilização de recursos avançados.

Vamos começar o exemplo abrindo o Visual Studio e criando um novo projeto do tipo Console Application chamado ConsoleVNC. Em seguida é preciso fazer a referência ao assembly VncSharp.dll que é disponibilizado no arquivo vncsharp-1.0-bin.zip do projeto VNCSharp.

Para fazer a referência corretamente é preciso clicar com o botão direito do mouse sobre o item References na janela Solution Explorer e escolher a opção Add References. Em seguida escolha a aba Browser e navegue para a pasta onde o arquivo VncSharp.dll está armazenado, selecione o arquivo e clique no botão OK. Agora que a referência está completa e podemos começar a utilizar as classes do VNCSharp. O código fonte comentado deste exemplo é apresentado na **Listagem 1** que contém apenas o método Main().

Para este exemplo utilizaremos um objeto da classe VncClient chamado v. O primeiro passo é abrir a conexão e faremos isso com o método Connect(), que recebe o endereço do servidor, o número da tela (screen) e a porta. Neste exemplo utilizaremos o endereço 10.0.0.100 e a porta 5900. O valor numérico da tela deve ser 0, uma vez que não vamos trabalhar com múltiplas telas neste exemplo.

Um servidor VNC pode ser configurado para permitir conexões de usuário sem que haja a necessidade de senha. Contudo, tal opção não é recomendada por motivos de segurança. Na **Listagem 1** o método Connect() retorna um valor booleano indicando se é necessário ou não fornecer uma senha para se conectar neste servidor. Caso seja necessário, a variável needPassword receberá o valor verdadeiro (true) e devemos chamar o método Authenticate(). Caso não seja necessário fornecer uma senha, podemos chamar o método Initialize(). Para fornecer a senha chamamos o método Authenticate() passando como parâmetro a senha, que no exemplo da **Listagem 1** é abc123. Caso o retorno de Authenticate() seja verdadeiro, prosseguimos com conexão e chamamos o método Initialize().

O último método necessário para abrir corretamente a conexão é o método StartUpdates(), pois ele sinaliza para o servidor que

Listagem 1. Exemplo de conexão com o projeto VncSharp

```
01 using System;
02 using VncSharp;
03 using System.Windows;
04 using System.Drawing;
05 using System.Threading;
06
07 namespace ConsoleVNC
08{
09    class ConsoleVNC
10    {
11        // Esta variável representa o cliente VNC
12        static VncClient v = new VncClient();
13
14        static void Main(string[] args)
15        {
16            // Armazena se precisa ou não enviar a senha
17            bool needPassword;
18
19            try
20            {
21                // Abre a conexão como servidor 10.0.0.100
22                // na porta 5900 e na tela 0
23                needPassword = v.Connect("10.0.0.100", 0, 5900);
24                if (needPassword)
25                {
26                    // Envia a senha abc123
27                    if (v.Authenticate("abc123"))
28                    {
29                        // Realiza as inicializações necessárias
30                        v.Initialize();
31                    }
32                else
33                {
34                    // Problema: informar usuário e finalizar
35                    Console.WriteLine("Autentication problem!");
36                    return;
37                }
38            }
39            else
40                // Realiza as inicializações necessárias
41                v.Initialize();
42
43                // Indicando o método que vai receber o
44                // evento de desconexão
45                v.ConnectionLost +=
46                    new EventHandler(ConsoleVNC.ClientConnectionLost);
47
48                // Começa a receber as atualizações de tela
49                v.StartUpdates();
50            }
51            catch (Exception e)
52            {
53                // Problemas na conexão: informar usuário e finalizar
54                Console.WriteLine("Connection error:" + e.Message);
55                return;
56            }
57            // Mostra informações do host
58            Console.WriteLine("Connection OK!");
59            Console.WriteLine("Host Name:" + v.HostName);
60            Console.WriteLine("Screen Size:" +
61                v.Framebuffer.Width.ToString() + "x"
62                + v.Framebuffer.Height.ToString());
63
64            // Envia a frase 'Hello World with VNC' para o host
65            ConsoleVNC.SendString("Hello World with VNC");
66
67            // Desconecta e libera os recursos
68            v.Disconnect();
69
70        }
71    }
```

estamos prontos para receber as atualizações de tela. Neste exemplo não iremos trabalhar com a atualização de tela, pois apenas vamos enviar um conjunto de caracteres para o servidor VNC. Um ponto importante é que toda a sequência de conexão está protegida por um bloco try/catch e as devidas mensagens de erro são apresentadas, caso ocorra algum problema.

Após a conexão entre o cliente e o servidor ser estabelecida, é possível que, por algum motivo, o servidor encerre esta conexão. Por exemplo: o computador que contém o serviço VNC foi reiniciado. Devido a esta possibilidade a biblioteca VNCSharp possui um evento que será disparado no cliente caso o servidor encerre a conexão. Para indicar um método que será chamado nesta situação devemos preencher a propriedade ConnectionLost com um objeto da classe EventHandler, que permite o acesso a um método que tratará este evento. No código da **Listagem 2** o método estático ClientConnectionLost() é utilizado como evento para a propriedade ConnectionLost do objeto v, que é da classe VncClient.

Listagem 2. Método para tratar desconexão do servidor

```
01 // Evento gerado caso o servidor desconecte o cliente
02 static void ClientConnectionLost(object sender, EventArgs e)
03 {
04     // Liberando os recursos da conexão do cliente
05     ConsoleVNC.v.Disconnect();
06 }
```

O método ClientConnectionLost() segue a assinatura de um evento, ou seja, ele retorna void e possui dois parâmetros: um objeto da classe Object e um objeto da classe EventArgs. A utilização do evento ocorre quando o servidor finaliza a conexão e ela será simples, pois somente vamos chamar o método Disconnect() da classe VncClient para liberar os recursos alocados. Contudo, o leitor pode codificar outras funcionalidades neste evento, como notificar o usuário de alguma forma na interface gráfica que o servidor finalizou a conexão.

Uma vez que o primeiro passo esteja completo, ou seja, abrir a conexão, podem mostrar algumas características do host. A propriedade HostName da classe VncClient retorna o nome do host e as propriedades Framebuffer.Width e Framebuffer.Height retornam valores numéricos com a largura e altura da tela do host, respectivamente. O exemplo mostra estes valores para o usuário e segue seu fluxo para enviar uma frase pelo VNC.

O envio da frase “Hello World with VNC” é realizado através do método SendString(), que é mostrado na **Listagem 3**. Este método basicamente recebe uma String como parâmetro e faz um loop para pegar cada caractere individual deste String, uma vez que esta operação é necessária, pois o método WriteKeyboardEvent() da classe VncClient requer a conversão de cada caractere enviado para UInt32. Também é necessário passar o segundo parâmetro de WriteKeyboardEvent() como verdadeiro para simular que o usuário realmente digitou algo.

No exemplo do código apresentado na **Listagem 3** um texto foi enviado por meio da conversão de cada um dos caracteres para

um valor numérico do tipo UInt32. Caso seja necessário enviar teclas especiais como Enter, Tab, SHIFT ou ALT, é preciso enviar um código numérico do tipo UInt32 especial para cada tecla. Este código é definido no protocolo RFB, porém a **Tabela 1** apresenta algumas das teclas especiais mais utilizadas e também o respectivo valor no tipo de dados UInt32 representados por um valor hexadecimal.

Listagem 3. Método para envio de caracteres com o VncSharp

```
01 static void SendString(String s)
02 {
03     // Envia um caracter por vez
04     foreach (Char c in s.ToCharArray())
05     {
06         // É preciso converter o caracter para UInt32
07         // O segundo parâmetro é verdadeiro para simular que o
08         // usuário pressionou as teclas
09         v.WriteKeyboardEvent((UInt32) c, true);
10    }
11 }
```

| Tecla | Valor numérico (UInt32) | Tecla | Valor numérico (UInt32) |
|--------------------|-------------------------|-------|-------------------------|
| Tab | 0x0000FF09 | F1 | 0x0000FFBE |
| Enter | 0x0000FF0D | F2 | 0x0000FFBF |
| Esc | 0x0000FF1B | F3 | 0x0000FFC0 |
| Home | 0x0000FF50 | F4 | 0x0000FFC1 |
| Seta para esquerda | 0x0000FF51 | F5 | 0x0000FFC2 |
| Seta para cima | 0x0000FF52 | F6 | 0x0000FFC3 |
| Seta para direita | 0x0000FF53 | F7 | 0x0000FFC4 |
| Seta para baixo | 0x0000FF54 | F8 | 0x0000FFC5 |
| PageUp | 0x0000FF55 | F9 | 0x0000FFC6 |
| PageDown | 0x0000FF56 | F10 | 0x0000FFC7 |
| End | 0x0000FF57 | F11 | 0x0000FFC8 |
| Insert | 0x0000FF63 | F12 | 0x0000FFC9 |
| Shift | 0x0000FFE1 | | |
| Alt | 0x0000FFE9 | | |
| Ctrl | 0x0000FFE3 | | |
| Delete | 0x0000FFFF | | |

Tabela 1. Teclas especiais e seus respectivos valores UInt32

O envio de teclas deve ser realizado de forma individual, ou seja, uma tecla por vez. É preciso lembrar que certas teclas iniciam e finalizam um modo, por exemplo, a tecla Shift que inicia/termina o modo de letras maiúsculas.

Por fim, o último passo do programa de exemplo é chamar o método Disconnect() da classe VncClient para finalizar a conexão e liberar os recursos. Para testar o exemplo é preciso instalar corretamente um servidor VNC e abrir algum tipo de aplicação que aceite caracteres, como o editor Notepad do Windows. Se tudo estiver configurado corretamente (servidor VNC, firewall,

Criando uma aplicação para conexões remotas

senha, etc.), ao rodar a aplicação o usuário remoto verá a frase “Hello World with VNC” sendo escrita na aplicação que estiver com foco do cursor do mouse.

Enviando eventos do mouse

Agora que já vimos como se conectar a um servidor VNC remoto e enviar um conjunto de teclas como se fosse o usuário digitando algo no teclado, veremos como enviar eventos do mouse. Este tipo de funcionalidade é muito útil quando precisamos acessar algum ícone ou elemento visual da interface do usuário que requer o posicionamento e o clique do mouse.

Um cenário comum para este tipo de situação é no suporte a usuários. Vamos supor que um usuário está com dificuldade para acessar uma opção no menu de uma aplicação. O atendente de suporte que recebeu uma ligação deste usuário pode acionar um comando da sua interface e posicionar automaticamente o cursor do mouse do usuário sem que o atendente de suporte tenha acesso remoto completo ao sistema operacional do usuário. Neste cenário a possibilidade do atendente de suporte ter acesso a arquivos e conteúdos da máquina do usuário é reduzida, evitando uma possível violação da privacidade.

Antes de enviar os eventos do mouse é preciso levar em consideração que a resolução do computador remoto (o host com o servidor VNC) pode ser diferente da resolução do computador com o cliente VNC. Portanto, é preciso fazer um mapeamento e conversão de coordenadas locais para as coordenadas remotas do ponteiro para que o uso do mouse pareça natural e adequado. O controle disponibilizado pelo VncSharp implementa um algoritmo que faz este tipo de mapeamento de coordenadas de tela, porém veremos um exemplo simples de movimentação do cursor em um conjunto de coordenadas da tela pré-definido.

Para enviar eventos do mouse utilizando a classe VncClient é preciso utilizar o método WritePointerEvent(), que requer dois parâmetros: um valor numérico para indicar qual botão do mouse foi pressionado e um objeto da classe System.Drawing.Point com as coordenadas de onde o cursor do mouse será posicionado.

No primeiro exemplo veremos como enviar um comando para simular a movimentação do mouse. Para isso vamos criar um novo método chamado SendMouseMove(), que vai enviar comandos de movimentação do mouse para seguir o formato de um quadrado com 100 pixels de lado.

O quadrado terá seus vértices nos pontos (100,100), (200,100), (200,200) e (100,200) da interface do usuário e será desenhado ponto a ponto. A movimentação do cursor será feita através de quatro loops que utilizam o contador do loop para variar as coordenadas de um objeto da classe System.Drawing.Point, que será enviado como segundo parâmetro do método WritePointerEvent(). Para visualizarmos o caminho do cursor do mouse utilizaremos um intervalo de 10 milisegundos entre cada movimentação do cursor através da chamada do método Thread.Sleep().

O primeiro parâmetro do método WritePointerEvent() requer um valor numérico para indicar qual é o botão do mouse que

está sendo pressionado. Neste primeiro exemplo vamos apenas movimentar o cursor do mouse e, por isso, passaremos o valor 0 como primeiro parâmetro para o método WritePointerEvent(). A **Listagem 4** mostra o conteúdo completo do método SendMouseMove() que pode ser chamado logo após o método SendString() da **Listagem 1**.

Listagem 4. Método SendMouseMove() que envia comandos para movimentar o cursor do mouse

```
01 static void SendMouseMove()
02 {
03     // Topo do quadrado
04     for (int i = 100; i <= 200; i++)
05     {
06         Thread.Sleep(10);
07         v.WritePointerEvent(0, new System.Drawing.Point(i, 100));
08     }
09     // Lado direito do quadrado
10     for (int i = 100; i <= 200; i++)
11     {
12         Thread.Sleep(10);
13         v.WritePointerEvent(0, new System.Drawing.Point(200, i));
14     }
15     // Base do quadrado
16     for (int i = 200; i >= 100; i--)
17     {
18         Thread.Sleep(10);
19         v.WritePointerEvent(0, new System.Drawing.Point(i, 200));
20     }
21
22     // Lado esquerdo do quadrado
23     for (int i = 200; i >= 100; i--)
24     {
25         Thread.Sleep(10);
26         v.WritePointerEvent(0, new System.Drawing.Point(100, i));
27     }
28 }
```

O próximo exemplo que veremos envia o clique do mouse remotamente. Para enviar o clique do mouse também utilizaremos o método WritePointEvent() da classe VncClient, porém desta vez precisamos indicar no primeiro parâmetro qual botão do mouse será pressionado. A **Tabela 2** mostra a correspondência de valores números para o valor do primeiro parâmetro do método WritePointEvent() e os respectivos botões do mouse. Também são mostrados na **Tabela 2** os valores correspondentes à movimentação da roda do mouse (scroll) para frente e para trás.

| Valor do parâmetro | Botão do mouse |
|--------------------|---------------------------------------|
| 0 | Nenhum botão pressionado |
| 1 | Esquerdo |
| 2 | Meio |
| 4 | Direito |
| 8 | Roda do mouse movimentada para frente |
| 16 | Roda do mouse movimentada para trás |

Tabela 2. Valores numéricos do primeiro parâmetro de WritePointEvent() e botões do mouse

É possível simular a situação onde mais de um botão foi pressionando adicionando os valores numéricos. Por exemplo, se desejarmos indicar que os botões direito e esquerdo foram pressionados juntos utilizaremos o valor 5, pois $1+4 = 5$.

Devemos tomar cuidado com um último detalhe antes de enviar um evento de clique do mouse com o método WritePointEvent(): nos sistemas operacionais que seguem o modelo de interface WIMP (**BOX 1**) um clique do mouse geralmente é representado por dois eventos: MouseDown e MouseUp. Isto quer dizer que precisamos fazer duas chamadas aos métodos WritePointEvent() para simular um clique do mouse, sendo que a primeira delas deve indicar qual foi o botão utilizado e a segunda chamada indica o valor 0 (ausência de botão).

BOX 1. Interfaces WIMP

WIMP é um acrônimo usado para descrever todos os elementos de sistemas operacionais que definem o modo como interagimos com os computadores cuja GUI (Graphical User Interface Interface Gráfica de usuário) seja baseada em janelas. W significa Windows (janelas), I significa Icons (ícones), M significa Menus e P significa Pointers (ponteiros).

Em 1973 a Xerox desenvolveu o primeiro sistema operacional com um paradigma WIMP para o Xerox Alto. Desde aquela época até recentemente as GUI baseadas em WIMP foram as líderes da interação entre humanos e computadores. Sistemas operacionais como o Windows, Mac OS, OS/2 e o Linux adotaram os conceitos WIMP em suas interfaces, o que facilitou a interação para os usuários que utilizam mouse e teclado.

Com as novas tecnologias e recursos para desenvolvimento de páginas Web, muitos dos conceitos WIMP vêm sendo desafiados, como a ausência de menus tradicionais. Além disso, os sistemas operacionais móveis como iOS, Android e Windows 8 abandonaram muitos dos conceitos WIMP ao adotarem interfaces baseadas em toques e gestos realizados diretamente no dispositivo sensível a toque que apresenta as informações.

Se entre a primeira chamada (equivalente ao MouseDown) e a segunda chamada (equivalente ao MouseUp) não houver uma diferença nas coordenadas, ou seja, o valor do segundo parâmetro de WritePointEvent() for igual, temos o evento do clique do mouse. Porém, caso haja uma diferença de coordenadas entre a primeira chamada de WritePointEvent() e a segunda temos a representação do clicar, arrastar e soltar, também conhecido como drag-and-drop. Por fim, se desejarmos simular o evento de clique duplo basta fazer duas vezes as duas chamadas necessárias para um clique normal.

Para mostrar como simular o envio de um evento de clique do mouse pelo VNC, a **Listagem 5** mostra o método SendMouseClick(), que recebe como parâmetro um valor numérico correspondente ao botão do mouse como apresentado na **Tabela 2**. O conteúdo deste método é simples: ele cria uma variável chamada mask para indicar qual é o botão do mouse que está sendo acionado e simula um clique no centro da tela, utilizando o método WritePointerEvent. Para testar este exemplo leitor deve realizar a chamada de SendMouseClick() duas vezes, sendo que na primeira vez deve-se indicar o valor do botão do mouse (1, 2 ou 4) e na segunda vez deve-se indicar o valor 0 para que um clique seja enviado para o servidor VNC remoto.

Listagem 5. Método SendMouseClick() que envia comandos para simular o clique de um botão do mouse no centro da tela.

```
01 static void SendMouseClick(byte botao)
02 {
03     // Mandando o cursor do mouse para o centro da tela
04
05     byte mask;
06
07     // 1 = botão da esquerda
08     // 2 = botão do centro
09     // 4 = botão da direita
10     mask = botao;
11
12     v.WritePointerEvent(
13         mask,
14         new System.Drawing.Point
15             (v.Framebuffer.Width/2,v.Framebuffer.Height/2)
16     );
}
```

Conclusão

A utilização de tecnologias de compartilhamento de tela e controle remoto representa uma oportunidade para a realização de tarefas que necessita do acesso completo local a um computador. Apesar de ser possível realizar diversas operações com ferramentas de acesso remoto ao shell, tais como ferramentas de SSH ou mesmo servidores de FTP/SFTP, o acesso direto ao desktop de um computador pode fornecer diversas oportunidades para automação de tarefas, treinamento de usuário, resolução de problemas de suporte, deploy de aplicações, verificação de configurações e outras.

As principais ferramentas de compartilhamento de telas livres, agrupadas sob a sigla VNC, permitem o acesso remoto completo de um servidor, porém elas representam potenciais problemas de segurança e violação de privacidade. Por exemplo, através do acesso por VNC um usuário com más intenções pode acessar arquivos pessoais de usuários ou mesmo observar a digitação de senhas e outras informações sigilosas. Neste contexto, permitir o acesso remoto por meio do VNC se torna uma atitude arriscada.

Contudo, é possível criar uma aplicação na plataforma .NET que encapsula o acesso remoto a um servidor VNC. Esta aplicação pode se conectar a um host remoto e enviar determinadas teclas ou eventos do mouse e realizar certas tarefas sem nem ao menos mostrar para quem iniciou este cliente VNC a tela do desktop remoto. Desta maneira evitam-se possíveis problemas de violação de privacidade.

Apesar da tecnologia envolvida nos servidores VNC ser básica, ou seja, permite apenas o envio de teclas e eventos do mouse e recebe do host a atualização de telas, existem várias possibilidades para incluir novas funcionalidades. Criptografia das informações, ferramentas de chat com texto e VoIP, envio/transferência de arquivos e acesso por meio da Web são apenas algumas das funcionalidades adicionais que não são contempladas pelo protocolo RFB e que podem ser agregadas em uma solução que utilize tecnologias de compartilhamento de tela.

Criando uma aplicação para conexões remotas

Outro cenário de uso para esta tecnologia envolve o treinamento. Em uma situação onde não há recursos para treinamento adequados (projetores, sala de treinamento, etc) é possível treinar usuários de forma automatizada diretamente nas suas interfaces, sem a necessidade de locomoção do usuário. Este tipo de treinamento com demonstrações fornece a sensação de presença remota e o usuário que está sendo treinado pode acompanhar as ações sendo realizadas como se o instrutor estivesse trabalhando junto com ele.

Outra forma para utilizar o compartilhamento de conteúdo ou apresentação são seminários Web ou webinars. Em um webinar um apresentador envia um convite por e-mail a um grupo de participantes que inclui um link e um código de acesso. Quando os participantes clicarem no link e digitarem seus códigos de acesso eles estão registrados na apresentação virtual. Neste cenário só pode haver um apresentador de cada vez que pode compartilhar seu desktop, suas apresentações com programas como o PowerPoint, ou partes de outras aplicações e software no computador host. Com o software de webinar, que é baseado em tecnologias de compartilhamento de tela, cada apresentador pode compartilhar o seu próprio ambiente de trabalho e arquivos. Talvez o exemplo mais claro deste tipo de tecnologia seja representado pelos serviços da empresa WebEx (parte do grupo da Cisco) que fornece aplicativos de demanda, reunião online, web conferência e aplicações de vídeo conferência.

Com a utilização de um servidor VNC também é possível criar um serviço de monitoração constante do desktop de um computador remoto. Tal ferramenta pode ser útil para revisar ações realizadas ou medir o tempo gasto em determinada aplicação. Por exemplo: um gerente de projeto deseja saber quanto tempo um programador passou como a interface do Visual Studio aberto e quanto tempo ele passou navegando na internet. Ao invés de programar um serviço específico para este tipo de monitoração, é possível instalar um servidor VNC no computador do desenvolvedor e criar um cliente VNC em C# que periodicamente lê a tela do desenvolvedor e grava a tela em uma sequência de imagens ou em um banco de dados. Posteriormente o gerente de projeto pode revisar as imagens e verificar exatamente quanto tempo o desenvolvedor gastou na interface do Visual Studio ou no navegador. Além disso, o gerente de projeto também pode identificar como o programador utilizou o IDE ou mesmo qual tipo de site foi visitado pelo programador. Novamente, este uso da tecnologia VNC possui implicações no campo da privacidade e segurança

que devem ser consideradas com cautela por quem emprega este tipo de tecnologia para fins de monitoria.

O leitor que desejar automatizar tarefas remotas que exigem acesso direto ao servidor em diversas máquinas pode se beneficiar do conteúdo apresentado neste artigo, uma vez que ele mostrou como é possível desenvolver uma aplicação .NET na linguagem C# que age como um cliente de um servidor VNC através do uso das classes fornecidas pelo projeto VNCSharp.

Autor



Mauro Pichiliani

mauro@pichiliani.com.br – <http://pichiliani.com.br>

É bacharel em Ciência da Computação, mestre e doutorando pelo ITA (Instituto Tecnológico de Aeronáutica) e MCP, MCDBA e MCTS. Trabalha há mais de 10 anos utilizando diversos bancos de dados SQL e NoSQL. É colunista de SQL Server do web site iMasters (<http://www.imasters.com.br>) e mantenedor do DatabaseCast (@databasecast), o podcast brasileiro sobre banco de dados.



Links:

Projeto VNC-Client for .NET

<http://sourceforge.net/projects/dnvnc/>

Projeto.NET VNC Viewer

<http://sourceforge.net/projects/dotnetvnc/>

Projeto VNCSharp

<http://cdot.senecac.on.ca/projects/vncsharp>

Servidor e cliente VNC gratuito RealVNC para Windows, Mac OS e Linux

<https://www.realvnc.com>

Servidor e cliente VNC gratuito TightVNC para Windows, Mac OS e Linux

<http://www.tightvnc.com/>

OnLive Games

<http://games.onlive.com/>

Você gostou deste artigo?

Dê seu voto em www.devmedia.com.br/netmagazine/feedback

Ajude-nos a manter a qualidade da revista!



Somos tão apaixonados por tecnologia que o nome da empresa diz tudo.

Porta 80 é o melhor que a Internet pode oferecer para sua empresa.

Já completamos 8 anos e estamos a caminho dos 80, junto com nossos clientes.

Adoramos tecnologia.
Somos uma equipe composta de gente que entende e gosta do que faz,
assim como você.



Estrutura

100% NACIONAL.
Servidores de primeira linha, links de alta capacidade.

Suporte diferenciado

Treinamos nossa equipe para fazer mais e melhor. Muito além do esperado.

Serviços

Oferecemos a tecnologia mais moderna, serviços diferenciados e antenados com as suas necessidades.

1-to-1

Conhecemos nossos clientes. Atendemos cada necessidade de forma única.
[Conheça!](#)

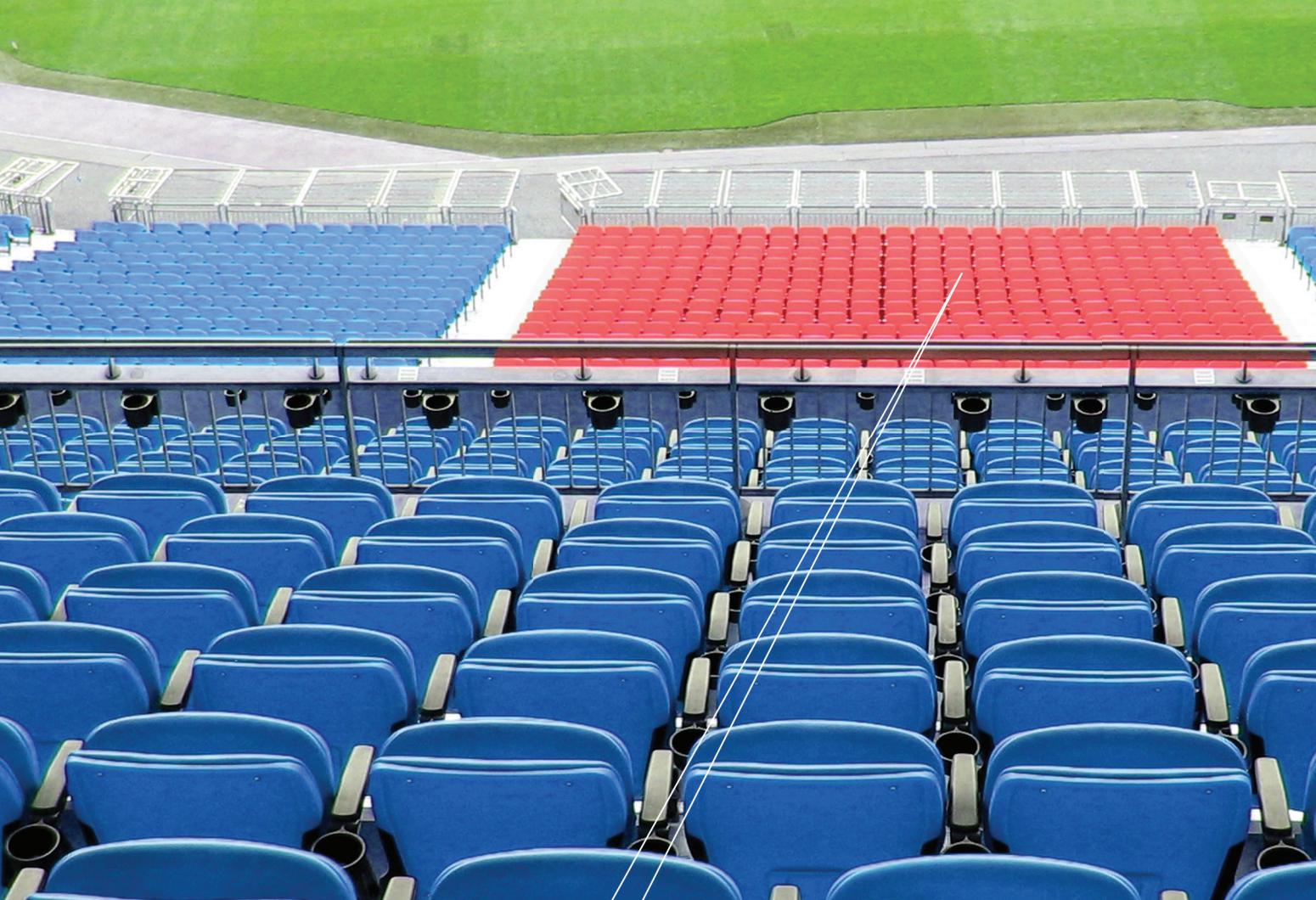


Porta 80
WEB HOSTING

Hospedagem | Cloud Computing | Dedicados | VoIP | Ecommerce |
Aplicações | Streaming | Email corporativo

porta80.com.br | comercial@porta80.com.br | twitter.com/porta80

SP 4063-8616 | RJ 4063-5092 | MG 4063-8120 | DF 4063-7486



Certificação é garantia
de acesso às melhores
vagas no mercado.

FORMAÇÃO DESENVOLVEDOR WEB .NET

Domine as mais modernas técnicas de programação para a plataforma .NET, incluindo os *frameworks* 4.5.

Seja um profissional capaz de enfrentar o dia a dia do desenvolvimento de um sistema corporativo Web e de gerar soluções que estejam alinhadas às necessidades das empresas.

MAIS INFORMAÇÕES:
www.infnet.edu.br/mcpdwebdeveloper