PARTE I

Introdução ao Microsoft Visual C# e ao Microsoft Visual Studio 2013

CAPÍTULO 1

Bem-vindo ao C#

Neste capítulo, você vai aprender a:

- Utilizar o ambiente de programação do Microsoft Visual Studio 2013.
- Criar um aplicativo de console em C#.
- Explicar o objetivo dos namespaces.
- Criar um aplicativo gráfico simples em C#.

Este capítulo apresenta o Visual Studio 2013, o ambiente de programação, e o conjunto de ferramentas projetadas para criar aplicativos para o Microsoft Windows. O Visual Studio 2013 é a ferramenta ideal para escrever código em C#, oferecendo muitos recursos que você vai conhecer à medida que avançar neste livro. Neste capítulo, você vai usar o Visual Studio 2013 para criar alguns aplicativos simples em C# e começar a construir soluções altamente funcionais para Windows.

Comece a programar com o ambiente do Visual Studio 2013

O Visual Studio 2013 é um ambiente de programação rico em recursos que contém a funcionalidade necessária para criar projetos grandes ou pequenos em C# que funcionam no Windows 7, no Windows 8 e no Windows 8.1. Você pode inclusive construir projetos que combinem módulos de diferentes linguagens, como C++, Visual Basic e F#. No primeiro exercício, você vai abrir o ambiente de programação do Visual Studio 2013 e aprender a criar um aplicativo de console.

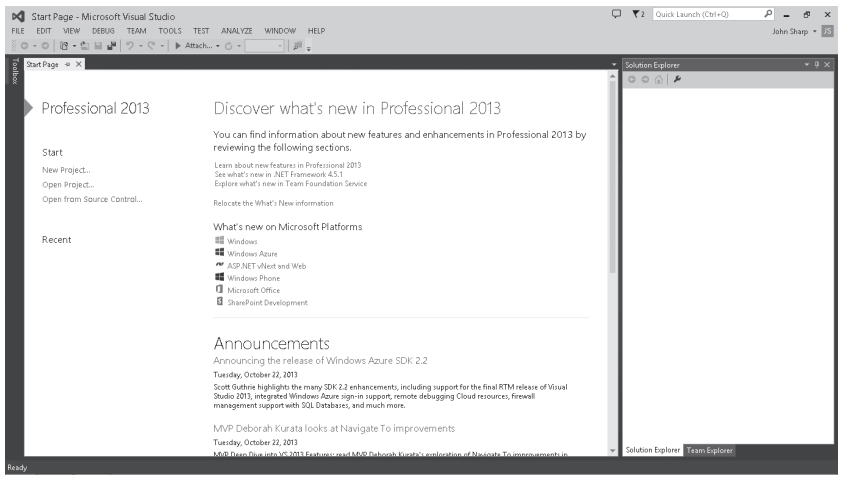
Nota Um aplicativo de console é executado em uma janela de prompt de comando, em vez de fornecer uma interface gráfica com o usuário (GUI).

Crie um aplicativo de console no Visual Studio 2013

- Se você estiver utilizando Windows 8.1 ou Windows 8, na tela Iniciar, digite **Visual Studio** e, no painel Resultados da Pesquisa, clique em Visual Studio 2013.

Nota No Windows 8 e no Windows 8.1, para encontrar um aplicativo, você pode digitar o nome dele literalmente (como Visual Studio) em qualquer parte em branco da tela Iniciar, longe de quaisquer tiles. O painel Resultados da Pesquisa aparecerá automaticamente.

O Visual Studio 2013 é iniciado e apresenta a Página Iniciar (Start page), semelhante à seguinte (sua Página Iniciar poderá ser diferente, dependendo da edição de Visual Studio 2013 que estiver usando).



Nota Se você estiver usando o Visual Studio 2013 pela primeira vez, talvez apareça uma caixa de diálogo solicitando a escolha das configurações de ambiente de desenvolvimento padrão. O Visual Studio 2013 pode ser personalizado de acordo com a sua linguagem de desenvolvimento preferida. As seleções padrão das diversas caixas de diálogo e ferramentas do ambiente de desenvolvimento integrado (Integrated Development Environment – IDE) são definidas para a linguagem que você escolher. Na lista, selecione Visual C# Development Settings e clique no botão Start Visual Studio. Após alguns instantes, o IDE do Visual Studio 2013 aparecerá.

- Se estiver usando o Windows 7, execute as seguintes operações para iniciar o Visual Studio 2013:

  a. Na barra de tarefas do Windows, clique no botão Iniciar, clique em Todos os Programas e, em seguida, clique no grupo de programas Microsoft Visual Studio 2013.

  b. No grupo de programas Microsoft Visual Studio 2013, clique em Visual Studio 2013.

  O Visual Studio 2013 é iniciado e apresenta a Página Iniciar.

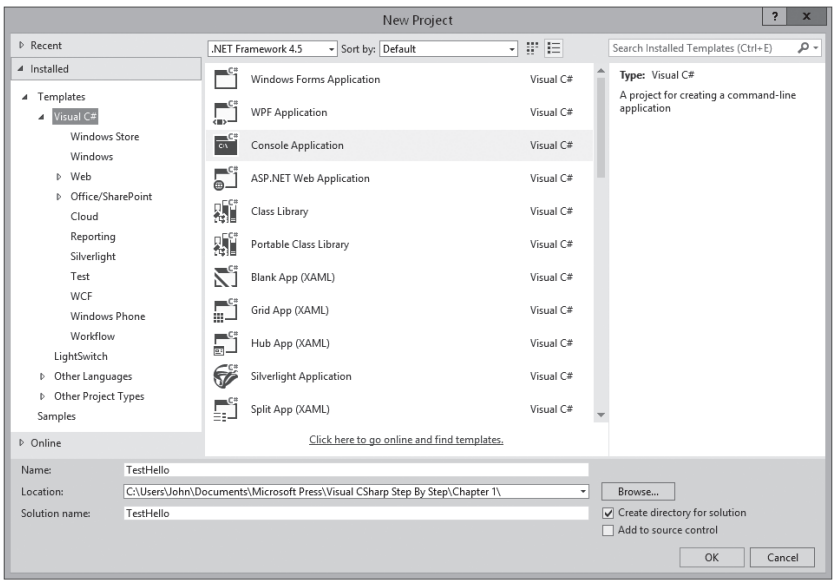
Nota Para não ser repetitivo e economizar espaço, escreverei apenas "Inicie o Visual Studio" quando você precisar abrir o Visual Studio 2013, independentemente do sistema operacional que esteja usando.

■ Siga estes passos para criar um novo aplicativo de console.

a. No menu File, aponte para New e então clique em Project.

A caixa de diálogo New Project se abre. Ela lista os templates que você pode utilizar como ponto de partida para construir um aplicativo. A caixa de diálogo categoriza os templates de acordo com a linguagem de programação que você está utilizando e o tipo de aplicativo.

b. No painel à esquerda, na seção Templates, clique em Visual C#. No painel central, verifique se a caixa de combinação posicionada no início do painel exibe o texto .NET Framework 4.5 e depois clique no ícone Console Application.



c. Na caixa Location, digite **C:\Users*SeuNome*\Documents\Microsoft Press\Visual CSharp Step By Step\Chapter 1**. Substitua o texto *SeuNome* nesse caminho pelo seu nome de usuário do Windows.

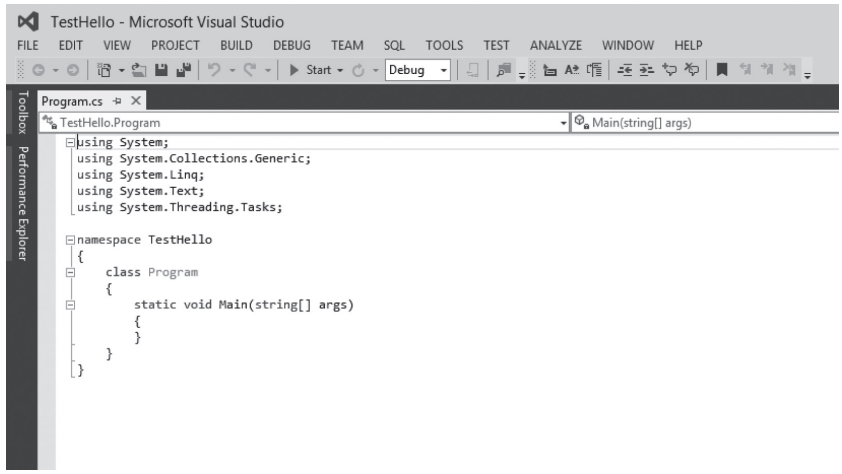
Nota Para não ser repetitivo e economizar espaço, no restante deste livro vou me referir ao caminho C:\Users\SeuNome\Documentos simplesmente como sua pasta Documentos.

Dica Se a pasta especificada não existir, o Visual Studio 2013 criará uma nova para você.

d. Na caixa Name, digite **TestHello** (digite sobre o nome existente, ConsoleApplication1).

e. Certifique-se de que a caixa de seleção Create Directory For Solution está selecionada e clique em OK.

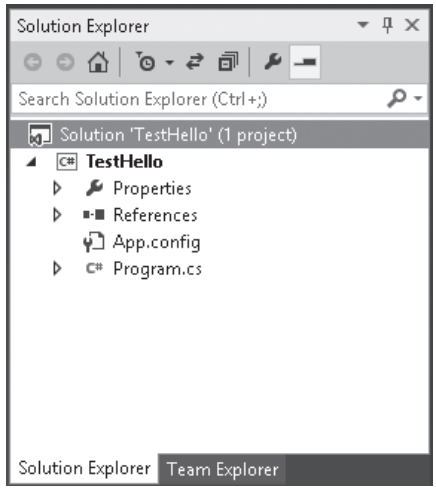
O Visual Studio cria o projeto utilizando o template Console Application e exibe o código básico para o projeto, como na ilustração:



A barra de menus na parte superior da tela fornece acesso aos recursos que você utilizará no ambiente de programação. Você pode usar o teclado ou o mouse para acessar os menus e os comandos, exatamente como faz em todos os programas baseados em Windows. A barra de ferramentas está localizada abaixo da barra de menus. Ela oferece botões de atalho para executar os comandos utilizados com mais frequência.

A janela Code and Text Editor, que ocupa a parte principal da tela, exibe o conteúdo dos arquivos-fonte. Em um projeto com vários arquivos, quando você edita mais de um deles, cada arquivo-fonte tem uma guia própria com seu nome. Você pode clicar na guia para trazer o arquivo-fonte nomeado para o primeiro plano na janela Code and Text Editor.

O painel Solution Explorer aparece no lado direito da caixa de diálogo:



O Solution Explorer exibe os nomes dos arquivos associados ao projeto, entre outros itens. Você pode clicar duas vezes em um nome de arquivo no painel Solution Explorer para trazer esse arquivo-fonte para o primeiro plano na janela do Code and Text Editor.

Antes de escrever o código, examine os arquivos listados no Solution Explorer, criados pelo Visual Studio 2013 como parte do seu projeto:

- **Solution 'TestHello'** É o arquivo de solução de nível superior. Cada aplicativo contém apenas um arquivo de solução. Uma solução pode conter um ou mais projetos; o Visual Studio 2013 cria o arquivo de solução para ajudar a organizar esses projetos. Se utilizar o Windows Explorer para examinar a pasta Documentos\Microsoft Press\Visual CSharp Step By Step\Chapter 1\TestHello, você verá que o nome real desse arquivo é TestHello.sln.
- **TestHello** É o arquivo de projeto do C#. Cada arquivo de projeto faz referência a um ou mais arquivos que contêm o código-fonte e outros artefatos do projeto, como imagens gráficas. Todos os códigos-fonte de um mesmo projeto devem ser escritos na mesma linguagem de programação. No Windows Explorer, esse arquivo se chama TestHello.csproj e está armazenado na pasta \Microsoft Press\Visual CSharp Step By Step\Chapter 1\TestHello\TestHello em sua pasta Documentos.
- **Properties** É uma pasta do projeto TestHello. Se for expandida (clique na seta ao lado de Properties), você verá que ela contém um arquivo chamado AssemblyInfo.cs. Esse é um arquivo especial que você pode utilizar para adicionar atributos a um programa, como o nome do autor, a data em que o programa foi escrito, etc. Você pode especificar atributos adicionais para modificar a maneira como o programa é executado. Explicar como esses atributos são utilizados está além dos objetivos deste livro.
- **References** Essa pasta contém as referências às bibliotecas de código compilado que seu aplicativo pode utilizar. Quando o código C# é compilado, ele é convertido em uma biblioteca e recebe um nome exclusivo. No Microsoft .NET Framework, essas bibliotecas são chamadas *assemblies*. Desenvolvedores utilizam assemblies para empacotar funcionalidade útil que escreveram, podendo distri-

buí-los para outros desenvolvedores que queiram utilizar esses recursos nos seus aplicativos. Se você expandir a pasta References, verá o conjunto de referências padrão adicionado em seu projeto pelo Visual Studio 2013. Esses assemblies dão acesso a muitos dos recursos normalmente utilizados do .NET Framework e são fornecidos pela Microsoft com o Visual Studio 2013. Você vai aprender sobre muitos desses assemblies à medida que avançar nos exercícios do livro.

- **App.config** É o arquivo de configuração de aplicativo. Ele é opcional e poderá não estar presente todas as vezes. É possível especificar, durante a execução, as configurações que seu aplicativo pode utilizar para modificar seu comportamento, como a versão do .NET Framework a ser utilizada para executar o aplicativo. Você vai aprender mais sobre esse arquivo nos capítulos posteriores deste livro.
- **Program.cs** É um arquivo-fonte do C# exibido na janela Code and Text Editor quando o projeto é criado. Você escreverá seu código para o aplicativo de console nesse arquivo. Ele contém um código que o Visual Studio 2013 fornece automaticamente, o qual será examinado a seguir.

Escreva seu primeiro programa

O arquivo Program.cs define uma classe chamada *Program* que contém um método chamado *Main*. Em C#, todo código executável deve ser definido dentro de um método e todos os métodos devem pertencer a uma classe ou a uma estrutura. Você aprenderá mais sobre classes no Capítulo 7, "Criação e gerenciamento de classes e objetos", e sobre estruturas, no Capítulo 9, "Como criar tipos-valor com enumerações e estruturas".

O método *Main* designa o ponto de entrada do programa. Ele deve ser definido como um método estático, da maneira especificada na classe *Program*; caso contrário, o .NET Framework poderá não reconhecê-lo como ponto de partida de seu aplicativo, quando for executado. (Veremos métodos em detalhes no Capítulo 3, "Como escrever métodos e aplicar escopo", e o Capítulo 7 fornece mais informações sobre os métodos estáticos.)

Importante O C# é uma linguagem que diferencia maiúsculas de minúsculas: Você deve escrever *Main* com *M* maiúsculo.

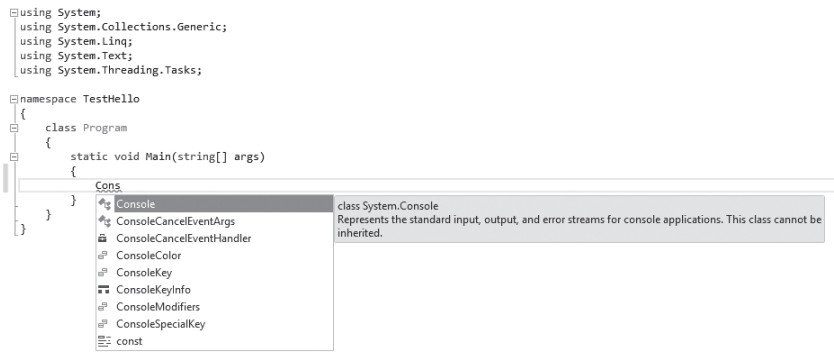
Nos exercícios a seguir, você vai escrever um código para exibir a mensagem "Hello World!" na janela do console, vai compilar e executar seu aplicativo de console Hello World e vai aprender como os namespaces são utilizados para dividir elementos do código.

Escreva o código utilizando o Microsoft IntelliSense

1. Na janela Code and Text Editor que exibe o arquivo Program.cs, coloque o cursor no método *Main* logo após a chave de abertura, {, e pressione Enter para criar uma nova linha.

2. Nessa nova linha, digite a palavra **Console**; esse é o nome de outra classe fornecida pelos assemblies referenciados por seu aplicativo. Ela fornece os métodos para exibir mensagens na janela do console e ler entradas a partir do teclado.

   Ao digitar a letra **C** no início da palavra *Console*, uma lista IntelliSense aparecerá.



   Essa lista contém todas as palavras-chave válidas do C# e os tipos de dados válidos nesse contexto. Você pode continuar digitando ou rolar pela lista e clicar duas vezes no item Console com o mouse. Como alternativa, depois de digitar **Cons**, a lista IntelliSense focalizará automaticamente o item *Console* e você poderá pressionar as teclas Tab ou Enter para selecioná-lo.

   Main deve se parecer com isto:

```
static void Main(string[] args)
{
    Console
}
```

> **Nota** *Console* é uma classe interna.

3. Digite um ponto logo após *Console*.

   Outra lista IntelliSense aparece, exibindo os métodos, propriedades e campos da classe *Console*.

4. Role para baixo pela lista, selecione *WriteLine* e então pressione Enter. Você também pode continuar a digitar os caracteres **W**, **r**, **i**, **t**, **e**, **L**, até *WriteLine* estar selecionado e então pressionar Enter.

   A lista IntelliSense é fechada e a palavra *WriteLine* é adicionada ao arquivo-fonte. *Main* deve se parecer com isto:

```
static void Main(string[] args)
{
    Console.WriteLine
}
```

5. Digite um parêntese de abertura, (. Outra dica do IntelliSense aparece.

   Essa dica exibe os parâmetros que o método *WriteLine* pode receber. De fato, *WriteLine* é um *método sobrecarregado*, ou seja, a classe *Console* contém mais de um método chamado *WriteLine* – na verdade ela fornece 19 versões diferentes desse método. Cada versão do método *WriteLine* pode ser utilizada para emitir diferentes tipos de dados. (O Capítulo 3 descreve métodos sobrecarregados em mais detalhes.) *Main* deve se parecer com isto:

```
static void Main(string[] args)
{
    Console.WriteLine(
}
```

Dica Você pode clicar nas setas para cima e para baixo na dica para rolar pelas diferentes sobrecargas de *WriteLine*.

6. Digite um parêntese de fechamento,), seguido por um ponto e vírgula, ;.

   Main deve se parecer com isto:

```
static void Main(string[] args)
{
    Console.WriteLine();
}
```

7. Mova o cursor e digite a string **"Hello World!"**, incluindo as aspas, entre os parênteses esquerdo e direito depois do método *WriteLine*.

   Main deve se parecer com isto:

```
static void Main(string[] args)
{
    Console.WriteLine("Hello World!");
}
```

Dica Adquira o hábito de digitar pares de caracteres correspondentes, como parênteses (e) e chaves { e }, antes de preencher seus conteúdos. É fácil esquecer o caractere de fechamento se você esperar para digitá-lo depois de inserir o conteúdo.

Ícones IntelliSense

Quando você digita um ponto depois do nome de uma classe, o IntelliSense exibe o nome de cada membro dessa classe. À esquerda de cada nome de membro há um ícone que representa o tipo de membro. Os ícones mais comuns e seus tipos são:

Ícone	Significado
	Método (Capítulo 3)
	Propriedade (Capítulo 15, "Implementação de propriedades para acessar campos")
	Classe (Capítulo 7)
	Estrutura (Capítulo 9)
	Enumeração (Capítulo 9)
	Método de extensão (Capítulo 12)
	Interface (Capítulo 13, "Como criar interfaces e definir classes abstratas")
	Delegado (Capítulo 17, "Genéricos")
	Evento (Capítulo 17)
	Namespace (próxima seção deste capítulo)

Outros ícones IntelliSense aparecerão à medida que você digitar o código em contextos diferentes.

Muitas vezes, você verá linhas de código contendo duas barras (//) seguidas por um texto comum. Esses são comentários ignorados pelo compilador, mas muito úteis para os desenvolvedores, porque ajudam a documentar o que um programa está fazendo. Considere o seguinte exemplo:

```
Console.ReadLine(); // Espera o usuário pressionar a tecla Enter
```

O compilador pula todo o texto desde as duas barras até o fim da linha. Você também pode adicionar comentários de várias linhas, que iniciam com uma barra normal seguida por um asterisco (/*). O compilador pula tudo até localizar um asterisco seguido por barra normal (*/), que pode estar várias linhas abaixo. É um estímulo para documentar seu código com o maior número possível de comentários significativos.

Compile e execute o aplicativo de console

1. No menu Build, clique em Build Solution.

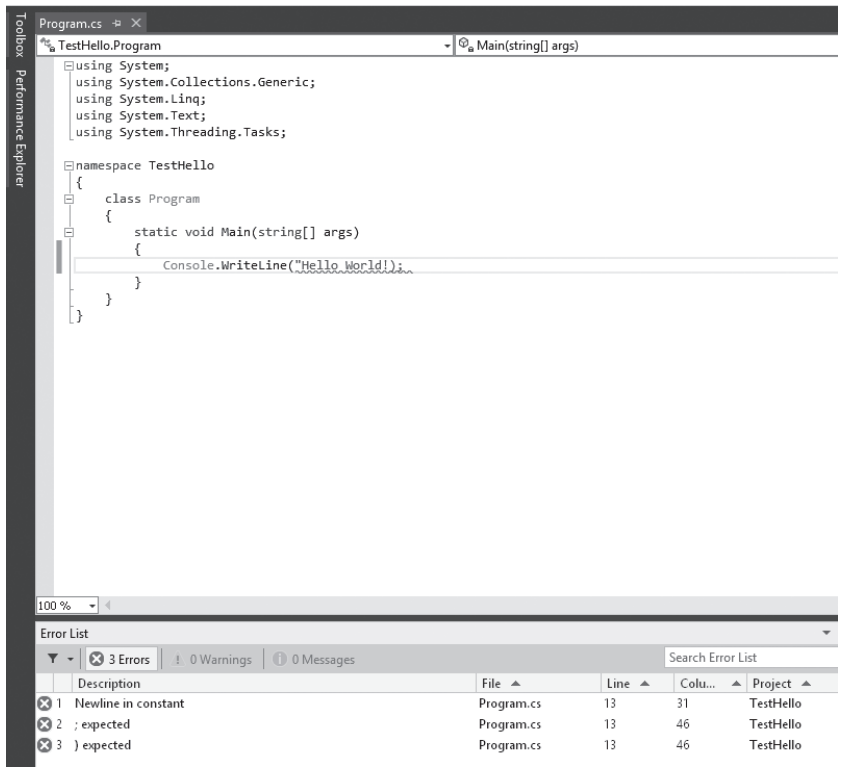
Essa ação compila o código C#, resultando em um programa que pode ser executado. A janela Output aparece abaixo da janela Code and Text Editor.

Dica Se a janela Output não aparecer no menu View, clique em Output para exibi-la.

Nessa janela, você deve ver mensagens semelhantes às seguintes, indicando como o programa está sendo compilado.

```
1>------ Build started: Project: TestHello, Configuration: Debug Any CPU ------
1>  TestHello -> C:\Users\John\Documents\Microsoft Press\Visual CSharp Step By
Step\Chapter
1\TestHello\TestHello\bin\Debug\TestHello.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Qualquer erro que você cometer aparecerá na janela Error List. A imagem a seguir mostra o que acontece se você esquecer de digitar as aspas de fechamento depois do texto *Hello World* na instrução *WriteLine*. Observe que um único erro às vezes pode causar vários erros de compilador.



Dica Para ir diretamente à linha que causou o erro, clique duas vezes em um item na janela Error List. Observe também que o Visual Studio exibe uma linha vermelha ondulada sob qualquer linha de código que não será compilada quando você a inserir.

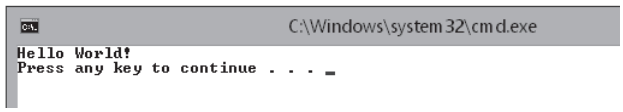
Se você seguiu as instruções anteriores cuidadosamente, não haverá erro ou aviso algum, e o programa deverá ser compilado com sucesso.

Dica Não há necessidade de salvar o arquivo explicitamente antes de compilá-lo, porque o comando Build Solution o salva automaticamente.

Um asterisco após o nome do arquivo na guia acima da janela Code and Text Editor indica que o arquivo foi alterado após ter sido salvo pela última vez.

2. No menu Debug, clique em Start Without Debugging.

Uma janela de comandos é aberta e o programa é executado. A mensagem "Hello World!" é exibida; o programa espera o usuário pressionar uma tecla, como mostra a ilustração a seguir:

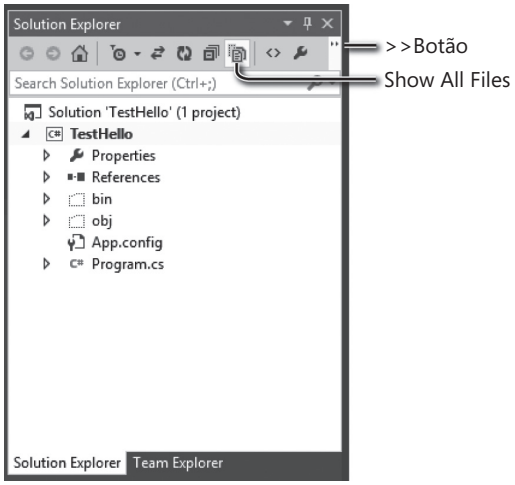


Nota O prompt "Press any key to continue" é gerado pelo Visual Studio sem que você tenha escrito código para fazer isso. Se executar o programa utilizando o comando Start Debugging no menu Debug, o aplicativo será executado, mas a janela de comando fechará imediatamente sem esperar que você pressione uma tecla.

3. Verifique se a janela de comandos que exibe a saída do programa tem o foco (significando que é a janela correntemente ativa) e, em seguida, pressione Enter.

A janela de comandos é fechada e você retorna ao ambiente de programação do Visual Studio 2013.

4. No Solution Explorer, clique no projeto *TestHello* (não na solução) e depois, na barra de ferramentas do Solution Explorer, clique no botão Show All Files. Observe que, para fazer esse botão aparecer, talvez seja necessário clicar no botão na margem direita da barra de ferramentas Solution Explorer.



Entradas chamadas *bin* e *obj* aparecem acima do arquivo Program.cs. Essas entradas correspondem diretamente às pastas chamadas bin e obj na pasta do projeto (Microsoft Press\ VisualCSharp Step By Step\Chapter 1\TestHello\TestHello). O Visual Studio as cria quando você compila seu aplicativo; elas contêm a versão executável do programa e alguns outros arquivos utilizados para compilar e depurar o aplicativo.

5. No Solution Explorer, expanda a entrada *bin*.

Outra pasta chamada Debug é exibida.

Nota Você também poderá ver uma pasta chamada Release.

6. No Solution Explorer, expanda a pasta Debug.

Aparecem diversos outros itens, incluindo um arquivo chamado *TestHello.exe*. Esse é o programa compilado, o qual é o arquivo executado quando você clica em Start Without Debugging no menu Debug. Os outros dois arquivos contêm informações que são utilizadas pelo Visual Studio 2013, se você executar o programa no modo de depuração (quando você clica em Start Debugging no menu Debug).

Namespaces

O exemplo que vimos até aqui é o de um programa muito pequeno. Mas programas pequenos podem crescer bastante. À medida que o programa se desenvolve, duas questões surgem. Primeiro, é mais difícil entender e manter programas grandes do que programas menores. Segundo, mais código normalmente significa mais classes,

com mais métodos, exigindo o acompanhamento de mais nomes. Conforme o número de nomes aumenta, também aumenta a probabilidade de a compilação do projeto falhar porque dois ou mais nomes entram em conflito; por exemplo, você poderia criar duas classes com o mesmo nome. A situação se torna mais complicada quando um programa faz referência a assemblies escritos por outros desenvolvedores que também utilizaram uma variedade de nomes.

Antigamente, os programadores tentavam resolver o conflito prefixando os nomes com algum tipo de qualificador (ou conjunto de qualificadores). Essa não é uma boa solução, pois não é expansível; os nomes tornam-se maiores, e você gasta menos tempo escrevendo o software e mais tempo digitando (há uma diferença), e lendo e relendo nomes longos e incompreensíveis.

Os namespaces ajudam a resolver esse problema criando um contêiner para itens, como classes. Duas classes com o mesmo nome não serão confundidas se elas estiverem em namespaces diferentes. Você pode criar uma classe chamada *Greeting* em um namespace chamado *TestHello*, utilizando a palavra-chave *namespace*, como mostrado a seguir:

```
namespace TestHello
{
   class Greeting
   {
     ...
   }
}
```

Você pode então referenciar a classe *Greeting* como *TestHello.Greeting* em seus programas. Se outro desenvolvedor também criar uma classe *Greeting* em um namespace diferente, como *NewNamespace*, e você instalar o assembly que contém essa classe no seu computador, seus programas ainda funcionarão conforme o esperado, pois usarão a classe *TestHello.Greeting*. Se quiser referenciar a classe *Greeting* do outro desenvolvedor, você deverá especificá-la como *NewNamespace.Greeting*.

É uma boa prática definir todas as suas classes em namespaces, e o ambiente do Visual Studio 2013 segue essa recomendação utilizando o nome do seu projeto como o namespace de nível mais alto. A biblioteca de classes do .NET Framework também segue essa recomendação: toda classe no .NET Framework está situada em um namespace. Por exemplo, a classe *Console* reside no namespace *System*. Isso significa que seu nome completo é, na verdade, *System.Console*.

Porém, se você tivesse que escrever o nome completo de uma classe sempre que ela fosse utilizada, seria melhor prefixar qualificadores ou então atribuir à classe um nome globalmente único, como *SystemConsole*. Felizmente, é possível resolver esse problema com uma diretiva *using* nos seus programas. Se você retornar ao programa *TestHello* no Visual Studio 2013 e examinar o arquivo *Program.cs* na janela Code and Text Editor, notará as seguintes linhas no início do arquivo:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

Essas linhas são diretivas *using*. Uma diretiva *using* adiciona um namespace ao escopo. No código subsequente, no mesmo arquivo, você não precisa mais qualificar explicitamente os objetos com o namespace ao qual eles pertencem. Os cinco namespaces mostrados contêm classes utilizadas com tanta frequência que o Visual Studio 2013 adiciona essas instruções *using* automaticamente toda vez que você cria um novo projeto. Você pode adicionar outras diretivas *using* no início de um arquivo-fonte, caso precise referenciar outros namespaces.

O exercício a seguir demonstra o conceito dos namespaces com mais detalhes.

Experimente os nomes longos

1. Na janela Code and Text Editor que exibe o arquivo *Program.cs*, transforme em comentário a primeira diretiva *using* na parte superior do arquivo, desta maneira:

   ```
   //using System;
   ```

2. No menu Build, clique em Build Solution.

   A compilação falha e a janela Error List exibe a seguinte mensagem de erro:

   ```
   The name 'Console' does not exist in the current context.
   ```

3. Na janela Error List, clique duas vezes na mensagem de erro.

   O identificador que causou o erro é destacado no arquivo-fonte *Program.cs*.

4. Na janela Code and Text Editor, edite o método *Main* para utilizar o nome completo *System.Console*.

   Main deve se parecer com isto:

   ```
   static void Main(string[] args)
   {
       System.Console.WriteLine("Hello World!");
   }
   ```

Nota Quando você digita o ponto final após *System*, os nomes de todos os itens no namespace *System* são exibidos pelo IntelliSense.

5. No menu Build, clique em Build Solution.

   A compilação do projeto deve ser bem-sucedida desta vez. Se não for, certifique-se de que o código *Main* está exatamente como aparece no código precedente e, em seguida, tente recompilar outra vez.

6. Execute o aplicativo para verificar se ele ainda funciona, clicando em Start Without Debugging no menu Debug.

7. Depois que o programa for executado e exibir "Hello World!", na janela do console, pressione Enter para retornar ao Visual Studio 2013.

Namespaces e assemblies

Uma diretiva *using* coloca em escopo os itens de um namespace, e você não precisa qualificar completamente os nomes das classes no seu código. As classes são compiladas em *assemblies*. Um assembly é um arquivo que tem, em geral, a extensão de nome de arquivo .dll, embora programas executáveis com a extensão de nome de arquivo .exe também sejam assemblies.

Um assembly pode conter muitas classes. As classes de biblioteca abrangidas pela biblioteca de classes do .NET Framework, como *System.Console*, são fornecidas nos assemblies instalados no seu computador junto com o Visual Studio. Você descobrirá que a biblioteca de classes do .NET Framework contém milhares de classes. Se todas fossem armazenadas nos mesmos assemblies, estes seriam enormes e difíceis de manter. (Se a Microsoft atualizasse um único método em uma única classe, ela teria de distribuir toda a biblioteca de classes a todos os desenvolvedores!)

Por essa razão, a biblioteca de classes do .NET Framework é dividida em alguns assemblies, agrupados de acordo com a área funcional a que as classes estão relacionadas. Por exemplo, um assembly "básico" (na verdade, chamado *mscorlib.dll*) contém todas as classes comuns, como *System.Console*, e outros assemblies contêm classes para manipular bancos de dados, acessar web services, compilar GUIs e assim por diante. Se quiser utilizar uma classe em um assembly, você deve adicionar ao seu projeto uma referência a ele. Então, pode adicionar instruções *using* ao seu código, colocando em escopo os itens do namespace nesse assembly.

Observe que não há necessariamente uma equivalência 1:1 entre um assembly e um namespace. Um único assembly pode conter classes definidas para muitos namespaces e um único namespace pode abranger vários assemblies. Por exemplo, as classes e itens do namespace *System* são, na verdade, implementados por vários assemblies, incluindo *mscorlib.dll*, *System.dll* e *System.Core.dll*, dentre outros. Isso parece muito confuso agora, mas você logo irá se acostumar.

Ao utilizar o Visual Studio para criar um aplicativo, o template que você seleciona inclui automaticamente referências aos assemblies adequados. Por exemplo, no Solution Explorer do projeto TestHello, expanda a pasta References. Você verá que um aplicativo de console contém automaticamente referências a assemblies chamados *Microsoft.CSharp*, *System*, *System.Core*, *System.Data*, *System.Data.DataExtensions*, *System.Xml* e *System.Xml.Linq*. Talvez você fique surpreso ao ver que *mscorlib.dll* não está nessa lista. Isso acontece porque todos os aplicativos do .NET Framework devem usar esse assembly, pois ele contém a funcionalidade de tempo de execução fundamental. A pasta References lista somente os assemblies opcionais; é possível adicionar ou remover assemblies dessa pasta, conforme for necessário.

Para acrescentar referências para assemblies adicionais em um projeto, clique com o botão direito do mouse na pasta References e então, no menu de atalho que aparece, clique em Add Reference – você fará isso nos próximos exercícios. Você também pode remover um assembly, clicando nele com o botão direito do mouse na pasta References e, então, clicando em Remove.

Crie um aplicativo gráfico

Até aqui, você usou o Visual Studio 2013 para criar e executar um aplicativo de console básico. O ambiente de programação do Visual Studio 2013 também contém tudo que você precisa para criar aplicativos gráficos para Windows 7, Windows 8 e Windows 8.1. Você pode projetar a interface de usuário (IU) de um aplicativo para Windows de modo interativo. O Visual Studio 2013 então gera as instruções do programa para implementar a interface de usuário que você projetou.

O Visual Studio 2013 fornece duas visualizações de um aplicativo gráfico: a visualização de projeto (*design view*) e a visualização de código (*code view*). Utilize a janela Code and Text Editor para modificar e manter o código e a lógica do programa para um aplicativo gráfico, e a janela Design View para organizar sua interface do usuário. Você pode alternar entre as duas visualizações sempre que quiser.

Nos exercícios a seguir, você aprenderá a criar um aplicativo gráfico utilizando o Visual Studio 2013. Esse programa exibe um formulário simples, contendo uma caixa de texto em que você pode inserir seu nome e um botão que, quando clicado, exibe uma saudação personalizada.

Importante No Windows 7 e no Windows 8, O Visual Studio 2013 fornece dois templates para compilar aplicativos gráficos: o template Windows Forms Application e o template WPF Application. Windows Forms é uma tecnologia que surgiu no .NET Framework versão 1.0. O WPF, ou Windows Presentation Foundation, é uma tecnologia aprimorada que apareceu na versão 3.0 do .NET Framework. O WPF oferece muitos recursos adicionais em relação ao Windows Forms, e você deve considerar o seu uso no lugar do Windows Forms para todos os novos desenvolvimentos para Windows 7.

Também é possível compilar aplicativos Windows Forms e WPF no Windows 8.1. Contudo, o Windows 8 e o Windows 8.1 oferecem um novo tipo de interface do usuário, denominado estilo "Windows Store". Os aplicativos que utilizam esse estilo de interface são chamados aplicativos Windows Store. O Windows 8 foi projetado para funcionar em uma variedade de hardware, incluindo computadores com telas sensíveis ao toque e tablets ou slates. Esses computadores permitem aos usuários interagir com os aplicativos por meio de gestos baseados em toques — por exemplo, os usuários podem passar o dedo nos aplicativos para movê-los na tela e girá-los ou "apertar" e "alongar" aplicativos para diminuí-los e ampliá-los novamente. Além disso, muitos tablets contêm sensores que detectam a orientação do dispositivo, e o Windows 8 pode passar essa informação para um aplicativo, o qual pode então ajustar a interface do usuário dinamicamente, de acordo com a orientação (pode trocar do modo paisagem para retrato, por exemplo). Se você tiver instalado o Visual Studio 2013 em um computador Windows 8.1, receberá um conjunto adicional de templates para compilar aplicativos Windows Store. *Contudo, esses templates dependem dos recursos fornecidos pelo Windows 8.1; portanto, se você estiver usando o Windows 8, os templates do Windows Store não estarão disponíveis*.

Para satisfazer os desenvolvedores de Windows 7, de Windows 8 e de Windows 8.1, em muitos dos exercícios, forneci instruções para uso dos templates WPF. Se estiver usando o Windows 7 ou o Windows 8, você deverá seguir as instruções do Windows 7. Se quiser usar o estilo de interface de usuário Windows Store, você deve seguir as instruções do Windows 8.1. Evidentemente, você pode seguir as instruções para Windows 7 e para Windows 8 para usar os templates WPF no Windows 8.1, se preferir.

Caso queira mais informações sobre os pormenores de como escrever aplicativos para Windows 8.1, os capítulos finais da Parte IV deste livro fornecem mais detalhes e orientações.

Crie um aplicativo gráfico no Visual Studio 2013

- Se estiver usando o Windows 8.1, execute as seguintes operações para criar um novo aplicativo gráfico:

  a. Inicie o Visual Studio 2013, se ele ainda não estiver em execução.

  b. No menu File, aponte para New e então clique em Project.

  A caixa de diálogo New Project se abre.

  c. No painel da esquerda, na seção Installed Templates, expanda Visual C# (se ainda não estiver expandido) e então clique na pasta Windows Store.

  d. No painel central, clique no ícone Blank App (XAML).

Nota XAML significa Extensible Application Markup Language, que é a linguagem utilizada por aplicativos Windows Store para definir o layout de sua GUI. Você vai aprender mais sobre XAML à medida que avançar nos exercícios do livro.

  e. Certifique-se de que o campo Location refere-se à pasta \Microsoft Press\Visual CSharp Step By Step\Chapter 1, na pasta Documentos.

  f. Na caixa Name, digite **Hello**.

  g. Na caixa Solution, assegure-se de que Create New Solution está selecionado.

  Essa ação cria uma nova solução para armazenar o projeto. A alternativa Add To Solution adiciona o projeto à solução TestHello, mas não é isso que você quer para este exercício.

  h. Clique em OK.

  Se essa for a primeira vez que você criou um aplicativo Windows Store, será solicitado a apresentar uma licença de desenvolvedor. Você deve concordar com os termos e condições indicados na caixa de diálogo,

antes de continuar a compilar aplicativos Windows Store. Se estiver de acordo com essas condições, clique em I Agree, como mostrado na ilustração a seguir. Será solicitado que você entre no Windows Live (nesse ponto, é possível criar uma nova conta, se necessário) e uma licença de desenvolvedor será criada e reservada para você.

i. Após a criação do aplicativo, examine a janela Solution Explorer.

Não se engane com o nome do template de aplicativo — embora seja chamado Blank App, na verdade esse template fornece vários arquivos e contém algum código. Por exemplo, se você expandir a pasta MainPage.xaml, encontrará um arquivo C# chamado MainPage.xaml.cs. Esse arquivo é onde você insere o código executado quando a interface do usuário definida pelo arquivo MainPage.xaml é exibida.

j. No Solution Explorer, clique duas vezes em MainPage.xaml.

Esse arquivo contém o layout da interface do usuário. A janela Design View mostra duas representações desse arquivo:

Na parte superior está uma visualização gráfica representando a tela de um computador tablet. O painel inferior contém uma descrição do conteúdo dessa tela em XAML. XAML é uma linguagem tipo XML utilizada por aplicativos Windows Store e WPF para definir o layout de um formulário e seu conteúdo. Se você conhece XML, a XAML deverá lhe parecer familiar.

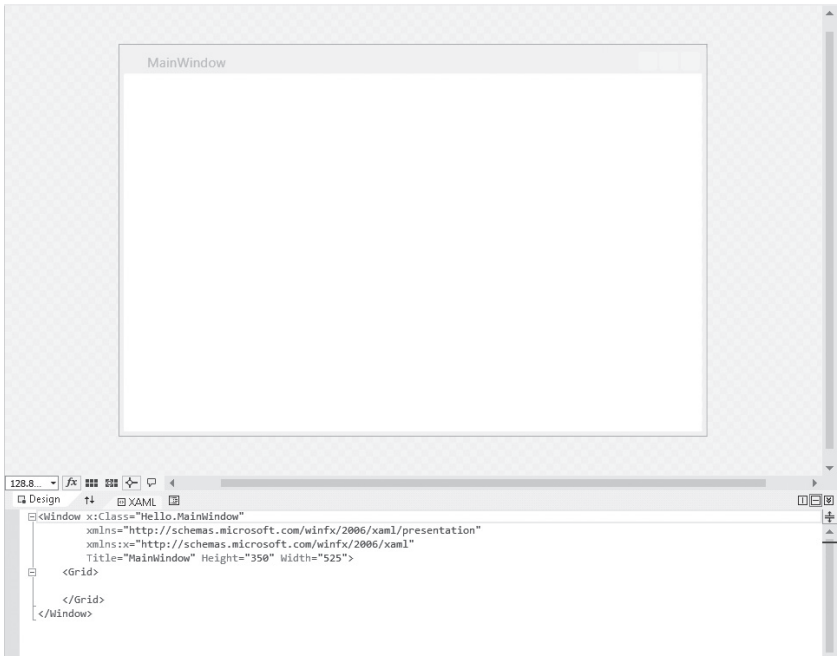
No próximo exercício, você vai usar a janela Design View para organizar a interface do usuário do aplicativo e vai examinar o código XAML gerado por esse layout.

- Se estiver usando o Windows 8 ou o Windows 7, execute as seguintes tarefas:
  - **a.** Inicie o Visual Studio 2013, se ele ainda não estiver em execução.
  - **b.** No menu File, aponte para New e então clique em Project.

    A caixa de diálogo New Project se abre.
  - **c.** No painel da esquerda, na seção Installed Templates, expanda Visual C# (se ainda não estiver expandido) e então clique na pasta Windows.
  - **d.** No painel central, clique no ícone WPF Application.
  - **e.** Certifique-se de que a caixa Location refere-se à pasta \Microsoft Press\Visual CSharp Step By Step\Chapter 1, na pasta Documentos.
  - **f.** Na caixa Name, digite **Hello**.
  - **g.** Na caixa Solution, assegure-se de que Create New Solution está selecionado e clique em OK.

    O template WPF Application gera menos itens do que o template Windows Store Blank App; ele não contém os estilos gerados pelo template Blank App, pois a funcionalidade incorporada nesses estilos é específica para o Windows 8.1. Contudo, o template WPF Application gera uma janela padrão para seu aplicativo. Como em um aplicativo Windows Store, essa janela é definida com XAML, mas, neste caso, é chamada MainWindow.xaml por padrão.

h. No Solution Explorer, clique duas vezes em MainWindow.xaml para exibir o conteúdo desse arquivo na janela Design View.



Dica Feche as janelas Output e Error List para dar mais espaço à exibição da janela Design View.

Nota Antes de prosseguirmos, é importante explicarmos alguma terminologia. Em um aplicativo WPF típico, a interface do usuário consiste em uma ou mais *janelas*, mas em um aplicativo Windows Store os itens correspondentes são chamados de *páginas* (rigorosamente falando, um aplicativo WPF também pode conter páginas, mas não quero confundir as coisas neste ponto). Para não ficar repetindo a frase bastante prolixa "janela WPF ou página de aplicativo Windows Store" no livro, vou simplesmente me referir aos dois itens usando o termo geral *formulário*. Entretanto, continuarei usando a palavra *janela* para me referir aos itens do IDE do Visual Studio 2013, como a janela Design View.

Nos próximos exercícios, você vai utilizar a janela Design View para adicionar três controles ao formulário exibido por seu aplicativo e examinar alguns dos códigos C# gerados automaticamente pelo Visual Studio 2013 para implementar esses controles.

Nota Os passos dos próximos exercícios são comuns para o Windows 7, para o Windows 8 e para o Windows 8.1, exceto onde quaisquer diferenças sejam explicitamente indicadas.

Crie a interface do usuário

1. Clique na guia Toolbox exibida à esquerda do formulário na janela Design View.

   A Toolbox aparece, ocultando parcialmente o formulário, e exibe os vários componentes e controles que você pode colocar em um formulário.

2. Se estiver utilizando o Windows 8.1, expanda a seção Common XAML Controls.

   Se estiver utilizando o Windows 7 ou o Windows 8, expanda a seção Common WPF Controls.

   Essa seção exibe uma lista de controles utilizados pela maioria dos aplicativos gráficos.

Dica A seção All XAML Controls (Windows 8.1) ou All WPF Controls (Windows 7 e Windows 8) exibe uma lista mais extensa de controles.

3. Na seção Common XAML Controls ou Common WPF Controls, clique em TextBlock e arraste o controle *TextBlock* para o formulário exibido na janela Design View.

Dica Certifique-se de selecionar o controle *TextBlock* e não o controle *TextBox*. Se acidentalmente você colocar o controle errado em um formulário, pode removê-lo com facilidade, clicando no item no formulário e pressionando Delete.

   Um controle *TextBlock* é adicionado ao formulário (você o moverá para o local correto mais adiante), e a Toolbox é ocultada.

Dica Se quiser que a Toolbox permaneça visível, mas não oculte nenhuma parte do formulário, na extremidade direita da barra de título da Toolbox, clique no botão Auto Hide (ele parece um alfinete). A Toolbox aparece permanentemente no lado esquerdo da janela do Visual Studio 2013 e a janela Design View é reduzida para acomodá-la. (Talvez você perca muito espaço se tiver uma tela com baixa resolução.) Clicar no botão Auto Hide mais uma vez fará a Toolbox desaparecer novamente.

4. É provável que o controle *TextBlock* no formulário não esteja exatamente onde você quer. Você pode clicar e arrastar os controles que adicionou a um formulário para reposicioná-los. Utilizando essa técnica, mova o controle *TextBlock* para posicioná-lo próximo ao canto superior esquerdo do formulário. (O local exato

não é importante para esse aplicativo.) Observe que talvez seja preciso clicar longe do controle e, então, clicar nele novamente, antes que você possa movê-lo na janela Design View.

No painel inferior, a descrição XAML do formulário agora inclui o controle *TextBlock*, junto com propriedades, como sua localização no formulário (controlada pela propriedade *Margin*), o texto padrão exibido por esse controle (na propriedade *Text*), o alinhamento do texto exibido por esse controle (especificado pelas propriedades *HorizontalAlignment* e *VerticalAlignment*) e se o texto deve passar para a próxima linha se ultrapassar a largura do controle *TextWrapping*.

Se você estiver usando Windows 8.1, o código XAML do controle *TextBlock* será parecido com este (seus valores para a propriedade *Margin* poderão ser um pouco diferentes, dependendo de onde você posicionou o controle *TextBlock* no formulário):

```
<TextBlock HorizontalAlignment="Left" Margin="400,200,0,0" TextWrapping="Wrap"
Text="TextBlock" VerticalAlignment="Top"/>
```

Se estiver usando Windows 7 ou Windows 8, o código XAML será praticamente o mesmo, exceto que as unidades utilizadas pela propriedade *Margin* operam em uma escala diferente, devido à resolução maior dos dispositivos Windows 8.1.

O painel XAML e a janela Design View têm uma relação bilateral entre si. Você pode editar os valores no painel XAML e as alterações serão refletidas na janela Design View. Por exemplo, você pode mudar o local do controle *TextBlock* modificando os valores da propriedade *Margin*.

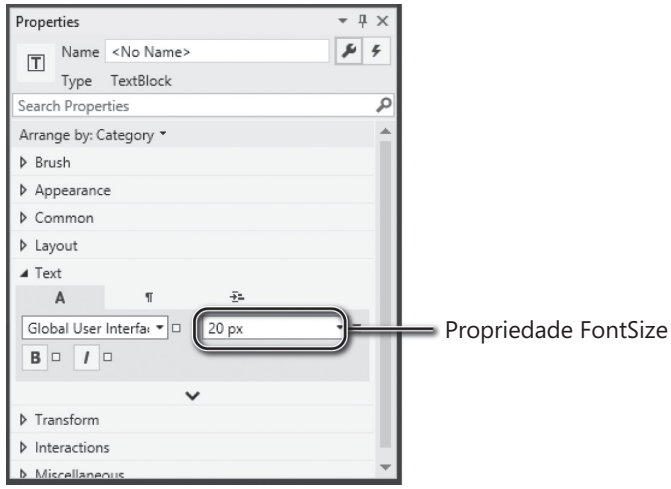
5. No menu View, clique em Properties Window.

   Se já estava aberta, a janela Properties aparece no canto inferior direito da tela, sob o Solution Explorer. É possível especificar as propriedades dos controles usando o painel XAML sob a janela Design View, mas a janela Properties é uma maneira mais prática de modificar as propriedades dos itens em um formulário, assim como outros itens em um projeto.

   A janela Properties é sensível ao contexto, exibindo as propriedades do item selecionado. Se clicar no formulário exibido na janela Design View, fora do controle *TextBlock*, você verá que a janela Properties exibe as propriedades de um elemento *Grid*. Se examinar o painel XAML, você verá que o controle *TextBlock* está contido em um elemento *Grid*. Todos os formulários contêm um elemento *Grid* que controla o layout dos itens exibidos – é possível definir layouts tabulares adicionando linhas e colunas ao elemento *Grid*, por exemplo.

6. Na janela Design View, clique no controle *TextBlock*. A janela Properties exibe novamente as propriedades do controle *TextBlock*.

7. Na janela Properties, expanda a propriedade *Text*. Altere a propriedade *FontSize* para **20 px** e, em seguida, pressione Enter. Essa propriedade está localizada ao lado da lista suspensa que contém o nome da fonte, o qual será diferente para o Windows 8.1 (Global User Interface) e para o Windows 7 ou Windows 8 (Segoe UI):



Nota O sufixo **px** indica que o tamanho da fonte é medido em pixels.

8. No painel XAML, abaixo da janela Design View, examine o texto que define o controle *TextBlock*. Se você fizer uma rolagem até o final da linha, deverá ver o texto FontSize = "20". Todas as alterações feitas na janela Properties constarão automaticamente nas definições do XAML e vice-versa.

Digite sobre o valor da propriedade *FontSize* no painel XAML, alterando-o para **24**. O tamanho da fonte do texto do controle *TextBlock* na janela Design View e na janela Properties muda.

9. Na janela Properties, examine as outras propriedades do controle *TextBlock*. Sinta-se livre para fazer testes, alterando-as para ver seus efeitos.

Observe que, à medida que você altera os valores das propriedades, essas propriedades são adicionadas à definição do controle *TextBlock* no painel XAML. Cada controle adicionado a um formulário tem um conjunto de valores de propriedade padrão e esses valores não aparecem no painel XAML, a não ser que você os altere.

10. Altere o valor da propriedade *Text* do controle *TextBlock*, de TextBlock para **Please enter your name** (Digite seu nome). Isso pode ser feito editando-se o elemento *Text* no painel XAML ou alterando-se o valor na janela Properties (essa propriedade está localizada na seção Common da janela Properties).

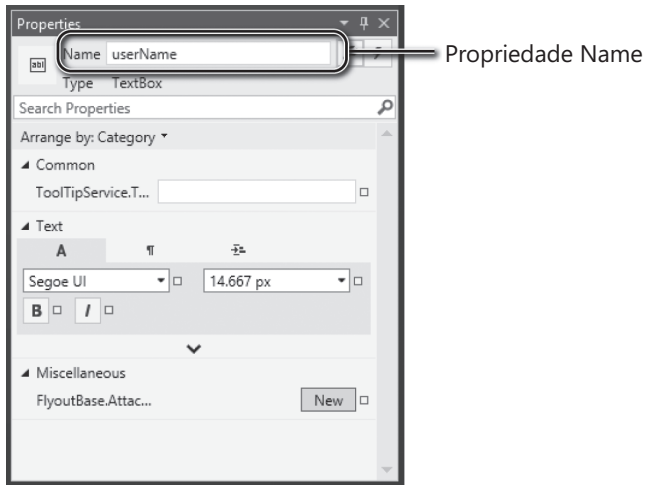
Observe que o texto exibido no controle *TextBlock* na janela Design View muda.

11. Clique no formulário na janela Design View e exiba a Toolbox novamente.

12. Na Toolbox, clique e arraste o controle *TextBox* para o formulário. Mova o controle *TextBox* para posicioná-lo imediatamente abaixo do controle *TextBlock*.

Dica Ao se arrastar um controle em um formulário, indicadores de alinhamento aparecem automaticamente quando o controle torna-se alinhado vertical ou horizontalmente a outros controles. É uma dica visual rápida para você se certificar de que esses controles estão alinhados de modo correto.

13. Na janela Design View, posicione o mouse sobre a borda direita do controle *TextBox*. O cursor do mouse deve mudar para uma seta de duas pontas, indicando que você pode redimensionar o controle. Arraste a borda direita do controle *TextBox* até que ele esteja alinhado com a borda direita do controle *TextBlock* acima; uma guia deverá aparecer quando as duas bordas estiverem alinhadas corretamente.
14. Com o controle *TextBox* ainda selecionado, altere o valor da propriedade *Name* exibida na parte superior da janela Properties, de <No Name> para **userName**, como ilustrado a seguir:



Nota Falaremos mais sobre as convenções de nomes para controles e variáveis no Capítulo 2, "Variáveis, operadores e expressões".

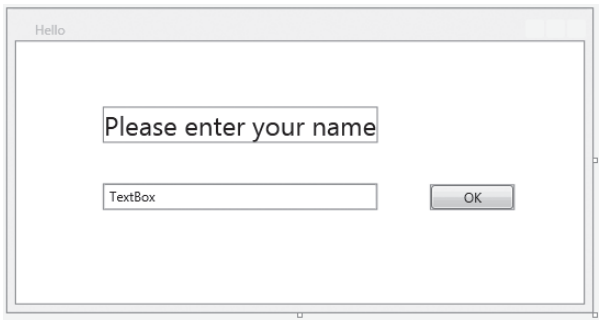
15. Exiba a Toolbox novamente, depois clique e arraste um controle *Button* para o formulário. Posicione o controle *Button* à direita da caixa do controle *TextBox* no formulário, de modo que a parte inferior do botão fique alinhada horizontalmente com a parte inferior da caixa de texto.
16. Na janela Properties, mude a propriedade *Name* do controle *Button* para **ok**, mude a propriedade *Content* (na seção Common) de Button para **OK** e pressione Enter. Verifique que a legenda do controle *Button* no formulário muda para exibir o texto *OK*.
17. Se estiver usando Windows 7 ou Windows 8, clique na barra de título do formulário na janela Design View. Na janela Properties, mude a propriedade *Title* (novamente, na seção Common) de MainWindow para **Hello**.

Nota Os aplicativos Windows Store não têm barra de título.

18. Se estiver usando Windows 7 ou Windows 8, na janela Design View, clique na barra de título do formulário Hello. Observe que uma alça de redimensionamento (um pequeno quadrado) aparece no canto inferior direito do formulário Hello. Mova o cursor do mouse sobre a alça de redimensionamento. Quando o cursor virar uma seta de duas pontas diagonal, arraste-o para redimensionar o formulário. Pare de arrastar e solte o botão do mouse quando o espaçamento em torno dos controles estiver igual.

Importante Clique na barra de título do formulário Hello e não no contorno da grade dentro do formulário, antes de redimensioná-lo. Se selecionar a grade, você modificará o layout dos controles no formulário, mas não o tamanho do formulário.

O formulário Hello deve ficar parecido com a figura a seguir:

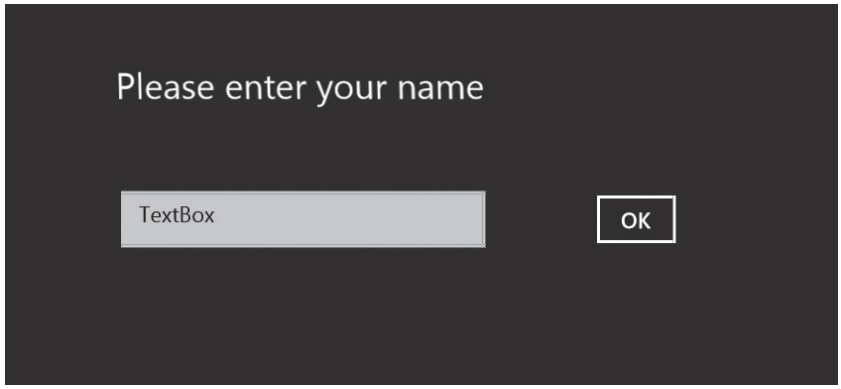


Nota Nos aplicativos Windows Store, as páginas não podem ser redimensionadas da mesma maneira que nos formulários WPF; quando são executados, eles ocupam automaticamente a tela inteira do dispositivo. Contudo, eles podem se adaptar a diferentes resoluções de tela e à orientação do dispositivo, apresentando diferentes visualizações quando são "encaixados". É fácil ver como seu aplicativo aparece em um dispositivo diferente, clicando em Device Window no menu Design e, então, selecionando as diferentes resoluções de tela disponíveis na lista suspensa Display. Também é possível ver como seu aplicativo aparece no modo retrato ou quando está encaixado, selecionando a orientação Portrait ou a visualização Snapped na lista de visualizações disponíveis.

19. No menu Build, clique em Build Solution e verifique se a compilação do projeto foi bem-sucedida.

20. No menu Debug, clique em Start Debugging.

O aplicativo deve ser executado, exibindo seu formulário. Se você está usando Windows 8.1, o formulário ocupa a tela inteira e aparece deste modo:



Nota Quando um aplicativo Windows Store é executado no modo Debug no Windows 8.1, aparecem dois pares de números nos cantos superior esquerdo e superior direito da tela. Esses números controlam a taxa de redesenho (*frame rate*) e os desenvolvedores podem utilizá-los para determinar quando um aplicativo começa a demorar mais do que devia para responder (possivelmente uma indicação de problemas de desempenho). Eles só aparecem quando um aplicativo é executado no modo Debug. Uma descrição completa do significado desses números está fora dos objetivos deste livro; portanto, você pode ignorá-los por enquanto.

Se você está usando Windows 7 ou Windows 8, o formulário aparece deste modo:

Na caixa de texto, você pode digitar sobre o que está lá, digitar seu nome e clicar em OK, mas nada acontecerá ainda. É necessário adicionar algum código para indicar o que deve acontecer quando o usuário clicar no botão OK, o que faremos em seguida.

21. Retorne ao Visual Studio 2013. No menu DEBUG, clique em Stop Debugging.

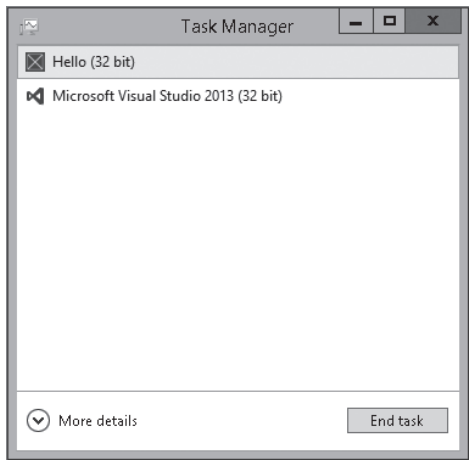
- Se você está usando o Windows 8.1, pressione a tecla Windows+B. Isso deve levá-lo de volta à Área de Trabalho do Windows que está executando o Visual Studio, a partir do qual é possível acessar o menu Debug.

- Se você está usando Windows 7 ou Windows 8, pode trocar diretamente para o Visual Studio. Também é possível clicar no botão de fechamento (o X no canto superior direito do formulário) para fechar o formulário, interromper a depuração e retornar ao Visual Studio.

Como fechar um aplicativo Windows Store

Se você está usando Windows 8.1 e clicou em Start Without Debugging no menu Debug para executar o aplicativo, precisará fechá-lo à força. Isso porque, ao contrário dos aplicativos de console, a vida de um aplicativo Windows Store é gerenciada pelo sistema operacional e não pelo usuário. O Windows 8.1 suspende um aplicativo quando não está sendo exibido e o terminará quando o sistema operacional precisar a liberartação dos recursos que ele consome. O modo mais confiável de interromper o aplicativo Hello à força é clicar (ou colocar o dedo, caso você tenha uma tela sensível ao toque) na parte superior da tela e, então, clicar e arrastar (ou deslizar) o aplicativo para a parte inferior, e segurá-lo até que sua imagem se dobre (se você soltar o aplicativo antes da imagem se dobrar, ele continuará sendo executado em segundo plano). Essa ação fecha o aplicativo e o leva de volta à tela Iniciar do Windows, onde você pode retornar ao Visual Studio. Como alternativa, você pode executar as seguintes tarefas:

1. Clique (ou coloque o dedo) no canto superior direito da tela e, então, arraste a imagem do Visual Studio para o meio da tela (ou pressione a tecla Windows+B).
2. Na parte inferior da área de trabalho, clique com o botão direito do mouse na barra de tarefas do Windows e, então, clique em Iniciar Gerenciador de Tarefas.
3. Na janela Gerenciador de Tarefas do Windows, clique no aplicativo Hello e, em seguida, clique em Finalizar Tarefa.



4. Feche a janela Gerenciador de Tarefas do Windows.

Você conseguiu criar um aplicativo gráfico sem escrever uma única linha de código em C#. Esse aplicativo ainda não faz muito (será necessário escrever algum código), mas o Visual Studio 2013 gera uma grande quantidade de código que trata das tarefas de rotina que todos os aplicativos gráficos devem realizar, como abrir e exibir uma janela. Antes de adicionar seu próprio código ao aplicativo, é importante entender o que Visual Studio produziu. A estrutura é um pouco diferente entre um aplicativo Windows Store e um aplicativo WPF, e as seções a seguir resumem esses estilos de aplicativo separadamente.

Examine o aplicativo Windows Store

Se estiver usando Windows 8.1, no Solution Explorer, clique na seta adjacente ao arquivo MainPage.xaml para expandir o nó. O arquivo MainPage.xaml.cs aparece; clique duas vezes nesse arquivo. O código a seguir, do formulário, é exibido na janela Code and Text Editor.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;

// O template do item Blank Page está documentado em http://go.microsoft.com/
fwlink/?LinkId=234238

namespace Hello
{
  /// <summary>
  /// Uma página vazia que pode ser usada sozinha ou acessada dentro de um Frame.
  /// </summary>
  public sealed partial class MainPage : Page
  {
    public MainPage()
    {
      this.InitializeComponent();
    }
  }
}
```

Além de muitas diretivas *using* que colocam no escopo alguns namespaces que a maioria dos aplicativos Windows Store utiliza, o arquivo contém apenas a definição de uma classe chamada *MainPage*. Há um pouco de código para a classe *MainPage*, conhecido como *construtor*, que chama um método denominado *InitializeComponent*. Um construtor é um método especial com o mesmo nome da classe. Ele é executado quando é criada uma instância da classe e pode conter um código para inicializar a instância. Discutiremos sobre construtores no Capítulo 7.

Na realidade, a classe contém muito mais código do que as poucas linhas mostradas no arquivo MainPage.xaml.cs, mas grande parte dele é gerada automaticamente com base na descrição XAML do formulário, e é ocultada. Esse código oculto realiza operações como criar e exibir o formulário e também criar e posicionar os vários controles no formulário.

Dica Você também pode exibir o arquivo do código C# para uma página em um aplicativo Windows Store, clicando em Code no menu View quando a janela Design View estiver exibida.

Você deve estar se perguntando onde está o método *Main* e como o formulário será exibido quando o aplicativo for executado. Lembre-se de que, em um aplicativo de console, *Main* define o ponto em que o programa inicia. Um aplicativo gráfico é um pouco diferente.

No Solution Explorer deve aparecer outro arquivo-fonte chamado App.xaml. Se expandir o nó desse arquivo, você verá outro arquivo, chamado App.xaml.cs. Em um aplicativo Windows Store, o arquivo App.xaml fornece o ponto de entrada no qual o aplicativo começa a executar. Se você clicar duas vezes em App.xaml.cs no Solution Explorer, verá código semelhante a este:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using Windows.ApplicationModel;
using Windows.ApplicationModel.Activation;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;

// O template do item Blank Application está documentado em http://go.microsoft.com/
fwlink/?LinkId=234227

namespace Hello
{
    /// <summary>
    /// Fornece comportamento específico do aplicativo para complementar a classe Application
padrão.
    /// </summary>
    sealed partial class App : Application
    {
        /// <summary>
        /// Inicializa o objeto aplicativo singleton. Esta é a primeira linha executada
        /// do código escrito e, como tal, é o equivalente lógico de main() ou WinMain().
        /// </summary>
        public App()
        {
            this.InitializeComponent();
            this.Suspending += OnSuspending;
```

```
        }

        /// <summary>
        /// Executado quando o aplicativo é chamado normalmente pelo usuário final. Outros pontos
        /// de entrada serão usados quando o aplicativo for chamado para abrir um arquivo
        /// específico, para exibir resultados de pesquisa e assim por diante.
        /// </summary>
        /// <param name="args">Details about the launch request and process.</param>
        protected override void OnLaunched(LaunchActivatedEventArgs e)
        {

#if DEBUG
            if (System.Diagnostics.Debugger.IsAttached)
            {
                this.DebugSettings.EnableFrameRateCounter = true;
            }
#endif

            Frame rootFrame = Window.Current.Content as Frame;

            // Não repete a inicialização do aplicativo quando a janela já tem conteúdo,
            // apenas garante que ela esteja ativa
            if (rootFrame == null)
            {
                // Cria um Frame para atuar como contexto de navegação e navega para a
primeira página
                rootFrame = new Frame();

                if (e.PreviousExecutionState == ApplicationExecutionState.Terminated)
                {
                    //TODO: carregar estado do aplicativo suspenso anteriormente
                }

                // Coloca o frame na janela atual
                Window.Current.Content = rootFrame;
            }

            if (rootFrame.Content == null)
            {
                // Quando a pilha de navegação não é restaurada, navega para a primeira
                // página, configurando a nova página passando as informações exigidas
                // como parâmetro de navegação
                if (!rootFrame.Navigate(typeof(MainPage), e.Arguments))
                {
                    throw new Exception("Failed to create initial page");
                }
            }
            // Garante que a janela atual esteja ativa
            Window.Current.Activate();
        }

        /// <summary>
        /// Chamado quando a execução do aplicativo está sendo suspensa. O estado do aplicativo
        /// é salvo sem saber se ele será terminado ou retomado com o conteúdo
        /// da memória ainda intacto.
```

```
        /// </summary>
        /// <param name="sender">The source of the suspend request.</param>
        /// <param name="e">Details about the suspend request.</param>
        private void OnSuspending(object sender, SuspendingEventArgs e)
        {
            var deferral = e.SuspendingOperation.GetDeferral();
            //TODO: salvar o estado do aplicativo e interromper qualquer atividade de segundo
plano
            deferral.Complete();
        }
    }
}
```

Grande parte desse código consiste em comentários (as linhas que começam com "///") e outras instruções que você ainda não precisa entender, mas os principais elementos estão localizados no método *OnLaunched*, realçado em negrito. Esse método é executado quando o aplicativo começa e o código presente nele faz com que o aplicativo crie um novo objeto *Frame*, exiba o formulário MainPage nesse quadro (frame) e, então, o ative. Neste estágio, não é necessário compreender completamente o funcionamento desse código ou a sintaxe de qualquer uma dessas instruções, mas é útil reconhecer que é assim que o aplicativo exibe o formulário, quando começa a ser executado.

Examine o aplicativo WPF

Se estiver usando o Windows 7 ou o Windows 8, no Solution Explorer, clique na seta adjacente ao arquivo MainWindow.xaml para expandir o nó. O arquivo MainWindow.xaml.cs aparece; clique duas vezes nesse arquivo. O código do formulário aparece na janela Code and Text Editor, como mostrado aqui:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace Hello
{
   /// <summary>
   /// Lógica de interação para MainWindow.xaml
   /// </summary>
   public partial class MainWindow : Window
   {
      public MainWindow()
      {
          InitializeComponent();
      }
   }
}
```

Esse código parece semelhante ao do aplicativo Windows Store, mas existem algumas diferenças significativas – muitos dos namespaces referenciados pelas diretivas *using* no início do arquivo são diferentes. Por exemplo, os aplicativos WPF utilizam objetos definidos em namespaces que começam com o prefixo *System.Windows*, enquanto os aplicativos Windows Store utilizam objetos definidos em namespaces que começam com *Windows.UI*. Essa diferença não é superficial. Esses namespaces são implementados por diferentes assemblies e os controles e a funcionalidade oferecidos por eles são diferentes entre os aplicativos WPF e Windows Store, embora possam ter nomes semelhantes. Voltando ao exercício anterior, você adicionou controles *TextBlock*, *TextBox* e *Button* ao formulário WPF e ao aplicativo Windows Store. Embora esses controles tenham o mesmo nome em cada estilo de aplicativo, eles são definidos em diferentes assemblies: *Windows.UI.Xaml.Controls* para aplicativos Windows Store e *System.Windows.Controls* para aplicativos WPF. Os controles de aplicativos Windows Store foram especificamente projetados e otimizados para interfaces de toque, enquanto os controles WPF são destinados, em especial, para uso em sistemas voltados para o mouse.

Assim como no código do aplicativo Windows Store, o construtor da classe *MainWindow* inicializa o formulário WPF chamando o método *InitializeComponent*. Novamente, como antes, o código desse método fica oculto e realiza operações como criar e exibir o formulário e também criar e posicionar os vários controles no formulário.

O modo pelo qual um aplicativo WPF especifica o formulário inicial a ser exibido é diferente de um aplicativo Windows Store. Assim como um aplicativo Windows Store, ele estipula um objeto *App* definido no arquivo App.xaml para fornecer o ponto de entrada para o aplicativo, mas o formulário a ser exibido é especificado de forma declarada como parte do código XAML, em vez de em forma de programa. Se você clicar duas vezes no arquivo App.xaml no Solution Explorer (não em App.xaml.cs), poderá examinar a descrição XAML. Há uma propriedade *StartupUri* no código XAML que se refere ao arquivo MainWindow.xaml, como mostrado em negrito no exemplo de código a seguir:

```
<Application x:Class="Hello.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com.winfx/2006/xaml"
             StartupUri="MainWindow.xaml">
    <Application.Resources>

    </Application.Resources>
</Application>
```

Em um aplicativo WPF, a propriedade *StartupUri* do objeto *App* indica o formulário a ser exibido.

Adicione código ao aplicativo gráfico

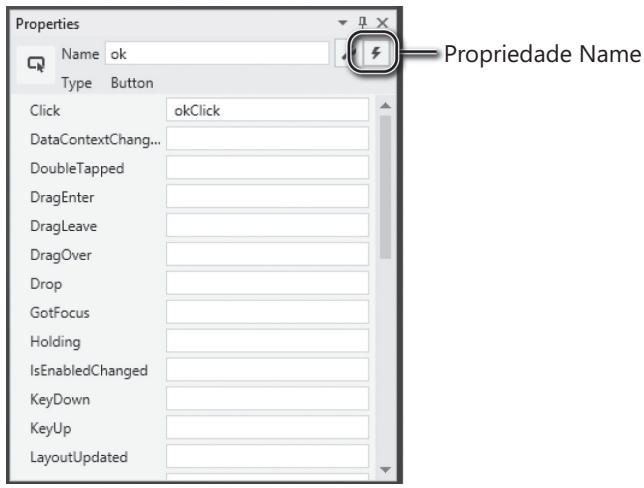
Agora que você conhece um pouco da estrutura de um aplicativo gráfico, chegou a hora de escrever código para que seu aplicativo realmente faça alguma coisa.

Escreva o código para o botão OK

1. Na janela Design View, abra o arquivo MainPage.xaml (Windows 8.1) ou o arquivo MainWindow.xaml (Windows 7 ou Windows 8) – para isso, clique duas vezes em MainPage.xaml ou em MainWindow.xaml no Solution Explorer.

2. Ainda na janela Design View, clique no botão OK do formulário para selecioná-lo.
3. Na janela Properties, clique no botão Event Handlers for the Selected Element.

   Esse botão exibe um ícone parecido com um relâmpago, como demonstrado aqui:



   A janela Properties exibe uma lista de nomes de evento para o controle *Button*. Um evento indica uma ação significativa que normalmente exige uma resposta, e você pode escrever seu código para executar essa resposta.

4. Na caixa adjacente ao evento *Click*, digite **okClick** e, em seguida, pressione Enter.

   O arquivo MainPage.xaml.cs (Windows 8.1) ou MainWindow.xaml.cs (Windows 7 ou Windows 8) aparece na janela Code and Text Editor e um novo método chamado *okClick* é adicionado à classe *MainPage* ou *MainWindow*. O método é semelhante a este:

```
private void okClick(object sender, RoutedEventArgs e)
{

}
```

   Não se preocupe com a sintaxe desse código ainda – você aprenderá tudo sobre métodos no Capítulo 3.

5. Se estiver usando o Windows 8.1, execute as seguintes tarefas:

   a. Adicione a seguinte diretiva *using*, mostrada em negrito, à lista do início do arquivo (o caractere de reticências [...] indica instruções que foram omitidas por brevidade):

```
using System;
...
using Windows.UI.Xaml.Navigation;
using Windows.UI.Popups;
```

   b. Adicione o seguinte código mostrado em negrito ao método *okClick*:

```
void okClick(object sender, RoutedEventArgs e)
{
  MessageDialog msg = new MessageDialog("Hello " + userName.Text);
  msg.ShowAsync();
}
```

Quando compilado, este código irá exibir em "warning" a respeito do uso de um método assíncrono. Não se preocupe com a mensagem; os metodos assíncronos serão explicados no Capítulo 24.

Esse código será executado quando o usuário clicar no botão OK. Novamente, não se preocupe com a sintaxe. Apenas certifique-se de copiar o código exatamente como mostrado; você vai descobrir o que essas instruções significam nos próximos capítulos. O mais importante a entender é que a primeira instrução cria um objeto *MessageDialog* com a mensagem "Hello <*SeuNome*>", onde <*SeuNome*> é o nome que você digita no controle *TextBox* do formulário. A segunda instrução exibe o objeto *MessageDialog*, fazendo-o aparecer na tela. A classe *MessageDialog* é definida no namespace *Windows.UI.Popups* e esse é o motivo pelo qual você o adicionou no passo a.

6. Se estiver usando Windows 7 ou Windows 8, basta adicionar ao método *okClick* a única instrução mostrada em negrito:

```
void okClick(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Hello " + userName.Text);
}
```

Esse código executa uma função semelhante à função do aplicativo Windows Store, exceto que utiliza uma classe diferente, chamada *MessageBox*. Essa classe é definida no namespace *System.Windows*, o qual já é referenciado pelas diretivas *using* existentes no início do arquivo; portanto, você não precisa adicioná-lo.

7. Clique na guia MainPage.xaml ou na guia MainWindow.xaml acima da janela Code and Text Editor para exibir o formulário na janela Design View novamente.

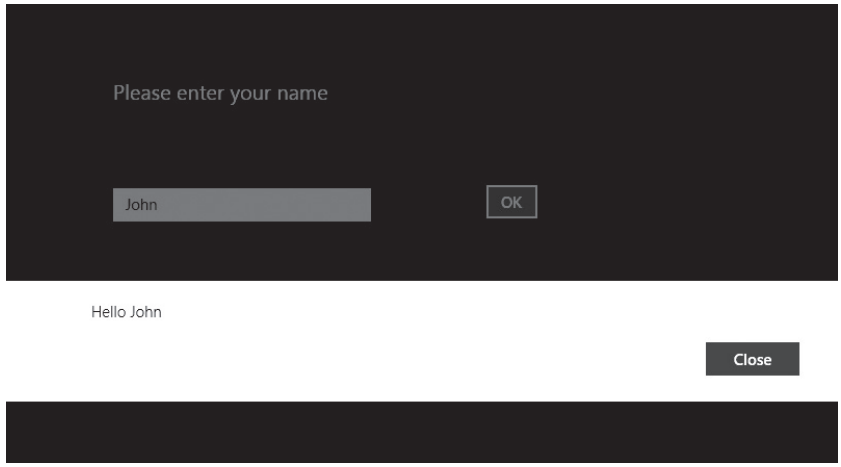
8. No painel inferior que exibe a descrição XAML do formulário, examine o elemento *Button*, mas tenha cuidado para não alterar nada. Observe que agora ele contém um elemento chamado *Click* que se refere ao método *okClick*:

```
<Button x:Name="ok" ... Click="okClick" />
```

9. No menu Debug, clique em Start Debugging.

10. Quando o formulário aparecer, digite seu nome sobre o texto existente na caixa de texto e então clique em OK.

Se você estiver usando o Windows 8.1, aparecerá um diálogo de mensagem no meio da tela, saudando-o pelo seu nome:



Se estiver usando Windows 7 ou Windows 8, aparecerá uma caixa de mensagem exibindo a seguinte saudação:

11. Clique em Close no diálogo de mensagem (Windows 8.1) ou em OK (Windows 7 ou Windows 8) na caixa de mensagem.

12. Volte para o Visual Studio 2013 e, então, no menu Debug, clique em Stop Debugging.

Resumo

Neste capítulo, você viu como é possível utilizar o Visual Studio 2013 para criar, construir e executar aplicativos. Você criou um aplicativo de console que exibe sua saída em uma janela de console e um aplicativo WPF com uma GUI simples.

- Se quiser continuar no próximo capítulo, mantenha o Visual Studio 2013 executando e vá para o Capítulo 2.
- Se quiser encerrar o Visual Studio 2013 agora, no menu File, clique em Exit. Se vir uma caixa de diálogo Save, clique em Yes para salvar o projeto.

Referência rápida

Para	Faça isto
Criar um novo aplicativo de console no Visual Studio 2013	No menu File, aponte para New e clique em Project para abrir a caixa de diálogo New Project. No painel à esquerda, em Installed Templates, clique em Visual C#. No painel central, clique em Console Application. Na caixa Location, especifique um diretório para os arquivos de projeto. Digite um nome para o projeto e clique em OK.
Criar um novo aplicativo gráfico Windows Store em branco para Windows 8.1 no Visual Studio 2013	No menu File, aponte para New e clique em Project para abrir a caixa de diálogo New Project. No painel da esquerda, na seção Installed Templates, expanda Visual C# e clique em Windows Store. No painel central, clique em Blank App (XAML). Na caixa Location, especifique um diretório para os arquivos de projeto. Digite um nome para o projeto e clique em OK.
Criar um novo aplicativo gráfico WPF para Windows 7 ou Windows 8 no Visual Studio 2013	No menu File, aponte para New e clique em Project para abrir a caixa de diálogo New Project. No painel da esquerda, na seção Installed Templates, expanda Visual C# e clique em Windows. No painel central, clique em WPF Application. Especifique um diretório para os arquivos do projeto na caixa Location. Digite um nome para o projeto e clique em OK.
Compilar o aplicativo	No menu Build, clique em Build Solution.
Executar o aplicativo no modo Debug	No menu Debug, clique em Start Debugging.
Executar o aplicativo sem depurar	No menu Debug, clique em Start Without Debugging.