

Contents

Windows Forms para .NET

Novidades

Introdução

- Visão geral

- Criar um aplicativo

Migração

- Migrar para .NET 5

Formulários

- Manipuladores de eventos

- Dimensionamento automático

- Tarefas comuns

 - Adicionar um formulário

 - Redimensionar um formulário

 - Posicionar um formulário

Controles

- Visão geral

- Opções de layout

- Rótulos

- Controles personalizados

- Pintura e desenho personalizados

- Aplicar informações sobre acessibilidade

- Tarefas comuns

 - Adicionar um controle para um formulário

 - Criar atalhos de chave de acesso

 - Definir o texto exibido por um controle

 - Definir a ordem de tabulação de um controle

 - Controles de encaixe e âncora

 - Definir a imagem exibida por um controle

Entrada de usuário – teclado

Visão geral

Usar eventos de teclado

Validar entrada

Tarefas comuns

- Alterar a tecla pressionada

- Determinar qual tecla modificadora foi pressionada

- Manipular a entrada no nível do formulário

- Simular eventos de teclado

Entrada de usuário – mouse

Visão geral

Usar eventos de mouse

Funcionalidade do tipo "arrastar e soltar"

Tarefas comuns

- Distinguir entre cliques e cliques duplos

- Controlar e modificar o ponteiro do mouse

- Simular eventos de mouse

O que há de novo (Windows Forms .NET)

14/05/2021 • 2 minutes to read

O Windows Forms para .NET 5,0 adiciona os seguintes recursos e aprimoramentos em .NET Framework.

Há algumas alterações significativas que você deve estar atento ao migrar do .NET Framework para o .NET 5,0. Para obter mais informações, consulte [alterações significativas em Windows Forms](#).

Recursos avançados

- Os padrões de automação da interface do usuário da Microsoft funcionam melhor com ferramentas de acessibilidade como Narrator e Jaws.
- Melhor desempenho.
- O modelo de projeto VB.NET usa como padrão as configurações de DPI SystemAware para resoluções de DPI alto, como monitores de 4K.
- A fonte padrão corresponde às recomendações de design atuais do Windows.

Caution

Isso pode afetar o layout dos aplicativos migrados do .NET Framework.

Novos controles

Os seguintes controles foram adicionados desde que Windows Forms foi transportado para .NET Framework:

- [System.Windows.Forms.TaskDialog](#)

Um diálogo de tarefa é uma caixa de diálogo que pode ser usada para exibir informações e receber a entrada simples do usuário. Como uma caixa de mensagem, ela é formatada pelo sistema operacional de acordo com os parâmetros definidos. A caixa de diálogo de tarefa tem mais recursos do que uma mensagem. Para obter mais informações, consulte o [exemplo da caixa de diálogo tarefa](#).

- [Microsoft.Web.WebView2.WinForms.WebView2](#)

Um novo controle de navegador da Web com suporte da Web moderno. Com base na borda (Chromium). Para obter mais informações, consulte [introdução ao WebView2 no Windows Forms](#).

Controles aprimorados

- [System.Windows.Forms.ListView](#)
 - Dá suporte a grupos recolhíveis
 - Rodapés
 - Subtítulo de grupo, tarefa e imagens de título
- [System.Windows.Forms.FolderBrowserDialog](#)

Essa caixa de diálogo foi atualizada para usar a experiência moderna do Windows em vez da experiência antiga do Windows 7.

- [System.Windows.Forms.FileDialog](#)
 - Adicionado suporte para [ClientGuid](#).

`ClientGuid` permite que um aplicativo de chamada associe um GUID ao estado persistente de uma caixa de diálogo. O estado de uma caixa de diálogo pode incluir fatores como a última pasta visitada e a posição e o tamanho da caixa de diálogo. Normalmente, esse estado é persistido com base no nome do arquivo executável. Com `ClientGuid` o, um aplicativo pode persistir Estados diferentes da caixa de diálogo dentro do mesmo aplicativo.

- [System.Windows.Forms.TextRenderer](#)

Suporte adicionado para [ReadOnlySpan<T>](#) para aprimorar o desempenho do processamento de texto.

Confira também

- [Alterações recentes no Windows Forms](#)
- [Tutorial: criar um novo aplicativo WinForms \(Windows Forms .NET\)](#)
- [Como migrar um aplicativo de área de trabalho Windows Forms para o .NET 5](#)

Guia da área de trabalho (Windows Forms .NET)

14/05/2021 • 6 minutes to read

Bem-vindo ao guia da área de trabalho para Windows Forms, uma estrutura de interface do usuário que cria aplicativos cliente de desktop avançados para Windows. A plataforma de desenvolvimento de Windows Forms dá suporte a um amplo conjunto de recursos de desenvolvimento de aplicativos, incluindo controles, elementos gráficos, vinculação de dados e entrada do usuário. Windows Forms apresenta um designer visual de arrastar e soltar no Visual Studio para criar facilmente Windows Forms aplicativos.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Há duas implementações de Windows Forms:

1. A implementação de código-fonte aberto hospedada no [GitHub](#).

Esta versão é executada no .NET 5 e no .NET Core 3,1. O designer de Windows Forms visual requer, no mínimo, o [Visual Studio 2019 versão 16,8 Preview](#).

2. A implementação .NET Framework 4 com suporte do Visual Studio 2019 e do Visual Studio 2017.

O .NET Framework 4 é uma versão somente para Windows do .NET e é considerado um componente do sistema operacional Windows. Esta versão do Windows Forms é distribuída com .NET Framework.

Este guia da área de trabalho foi escrito para Windows Forms no .NET 5. Para obter mais informações sobre a versão .NET Framework do Windows Forms, consulte [Windows Forms para .NET Framework](#).

Introdução

Windows Forms é uma estrutura de interface do usuário para criar aplicativos da área de trabalho do Windows. Ele fornece uma das maneiras mais produtivas de criar aplicativos de área de trabalho com base no designer visual fornecido no Visual Studio. Funcionalidades como o posicionamento do tipo "arrastar e soltar" de controles visuais facilitam a criação de aplicativos da área de trabalho.

Com o Windows Forms, você desenvolve aplicativos graficamente ricos que são fáceis de implantar, atualizar e trabalhar enquanto estiverem offline ou conectados à Internet. Windows Forms aplicativos podem acessar o hardware local e o sistema de arquivos do computador em que o aplicativo está sendo executado.

Para saber como criar um aplicativo Windows Forms, consulte [tutorial: criar um novo aplicativo WinForms \(Windows Forms .net\)](#).

Por que migrar do .NET Framework

O Windows Forms para .NET 5,0 fornece novos recursos e aprimoramentos em .NET Framework. Para obter mais informações, consulte [What ' s New in Windows Forms for .NET 5](#). Para saber como migrar um aplicativo, consulte [como migrar um aplicativo de área de trabalho Windows Forms para o .NET 5](#).

Crie interfaces de usuário sofisticadas e interativas

Windows Forms é uma tecnologia de interface do usuário para .NET, um conjunto de bibliotecas gerenciadas que simplificam tarefas comuns de aplicativos, como leitura e gravação no sistema de arquivos. Ao usar um

ambiente de desenvolvimento como o Visual Studio, você pode criar Windows Forms aplicativos de cliente inteligente que exibem informações, solicitam a entrada de usuários e se comunicam com computadores remotos em uma rede.

nos Windows Forms, um *formulário* é uma superfície visual na qual são exibidas informações para o usuário. Normalmente, você cria Windows Forms aplicativos adicionando controles a formulários e desenvolvendo respostas para ações do usuário, como cliques do mouse ou pressionamentos de tecla. Um *controle* é um elemento de interface do usuário discreto que exibe dados ou aceita entrada de dados.

Quando um usuário executa alguma ação em seu formulário ou em um de seus controles, a ação gera um evento. Seu aplicativo reage a esses eventos com o código e processa os eventos quando eles ocorrem.

O Windows Forms contém uma variedade de controles que você pode adicionar a formulários: controles que exibem caixas de texto, botões, caixas suspensas, botões de opção e até mesmo páginas da Web. Se um controle existente não atender às suas necessidades, Windows Forms também dará suporte à criação de seus próprios controles personalizados usando a [UserControl](#) classe.

Windows Forms tem controles avançados de interface do usuário que emulam recursos em aplicativos de alto nível, como Microsoft Office. Ao usar os [ToolStrip](#) controles e [MenuStrip](#), você pode criar barras de ferramentas e menus que contêm texto e imagens, exibir submenus e hospedar outros controles, como caixas de texto e caixas de combinação.

Com o **Designer de formulários do Windows** do tipo "arrastar e soltar" no Visual Studio, você pode facilmente criar Windows Forms aplicativos. Basta selecionar os controles com o cursor e colocá-los onde você deseja no formulário. O designer oferece ferramentas como linhas de grade e linhas de alinhamento para facilitar o alinhamento dos controles. Você pode usar os [FlowLayoutPanel](#) [TableLayoutPanel](#) controles, e [SplitContainer](#) para criar layouts de formulário avançados em menos tempo.

Por fim, se você precisar criar seus próprios elementos personalizados de interface do usuário, o [System.Drawing](#) namespace conterá uma grande seleção de classes para processar linhas, círculos e outras formas diretamente em um formulário.

Criar formulários e controles

Para obter informações passo a passo sobre como usar esses recursos, consulte os seguintes tópicos da Ajuda.

- [Como adicionar um formulário a um projeto](#)
- [Como adicionar controles a um formulário](#)

Exibir e manipular dados

Muitos aplicativos devem exibir dados de um banco de dados, um arquivo XML ou JSON, um serviço Web ou outra fonte. Windows Forms fornece um controle flexível chamado de [DataGridView](#) controle para exibir esses dados tabulares em um formato de linha e coluna tradicional, para que cada parte dos dados ocupe sua própria célula. Ao usar [DataGridView](#) o, você pode personalizar a aparência de células individuais, bloquear linhas e colunas arbitrárias em vigor e exibir controles complexos dentro de células, entre outros recursos.

Conectar-se a fontes de dados em uma rede é uma tarefa simples com Windows Forms. O [BindingSource](#) componente representa uma conexão com uma fonte de dados e expõe métodos para associação de dados a controles, navegando até os registros anteriores e seguintes, editando registros e salvando as alterações de volta na fonte original. O [BindingNavigator](#) controle fornece uma interface simples sobre o [BindingSource](#) componente para que os usuários naveguem entre os registros.

Você pode criar controles vinculados a dados facilmente usando a janela fontes de dados no Visual Studio. A janela exibe fontes de dados como, por exemplo, databases, serviços Web e objetos em seu projeto. Você pode criar controles de associação de dados ao arrastar itens dessa janela para os formulários do seu projeto. Você também pode associar controles existentes a dados ao arrastar objetos da janela Fontes de Dados para eles.

Outro tipo de vinculação de dados que você pode gerenciar nos Windows Forms é chamado de *configurações*. A maioria dos aplicativos deve reter algumas informações sobre o estado de tempo de execução, como o último tamanho de formulários, e manter os dados de preferência do usuário, como locais padrão para arquivos salvos. O recurso de configuração de aplicativo lida com esses requisitos ao oferecer uma forma fácil de armazenar ambos os tipos de configuração no computador cliente. Depois de definir essas configurações usando o Visual Studio ou um editor de código, as configurações são mantidas como XML e são automaticamente lidas de volta na memória em tempo de execução.

Implantar aplicativos em computadores cliente

Depois de escrever seu aplicativo, você deve enviar o aplicativo aos seus usuários para que eles possam instalá-lo e executá-lo em seus próprios computadores cliente. Ao usar a tecnologia ClickOnce, você pode implantar seus aplicativos de dentro do Visual Studio usando apenas alguns cliques e fornecer aos usuários uma URL apontando para seu aplicativo na Web. O ClickOnce gerencia todos os elementos e dependências em seu aplicativo e garante que o aplicativo esteja instalado corretamente no computador cliente.

Os aplicativos ClickOnce podem ser configurados para serem executados somente quando o usuário estiver conectado à rede ou para serem executados online e offline. Quando você especifica que um aplicativo deve dar suporte à operação offline, o ClickOnce adiciona um link ao seu aplicativo no menu **Iniciar** do usuário. O usuário pode abrir o aplicativo sem usar a URL.

Ao atualizar seu aplicativo, você publica um novo manifesto de implantação e uma nova cópia do seu aplicativo em seu servidor Web. O ClickOnce detectará que há uma atualização disponível e atualizará a instalação do usuário. Nenhuma programação personalizada é necessária para atualizar aplicativos antigos.

Confira também

- [Tutorial: criar um novo aplicativo WinForms \(Windows Forms .NET\)](#)
- [Como adicionar um formulário a um projeto \(Windows Forms .NET\)](#)
- [Adicionar um controle \(Windows Forms .NET\)](#)

Tutorial: criar um novo aplicativo WinForms (Windows Forms .NET)

14/05/2021 • 4 minutes to read

Neste breve tutorial, você aprenderá a criar um novo aplicativo Windows Forms (WinForms) com o Visual Studio. Depois que o aplicativo inicial tiver sido gerado, você aprenderá a adicionar controles e a manipular eventos. Ao final deste tutorial, você terá um aplicativo simples que adiciona nomes a uma caixa de listagem.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Neste tutorial, você aprenderá como:

- Criar um novo aplicativo WinForms
- Adicionar controles a um formulário
- Manipular eventos de controle para fornecer funcionalidade de aplicativo
- Executar o aplicativo

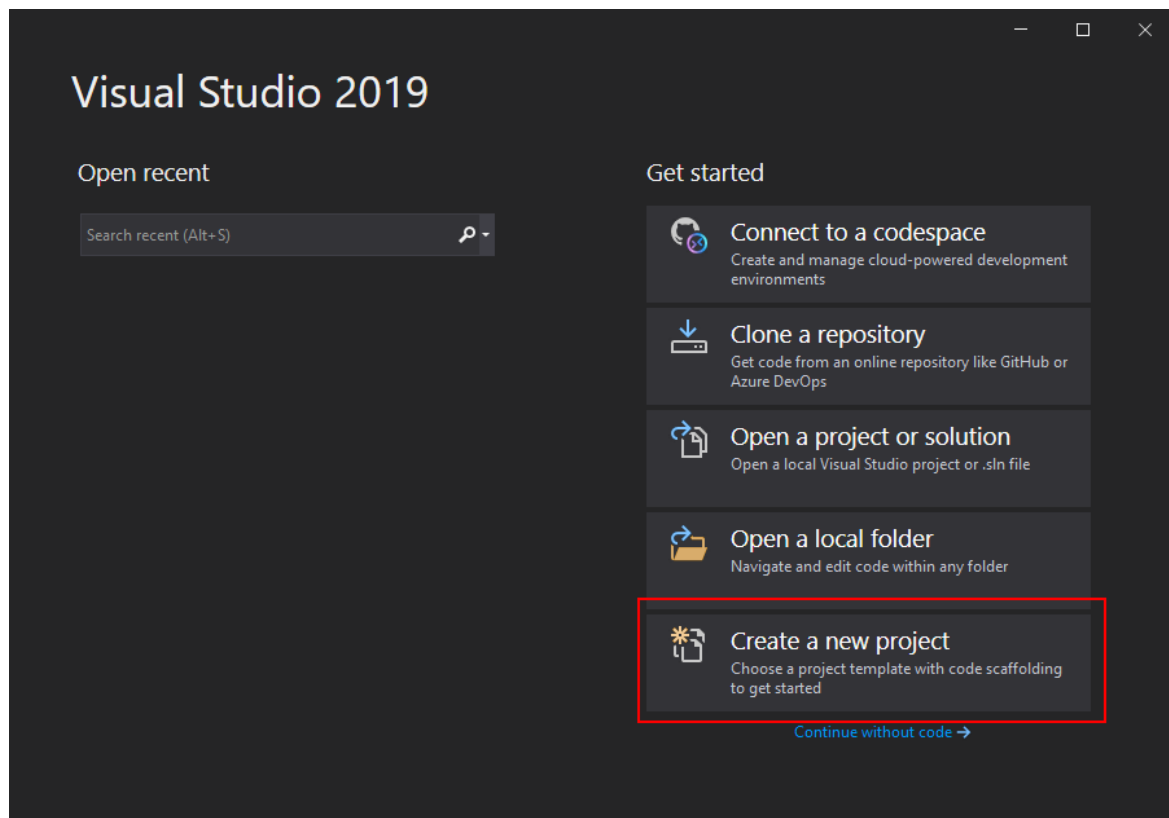
Pré-requisitos

- [Visual Studio 2019 versão 16,8 ou versões posteriores](#)
 - Selecione a [carga de trabalho do Visual Studio desktop](#)
 - Selecione o [componente individual do .NET 5](#)

Criar um aplicativo WinForms

A primeira etapa para criar um novo aplicativo é abrir o Visual Studio e gerar o aplicativo a partir de um modelo.

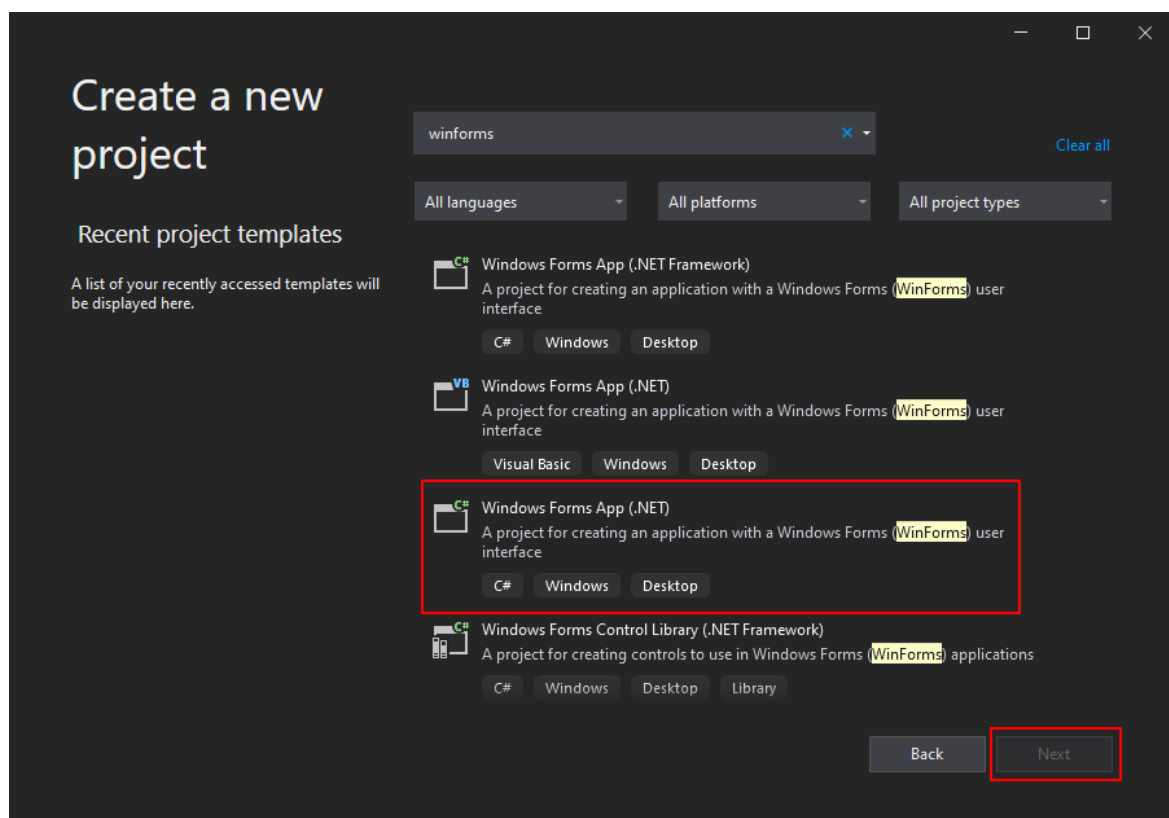
1. Abra o Visual Studio.
2. Selecione **Criar um novo projeto**.



3. Na caixa **Pesquisar modelos** , digite **WinForms** e pressione Enter.
4. Na lista suspensa **linguagem de código** , escolha **C#** ou **Visual Basic**.
5. Na lista modelos, selecione **Windows Forms aplicativo (.net)** e clique em **Avançar**.

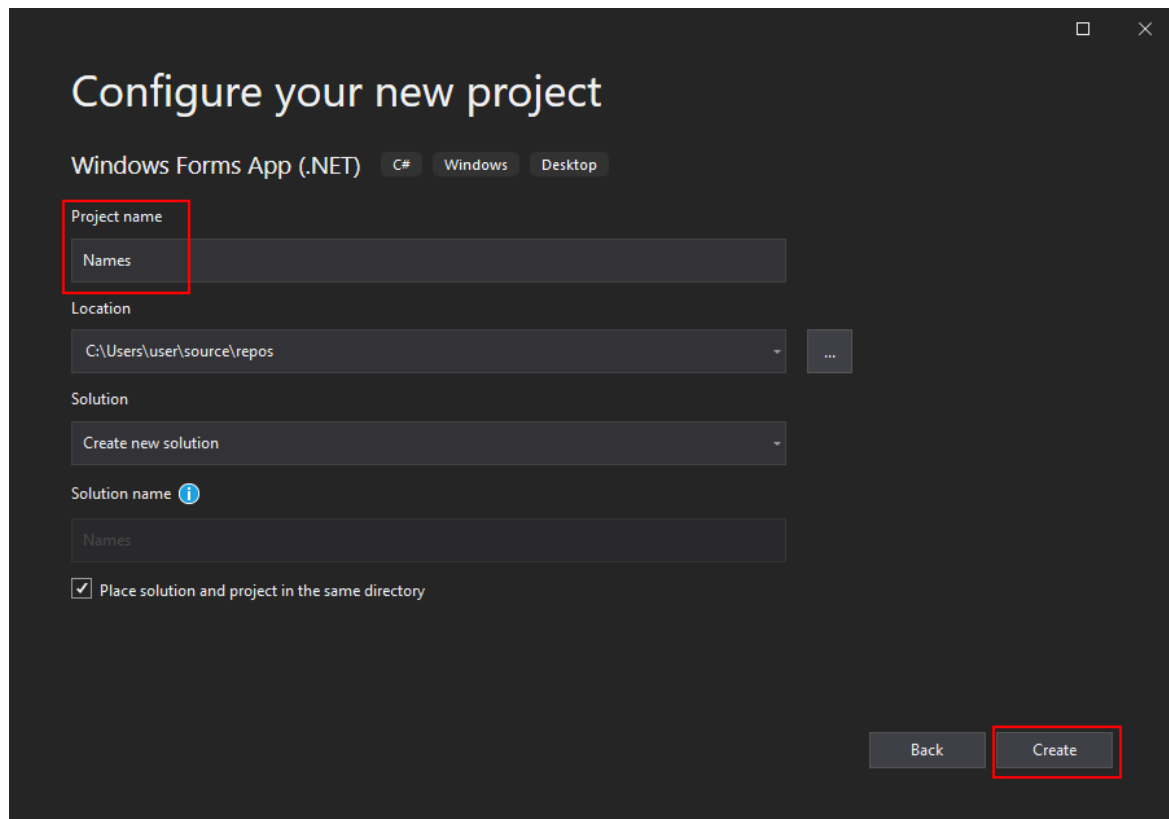
IMPORTANT

Não selecione o modelo aplicativo Windows Forms (.NET Framework) .



6. Na janela **configurar seu novo projeto**, defina o **nome do projeto** para **nomes** e clique em **criar**.

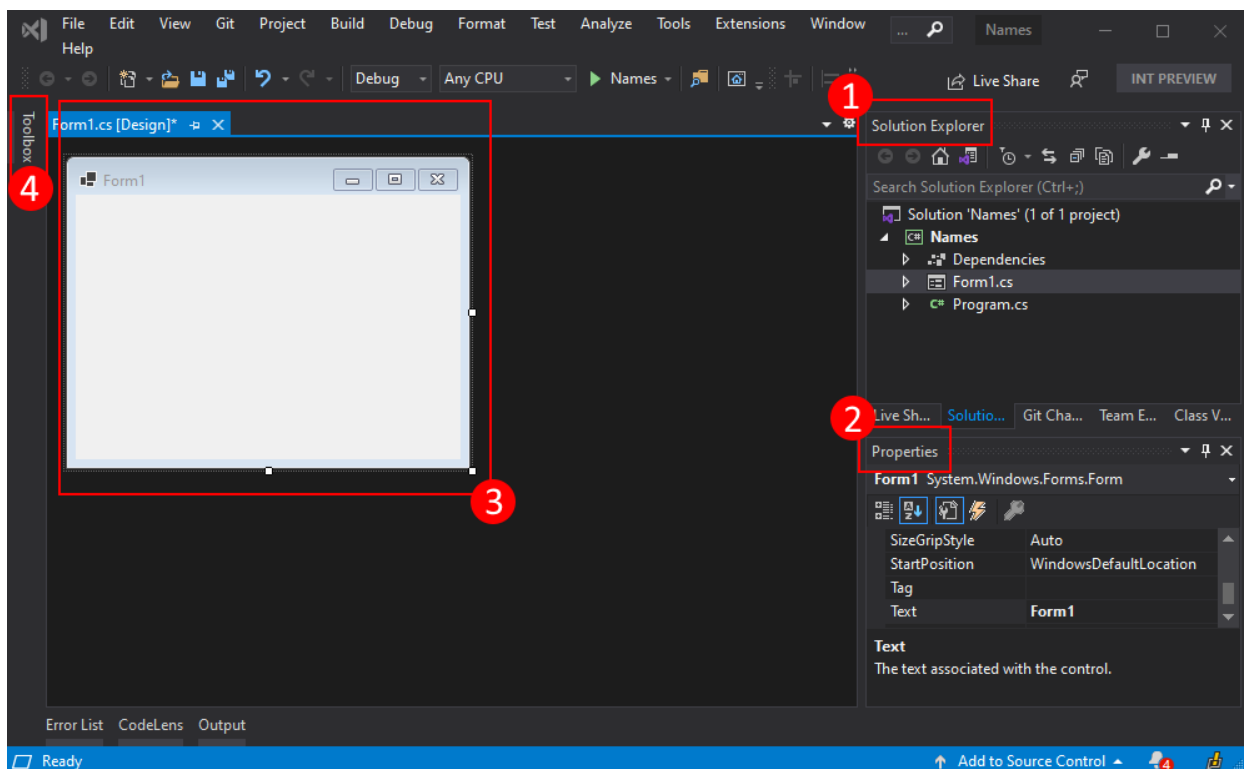
Você também pode salvar seu projeto em uma pasta diferente ajustando a configuração de **local**.



Depois que o aplicativo é gerado, o Visual Studio deve abrir o painel designer para o formulário padrão, *Form1*. Se o designer de formulário não estiver visível, clique duas vezes no formulário no painel de **Gerenciador de soluções** para abrir a janela do designer.

Partes importantes do Visual Studio

O suporte para WinForms no Visual Studio tem quatro componentes importantes com os quais você irá interagir ao criar um aplicativo:



1. Gerenciador de Soluções

Todos se os arquivos do projeto, o código, os formulários, os recursos, serão exibidos neste painel.

2. Propriedades

Esse painel mostra as configurações de propriedade que podem ser definidas com base no item selecionado. Por exemplo, se você selecionar um item de **Gerenciador de soluções**, verá as configurações de propriedade relacionadas ao arquivo. Se você selecionar um objeto no **Designer**, você verá as configurações para o controle ou formulário.

3. Designer de formulário

Este é o designer do formulário. É interativo e você pode arrastar e soltar objetos da **caixa de ferramentas**. Ao selecionar e mover itens no designer, você pode compor visualmente a interface do usuário para seu aplicativo.

4. Caixa de Ferramentas

A caixa de ferramentas contém todos os controles que você pode adicionar a um formulário. Para adicionar um controle ao formulário atual, clique duas vezes em um controle ou arraste e solte o controle.

Adicionar controles ao formulário

Com o designer de formulário do *Form1* aberto, use o painel **caixa de ferramentas** para adicionar os seguintes controles ao formulário:

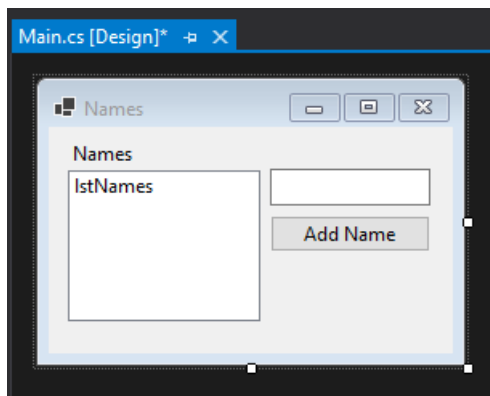
- Rotular
- Botão
- Caixas
- Caixa de texto

Você pode posicionar e dimensionar os controles de acordo com as configurações a seguir. Mova-os visualmente para corresponder à captura de tela que segue, ou clique em cada controle e defina as configurações no painel **Propriedades**. Você também pode clicar na área título do formulário para selecionar o formulário:

OBJETO	CONFIGURAÇÃO	VALOR
Formulário	Texto	<input type="text" value="Names"/>
	Tamanho	<input type="text" value="268, 180"/>
Rotular	Local	<input type="text" value="12, 9"/>
	Texto	<input type="text" value="Names"/>
Caixas	Nome	<input type="text" value="1stNames"/>
	Localização	<input type="text" value="12, 27"/>
	Tamanho	<input type="text" value="120, 94"/>


OBJETO	CONFIGURAÇÃO	VALOR
Caixa de texto	Nome	<code>txtName</code>
	Localização	<code>138, 26</code>
	Tamanho	<code>100, 23</code>
Botão	Nome	<code>btnAdd</code>
	Localização	<code>138, 55</code>
	Tamanho	<code>100, 23</code>
	Texto	<code>Add Name</code>

Você deve ter um formulário no designer parecido com o seguinte:



Tratar eventos

Agora que o formulário tem todos os seus controles dispostos, você precisa manipular os eventos dos controles para responder à entrada do usuário. Com o designer de formulário ainda aberto, execute as seguintes etapas:

1. Selecione o controle de botão no formulário.
2. No painel **Propriedades**, clique no ícone eventos  para listar os eventos do botão.
3. Localize o evento de **clique** e clique duas vezes nele para gerar um manipulador de eventos.

Essa ação adiciona o seguinte código ao formulário:

```
private void btnAdd_Click(object sender, EventArgs e)
{
}

```

```
Private Sub btnAdd_Click(sender As Object, e As EventArgs) Handles btnAdd.Click
End Sub

```

O código que colocaremos neste manipulador adicionará o nome especificado pelo `txtName` controle TextBox ao `lstNames` controle ListBox. No entanto, queremos que haja duas condições para adicionar o nome: o nome fornecido não deve estar em branco e o nome já não deve existir.

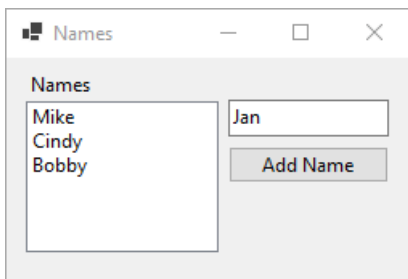
4. O código a seguir demonstra como adicionar um nome ao `lstNames` controle:

```
private void btnAdd_Click(object sender, EventArgs e)
{
    if (!string.IsNullOrEmpty(txtName.Text) && !lstNames.Items.Contains(txtName.Text))
        lstNames.Items.Add(txtName.Text);
}
```

```
Private Sub btnAdd_Click(sender As Object, e As EventArgs) Handles btnAdd.Click
    If Not String.IsNullOrEmpty(txtName.Text) And Not lstNames.Items.Contains(txtName.Text) Then
        lstNames.Items.Add(txtName.Text)
    End If
End Sub
```

Executar o aplicativo

Agora que o evento foi codificado, você pode executar o aplicativo pressionando a tecla F5 ou selecionando **depurar > Iniciar Depuração** no menu. O formulário é exibido e você pode inserir um nome na caixa de texto e, em seguida, adicioná-lo clicando no botão.



Próximas etapas

[Saiba mais sobre o Windows Forms](#)

Como migrar um aplicativo Windows Forms desktop para o .NET 5

19/05/2021 • 9 minutes to read

Este artigo descreve como migrar um aplicativo Windows Forms desktop do .NET Framework para o .NET 5 ou posterior. O SDK do .NET inclui suporte para Windows Forms aplicativos. O Windows Forms ainda é uma estrutura somente do Windows e só é executado nessa plataforma.

TIP

Experimente o [Assistente de Atualização do .NET](#) para ajudar a migrar seu projeto.

Migrar seu aplicativo do .NET Framework para o .NET 5 geralmente requer um novo arquivo de projeto. O .NET 5 usa arquivos de projeto no estilo SDK enquanto .NET Framework normalmente usa o arquivo Visual Studio projeto mais antigo. Se você já abriu um arquivo Visual Studio projeto em um editor de texto, sabe como ele é detalhado. Os projetos no estilo SDK são menores e não exigem tantas entradas quanto o formato de arquivo de projeto mais antigo.

Para saber mais sobre o .NET 5, consulte [Introdução ao .NET](#).

Pré-requisitos

- [Visual Studio versão 2019 versão 16.8 Preview](#)
 - Selecione a carga [de trabalho Visual Studio Desktop](#).
 - Selecione o [componente individual do .NET 5](#).

- Versão prévia do designer winForms Visual Studio.

Para habilitar o designer, acesse **Ferramentas** Opções Recursos de Visualização do Ambiente e selecione a opção Usar a versão > > > **prévia Windows Forms designer para aplicativos .NET Core**.

- Este artigo usa o aplicativo [de exemplo Correspondência](#) de jogos. Se você quiser acompanhar, baixe e abra o aplicativo no Visual Studio. Caso contrário, use seu próprio aplicativo.

Considerações

Ao migrar um .NET Framework Windows Forms, há algumas coisas que você deve considerar.

1. Verifique se o seu aplicativo é um bom candidato para a migração.

Use o [.net Portability Analyzer](#) para determinar se o seu projeto será migrado para o .NET 5. Se o seu projeto tiver problemas com o .NET 5, o analisador o ajudará a identificar esses problemas. A ferramenta do analisador de portabilidade .NET pode ser instalada como uma extensão do Visual Studio ou usada na linha de comando. Para obter mais informações, consulte [Analisador de Portabilidade do .NET](#).

2. Você está usando uma versão diferente do Windows Forms.

Quando o .NET Core 3,0 foi lançado, Windows Forms tornou [-se aberto no GitHub](#). O código para Windows Forms para .NET 5 é uma bifurcação da base de código .NET Framework Windows Forms. É possível haver algumas diferenças e seu aplicativo será difícil de migrar.

3. O [Pacote de Compatibilidade do Windows](#) pode ajudar você na migração.

Algumas APIs que estão disponíveis no .NET Framework não estão disponíveis no .NET 5. O [pacote de compatibilidade do Windows](#) adiciona muitas dessas APIs e pode ajudar seu Windows Forms aplicativo se tornar compatível com o .NET 5.

4. Atualize os pacotes do NuGet usados pelo seu projeto.

É sempre uma boa prática usar as versões mais recentes dos pacotes do NuGet antes de qualquer migração. Se seu aplicativo faz referência a pacotes do NuGet, atualize-os para a versão mais recente. Certifique-se de que seu aplicativo foi compilado com êxito. Após a atualização, se houver erros de pacote, faça downgrade do pacote para a versão mais recente que não interrompa seu código.

Fazer backup de seus projetos

A primeira etapa para migrar um projeto é fazer backup de seu projeto! Se algo der errado, você poderá restaurar seu código para seu estado original restaurando o backup. Não confie em ferramentas como o .NET Portability Analyzer para fazer backup de seu projeto, mesmo que eles pareçam. É melhor criar uma cópia do projeto original de forma pessoal.

Pacotes NuGet

Se o seu projeto estiver fazendo referência a pacotes NuGet, você provavelmente terá um arquivo **packages.config** na pasta do projeto. Com os projetos em estilo SDK, as referências de pacote NuGet são configuradas no arquivo de projeto. Os arquivos de projeto do Visual Studio podem opcionalmente definir pacotes NuGet no arquivo de projeto. O .NET 5 não usa **packages.config** para pacotes NuGet. As referências de pacote NuGet devem ser migradas para o arquivo de projeto antes da migração.

Para migrar o arquivo de **packages.config**, execute as seguintes etapas:

1. No **Gerenciador de soluções**, localize o projeto que você está migrando.
2. Clique com o botão direito **do packages.config > migrar packages.config para PackageReference**.
3. Select all dos pacotes de nível superior.

Um relatório de build é gerado para informar sobre os problemas de migração dos pacotes NuGet.

Arquivo de projeto

A próxima etapa na migração do aplicativo é converter o arquivo de projeto. Conforme mencionado anteriormente, o .NET 5 usa arquivos de projeto no estilo SDK e não carregará os Visual Studio de projeto que .NET Framework usa. No entanto, há a possibilidade de que você já esteja usando projetos no estilo SDK. Você pode identificar facilmente a diferença Visual Studio. Clique com o botão direito do mouse no arquivo de projeto no Explorador de Soluções e procure a **opção de** menu Editar Arquivo de Projeto. Se esse item de menu estiver ausente, você está usando o formato Visual Studio projeto antigo e precisa atualizar.

Converta cada projeto em sua solução. Se você estiver usando o aplicativo de exemplo referenciado anteriormente, os projetos **MatchingGame** e **MatchingGame.Logic** serão convertidos.

Para converter um projeto, faça as seguintes etapas:

1. No **Explorador de Soluções**, encontre o projeto que você está migrando.
2. Clique com o botão direito do mouse no projeto e selecione **Descarregar Projeto**.
3. Clique com o botão direito do mouse no projeto e **selecione Editar Arquivo de Projeto**.
4. Copie e copie e copie o XML do projeto em um editor de texto. Você deseja uma cópia para que seja fácil mover o conteúdo para o novo projeto.

5. Apagar o conteúdo do arquivo e colar o seguinte XML:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>WinExe</OutputType>
    <TargetFramework>net5.0-windows</TargetFramework>
    <UseWindowsForms>true</UseWindowsForms>
    <GenerateAssemblyInfo>false</GenerateAssemblyInfo>
  </PropertyGroup>

</Project>
```

IMPORTANT

As bibliotecas não precisam definir uma `<OutputType>` configuração. Remova essa entrada se você estiver atualizando um projeto de biblioteca.

Esse XML fornece a estrutura básica do projeto. No entanto, ele não contém nenhuma das configurações do arquivo de projeto antigo. Usando as informações antigas do projeto que você copiou anteriormente para um editor de texto, execute as seguintes etapas:

1. Copie os seguintes elementos do arquivo de projeto antigo para o `<PropertyGroup>` elemento no novo arquivo de projeto:

- `<RootNamespace>`
- `<AssemblyName>`

O arquivo de projeto deve ser semelhante ao seguinte XML:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>WinExe</OutputType>
    <TargetFramework>net5.0-windows</TargetFramework>
    <UseWindowsForms>true</UseWindowsForms>
    <GenerateAssemblyInfo>false</GenerateAssemblyInfo>

    <RootNamespace>MatchingGame</RootNamespace>
    <AssemblyName>MatchingGame</AssemblyName>
  </PropertyGroup>

</Project>
```

2. Copie os `<ItemGroup>` elementos do arquivo de projeto antigo que contém `<ProjectReference>` ou `<PackageReference>` para o novo arquivo após a `</PropertyGroup>` marca de fechamento.

O arquivo de projeto deve ser semelhante ao seguinte XML:


```

<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    (contains settings previously described)
  </PropertyGroup>

  <ItemGroup>
    <ProjectReference Include="..\MatchingGame.Logic\MatchingGame.Logic.csproj">
      <Project>{36b3e6e2-a9ae-4924-89ae-7f0120ce08bd}</Project>
      <Name>MatchingGame.Logic</Name>
    </ProjectReference>
  </ItemGroup>
  <ItemGroup>
    <PackageReference Include="MetroFramework">
      <Version>1.2.0.3</Version>
    </PackageReference>
  </ItemGroup>

</Project>

```

Os `<ProjectReference>` elementos não precisam dos `<Project>` `<Name>` filhos e, portanto, você pode remover essas configurações:

```

<ItemGroup>
  <ProjectReference Include="..\MatchingGame.Logic\MatchingGame.Logic.csproj" />
</ItemGroup>

```

Recursos e configurações

Uma coisa a ser observada sobre a diferença entre .NET Framework projetos e os projetos no estilo SDK usados pelo .NET 5 é que .NET Framework projetos usam um modelo de aceitação para arquivos de código. Qualquer arquivo de código que você queira compilar precisa ser explicitamente definido em seu arquivo de projeto. Os projetos no estilo SDK são invertidos; eles assumem como padrão o comportamento de recusa: todos os arquivos de código iniciados no diretório do projeto e abaixo são incluídos automaticamente no seu projeto. Você não precisará migrar essas entradas se elas forem simples e sem configurações. Isso é o mesmo para outros arquivos comuns, como *resx*.

Windows Forms projetos também podem referenciar os seguintes arquivos:

- *Configurações de propriedades \ . configurações*
- *Propriedades \ Resources. resx*
- *Propriedades \ app. manifest*

O arquivo *app. manifest* é automaticamente referenciado pelo seu projeto e você não precisa fazer nada especial para migrá-lo.

Qualquer arquivo **. resx* e **. Settings* na pasta *Propriedades* precisa ser migrado no projeto. Copie essas entradas do arquivo de projeto antigo em um `<ItemGroup>` elemento no novo projeto. Depois de copiar as entradas, altere todos os `<Compile Include="value">` elementos para, em vez disso, use o `Update` atributo em vez de `Include`.

- Importe a configuração do arquivo *Settings. Settings*.

```
<ItemGroup>
  <None Include="Properties\Settings.settings">
    <Generator>SettingsSingleFileGenerator</Generator>
    <LastGenOutput>Settings.Designer.cs</LastGenOutput>
  </None>
  <Compile Update="Properties\Settings.Designer.cs">
    <AutoGen>True</AutoGen>
    <DependentUpon>Settings.settings</DependentUpon>
    <DesignTimeSharedInput>True</DesignTimeSharedInput>
  </Compile>
</ItemGroup>
```

Observe que a entrada *Properties\Settings.settings* permaneceu `Include`. O projeto não inclui automaticamente os arquivos de *configuração*.

IMPORTANT

Os projetos de **Visual Basic** normalmente usam a pasta *meu projeto* enquanto projetos C# normalmente usam as *Propriedades* de pasta para o arquivo de configurações de projeto padrão.

- Importe a configuração para qualquer arquivo *resx*, como o arquivo *Properties | Resources.resx*.

Observe que o `Include` atributo foi definido como `Update` no `<Compile>` elemento e `<EmbeddedResource>` e `<SubType>` foi removido de `<EmbeddedResource>`:

```
<ItemGroup>
  <EmbeddedResource Update="Properties\Resources.resx">
    <Generator>ResXFileCodeGenerator</Generator>
    <LastGenOutput>Resources.Designer.cs</LastGenOutput>
  </EmbeddedResource>
  <Compile Update="Properties\Resources.Designer.cs">
    <AutoGen>True</AutoGen>
    <DependentUpon>Resources.resx</DependentUpon>
    <DesignTime>True</DesignTime>
  </Compile>
</ItemGroup>
```

IMPORTANT

Os projetos de **Visual Basic** normalmente usam a pasta *meu projeto* enquanto projetos C# normalmente usam as *Propriedades* de pasta para o arquivo de recurso de projeto padrão.

Visual Basic

Visual Basic projetos de idioma exigem configuração adicional.

1. Importe a configuração do arquivo de configuração *My Project | MyApp*. Observe que o `<Compile>` elemento usa o `Update` atributo em vez do `Include` atributo.

```

<ItemGroup>
  <None Include="My Project\Application.myapp">
    <Generator>MyApplicationCodeGenerator</Generator>
    <LastGenOutput>Application.Designer.vb</LastGenOutput>
  </None>
  <Compile Update="My Project\Application.Designer.vb">
    <AutoGen>True</AutoGen>
    <DependentUpon>Application.myapp</DependentUpon>
    <DesignTime>True</DesignTime>
  </Compile>
</ItemGroup>

```

2. Adicione a `<MyType>WindowsForms</MyType>` configuração ao `<PropertyGroup>` elemento:

```

<PropertyGroup>
  (contains settings previously described)

  <MyType>WindowsForms</MyType>
</PropertyGroup>

```

Essa configuração importa os `My` membros do namespace Visual Basic os programadores estão familiarizados com o.

3. Importe os namespaces definidos pelo seu projeto.

Visual Basic projetos podem importar automaticamente namespaces para cada arquivo de código. Copie os `<ItemGroup>` elementos do arquivo de projeto antigo que contém `<Import>` para o novo arquivo após a `</PropertyGroup>` marca de fechamento.

```

<ItemGroup>
  <Import Include="Microsoft.VisualBasic" />
  <Import Include="System" />
  <Import Include="System.Collections" />
  <Import Include="System.Collections.Generic" />
  <Import Include="System.Data" />
  <Import Include="System.Drawing" />
  <Import Include="System.Diagnostics" />
  <Import Include="System.Windows.Forms" />
  <Import Include="System.Linq" />
  <Import Include="System.Xml.Linq" />
  <Import Include="System.Threading.Tasks" />
</ItemGroup>

```

Se você não encontrar instruções ou se não for possível `<Import>` compilar o projeto, certifique-se de que pelo menos tenha as seguintes `<Import>` instruções definidas em seu projeto:

```

<ItemGroup>
  <Import Include="System.Data" />
  <Import Include="System.Drawing" />
  <Import Include="System.Windows.Forms" />
</ItemGroup>

```

4. No projeto original, copie as `<Option*>` configurações e `<StartupObject>` para o `<PropertyGroup>` elemento:

```
<PropertyGroup>
  (contains settings previously described)

  <OptionExplicit>On</OptionExplicit>
  <OptionCompare>Binary</OptionCompare>
  <OptionStrict>Off</OptionStrict>
  <OptionInfer>On</OptionInfer>
  <StartupObject>MatchingGame.My.MyApplication</StartupObject>
</PropertyGroup>
```

Recarregar o projeto

Depois de converter um projeto no novo formato de estilo do SDK, recarregue o projeto Visual Studio:

1. No **Gerenciador de Soluções**, encontre o projeto que você converteu.
2. Clique com o botão direito do mouse no projeto e **selecione Recarregar Projeto**.

Se o projeto não for carregado, você poderá ter introduzido um erro no XML do projeto. Abra o arquivo de projeto para edição e tente identificar e corrigir o erro. Se você não encontrar um erro, tente começar novamente.

Editar App.config

Se o aplicativo tiver um *arquivoApp.config*, remova o `<supportedRuntime>` elemento :

```
<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
```

Há algumas coisas que você deve considerar com o *arquivoApp.config* dados. O *App.config* no .NET Framework foi usado não apenas para configurar o aplicativo, mas para definir configurações de runtime e comportamento, como registro em log. O *App.config* no .NET 5+ (e no .NET Core) não é mais usado para configuração de runtime. Se o *App.config* arquivo tiver essas seções, eles não serão respeitados.

Adicionar o pacote de compatibilidade

Se o arquivo de projeto estiver sendo carregado corretamente, mas a compilação falhar para seu projeto e você receber erros semelhantes aos seguintes:

- Não foi possível encontrar o tipo ou o namespace `<some name>`
- O nome `<some name>` não existe no contexto atual

Talvez seja necessário adicionar o `Microsoft.Windows.Compatibility` pacote ao seu aplicativo. Esse pacote adiciona cerca de 21.000 APIs do .NET .NET Framework, como a classe e as APIs para interagir com o

`System.Configuration.ConfigurationManager` Registro do Windows. Adicione o pacote `Microsoft.Windows.Compatibility`.

Edite o arquivo de projeto e adicione o seguinte `<ItemGroup>` elemento:

```
<ItemGroup>
  <PackageReference Include="Microsoft.Windows.Compatibility" Version="5.0.0" />
</ItemGroup>
```

Testar seu aplicativo

Depois de terminar de migrar seu aplicativo, teste-o!

Próximas etapas

- Experimente o [Assistente de Atualização do .NET](#) para migrar seu aplicativo.
- Saiba mais sobre [alterações significativas no Windows Forms](#).
- Leia mais sobre o [Pacote de Compatibilidade do Windows](#).

Visão geral de eventos (Windows Forms .NET)

14/05/2021 • 3 minutes to read

Um evento é uma ação que você pode responder, ou "manipular", no código. Os eventos podem ser gerados por uma ação do usuário, como clicar no mouse ou pressionar uma tecla, por código do programa ou pelo sistema.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Os aplicativos controlados por eventos executam código em resposta a um evento. Cada formulário e controle expõe um conjunto predefinido de eventos com base nos quais você pode programar. Se um desses eventos ocorrer e houver um código de manipulador de eventos associado, esse código será invocado.

Os tipos de eventos gerados por um objeto variam, mas muitos tipos são comuns à maioria dos controles. Por exemplo, a maioria dos objetos lidará com um evento [Click](#). Se um usuário clicar em um formulário, o código no manipulador de eventos [Click](#) do formulário é executado.

NOTE

Muitos eventos ocorrem com outros eventos. Por exemplo, enquanto o evento [DoubleClick](#) ocorre, os eventos [MouseDown](#), [MouseUp](#) e [Click](#) ocorrem.

Para obter informações sobre como gerar e consumir um evento, consulte [manipulando e gerando eventos](#).

Delegados e sua função

Delegados são classes comumente usadas no .NET para criar mecanismos de manipulação de eventos. Delegados são praticamente equivalentes a ponteiros de função, geralmente usados em Visual C++ e em outras linguagens orientadas a objeto. No entanto, diferente dos ponteiros de função, as classes delegate são orientadas a objeto, fortemente tipadas e seguras. Além disso, onde um ponteiro de função contém apenas uma referência a uma função específica, um delegado consiste em uma referência a um objeto e referências a um ou mais métodos dentro do objeto.

Esse modelo de evento usa *delegados* para associar eventos aos métodos que são usados para tratá-los. A classe delegate permite que outras classes se registrem para a notificação de eventos, especificando um método de manipulador. Quando o evento ocorre, a classe delegate chama o método associado. Para obter mais informações sobre como definir delegados, consulte [manipulando e gerando eventos](#).

Essas classes podem ser associadas a um único método ou a vários métodos, o que chamamos de multicasting. Ao criar um delegado para um evento, você normalmente cria um evento de multicast. Uma exceção rara pode ser um evento que resulta em um procedimento específico (como exibir uma caixa de diálogo) que não repetiria logicamente várias vezes por evento. Para obter informações sobre como criar um delegado de multicast, consulte [como combinar delegados \(delegados de multicast\)](#).

Um delegado de multicast mantém uma lista de invocação dos métodos aos quais está associado. Essa classe é compatível com um método [Combine](#) para adicionar um método à lista de invocação e um método [Remove](#) para removê-lo.

Quando um evento é registrado pelo aplicativo, o controle gera o evento invocando a classe delegate para esse evento. A classe delegate chama o método vinculado. No caso mais comum (um delegado de multicast), o

delegado chama cada método associado na lista de invocação, por sua vez, que fornece uma notificação de um para muitos. Essa estratégia significa que o controle não precisa manter uma lista de objetos de destino para notificação de eventos – o delegado manipula todo o registro e a notificação.

Essas classes também permitem que vários eventos sejam associados ao mesmo método, permitindo uma notificação muitos para um. Por exemplo, um evento de clique de botão e um evento de clique de comando de menu podem invocar a mesma classe delegate, que chama um único método para lidar com esses eventos separados da mesma maneira.

O mecanismo de associação usado com delegados é dinâmico: um delegado pode ser associado em tempo de execução para qualquer método cuja assinatura corresponda à do manipulador de eventos. Com esse recurso, você pode configurar ou alterar o método associado de acordo com uma condição e anexar dinamicamente um manipulador de eventos a um controle.

Confira também

- [Manipulando e gerando eventos](#)

Dimensionamento automático (Windows Forms .NET)

14/05/2021 • 4 minutes to read

O dimensionamento automático permite que um formulário e seus controles, projetados em um computador com determinada resolução de vídeo ou fonte, sejam exibidos adequadamente em outro computador com uma fonte ou resolução de vídeo diferente. Garante que o formulário e seus controles sejam redimensionados de forma inteligente para serem consistentes com o Windows nativo e outros aplicativos nas máquinas dos usuários e de outros desenvolvedores. O dimensionamento automático e os estilos visuais permitem que Windows Forms aplicativos mantenham uma aparência consistente em comparação com os aplicativos nativos do Windows na máquina de cada usuário.

Na maior parte, o dimensionamento automático funciona conforme o esperado no Windows Forms. No entanto, as alterações no esquema de fontes podem ser problemáticas.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Necessidade de dimensionamento automático

Sem o dimensionamento automático, um aplicativo criado para uma resolução ou fonte de vídeo parecerá muito pequeno ou muito grande quando a resolução ou a fonte for alterada. Por exemplo, se o aplicativo for projetado usando o ponto Tahoma 9 como uma linha de base, sem ajuste, ele aparecerá muito pequeno se for executado em um computador em que a fonte do sistema for Tahoma 12 Point. Elementos de texto, como títulos, menus, conteúdo de caixa de texto e assim por diante, serão processados menores do que outros aplicativos. Além disso, o tamanho dos elementos da interface do usuário que contêm texto, como a barra de título, os menus e muitos controles, dependem da fonte usada. Neste exemplo, esses elementos também serão relativamente menores.

Uma situação análoga ocorre quando um aplicativo é projetado para uma determinada resolução de vídeo. A resolução de vídeo mais comum é de 96 pontos por polegada (DPI), que é igual a 100% de escala de exibição, mas a resolução mais alta é exibida com suporte a 125%, 150%, 200% (que, respectivamente, igual a 120, 144 e 192 DPI) e acima estão se tornando mais comuns. Sem ajuste, um aplicativo, especialmente um baseado em gráficos, projetado para uma resolução aparecerá muito grande ou muito pequeno quando executado em outra resolução.

O dimensionamento automático busca resolver esses problemas redimensionando automaticamente o formulário e seus controles filho de acordo com o tamanho relativo da fonte ou a resolução da tela. O sistema operacional Windows dá suporte ao dimensionamento automático de caixas de diálogo usando uma unidade relativa de medida chamada unidades de caixa de diálogo. Uma unidade de caixa de diálogo é baseada na fonte do sistema e sua relação com os pixels pode ser determinada por meio da função do SDK do Win32

`GetDialogBaseUnits`. Quando um usuário altera o tema usado pelo Windows, todas as caixas de diálogo são ajustadas automaticamente de acordo. Além disso, o Windows Forms dá suporte ao dimensionamento automático de acordo com a fonte do sistema padrão ou a resolução de vídeo. Opcionalmente, o dimensionamento automático pode ser desabilitado em um aplicativo.

Caution

Não há suporte para mesclagens arbitrárias de DPI e modos de dimensionamento de fonte. Embora você possa

dimensionar um controle de usuário usando um modo (por exemplo, DPI) e colocá-lo em um formulário usando outro modo (fonte) sem problemas, mas misturar um formulário base em um modo e um formulário derivado em outro pode levar a resultados inesperados.

Dimensionamento automático em ação

Windows Forms usa a seguinte lógica para dimensionar automaticamente os formulários e seus conteúdos:

1. No momento do design, cada um [ContainerControl](#) registra o modo de dimensionamento e a resolução atual no [AutoScaleMode](#) e [AutoScaleDimensions](#) , respectivamente.
2. Em tempo de execução, a resolução real é armazenada na [CurrentAutoScaleDimensions](#) propriedade. A [AutoScaleFactor](#) Propriedade calcula dinamicamente a taxa entre a resolução de escala em tempo de execução e de design.
3. Quando o formulário for carregado, se os valores de [CurrentAutoScaleDimensions](#) e [AutoScaleDimensions](#) forem diferentes, o [PerformAutoScale](#) método será chamado para dimensionar o controle e seus filhos. Esse método suspende o layout e chama o [Scale](#) método para executar o dimensionamento real. Posteriormente, o valor de [AutoScaleDimensions](#) é atualizado para evitar o dimensionamento progressivo.
4. [PerformAutoScale](#) também é invocado automaticamente nas seguintes situações:
 - Em resposta ao [OnFontChanged](#) evento, se o modo de dimensionamento for [Font](#) .
 - Quando o layout do controle de contêiner é retomado e uma alteração é detectada nas [AutoScaleDimensions](#) [AutoScaleMode](#) Propriedades ou.
 - Como implícito acima, quando um pai [ContainerControl](#) está sendo dimensionado. Cada controle de contêiner é responsável por dimensionar seus filhos usando seus próprios fatores de dimensionamento e não aquele de seu contêiner pai.
5. Os controles filho podem modificar seu comportamento de dimensionamento por vários meios:
 - A [ScaleChildren](#) propriedade pode ser substituída para determinar se os controles filho devem ser dimensionados ou não.
 - O [GetScaledBounds](#) método pode ser substituído para ajustar os limites para os quais o controle é dimensionado, mas não a lógica de dimensionamento.
 - O [ScaleControl](#) método pode ser substituído para alterar a lógica de dimensionamento do controle atual.

Confira também

- [AutoScaleMode](#)
- [Scale](#)
- [PerformAutoScale](#)
- [AutoScaleDimensions](#)

Como adicionar um formulário a um projeto (Windows Forms .NET)

14/05/2021 • 2 minutes to read

Adicione formulários ao seu projeto com o Visual Studio. Quando seu aplicativo tem vários formulários, você pode escolher qual é o formulário de inicialização para seu aplicativo e pode exibir vários formulários ao mesmo tempo.

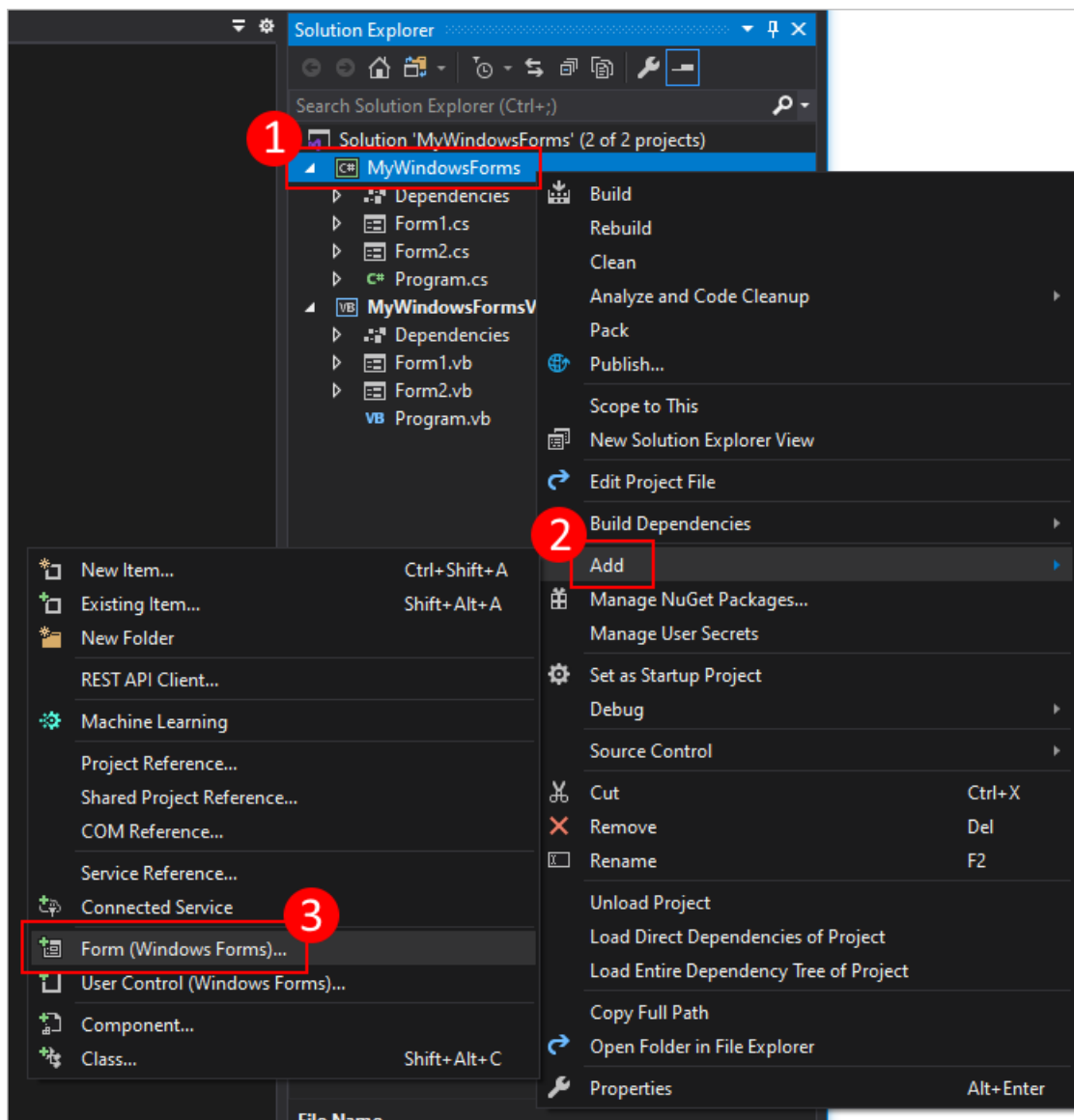
IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

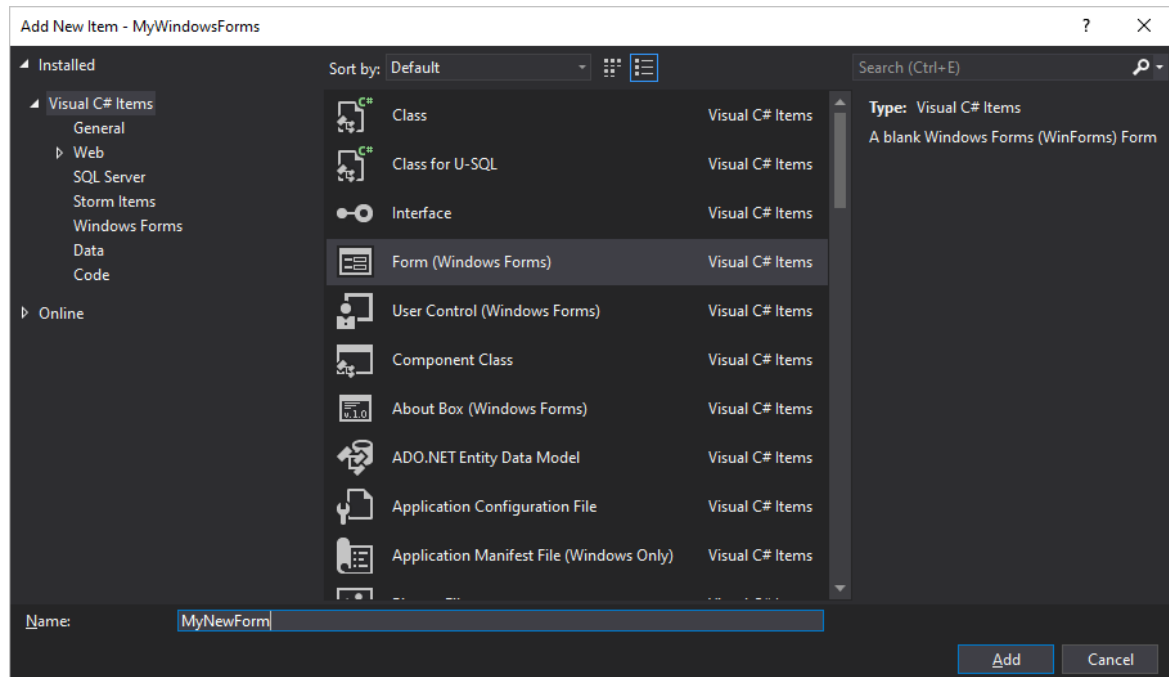
Adicionar um novo formulário

Adicione um novo formulário com o Visual Studio.

1. No Visual Studio, localize o painel **Explorador de projeto**. Clique com o botão direito do mouse no projeto e escolha **Adicionar > formulário (Windows Forms)**.



2. Na caixa **nome** , digite um nome para o formulário, como *MyNewForm*. O Visual Studio fornecerá um nome padrão e exclusivo que você pode usar.



Depois que o formulário tiver sido adicionado, o Visual Studio abrirá o designer de formulário para o formulário.

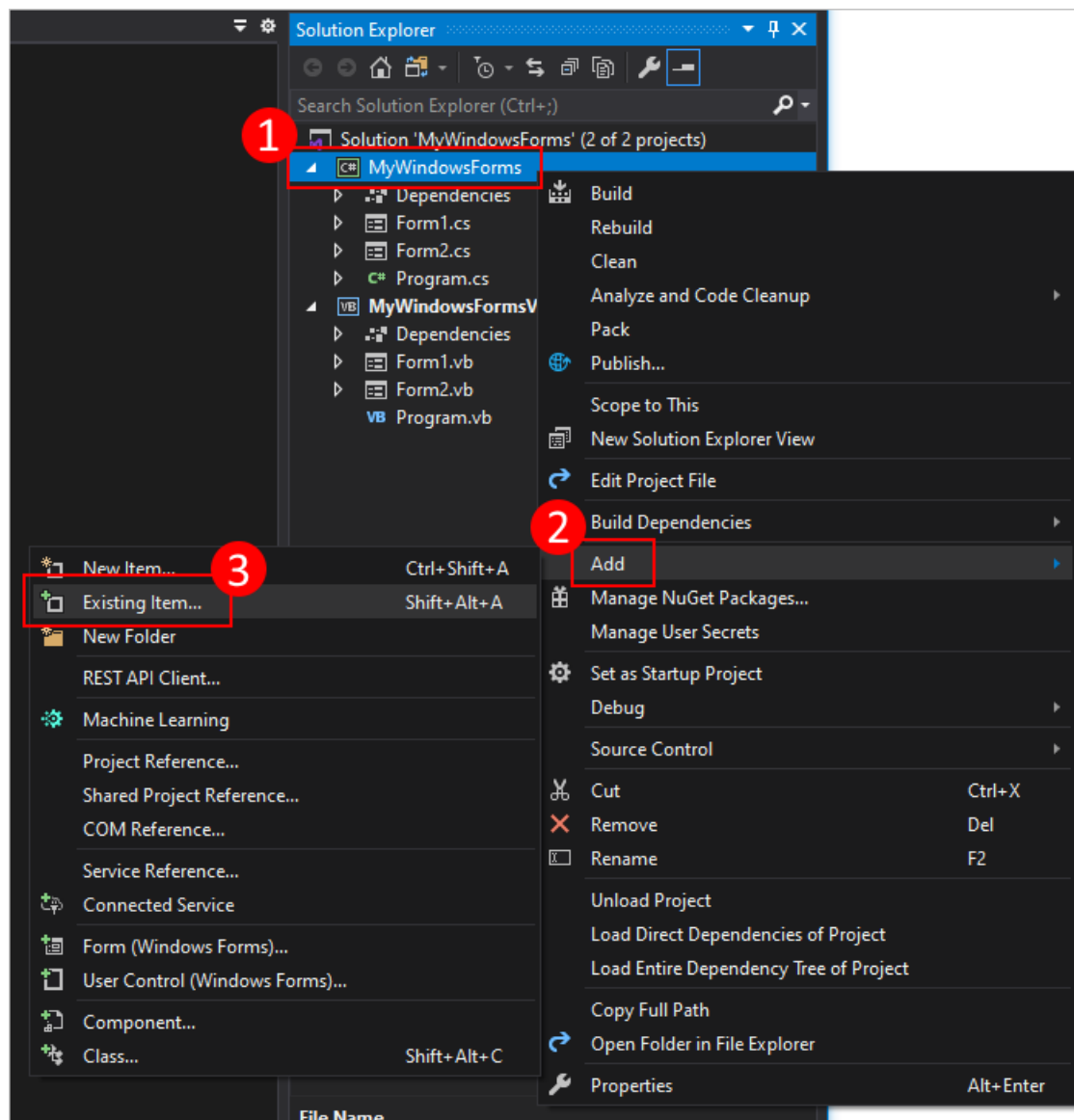
Adicionar uma referência de projeto a um formulário

Se você tiver os arquivos de origem para um formulário, poderá adicionar o formulário ao seu projeto copiando os arquivos para a mesma pasta que o seu projeto. O projeto faz referência automaticamente a todos os arquivos de código que estão na mesma pasta ou pasta filho do seu projeto.

Os formulários são compostos de dois arquivos que compartilham o mesmo nome: *Form2.cs* (*Form2* sendo um exemplo de um nome de arquivo) e *Form2.Designer.cs*. Às vezes, um arquivo de recurso existe, compartilhando o mesmo nome, *Form2.resx*. No exemplo anterior, *Form2* representa o nome do arquivo base. Você desejará copiar todos os arquivos relacionados para a pasta do projeto.

Como alternativa, você pode usar o Visual Studio para importar um arquivo para o seu projeto. Quando você adiciona um arquivo existente ao seu projeto, o arquivo é copiado para a mesma pasta que o seu projeto.

1. No Visual Studio, localize o painel **Explorador de projeto** . Clique com o botão direito do mouse no projeto e escolha **Adicionar > Item existente**.



2. Navegue até a pasta que contém os arquivos de formulário.
3. Selecione o arquivo *Form2.cs*, em que *Form2* é o nome de arquivo base dos arquivos de formulário relacionados. Não selecione os outros arquivos, como *Form2.Designer.cs*.

Confira também

- [Como posicionar e dimensionar um formulário \(Windows Forms .NET\)](#)
- [Visão geral de eventos \(Windows Forms .NET\)](#)
- [Posição e layout de controles \(Windows Forms .NET\)](#)

Como posicionar e dimensionar um formulário (Windows Forms .NET)

14/05/2021 • 4 minutes to read

Quando um formulário é criado, o tamanho e o local são inicialmente definidos como um valor padrão. O tamanho padrão de um formulário geralmente é uma largura e altura de *800x500* pixels. O local inicial, quando o formulário é exibido, depende de algumas configurações diferentes.

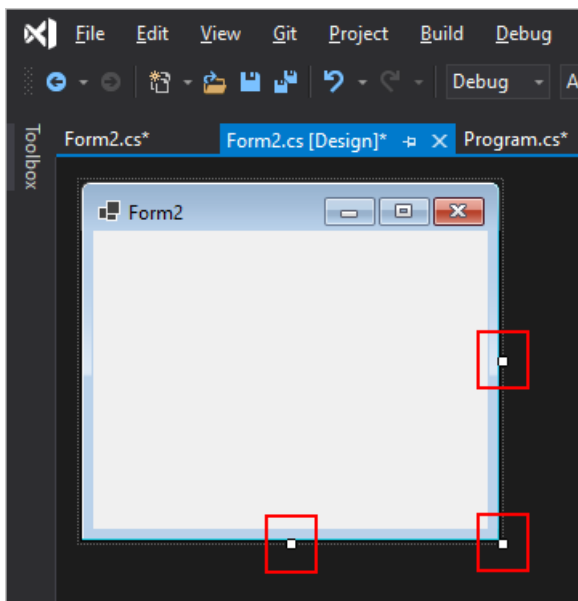
Você pode alterar o tamanho de um formulário em tempo de design com o Visual Studio e em tempo de execução com o código.

IMPORTANT

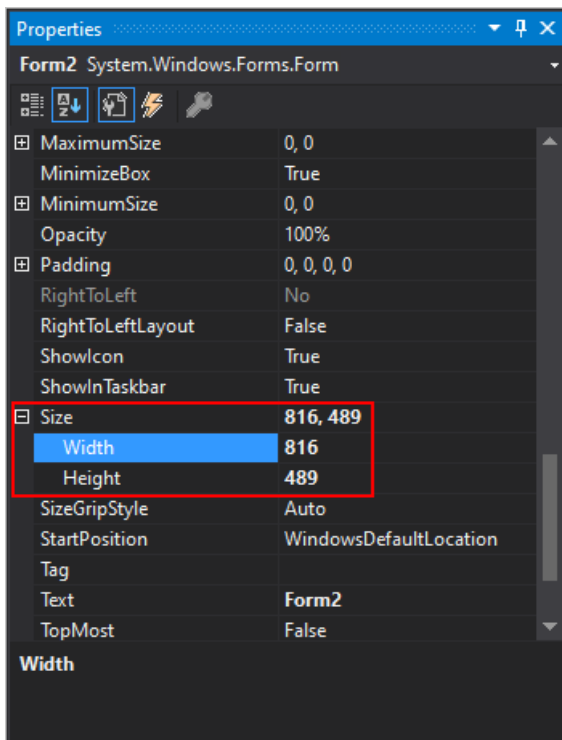
A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Redimensionar com o designer

Depois de [Adicionar um novo formulário](#) ao projeto, o tamanho de um formulário é definido de duas maneiras diferentes. Primeiro, você pode defini-lo com o tamanho das alças no designer. Arrastando a borda direita, a borda inferior ou o canto, você pode redimensionar o formulário.



A segunda maneira de redimensionar o formulário enquanto o designer está aberto é por meio do painel Propriedades. Selecione o formulário e, em seguida, localize o painel **Propriedades** no Visual Studio. Role para baixo até **tamanho** e expanda-o. Você pode definir a **largura** e a **altura** manualmente.



Redimensionar no código

Embora o designer defina o tamanho inicial de um formulário, você pode redimensioná-lo por meio de código. O uso de código para redimensionar um formulário é útil quando algo sobre seu aplicativo determina que o tamanho padrão do formulário é insuficiente.

Para redimensionar um formulário, altere o [Size](#), que representa a largura e a altura do formulário.

Redimensionar o formulário atual

Você pode alterar o tamanho do formulário atual, desde que o código esteja sendo executado dentro do contexto do formulário. Por exemplo, se você tiver `Form1` um botão, que quando clicado invoca o `Click` manipulador de eventos para redimensionar o formulário:

```
private void button1_Click(object sender, EventArgs e) =>
    Size = new Size(250, 200);
```

```
Private Sub Button1_Click(sender As Object, e As EventArgs)
    Size = New Drawing.Size(250, 200)
End Sub
```

Redimensionar um formulário diferente

Você pode alterar o tamanho de outro formulário depois que ele é criado usando a variável que faz referência ao formulário. Por exemplo, digamos que você tenha duas formas, `Form1` (o formulário de inicialização neste exemplo) e `Form2`. `Form1` tem um botão que, quando clicado, invoca o `Click` evento. O manipulador desse evento cria uma nova instância do `Form2` formulário, define o tamanho e, em seguida, exibe:

```
private void button1_Click(object sender, EventArgs e)
{
    Form2 form = new Form2();
    form.Size = new Size(250, 200);
    form.Show();
}
```

```

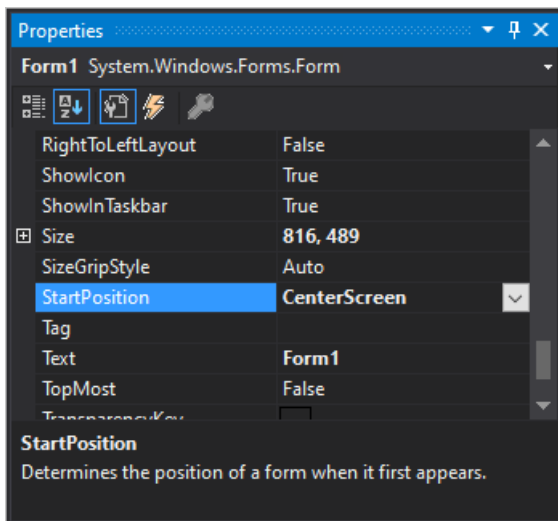
Private Sub Button1_Click(sender As Object, e As EventArgs)
    Dim form = New Form2 With {
        .Size = New Drawing.Size(250, 200)
    }
    form.Show()
End Sub

```

Se o `Size` não estiver definido manualmente, o tamanho padrão do formulário será o que foi definido durante o tempo de design.

Posição com o designer

Quando uma instância de formulário é criada e exibida, o local inicial do formulário é determinado pela `StartPosition` propriedade. A `Location` propriedade contém o local atual do formulário. Ambas as propriedades podem ser definidas por meio do designer.



FORMSTARTPOSITION ENUM	DESCRIÇÃO
CenterParent	O formulário é centralizado dentro dos limites do formulário pai.
CenterScreen	O formulário é centralizado na exibição atual.
Manual	A posição do formulário é determinada pela propriedade <code>Location</code> .
WindowsDefaultBounds	O formulário é posicionado no local padrão do Windows e é redimensionado para o tamanho padrão determinado pelo Windows.
WindowsDefaultLocation	O formulário é posicionado no local padrão do Windows e não é redimensionado.

O valor `CenterParent` só funciona com formulários que são um formulário filho MDI (interface de vários documentos) ou um formulário normal que é exibido com o `ShowDialog` método. `CenterParent` não afeta um formulário normal que é exibido com o `Show` método. Para centralizar um formulário (`form` variável) em outro formulário (`parentForm` variável), use o seguinte código:

```
form.StartPosition = FormStartPosition.Manual;
form.Location = new Point(parentForm.Width / 2 - form.Width / 2 + parentForm.Location.X,
                          parentForm.Height / 2 - form.Height / 2 + parentForm.Location.Y);
form.Show();
```

```
form.StartPosition = Windows.Forms.FormStartPosition.CenterParent.Manual
form.Location = New Drawing.Point(parentForm.Width / 2 - form.Width / 2 + parentForm.Location.X,
                                  parentForm.Height / 2 - form.Height / 2 + parentForm.Location.Y)

form.Show()
```

Posição com código

Embora o designer possa ser usado para definir o local inicial de um formulário, você pode usar o código para alterar o modo de posição inicial ou definir o local manualmente. Usar o código para posicionar um formulário será útil se você precisar posicionar e dimensionar manualmente um formulário em relação à tela ou a outros formulários.

Mover o formulário atual

Você pode mover o formulário atual, desde que o código esteja sendo executado dentro do contexto do formulário. Por exemplo, se você tiver `Form1` um botão, quando clicado invoca o `Click` manipulador de eventos. O manipulador neste exemplo altera o local do formulário para o canto superior esquerdo da tela, definindo a `Location` Propriedade:

```
private void button1_Click(object sender, EventArgs e) =>
    Location = new Point(0, 0);
```

```
Private Sub Button1_Click(sender As Object, e As EventArgs)
    Location = New Drawing.Point(0, 0)
End Sub
```

Posicionar um formulário diferente

Você pode alterar o local de outro formulário depois que ele é criado usando a variável que faz referência ao formulário. Por exemplo, digamos que você tenha duas formas, `Form1` (o formulário de inicialização neste exemplo) e `Form2`. `Form1` tem um botão que, quando clicado, invoca o `Click` evento. O manipulador desse evento cria uma nova instância do `Form2` formulário e define o tamanho:

```
private void button1_Click(object sender, EventArgs e)
{
    Form2 form = new Form2();
    form.Size = new Size(250, 200);
    form.Show();
}
```

```
Private Sub Button1_Click(sender As Object, e As EventArgs)
    Dim form = New Form2 With {
        .Size = New Drawing.Size(250, 200)
    }
    form.Show()
End Sub
```

Se o `Size` não estiver definido, o tamanho padrão do formulário será o que foi definido como em tempo de

design.

Confira também

- [Como adicionar um formulário a um projeto \(Windows Forms .NET\)](#)
- [Visão geral de eventos \(Windows Forms .NET\)](#)
- [Posição e layout de controles \(Windows Forms .NET\)](#)

Como posicionar e dimensionar um formulário (Windows Forms .NET)

14/05/2021 • 4 minutes to read

Quando um formulário é criado, o tamanho e o local são inicialmente definidos como um valor padrão. O tamanho padrão de um formulário geralmente é uma largura e altura de *800x500* pixels. O local inicial, quando o formulário é exibido, depende de algumas configurações diferentes.

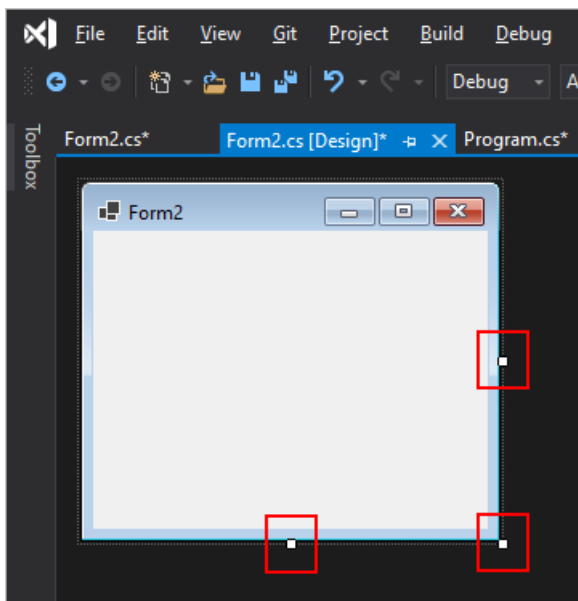
Você pode alterar o tamanho de um formulário em tempo de design com o Visual Studio e em tempo de execução com o código.

IMPORTANT

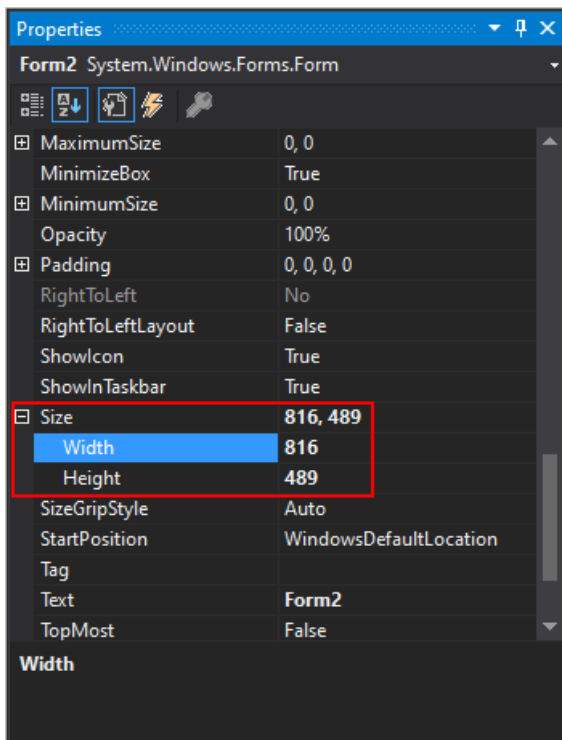
A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Redimensionar com o designer

Depois de [Adicionar um novo formulário](#) ao projeto, o tamanho de um formulário é definido de duas maneiras diferentes. Primeiro, você pode defini-lo com o tamanho das alças no designer. Arrastando a borda direita, a borda inferior ou o canto, você pode redimensionar o formulário.



A segunda maneira de redimensionar o formulário enquanto o designer está aberto é por meio do painel Propriedades. Selecione o formulário e, em seguida, localize o painel **Propriedades** no Visual Studio. Role para baixo até **tamanho** e expanda-o. Você pode definir a **largura** e a **altura** manualmente.



Redimensionar no código

Embora o designer defina o tamanho inicial de um formulário, você pode redimensioná-lo por meio de código. O uso de código para redimensionar um formulário é útil quando algo sobre seu aplicativo determina que o tamanho padrão do formulário é insuficiente.

Para redimensionar um formulário, altere o [Size](#), que representa a largura e a altura do formulário.

Redimensionar o formulário atual

Você pode alterar o tamanho do formulário atual, desde que o código esteja sendo executado dentro do contexto do formulário. Por exemplo, se você tiver `Form1` um botão, que quando clicado invoca o `Click` manipulador de eventos para redimensionar o formulário:

```
private void button1_Click(object sender, EventArgs e) =>
    Size = new Size(250, 200);
```

```
Private Sub Button1_Click(sender As Object, e As EventArgs)
    Size = New Drawing.Size(250, 200)
End Sub
```

Redimensionar um formulário diferente

Você pode alterar o tamanho de outro formulário depois que ele é criado usando a variável que faz referência ao formulário. Por exemplo, digamos que você tenha duas formas, `Form1` (o formulário de inicialização neste exemplo) e `Form2`. `Form1` tem um botão que, quando clicado, invoca o `Click` evento. O manipulador desse evento cria uma nova instância do `Form2` formulário, define o tamanho e, em seguida, exibe:

```
private void button1_Click(object sender, EventArgs e)
{
    Form2 form = new Form2();
    form.Size = new Size(250, 200);
    form.Show();
}
```

```

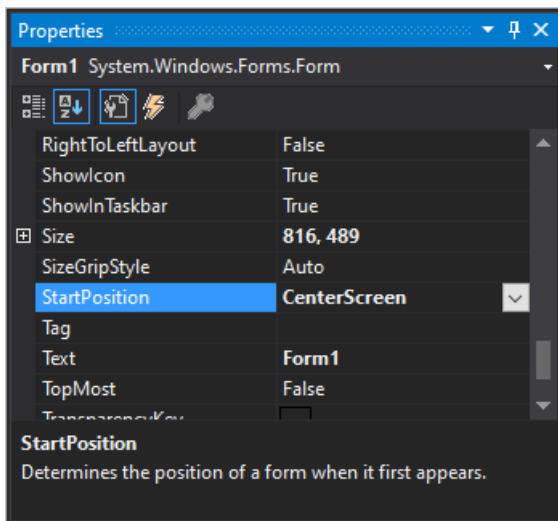
Private Sub Button1_Click(sender As Object, e As EventArgs)
    Dim form = New Form2 With {
        .Size = New Drawing.Size(250, 200)
    }
    form.Show()
End Sub

```

Se o `Size` não estiver definido manualmente, o tamanho padrão do formulário será o que foi definido durante o tempo de design.

Posição com o designer

Quando uma instância de formulário é criada e exibida, o local inicial do formulário é determinado pela `StartPosition` propriedade. A `Location` propriedade contém o local atual do formulário. Ambas as propriedades podem ser definidas por meio do designer.



FORMSTARTPOSITION ENUM	DESCRIÇÃO
CenterParent	O formulário é centralizado dentro dos limites do formulário pai.
CenterScreen	O formulário é centralizado na exibição atual.
Manual	A posição do formulário é determinada pela propriedade <code>Location</code> .
WindowsDefaultBounds	O formulário é posicionado no local padrão do Windows e é redimensionado para o tamanho padrão determinado pelo Windows.
WindowsDefaultLocation	O formulário é posicionado no local padrão do Windows e não é redimensionado.

O valor `CenterParent` só funciona com formulários que são um formulário filho MDI (interface de vários documentos) ou um formulário normal que é exibido com o `ShowDialog` método. `CenterParent` não afeta um formulário normal que é exibido com o `Show` método. Para centralizar um formulário (`form` variável) em outro formulário (`parentForm` variável), use o seguinte código:

```
form.StartPosition = FormStartPosition.Manual;
form.Location = new Point(parentForm.Width / 2 - form.Width / 2 + parentForm.Location.X,
    parentForm.Height / 2 - form.Height / 2 + parentForm.Location.Y);
form.Show();
```

```
form.StartPosition = Windows.Forms.FormStartPosition.CenterParent.Manual
form.Location = New Drawing.Point(parentForm.Width / 2 - form.Width / 2 + parentForm.Location.X,
    parentForm.Height / 2 - form.Height / 2 + parentForm.Location.Y)

form.Show()
```

Posição com código

Embora o designer possa ser usado para definir o local inicial de um formulário, você pode usar o código para alterar o modo de posição inicial ou definir o local manualmente. Usar o código para posicionar um formulário será útil se você precisar posicionar e dimensionar manualmente um formulário em relação à tela ou a outros formulários.

Mover o formulário atual

Você pode mover o formulário atual, desde que o código esteja sendo executado dentro do contexto do formulário. Por exemplo, se você tiver `Form1` um botão, quando clicado invoca o `Click` manipulador de eventos. O manipulador neste exemplo altera o local do formulário para o canto superior esquerdo da tela, definindo a `Location` Propriedade:

```
private void button1_Click(object sender, EventArgs e) =>
    Location = new Point(0, 0);
```

```
Private Sub Button1_Click(sender As Object, e As EventArgs)
    Location = New Drawing.Point(0, 0)
End Sub
```

Posicionar um formulário diferente

Você pode alterar o local de outro formulário depois que ele é criado usando a variável que faz referência ao formulário. Por exemplo, digamos que você tenha duas formas, `Form1` (o formulário de inicialização neste exemplo) e `Form2`. `Form1` tem um botão que, quando clicado, invoca o `Click` evento. O manipulador desse evento cria uma nova instância do `Form2` formulário e define o tamanho:

```
private void button1_Click(object sender, EventArgs e)
{
    Form2 form = new Form2();
    form.Size = new Size(250, 200);
    form.Show();
}
```

```
Private Sub Button1_Click(sender As Object, e As EventArgs)
    Dim form = New Form2 With {
        .Size = New Drawing.Size(250, 200)
    }
    form.Show()
End Sub
```

Se o `Size` não estiver definido, o tamanho padrão do formulário será o que foi definido como em tempo de

design.

Confira também

- [Como adicionar um formulário a um projeto \(Windows Forms .NET\)](#)
- [Visão geral de eventos \(Windows Forms .NET\)](#)
- [Posição e layout de controles \(Windows Forms .NET\)](#)

Visão geral do uso de controles (Windows Forms .NET)

27/05/2021 • 2 minutes to read

Windows Forms controles são componentes reutilizáveis que encapsulam a funcionalidade da interface do usuário e são usados em aplicativos do lado do cliente baseados no Windows. O Windows Forms não só fornece vários controles prontos para usar como também proporciona a infraestrutura para desenvolver seus próprios controles. É possível combinar os controles existentes, ampliar os controles existentes e fazer seus controles personalizados. Para obter mais informações, consulte [Tipos de controles personalizados](#).

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Adicionando controles

Os controles são adicionados por meio do designer Visual Studio. Com o Designer, você pode colocar, tamanho, alinhar e mover controles. Como alternativa, os controles podem ser adicionados por meio do código. Para obter mais informações, consulte [Adicionar um controle \(Windows Forms\)](#).

Opções de layout

A posição em que um controle aparece em um pai é determinada pelo valor da propriedade em relação à parte superior [Location](#) esquerda da superfície pai. A coordenada de posição superior esquerda no pai é `(x0,y0)`. O tamanho do controle é determinado pela [Size](#) propriedade e representa a largura e a altura do controle.

Além do posicionamento e do tamanho manuais, vários controles de contêiner são fornecidos que ajudam com o posicionamento automático de controles.

Para obter mais informações, consulte [Posição e layout de controles](#).

Confira também

- [Posição e layout de controles](#)
- [Visão geral do controle de rótulo](#)
- [Adicionar um controle para um formulário](#)

Posição e layout de controles (Windows Forms .NET)

04/06/2021 • 11 minutes to read

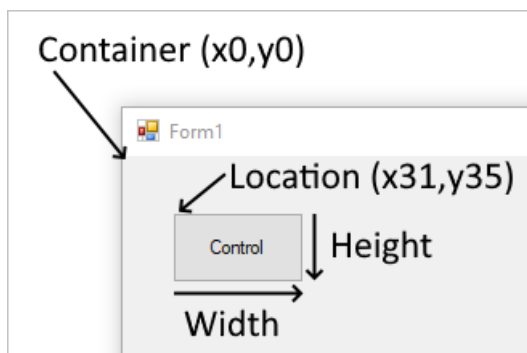
O posicionamento de controle no Windows Forms é determinado não apenas pelo controle, mas também pelo pai do controle. Este artigo descreve as diferentes configurações fornecidas pelos controles e os diferentes tipos de contêineres pai que afetam o layout.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Posição e tamanho fixos

A posição que um controle aparece em um pai é determinada pelo valor da [Location](#) propriedade em relação à parte superior esquerda da superfície pai. A coordenada da posição superior esquerda no pai é $(x0, y0)$. O tamanho do controle é determinado pela [Size](#) propriedade e representa a largura e a altura do controle.



Quando um controle é adicionado a um pai que impõe o posicionamento automático, a posição e o tamanho do controle são alterados. Nesse caso, a posição e o tamanho do controle podem não ser ajustados manualmente, dependendo do tipo de pai.

As [MaximumSize](#) [MinimumSize](#) Propriedades e ajudam a definir o espaço mínimo e máximo que um controle pode usar.

Posicionamento e tamanho automáticos

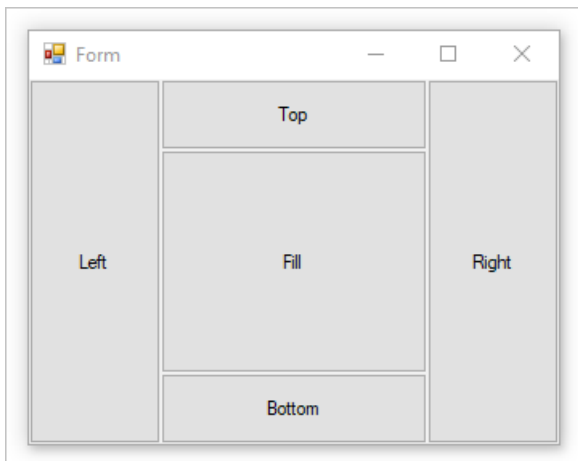
Os controles podem ser colocados automaticamente em seu pai. Alguns contêineres pai forçam o posicionamento enquanto outros respeitam as configurações de controle que orientam o posicionamento. Há duas propriedades em um controle que ajudam o posicionamento e o tamanho automáticos dentro de um pai: [Dock](#) e [Anchor](#).

A ordem de desenho pode afetar o posicionamento automático. A ordem na qual um controle é desenhado determinado pelo índice do controle na coleção do pai [Controls](#). Esse índice é conhecido como o **Z-order**. Cada controle é desenhado na ordem inversa que aparecem na coleção. Ou seja, a coleção é uma coleção first-in-Last-desenhada e Last-in-first-desenhada.

As [MinimumSize](#) [MaximumSize](#) Propriedades e ajudam a definir o espaço mínimo e máximo que um controle pode usar.

Dock

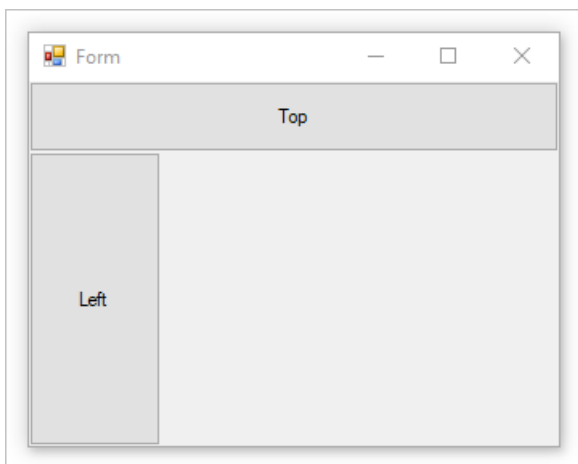
A `Dock` propriedade define qual borda do controle é alinhada ao lado correspondente do pai e como o controle é redimensionado dentro do pai.



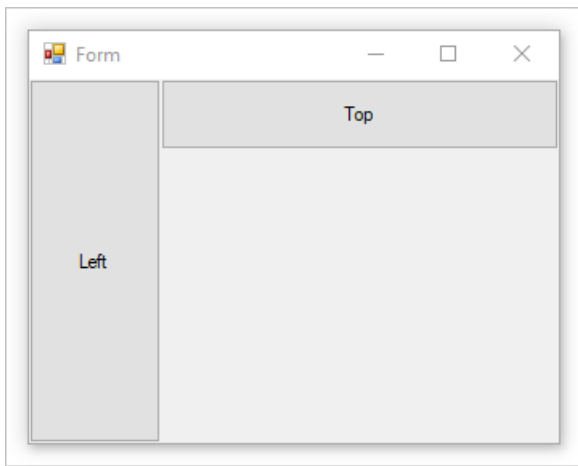
Quando um controle é encaixado, o contêiner determina o espaço que ele deve ocupar e redimensiona e coloca o controle. A largura e a altura do controle ainda são respeitadas com base no estilo de encaixe. Por exemplo, se o controle for encaixado na parte superior, o `Height` do controle será respeitado, mas o `Width` será ajustado automaticamente. Se um controle for encaixado à esquerda, o `Width` do controle será respeitado, mas o `Height` será ajustado automaticamente.

O `Location` do controle não pode ser definido manualmente, pois o encaixe de um controle controla automaticamente sua posição.

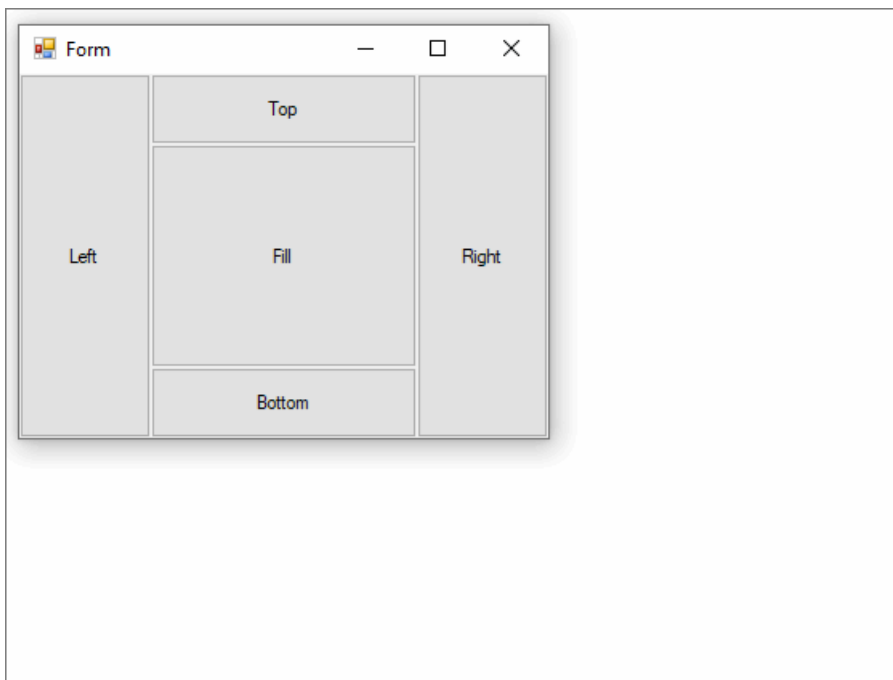
O **Z-order** do controle afeta o encaixe. À medida que os controles encaixados são dispostos, eles usam o espaço disponível para eles. Por exemplo, se um controle for desenhado primeiro e encaixado na parte superior, ele ocupará toda a largura do contêiner. Se um próximo controle for encaixado à esquerda, ele terá menos espaço vertical disponível para ele.



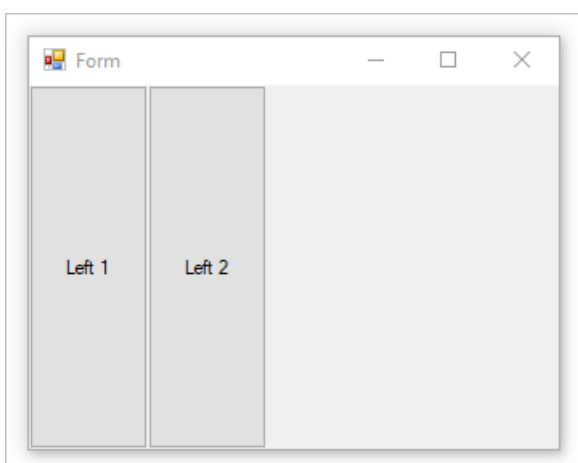
Se o controle **Z-order** for invertido, o controle que está encaixado à esquerda agora tem mais espaço inicial disponível. O controle usa toda a altura do contêiner. O controle encaixado na parte superior tem menos espaço horizontal disponível para ele.



À medida que o contêiner aumenta e diminui, os controles encaixados no contêiner são reposicionados e redimensionados para manter suas posições e tamanhos aplicáveis.



Se vários controles estiverem encaixados no mesmo lado do contêiner, eles serão empilhados de acordo com seus **Z-order**.

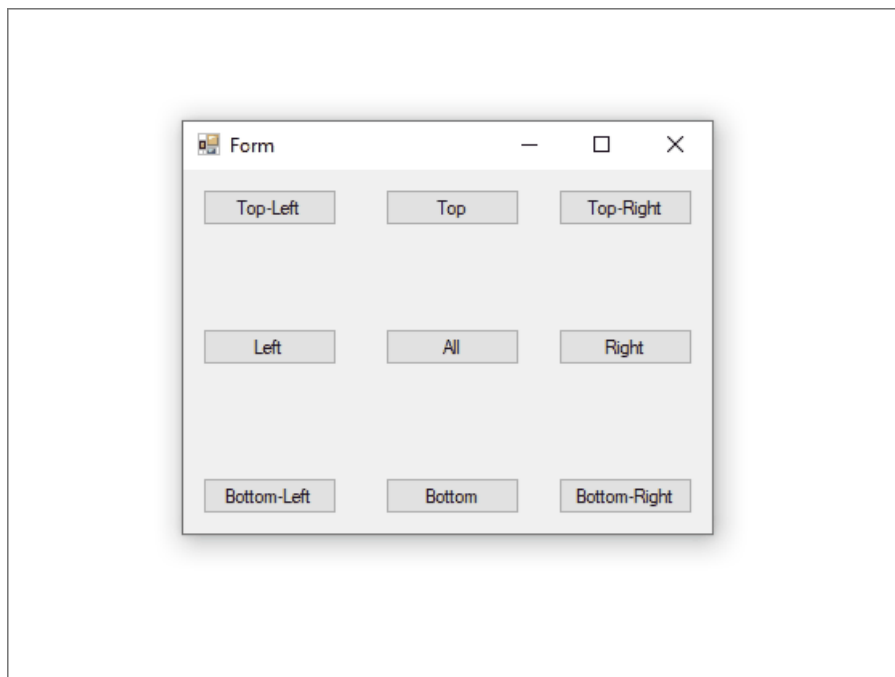


Âncora

A ancoragem de um controle permite que você vincule o controle a um ou mais lados do contêiner pai. À medida que o contêiner muda de tamanho, qualquer controle filho manterá sua distância com o lado ancorado.

Um controle pode ser ancorado a um ou mais lados, sem restrição. A âncora é definida com a [Anchor](#)

propriedade.



Dimensionamento automático

A [AutoSize](#) propriedade permite que um controle altere seu tamanho, se necessário, para se ajustar ao tamanho especificado pela [PreferredSize](#) propriedade. Você ajusta o comportamento de dimensionamento para controles específicos configurando a propriedade `AutoSizeMode`.

Somente alguns controles dão suporte à [AutoSize](#) propriedade. Além disso, alguns controles que oferecem suporte à [AutoSize](#) propriedade também dão suporte à `AutoSizeMode` propriedade.

COMPORTAMENTO SEMPRE VERDADEIRO	DESCRIÇÃO
O dimensionamento automático é um recurso de tempo de execução.	Isso significa que ele nunca aumenta ou diminui um controle e, em seguida, não tem mais efeito.
Se um controle mudar de tamanho, o valor de sua Location propriedade sempre permanecerá constante.	Quando o conteúdo de um controle faz com que ele cresça, o controle aumenta para a direita e para baixo. Controles não crescem para a esquerda.
As Dock Anchor Propriedades e são respeitadas quando AutoSize é <code>true</code> .	<p>O valor da Propriedade do controle Location é ajustado para o valor correto.</p> <p>O Label controle é a exceção a essa regra. Quando você define o valor de uma propriedade de controle encaixado Label AutoSize como <code>true</code>, o Label controle não será ampliado.</p>
MaximumSize As propriedades e de um controle MinimumSize são sempre respeitadas, independentemente do valor de sua AutoSize propriedade.	As MaximumSize MinimumSize Propriedades e não são afetadas pela AutoSize propriedade.
Não há um tamanho mínimo definido por padrão.	Isso significa que, se um controle for definido para reduzir em AutoSize e não tiver conteúdo, o valor de sua Size propriedade será <code>(0x,0y)</code> . Nesse caso, o controle será reduzido a um ponto e não ficará visível imediatamente.

COMPORTAMENTO SEMPRE VERDADEIRO	DESCRIÇÃO
Se um controle não implementar o GetPreferredSize método, o GetPreferredSize método retornará o último valor atribuído à Size propriedade.	Isso significa que AutoSize a configuração para <code>true</code> não terá nenhum efeito.
Um controle em uma TableLayoutPanel célula sempre é reduzido para caber na célula até que seu MinimumSize seja atingido.	Esse tamanho é imposto como um tamanho máximo. Esse não é o caso quando a célula faz parte de uma AutoSize linha ou coluna.

Propriedade [AutoSizeMode](#)

A [AutoSizeMode](#) propriedade fornece um controle mais refinado sobre o [AutoSize](#) comportamento padrão. A propriedade `AutoSizeMode` especifica como um controle dimensiona a si mesmo de acordo com seu conteúdo. O conteúdo, por exemplo, pode ser o texto para um [Button](#) controle ou os controles filho de um contêiner.

A lista a seguir mostra os `AutoSizeMode` valores e seu comportamento.

- [AutoSizeMode.GrowAndShrink](#)

O controle aumenta ou diminui para conter o conteúdo.

Os [MinimumSize](#) [MaximumSize](#) valores e são respeitados, mas o valor atual da [Size](#) propriedade é ignorado.

Esse é o mesmo comportamento que os controles com a [AutoSize](#) propriedade e nenhuma `AutoSizeMode` propriedade.

- [AutoSizeMode.GrowOnly](#)

O controle cresce tanto quanto necessário para abranger seu conteúdo, mas não reduzirá a menor do que o valor especificado por sua [Size](#) propriedade.

Esse é o valor padrão de `AutoSizeMode`.

Controles que dão suporte à propriedade [AutoSize](#)

A tabela a seguir descreve o nível de suporte de dimensionamento automático por controle:

CONTROL	<code>AUTOSIZE</code> PORTA	<code>AUTOSIZEMODE</code> PORTA
Button	✓	✓
CheckedListBox	✓	✓
FlowLayoutPanel	✓	✓
Form	✓	✓
GroupBox	✓	✓
Panel	✓	✓
TableLayoutPanel	✓	✓
CheckBox	✓	✗
DomainUpDown	✓	✗

CONTROL	AUTOSIZE PORTA	AUTOSIZEMODE PORTA
Label	✓	✗
LinkLabel	✓	✗
MaskedTextBox	✓	✗
NumericUpDown	✓	✗
RadioButton	✓	✗
TextBox	✓	✗
TrackBar	✓	✗
CheckedListBox	✗	✗
ComboBox	✗	✗
DataGridView	✗	✗
DateTimePicker	✗	✗
ListBox	✗	✗
ListView	✗	✗
MaskedTextBox	✗	✗
MonthCalendar	✗	✗
ProgressBar	✗	✗
PropertyGrid	✗	✗
RichTextBox	✗	✗
SplitContainer	✗	✗
TabControl	✗	✗
TabPage	✗	✗
TreeView	✗	✗
WebBrowser	✗	✗
ScrollBar	✗	✗

AutoSize no ambiente de design

A tabela a seguir descreve o comportamento deizing de um controle em tempo de design, com base no valor de suas propriedades [AutoSize](#) e `AutoSizeMode`.

Substitua a propriedade para determinar se um determinado controle está em um estado [SelectionRules](#) redimensionável pelo usuário. Na tabela a seguir, "não é possível redimensionar" significa [Moveable](#) apenas " pode redimensionar" significa [AllSizeable](#) e [Moveable](#) .

CONFIGURAÇÃO DE AUTOSIZE	CONFIGURAÇÃO DE AUTOSIZEMODE	COMPORTAMENTO
true	Propriedade não disponível.	O usuário não pode reorganizar o controle em tempo de design, exceto para os seguintes controles: - TextBox - MaskedTextBox - RichTextBox - TrackBar
true	GrowAndShrink	O usuário não pode reorganizar o controle em tempo de design.
true	GrowOnly	O usuário pode redimensionar o controle em tempo de design. Quando a Size propriedade é definida, o usuário só pode aumentar o tamanho do controle.
false ou AutoSize está oculto	Não aplicável.	O usuário pode redimensionar o controle em tempo de design.

NOTE

Para maximizar a produtividade, o Designer de Formulários do Windows em Visual Studio sombreada [AutoSize](#) da propriedade para a classe [Form](#) . Em tempo de design, o formulário se comporta como se a [AutoSize](#) propriedade fosse definida como , [false](#) independentemente de sua configuração real. Em runtime, nenhuma acomodação especial é feita e a [AutoSize](#) propriedade é aplicada conforme especificado pela configuração de propriedade.

Contêiner: Formulário

O [Form](#) é o objeto principal de Windows Forms. Um Windows Forms aplicativo geralmente terá um formulário exibido em todos os momentos. Os formulários contêm controles e [Location](#) respeitam as [Size](#) propriedades e do controle para posicionamento manual. Os formulários também respondem à propriedade [Dock](#) para posicionamento automático.

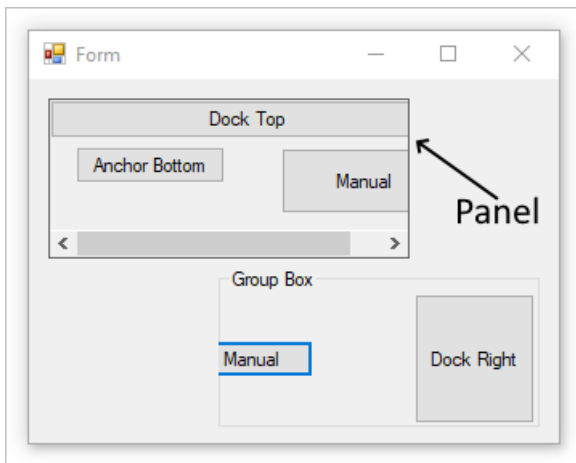
Na maioria das vezes, um formulário terá controles nas bordas que permitem que o usuário redimensione o formulário. A [Anchor](#) propriedade de um controle permitirá que o controle cresça e reduza conforme o formulário é redimensionado.

Contêiner: Painel

O [Panel](#) controle é semelhante a um formulário em que ele simplesmente reúne controles. Ele dá suporte aos mesmos estilos de posicionamento manual e automático que um formulário faz. Para obter mais informações, consulte a [seção Contêiner: Formulário](#).

Um painel é mesclado perfeitamente com o pai e corta qualquer área de um controle que está fora dos limites do painel. Se um controle ficar fora dos limites do painel e estiver definido como , as barras de rolagem aparecerão e o usuário [AutoScroll](#) [true](#) poderá rolar o painel.

Ao contrário do controle de caixa de grupo, um painel não tem uma legenda e uma borda.



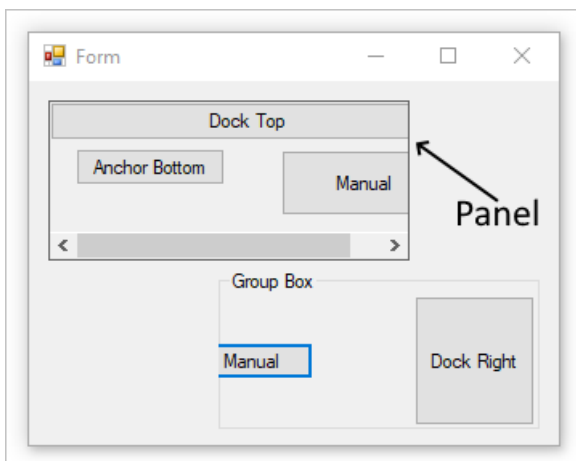
A imagem acima tem um painel com a [BorderStyle](#) propriedade definida para demonstrar os limites do painel.

Contêiner: caixa Grupo

O [GroupBox](#) controle fornece um grupo identificável para outros controles. Normalmente, você usa uma caixa de grupo para subdividir um formulário por função. Por exemplo, você pode ter um formulário que representa informações pessoais e os campos relacionados a um endereço seriam agrupados. Em tempo de design, é fácil mover a caixa de grupo junto com seus controles contidos.

A caixa de grupo dá suporte aos mesmos estilos de posicionamento manual e automático que um formulário faz. Para obter mais informações, consulte a [seção Contêiner: Formulário](#). Uma caixa de grupo também corta qualquer parte de um controle que cai fora dos limites do painel.

Ao contrário [do controle](#) de painel, uma caixa de grupo não tem a capacidade de rolar o conteúdo e exibir barras de rolagem.



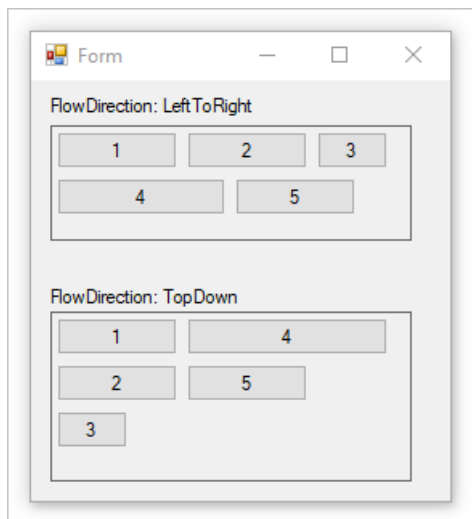
Contêiner: layout de fluxo

O [FlowLayoutPanel](#) controle organiza seu conteúdo em uma direção de fluxo horizontal ou vertical. Você pode encapsular o conteúdo do controle de uma linha para outra ou de uma coluna para a próxima. Como alternativa, você pode recortar, em vez de encapsular seu conteúdo.

Você pode especificar a direção do fluxo definindo o valor da [FlowDirection](#) propriedade. O controle inverte corretamente sua direção de fluxo em [FlowLayoutPanel](#) layouts RTL (da direita para a esquerda). Você também pode especificar se o conteúdo do controle é empacotado ou recortado [FlowLayoutPanel](#) definindo o valor da [WrapContents](#) propriedade.

O [FlowLayoutPanel](#) controle é automaticamente tamanhos para seu conteúdo quando você definir [AutoSize](#) a propriedade como `true`. Ele também fornece uma [FlowBreak](#) propriedade para seus controles filho. Definir o

valor da propriedade como faz com que o controle pare de definir controles na direção do fluxo atual e wrap para a `FlowBreak` `true` próxima linha ou `FlowLayoutPanel` coluna.



A imagem acima tem dois `FlowLayoutPanel` controles com `BorderStyle` a propriedade definida para demonstrar os limites do controle.

Contêiner: layout da tabela

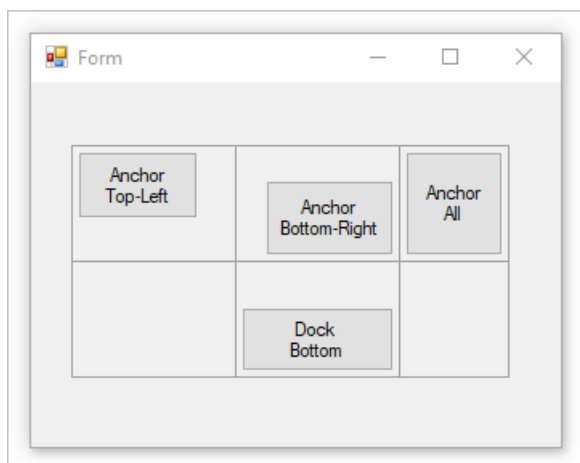
O `TableLayoutPanel` controle organiza seu conteúdo em uma grade. Como o layout é feito em tempo de design e em tempo de executar, ele pode mudar dinamicamente conforme o ambiente do aplicativo muda. Isso dá aos controles no painel a capacidade de reesquilar proporcionalmente, de modo que eles possam responder a alterações, como o resizing do controle pai ou a alteração do comprimento do texto devido à localização.

Qualquer Windows Forms controle pode ser um filho do `TableLayoutPanel` controle, incluindo outras instâncias do `TableLayoutPanel`. Isso permite criar layouts sofisticados que se adaptam às alterações no tempo de execução.

Você também pode controlar a direção da expansão (horizontal ou vertical) depois que o `TableLayoutPanel` controle está cheio de controles filho. Por padrão, o `TableLayoutPanel` controle se expande para baixo adicionando linhas.

Você pode controlar o tamanho e o estilo das linhas e colunas usando as `RowStyles` propriedades `ColumnStyles` e . É possível definir as propriedades de linhas ou colunas individualmente.

O `TableLayoutPanel` controle adiciona as seguintes propriedades aos seus controles filho: `Cell` , , e `Column` `Row` `ColumnSpan` `RowSpan` .

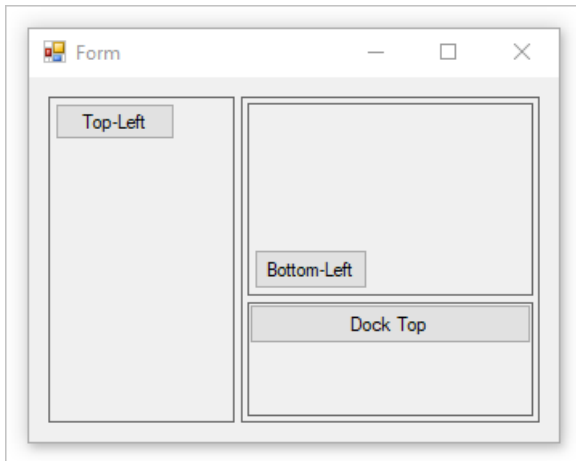


A imagem acima tem uma tabela com a `CellBorderStyle` propriedade definida para demonstrar os limites de cada célula.

Contêiner: Dividir contêiner

O Windows Forms controle pode ser pensado como um controle composto; são dois [SplitContainer](#) painéis separados por uma barra móvel. Quando o ponteiro do mouse passa sobre a barra, ele muda de forma para mostrar que a barra é móvel.

Com o controle, você pode criar interfaces de usuário complexas; geralmente, uma seleção em um painel determina quais objetos são [SplitContainer](#) mostrados no outro painel. Essa disposição é eficaz para exibir e navegar por informações. Ter dois painéis permite agregar as informações em áreas e a barra, ou “divisor”, tornando mais fácil para os usuários redimensionar os painéis.

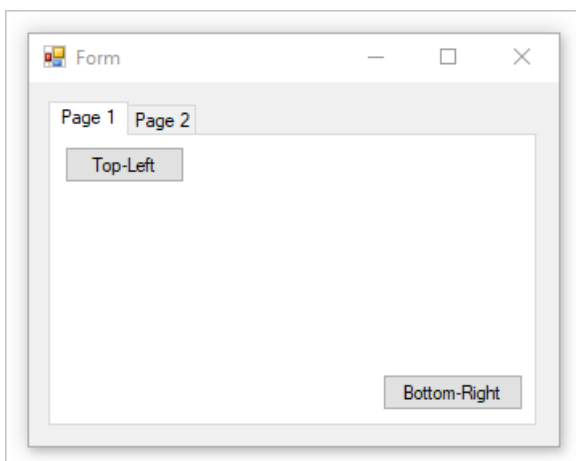


A imagem acima tem um contêiner dividido para criar um painel esquerdo e direito. O painel direito contém um segundo contêiner de divisão com o [Orientation](#) definido como [Vertical](#). A [BorderStyle](#) propriedade é definida para demonstrar os limites de cada painel.

Contêiner: controle tab

O [exibe](#) várias guias, como divisores em um notebook ou rótulos em um conjunto de pastas [TabControl](#) em um gabinete de arquivamento. As guias podem conter imagens e outros controles. Use o controle de tabulação para produzir o tipo de caixa de diálogo de várias páginas que aparece em muitos locais no sistema operacional Windows, como os Painel de Controle e Propriedades de Vídeo. Além disso, pode ser usado para criar páginas de propriedades, que são usadas [TabControl](#) para definir um grupo de propriedades relacionadas.

A propriedade mais importante do [TabControl](#) é, que contém as guias [TabPage](#) individuais. Cada guia individual é um [TabPage](#) objeto.



Visão geral do controle de rótulo (Windows Forms .NET)

14/05/2021 • 2 minutes to read

Windows Forms [Label](#) controles são usados para exibir texto que não pode ser editado pelo usuário. Eles são usados para identificar objetos em um formulário e fornecer uma descrição do que determinado controle representa ou faz. Por exemplo, você pode usar rótulos para adicionar legendas descritivas em caixas de texto, caixas de listagem, caixas de combinação e assim por diante. Também é possível escrever código para alterar o texto exibido por um rótulo em resposta a eventos em tempo de execução.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Trabalhando com o controle de rótulo

Como o [Label](#) controle não pode receber o foco, ele pode ser usado para criar chaves de acesso para outros controles. Uma chave de acesso permite que um usuário focalize o próximo controle na ordem de tabulação pressionando a tecla ALT com a tecla de acesso escolhida. Para obter mais informações, consulte [usar um rótulo para focalizar um controle](#).

A legenda exibida no rótulo está contida na [Text](#) propriedade. A [TextAlign](#) propriedade permite que você defina o alinhamento do texto dentro do rótulo. Para obter mais informações, consulte [Como definir o texto exibido por um controle dos Windows Forms](#).

Confira também

- [Usar um rótulo para focalizar um controle \(Windows Forms .NET\)](#)
- [Como definir o texto exibido por um controle \(Windows Forms .NET\)](#)
- [AutoScaleMode](#)
- [Scale](#)
- [PerformAutoScale](#)
- [AutoScaleDimensions](#)

Tipos de controles personalizados (Windows Forms .NET)

14/05/2021 • 5 minutes to read

Com Windows Forms, você pode desenvolver e implementar novos controles. Você pode criar um novo controle de usuário, modificar os controles existentes por meio de herança e escrever um controle personalizado que faça sua própria pintura.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Decidir qual tipo de controle criar pode ser confuso. Este artigo destaca as diferenças entre os vários tipos de controles dos quais você pode herdar e fornece informações sobre como escolher um tipo específico de controle para seu projeto.

SE...	CRIAR UM...
<ul style="list-style-type: none">• Você deseja combinar a funcionalidade de vários controles dos Windows Forms em uma única unidade reutilizável.	Controle composto herdando de System. Windows. Forms. UserControl .
<ul style="list-style-type: none">• A maioria da funcionalidade que você precisa já é idêntica a um controle Windows Forms existente.• Você não precisa de uma interface gráfica do usuário personalizada ou deseja criar uma nova interface gráfica do usuário para um controle existente.	Controle estendido herdando de um controle de Windows Forms específico.
<ul style="list-style-type: none">• Você deseja fornecer uma representação gráfica personalizada do seu controle.• Você precisa implementar uma funcionalidade personalizada que não esteja disponível por meio de controles padrão.	Controle personalizado herdando de System. Windows. Forms. Control .

Classe de controle base

A [Control](#) classe é a classe base para controles de Windows Forms. Ele fornece a infraestrutura necessária para a exibição Visual em aplicativos Windows Forms e fornece os seguintes recursos:

- Expõe um identificador de janela.
- Gerencia o roteamento de mensagens.
- Fornece eventos de teclado e mouse, além de muitos outros eventos da interface do usuário.
- Fornece recursos de layout avançados.
- Contém muitas propriedades específicas para a exibição Visual, como [ForeColor](#) , [BackColor](#) , [Height](#) e [Width](#) .
- Fornece a segurança e o suporte a threading necessários para um controle dos Windows Forms atuar como um controle do Microsoft® ActiveX®.

Como grande parte da infraestrutura é fornecida pela classe base, é relativamente fácil desenvolver seus próprios controles de Windows Forms.

Controles de composição

Um controle de composição é uma coleção de controles dos Windows Forms encapsulados em um contêiner comum. Esse tipo de controle é, às vezes, chamado de *controle de usuário*. Os controles contidos são chamados *controles constituintes*.

Um controle de composição contém todas as funcionalidades inerentes associadas a cada um dos controles dos Windows Forms contidos e permite que você exponha seletivamente e associe suas propriedades. Um controle de composição também é ideal para a funcionalidade de manipulação do teclado padrão sem nenhum esforço adicional de desenvolvimento de sua parte.

Por exemplo, um controle de composição poderia ser criado para exibir dados de endereço de cliente de um banco de dados. Esse controle incluiria um [DataGridView](#) controle para exibir os campos do banco de [BindingSource](#) dados, um para tratar a associação a uma fonte de dado e um [BindingNavigator](#) controle para percorrer os registros. Você pode expor seletivamente propriedades de vinculação de dados, além de poder empacotar e reutilizar todo o controle do aplicativo para o aplicativo.

Para criar um controle composto, derive da [UserControl](#) classe. A [UserControl](#) classe base fornece roteamento de teclado para controles filho e permite que controles filho funcionem como um grupo.

Controles estendidos

Você pode derivar um controle herdado de qualquer controle Windows Forms existente. Com essa abordagem, você pode manter toda a funcionalidade inerente de um controle de Windows Forms e, em seguida, estender essa funcionalidade adicionando propriedades personalizadas, métodos ou outros recursos. Com essa opção, você pode substituir a lógica de pintura do controle base e estender a interface do usuário alterando sua aparência.

Por exemplo, você pode criar um controle derivado do [Button](#) controle que controla quantas vezes um usuário clicou nele.

Em alguns controles, você também pode adicionar uma aparência personalizada à interface gráfica do usuário do seu controle, substituindo o [OnPaint](#) método da classe base. Para um botão estendido que controla cliques, você pode substituir o [OnPaint](#) método para chamar a implementação base de [OnPaint](#) e, em seguida, desenhar a contagem de cliques em um canto da [Button](#) área do cliente do controle.

Controles personalizados

Outra maneira de criar um controle é criar um substancialmente desde o início, herdando de [Control](#). A [Control](#) classe fornece toda a funcionalidade básica exigida pelos controles, incluindo eventos de manipulação de mouse e teclado, mas nenhuma funcionalidade específica de controle ou interface gráfica.

A criação de um controle por herança da [Control](#) classe requer muito mais pensamento e esforço do que a herança de [UserControl](#) um controle de Windows Forms existente. Como uma grande quantidade de implementação é deixada para você, o controle pode ter maior flexibilidade do que um controle de composição ou estendido e você pode personalizar o controle para atender às suas necessidades.

Para implementar um controle personalizado, você deve escrever código para o [OnPaint](#) evento do controle, bem como qualquer código específico de recurso que você precise. Você também pode substituir o [WndProc](#) método e tratar as mensagens do Windows diretamente. Essa é a maneira mais eficiente para criar um controle, mas, para usar essa técnica com eficiência, você precisa estar familiarizado com a API do Win32® da Microsoft.

Um exemplo de um controle personalizado é um controle de relógio que duplica a aparência e o

comportamento de um relógio analógico. A pintura personalizada é chamada para fazer com que as mãos do relógio sejam movidas em resposta a [Tick](#) eventos de um [Timer](#) componente interno.

Controles ActiveX

Embora a infraestrutura dos Windows Forms tenha sido otimizada para hospedar controles dos Windows Forms, você ainda poderá usar controles ActiveX. Há suporte para essa tarefa no Visual Studio.

Controles sem janelas

As tecnologias Microsoft Visual Basic® 6,0 e ActiveX oferecem suporte a controles *sem janelas*. Não há suporte para controles sem janela no Windows Forms.

Experiência de design personalizado

Se você precisar implementar uma experiência de tempo de design personalizada, você poderá criar seu próprio designer. Para controles de composição, derive sua classe de designer personalizada das [ParentControlDesigner](#) [DocumentDesigner](#) classes ou. Para controles estendidos e personalizados, derive sua classe de designer personalizada da [ControlDesigner](#) classe.

Use o [DesignerAttribute](#) para associar seu controle com seu designer.

As informações a seguir estão desatualizadas, mas podem ajudá-lo.

- [\(Visual Studio 2013\) extensão do suporte a Design-Time.](#)
- [\(Visual Studio 2013\) como: criar um controle de Windows Forms que aproveita os recursos Design-Time.](#)

Confira também

- [Visão geral do uso de controles \(Windows Forms .NET\)](#)

Pintura e desenho em controles (Windows Forms .NET)

14/05/2021 • 5 minutes to read

A pintura personalizada de controles é uma das muitas tarefas complicadas facilitadas pelo Windows Forms. Ao criar um controle personalizado, você tem várias opções disponíveis para lidar com a aparência gráfica do seu controle. Se você estiver criando um [controle personalizado](#), ou seja, um controle que herda de [Control](#), você deve fornecer código para renderizar sua representação gráfica.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Se você estiver criando um [controle composto](#), que é um controle herdado de [UserControl](#) um dos controles de Windows Forms existentes, poderá substituir a representação gráfica padrão e fornecer seu próprio código gráfico.

Se você quiser fornecer renderização personalizada para um controle existente sem criar um novo controle, suas opções se tornarão mais limitadas. No entanto, ainda há uma ampla variedade de possibilidades gráficas para seus controles e aplicativos.

Os elementos a seguir estão envolvidos na renderização de controles:

- A funcionalidade de desenho fornecida pela classe base [System.Windows.Forms.Control](#).
- Os elementos essenciais da biblioteca gráfica GDI.
- A geometria da região de desenho.
- O procedimento para liberar recursos gráficos.

Desenho fornecido pelo controle

A classe base [Control](#) fornece a funcionalidade de desenho por meio de seu [Paint](#) evento. Um controle gera o [Paint](#) evento sempre que precisa atualizar sua exibição. Para obter mais informações sobre eventos no .NET, consulte [manipulando e gerando eventos](#).

A classe de dados de evento para o [Paint](#) evento, [PaintEventArgs](#), mantém os dados necessários para desenhar um controle-um identificador para um objeto gráfico e um retângulo que representa a região para desenhar.

```
public class PaintEventArgs : EventArgs, IDisposable
{
    public System.Drawing.Rectangle ClipRectangle {get;}
    public System.Drawing.Graphics Graphics {get;}

    // Other properties and methods.
}
```

```
Public Class PaintEventArgs
    Inherits EventArgs
    Implements IDisposable

    Public ReadOnly Property ClipRectangle As System.Drawing.Rectangle
    Public ReadOnly Property Graphics As System.Drawing.Graphics

    ' Other properties and methods.
End Class
```

[Graphics](#) é uma classe gerenciada que encapsula a funcionalidade de desenho, conforme descrito na discussão do GDI, mais adiante neste artigo. O [ClipRectangle](#) é uma instância da [Rectangle](#) estrutura e define a área disponível na qual um controle pode desenhar. Um desenvolvedor de controle pode calcular o [ClipRectangle](#) usando a [ClipRectangle](#) propriedade de um controle, conforme descrito na discussão de geometry mais adiante neste artigo.

OnPaint

Um controle deve fornecer lógica de renderização substituindo o [OnPaint](#) método do qual ele herda [Control](#). [OnPaint](#) Obtém acesso a um objeto de gráfico e um retângulo para desenhar por meio de [Graphics](#) e as [ClipRectangle](#) Propriedades da [PaintEventArgs](#) instância passada a ele.

O código a seguir usa o `System.Drawing` namespace:

```
protected override void OnPaint(PaintEventArgs e)
{
    // Call the OnPaint method of the base class.
    base.OnPaint(e);

    // Declare and instantiate a new pen that will be disposed of at the end of the method.
    using var myPen = new Pen(Color.Aqua);

    // Create a rectangle that represents the size of the control, minus 1 pixel.
    var area = new Rectangle(new Point(0, 0), new Size(this.Size.Width - 1, this.Size.Height - 1));

    // Draw an aqua rectangle in the rectangle represented by the control.
    e.Graphics.DrawRectangle(myPen, area);
}
```

```
Protected Overrides Sub OnPaint(e As PaintEventArgs)
    MyBase.OnPaint(e)

    ' Declare and instantiate a drawing pen.
    Using myPen = New System.Drawing.Pen(Color.Aqua)

        ' Create a rectangle that represents the size of the control, minus 1 pixel.
        Dim area = New Rectangle(New Point(0, 0), New Size(Me.Size.Width - 1, Me.Size.Height - 1))

        ' Draw an aqua rectangle in the rectangle represented by the control.
        e.Graphics.DrawRectangle(myPen, area)

    End Using
End Sub
```

O [OnPaint](#) método da classe base [Control](#) não implementa nenhuma funcionalidade de desenho, mas meramente invoca os delegados de evento que são registrados com o [Paint](#) evento. Ao substituir [OnPaint](#), você normalmente deve invocar o [OnPaint](#) método da classe base para que os delegados registrados recebam o [Paint](#) evento. No entanto, os controles que pintam toda a superfície não devem invocar a classe base [OnPaint](#), pois isso apresenta cintilação.

NOTE

Não invoque [OnPaint](#) diretamente do seu controle; em vez disso, invoque o [Invalidate](#) método (Herdado de [Control](#)) ou algum outro método que invoça [Invalidate](#). O [Invalidate](#) método, por sua vez, invoça [OnPaint](#). O [Invalidate](#) método está sobrecarregado e, dependendo dos argumentos fornecidos para [Invalidate](#) `e`, o redesenha alguma ou toda a área da tela.

O código no [OnPaint](#) método do seu controle será executado quando o controle for desenhado primeiro e sempre que for atualizado. Para garantir que o controle seja redesenhado sempre que ele for redimensionado, adicione a seguinte linha ao construtor do seu controle:

```
SetStyle(ControlStyles.ResizeRedraw, true);
```

```
SetStyle(ControlStyles.ResizeRedraw, True)
```

OnPaintBackground

A [Control](#) classe base define outro método que é útil para desenhar, o [OnPaintBackground](#) método.

```
protected virtual void OnPaintBackground(PaintEventArgs e);
```

```
Protected Overridable Sub OnPaintBackground(e As PaintEventArgs)
```

[OnPaintBackground](#) pinta o plano de fundo (e, dessa forma, a forma) da janela e é garantido que seja rápido, enquanto [OnPaint](#) pinta os detalhes e pode ser mais lento porque as solicitações de pintura individuais são combinadas em um [Paint](#) evento que abrange todas as áreas que precisam ser redenhadas. Talvez você queira invocar o [OnPaintBackground](#) If, por exemplo, que deseja desenhar um plano de fundo colorido em gradiente para seu controle.

Embora [OnPaintBackground](#) tenha um evento como nomenclatura e assuma o mesmo argumento que o [OnPaint](#) método, [OnPaintBackground](#) não é um método de evento verdadeiro. Não há [PaintBackground](#) evento e [OnPaintBackground](#) não invoca delegados de evento. Ao substituir o [OnPaintBackground](#) método, uma classe derivada não é necessária para invocar o [OnPaintBackground](#) método de sua classe base.

Noções básicas sobre a GDI+

A [Graphics](#) classe fornece métodos para desenhar várias formas, como círculos, triângulos, arcos e elipses, e métodos para exibir texto. O [System.Drawing](#) namespace contém namespaces e classes que encapsulam elementos gráficos, como formas (círculos, retângulos, arcos e outros), cores, fontes, pincéis e assim por diante.

Geometria da região de desenho

A [ClientRectangle](#) propriedade de um controle Especifica a região retangular disponível para o controle na tela do usuário, enquanto a [ClipRectangle](#) propriedade de [PaintEventArgs](#) especifica a área pintada. Um controle pode precisar pintar apenas uma parte da sua área disponível, como é o caso quando uma pequena seção da exibição do controle é alterada. Nessas situações, um desenvolvedor de controle deve calcular o retângulo real para desenhar e passá-lo para [Invalidate](#). As versões sobrecarregadas do [Invalidate](#) que usam um [Rectangle](#) ou [Region](#) como um argumento usam esse argumento para gerar a [ClipRectangle](#) propriedade de [PaintEventArgs](#).

Liberando recursos gráficos

Objetos gráficos são caros porque usam recursos do sistema. Esses objetos incluem instâncias da [System.Drawing.Graphics](#) classe e instâncias do [System.Drawing.Brush](#) , [System.Drawing.Pen](#) e outras classes gráficas. É importante que você crie um recurso de gráficos somente quando precisar dele e libere-o logo que terminar de usá-lo. Se você criar uma instância de um tipo que implementa a [IDisposable](#) interface, chame seu [Dispose](#) método quando tiver terminado de fazê-lo para liberar recursos.

Confira também

- [Tipos de controles personalizados](#)

Fornecendo informações de acessibilidade para controles (Windows Forms .NET)

14/05/2021 • 2 minutes to read

Os recursos de acessibilidade são programas e dispositivos especializados que ajudam as pessoas com deficiência a usarem computadores de forma mais eficaz. Alguns exemplos incluem leitores de tela para pessoas cegas e utilitários de entrada de voz para as pessoas que fornecem comandos verbais em vez de usar o mouse ou teclado. Esses recursos de acessibilidade interagem com as propriedades de acessibilidade expostas pelos controles dos Windows Forms. Essas propriedades são:

- [System.Windows.Forms.AccessibleObject](#)
- [System.Windows.Forms.Control.AccessibleDefaultActionDescription](#)
- [System.Windows.Forms.Control.AccessibleDescription](#)
- [System.Windows.Forms.Control.AccessibleName](#)
- [System.Windows.Forms.AccessibleRole](#)

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Propriedade AccessibilityObject

Essa propriedade somente leitura contém uma [AccessibleObject](#) instância. O `AccessibleObject` implementa a [IAccessible](#) interface, que fornece informações sobre a descrição do controle, o local da tela, as capacidades de navegação e o valor. O designer define esse valor quando o controle é adicionado ao formulário.

Propriedade AccessibleDefaultActionDescription

Essa cadeia de caracteres descreve as ações do controle. Ela não aparece na janela Propriedades e só pode ser definido no código. O exemplo a seguir define a [AccessibleDefaultActionDescription](#) propriedade para um controle de botão:

```
Button1.AccessibleDefaultActionDescription = "Closes the application."
```

```
button1.AccessibleDefaultActionDescription = "Closes the application.";
```

Propriedade AccessibleDescription

Essa cadeia de caracteres descreve o controle. A [AccessibleDescription](#) propriedade pode ser definida no janela Propriedades ou no código da seguinte maneira:

```
Button1.AccessibleDescription = "A button with text 'Exit'."
```

```
button1.AccessibleDescription = "A button with text 'Exit'";
```

Propriedade AccessibleName

Esse é o nome de um controle relatado para os recursos de acessibilidade. A [AccessibleName](#) propriedade pode ser definida na janela Propriedades ou no código da seguinte maneira:

```
Button1.AccessibleName = "Order"
```

```
button1.AccessibleName = "Order";
```

Propriedade AccessibleRole

Essa propriedade, que contém uma [AccessibleRole](#) enumeração, descreve a função de interface do usuário do controle. Um novo controle tem o valor definido como `Default`. Isso significa que, por padrão, um `Button` controle atua como um `Button`. Pode ser útil redefinir essa propriedade se um controle tiver outra função. Por exemplo, você pode estar usando um `PictureBox` controle como um `Chart`, e talvez queira que os recursos de acessibilidade relatem a função como um `Chart`, e não como um `PictureBox`. Também pode ser útil especificar essa propriedade para controles personalizados desenvolvidos por você. Essa propriedade pode ser definida na janela Propriedades ou no código da seguinte maneira:

```
PictureBox1.AccessibleRole = AccessibleRole.Chart
```

```
pictureBox1.AccessibleRole = AccessibleRole.Chart;
```

Confira também

- [Visão geral do controle de rótulo \(Windows Forms .NET\)](#)
- [AccessibleObject](#)
- [Control.AccessibilityObject](#)
- [Control.AccessibleDefaultActionDescription](#)
- [Control.AccessibleDescription](#)
- [Control.AccessibleName](#)
- [Control.AccessibleRole](#)
- [AccessibleRole](#)

Adicionar um controle a um formulário (Windows Forms .NET)

27/05/2021 • 2 minutes to read

A maioria dos formulários é projetada adicionando controles à superfície do formulário para definir uma interface do usuário (IU). Um *controle* é um componente em um formulário usado para exibir informações ou aceitar a entrada do usuário.

A principal maneira como um controle é adicionado a um formulário é por meio do designer Visual Studio, mas você também pode gerenciar os controles em um formulário em tempo de executar por meio do código.

IMPORTANT

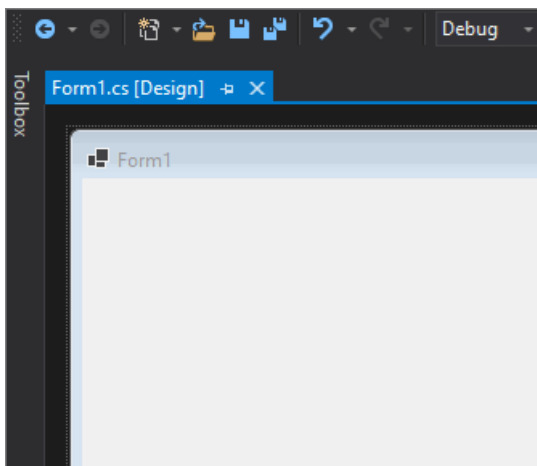
A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Adicionar com o Designer

Visual Studio usa o Designer de Formulários para criar formulários. Há um painel Controles que lista todos os controles disponíveis para seu aplicativo. Você pode adicionar controles do painel de duas maneiras:

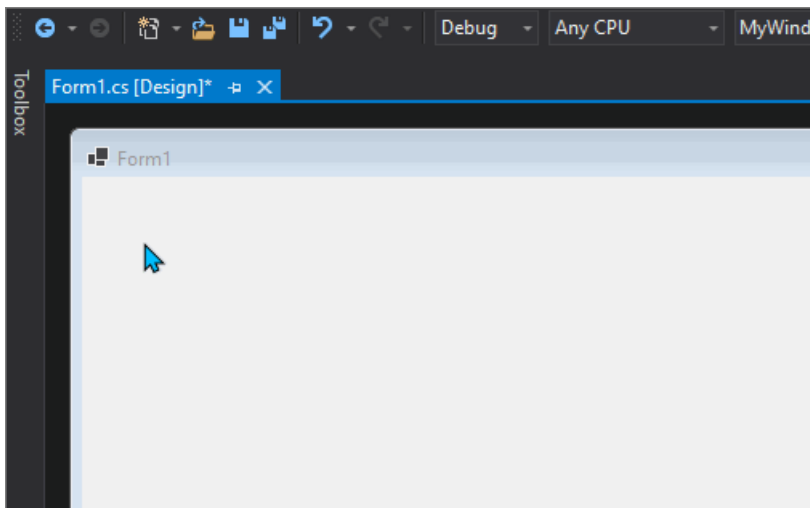
Adicione o controle clicando duas vezes

Quando um controle é clicado duas vezes, ele é adicionado automaticamente ao formulário aberto atual com as configurações padrão.



Adicionar o controle desenhando

Selecione o controle clicando nele. No formulário, arraste e selecione uma região. O controle será colocado para se ajustar ao tamanho da região selecionada.



Adicionar com código

Os controles podem ser criados e, em seguida, adicionados a um formulário em tempo de executar com a coleção do [Controls](#) formulário. Essa coleção também pode ser usada para remover controles de um formulário.

O código a seguir adiciona e posiciona dois controles, um [Label](#) e um [TextBox](#):

```
Label label1 = new Label()
{
    Text = "&First Name",
    Location = new Point(10, 10),
    TabIndex = 10
};

TextBox field1 = new TextBox()
{
    Location = new Point(label1.Location.X, label1.Bounds.Bottom + Padding.Top),
    TabIndex = 11
};

Controls.Add(label1);
Controls.Add(field1);
```

```
Dim label1 As New Label With {.Text = "&First Name",
                              .Location = New Point(10, 10),
                              .TabIndex = 10}

Dim field1 As New TextBox With {.Location = New Point(label1.Location.X,
                                                       label1.Bounds.Bottom + Padding.Top),
                               .TabIndex = 11}

Controls.Add(label1)
Controls.Add(field1)
```

Confira também

- [Definir o texto exibido por um controle do Windows Forms](#)
- [Adicionar um atalho de tecla de acesso a um controle](#)
- [System.Windows.Forms.Label](#)
- [System.Windows.Forms.TextBox](#)
- [System.Windows.Forms.Button](#)

Adicionar um atalho de tecla de acesso a um controle (Windows Forms .NET)

27/05/2021 • 2 minutes to read

Uma *chave de acesso* é um caractere de sublinhado no texto de um menu, item de menu ou rótulo de um controle como um botão. Com uma chave de acesso, o usuário pode "clique" em um botão pressionando a tecla ALT em combinação com a chave de acesso predefinida. Por exemplo, se um botão executa um procedimento para imprimir um formulário e, portanto, sua propriedade `Text` estiver definida como "Print", adicionar um e comercial (&) antes da letra "P" faz com que a letra "P" seja sublinhada no texto do botão no tempo de execução. O usuário pode executar o comando associado ao botão pressionando ALT.

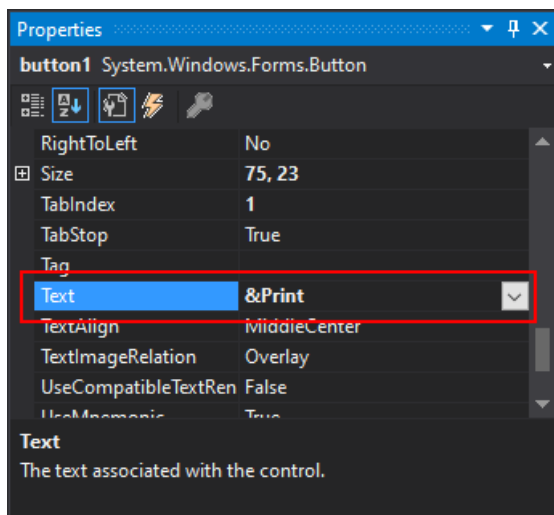
Controles que não podem receber foco não podem ter chaves de acesso, exceto controles de rótulo.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Designer

Na janela **Propriedades** do Visual Studio, defina a propriedade **Text** como uma cadeia de caracteres que inclui um e comercial (&) antes da letra que será a chave de acesso. Por exemplo, para definir a letra "P" como a chave de acesso, insira **&imprimir**.



Programático

Defina a `Text` propriedade como uma cadeia de caracteres que inclui um e comercial (&) antes da letra que será o atalho.

```
' Set the letter "P" as an access key.  
Button1.Text = "&Print"
```

```
// Set the letter "P" as an access key.  
button1.Text = "&Print";
```

Usar um rótulo para focalizar um controle

Embora um rótulo não possa ser focado, ele tem a capacidade de concentrar o próximo controle na ordem de tabulação do formulário. Cada controle recebe um valor para a [TabIndex](#) propriedade, geralmente em ordem sequencial crescente. Quando a tecla de acesso é atribuída à propriedade [Label.Text](#), o próximo controle na ordem de tabulação sequencial é focado.

Usando o exemplo da seção [programática](#), se o botão não tiver nenhum conjunto de texto, mas, em vez disso, tiver apresentado uma imagem de uma impressora, você poderá usar um rótulo para concentrar o botão.

```
' Set the letter "P" as an access key.  
Label1.Text = "&Print"  
Label1.TabIndex = 9  
Button1.TabIndex = 10
```

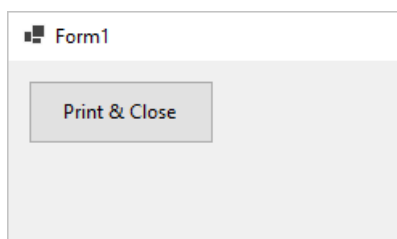
```
// Set the letter "P" as an access key.  
label1.Text = "&Print";  
label1.TabIndex = 9  
button1.TabIndex = 10
```

Exibir um e comercial

Ao definir o texto ou a legenda de um controle que interpreta um e comercial (&) como uma tecla de acesso, use dois e-ou-ou-comercial (&&) consecutivos para exibir um único "e". Por exemplo, o texto de um botão definido como é "Print && Close" exibido na legenda de `Print & Close`:

```
' Set the letter "P" as an access key.  
Button1.Text = "Print && Close"
```

```
// Set the letter "P" as an access key.  
button1.Text = "Print && Close";
```



Confira também

- [Definir o texto exibido por um controle de Windows Forms](#)
- [System.Windows.Forms.Button](#)
- [System.Windows.Forms.Label](#)

Como definir o texto exibido por um controle (Windows Forms .NET)

27/05/2021 • 2 minutes to read

Os controles de Windows Forms geralmente exibem algum texto relacionado à função principal do controle. Por exemplo, um [Button](#) controle geralmente exibe uma legenda indicando qual ação será executada se o botão for clicado. Para todos os controles, você pode definir ou retornar o texto usando a [Text](#) propriedade. Você pode alterar a fonte usando a [Font](#) propriedade.

Você também pode definir o texto usando o [Designer](#).

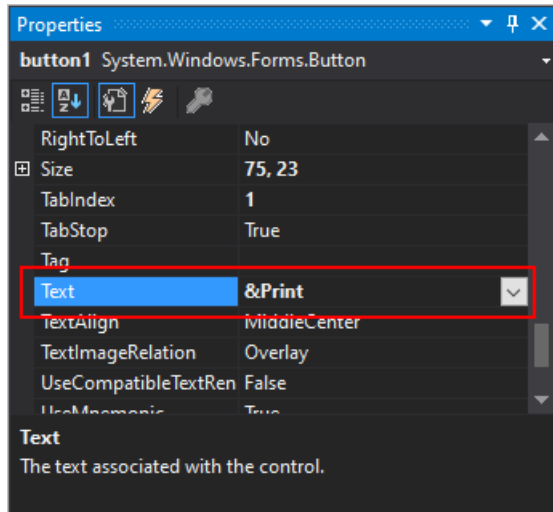
IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

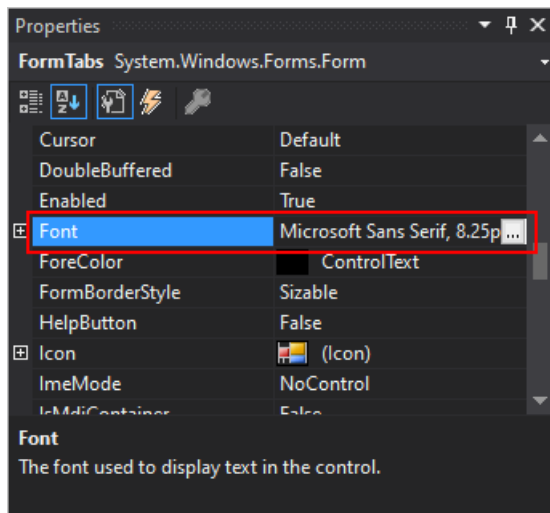
Designer

1. Na janela **Propriedades** no Visual Studio, defina a propriedade **Text** do controle como uma cadeia de caracteres apropriada.

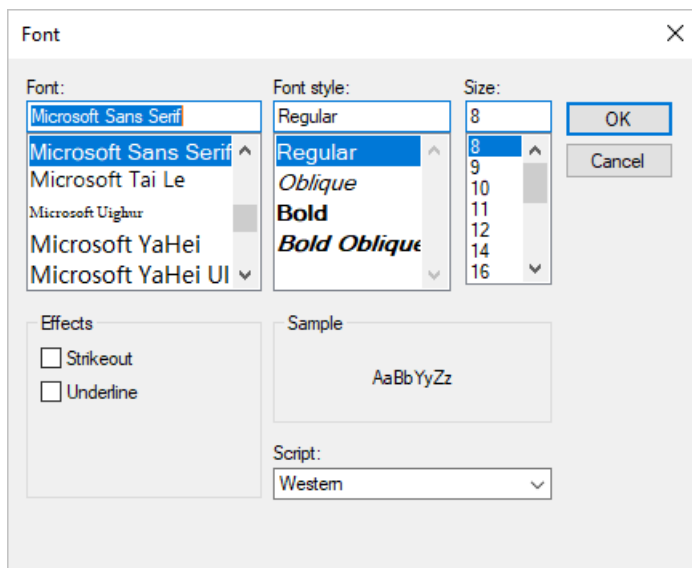
Para criar uma chave de atalho sublinhada, inclua um e comercial (&) antes da letra que será a tecla de atalho.



2. Na janela **Propriedades**, selecione o botão de reticências (⋮) ao lado da propriedade **Font**.



Na caixa de diálogo fonte padrão, ajuste a fonte com configurações como tipo, tamanho e estilo.



Programático

1. Defina a **Text** propriedade como uma cadeia de caracteres.

Para criar uma chave de acesso sublinhada, inclua um e comercial (&) antes da letra que será a chave de acesso.

2. Defina a **Font** propriedade como um objeto do tipo **Font**.

```
Button1.Text = "Click here to save changes"
Button1.Font = New Font("Arial", 10, FontStyle.Bold, GraphicsUnit.Point)
```

```
button1.Text = "Click here to save changes";
button1.Font = new Font("Arial", 10, FontStyle.Bold, GraphicsUnit.Point);
```

NOTE

Você pode usar um caractere de escape para exibir um caractere especial nos elementos de interface do usuário que seriam normalmente interpretados de maneira diferente, como itens de menu. Por exemplo, a linha de código a seguir define o texto do item de menu para ler "& agora para algo completamente diferente":

```
MpMenuItem.Text = "&& Now For Something Completely Different"
```

```
mpMenuItem.Text = "&& Now For Something Completely Different";
```

Confira também

- [Criar chaves de acesso para controles do Windows Forms](#)
- [System.Windows.Forms.Control.Text](#)

Como definir a ordem de tabulação Windows Forms (Windows Forms .NET)

04/06/2021 • 2 minutes to read

A ordem de tabulação é a ordem na qual um usuário move o foco de um controle para outro pressionando a tecla `Tab`. Cada formulário tem sua própria ordem de tabulação. Por padrão, a ordem de tabulação é igual à ordem em que você criou os controles. A numeração de ordem de tabulação começa com zero e é crescente no valor e é definida com a `TabIndex` propriedade.

Você também pode definir a ordem de tabulação usando o [designer](#).

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

A ordem de tabulação pode ser definida na **janela** Propriedades do designer usando a `TabIndex` propriedade. A `TabIndex` propriedade de um controle determina onde ele está posicionado na ordem de tabulação. Por padrão, o primeiro controle adicionado ao designer tem um valor de 0, o segundo tem um `TabIndex` de 1 e assim por diante. Depois que o mais `TabIndex` alto tiver sido focalizado, pressionar `Tab` fará o ciclo e concentrará o controle com o valor `TabIndex` mais baixo.

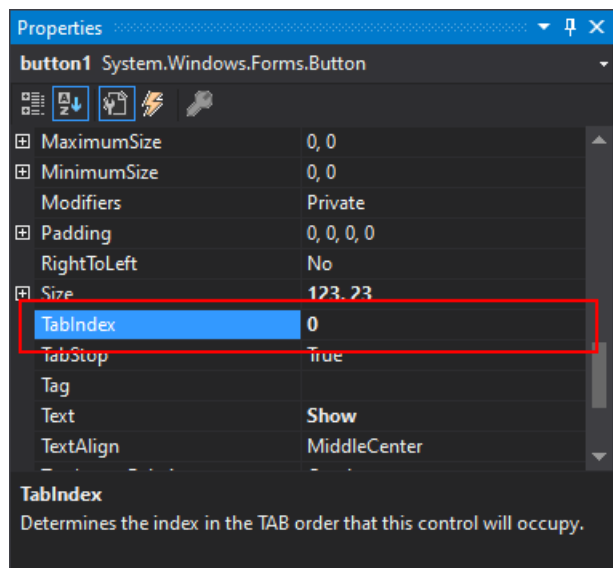
Controles de contêiner, como um controle, tratam seus filhos como `GroupBox` separados do restante do formulário. Cada filho no contêiner tem seu próprio `TabIndex` valor. Como um controle de contêiner não pode ser focalizado, quando a ordem de tabulação atinge o controle de contêiner, o controle filho do contêiner com o menor `TabIndex` é focalizado. À medida que a guia é pressionada, cada controle filho é focalizado de acordo com seu valor até o último `TabIndex` controle. Quando `Tab` é pressionado no último controle, o foco é retomado para o próximo controle no pai do contêiner, com base no próximo `TabIndex` valor.

Qualquer controle de muitos em seu formulário pode ser ignorado na ordem de tabulação. Normalmente, pressionar `Tab` sucessivamente em tempo de executar seleciona cada controle na ordem de tabulação. Ao desligar a `TabStop` propriedade, um controle é passado na ordem de tabulação do formulário.

Designer

Use a Visual Studio **propriedades** do designer de dados para definir a ordem de tabulação de um controle.

1. Selecione o controle no designer.
2. Na janela **Propriedades** no Visual Studio, de definir a **propriedade** `TabIndex` do controle como um número apropriado.



Programático

1. De definir `TabIndex` a propriedade como um valor numérico.

```
Button1.TabIndex = 1
```

```
Button1.TabIndex = 1;
```

Remover um controle da ordem de tabulação

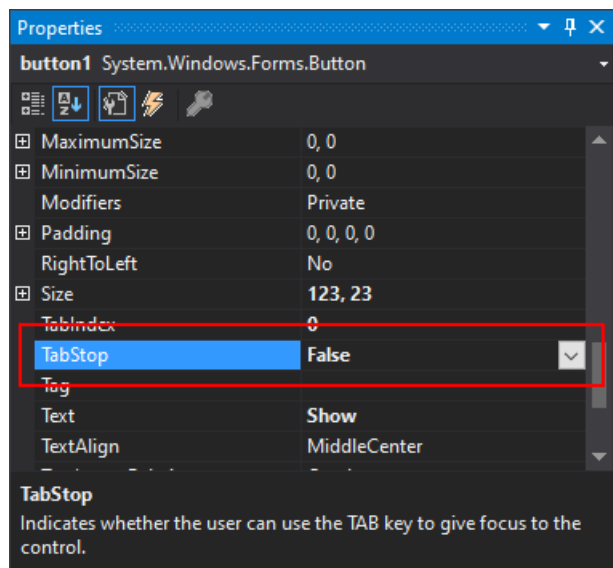
Você pode impedir que um controle receba o foco quando a tecla `Tab` for pressionada, definindo a propriedade `TabStop` como `false`. O controle é ignorado quando você passa pelos controles com a tecla `Tab`. O controle não perde sua ordem de tabulação quando essa propriedade é definida como `false`.

NOTE

Um grupo de botões de rádio tem uma única parada de tabulação em tempo de executar. O botão selecionado, o botão com sua propriedade definida como `Checked` `true`, `TabStop` automaticamente como `true`. Outros botões no grupo de botões de rádio têm suas `TabStop` propriedades definidas como `false`.

Definir `TabStop` com o designer

1. Selecione o controle no designer.
2. Na janela **Propriedades** no Visual Studio, de definir a **propriedade** `TabStop` como `False`.



Definir TabStop programaticamente

1. Defina a propriedade `TabStop` como `false`.

```
Button1.TabStop = false;
```

```
Button1.TabStop = False
```

Confira também

- [Adicionar um controle para um formulário](#)
- [System.Windows.Forms.Control.TabIndex](#)
- [System.Windows.Forms.Control.TabStop](#)

Como encaixar e ancorar controles (Windows Forms .NET)

04/06/2021 • 2 minutes to read

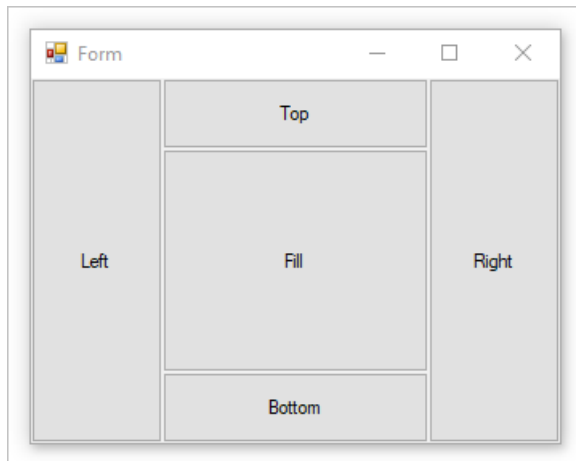
Se você estiver projetando um formulário que o usuário possa reescalar em tempo de executar, os controles em seu formulário deverão ser reposicionados e reposicionados corretamente. Os controles têm duas propriedades que ajudam com o posicionamento e o tamanho automáticos, quando o formulário altera o tamanho.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

- [Control.Dock](#)

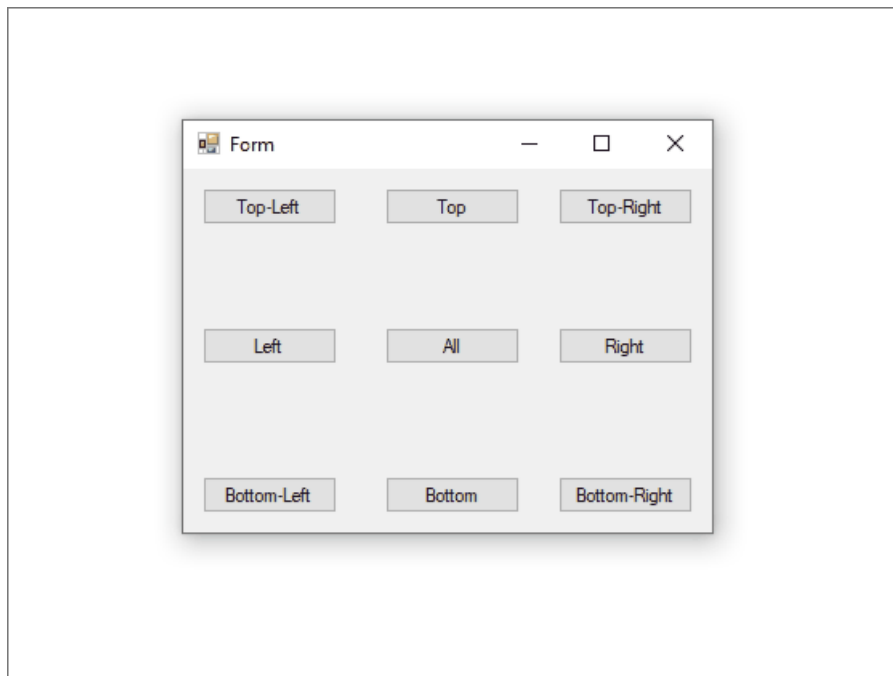
Os controles encaixados preenchem as bordas do contêiner do controle, seja o formulário ou um controle de contêiner. Por exemplo, Windows Explorer encaixa seu controle no lado esquerdo da janela e seu controle à [TreeView ListView](#) direita da janela. O modo de encaixe pode ser qualquer lado do contêiner do controle ou definido para preencher o espaço restante do contêiner.



Os controles são encaixados na ordem z inversa e [Dock](#) a propriedade interage com a [AutoSize](#) propriedade. Para obter mais informações, consulte [O ajuste automático](#).

- [Control.Anchor](#)

Quando o formulário de um controle ancorado é reessado, o controle mantém a distância entre o controle e as posições de âncora. Por exemplo, se você tiver um controle ancorado às bordas esquerda, direita e inferior do formulário, à medida que o formulário é reessado, o controle será reessado horizontalmente para manter a mesma distância dos lados direito e esquerdo do [TextBox TextBox](#) formulário. O controle também se posiciona verticalmente para que seu local sempre tenha a mesma distância da borda inferior do formulário. Se um controle não estiver ancorado e o formulário for reessado, a posição do controle em relação às bordas do formulário será alterada.



Para obter mais informações, consulte [Posição e layout de controles](#).

Encaixar um controle

Um controle é encaixado definindo sua [Dock](#) propriedade.

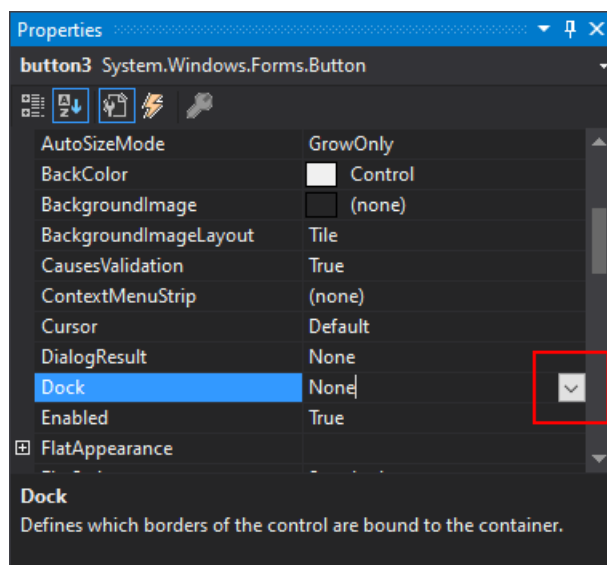
NOTE

Controles herdados devem ser `Protected` para poderem ser encaixados. Para alterar o nível de acesso de um controle, de definido sua **propriedade Modificador** na janela Propriedades.

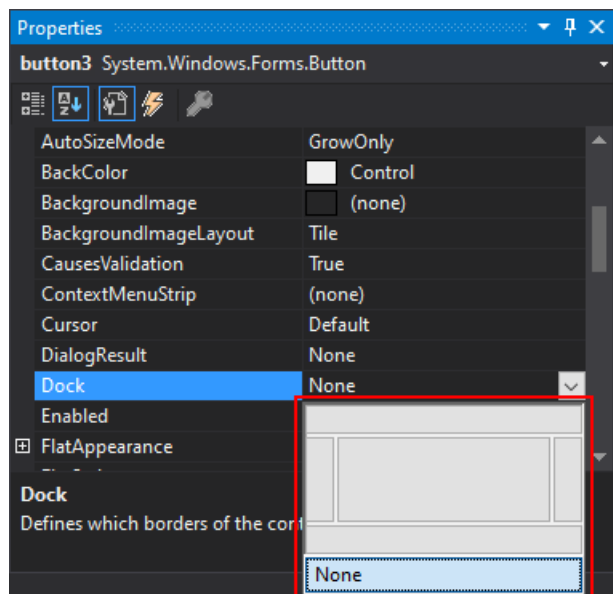
Usar o designer

Use a Visual Studio **propriedades** do designer de dados para definir o modo de encaixe de um controle.

1. Selecione o controle no designer.
2. Na janela **Propriedades**, selecione a seta à direita da **propriedade Dock**.



3. Selecione o botão que representa a borda do contêiner em que você deseja encaixar o controle. Para preencher o conteúdo do formulário ou do controle de contêiner do controle, pressione a caixa central. Pressione **(nenhum)** para desabilitar o encaixe.



O controle é redimensionado automaticamente para ajustar os limites da borda encaixada.

Definir Dock programaticamente

1. De definir `Dock` a propriedade em um controle . Neste exemplo, um botão é encaixado no lado direito do contêiner:

```
button1.Dock = DockStyle.Right;
```

```
button1.Dock = DockStyle.Right
```

Ancorar um controle

Um controle é ancorado em uma borda definindo sua [Anchor](#) propriedade como um ou mais valores.

NOTE

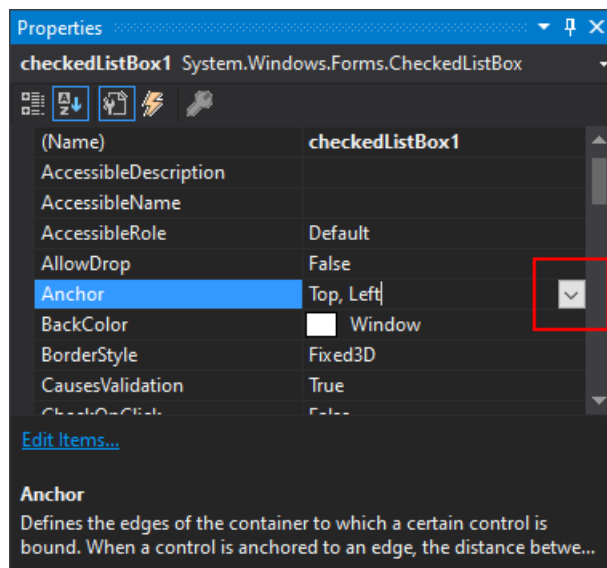
Determinados controles, como [ComboBox](#) o controle , têm um limite para sua altura. Ancorar o controle na parte inferior do formulário ou contêiner não pode forçar o controle a exceder o limite de altura.

Controles herdados devem ser `Protected` para serem ancorados. Para alterar o nível de acesso de um controle, defina sua propriedade `Modifiers` na janela **Propriedades**.

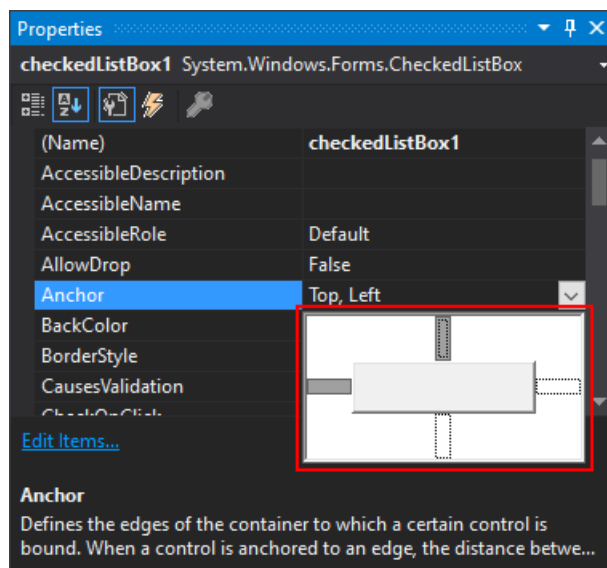
Usar o designer

Use a Visual Studio **propriedades** do designer de dados para definir as bordas ancoradas de um controle.

1. Selecione o controle no designer.
2. Na janela **Propriedades**, selecione a seta à direita da **propriedade Âncora**.



3. Para definir ou desancorar uma âncora, selecione o braço superior, esquerdo, direito ou inferior da cruz.



Definir Âncora programaticamente

1. De definir **Anchor** a propriedade em um controle . Neste exemplo, um botão é ancorado nos lados direito e inferior de seu contêiner:

```
button1.Anchor = AnchorStyles.Bottom | AnchorStyles.Right;
```

```
button1.Anchor = AnchorStyles.Bottom Or AnchorStyles.Right
```

Confira também

- [Posição e layout dos controles.](#)
- [System.Windows.Forms.Control.Anchor](#)
- [System.Windows.Forms.Control.Dock](#)

Como exibir uma imagem em um controle (Windows Forms .NET)


27/05/2021 • 2 minutes to read

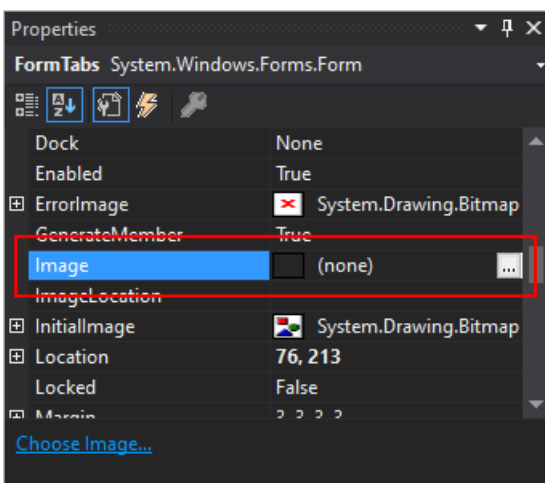
Vários controles de Windows Forms podem exibir imagens. Essas imagens podem ser ícones que esclarecem a finalidade do controle, como um ícone de disquete em um botão que indica o comando salvar. Como alternativa, os ícones podem ser imagens de plano de fundo para dar ao controle a aparência e o comportamento que você deseja.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Designer

Na janela **Propriedades** do Visual Studio, selecione a propriedade **Image** ou **BackgroundImage** do controle e, em seguida, selecione as reticências () para exibir a caixa de diálogo **selecionar recurso** e, em seguida, selecione a imagem que deseja exibir.



Programático

Defina a `Image` propriedade ou o controle `BackgroundImage` como um objeto do tipo `Image`. Em geral, você carregará a imagem de um arquivo usando o `FromFile` método.

No exemplo de código a seguir, o caminho definido para o local da imagem é a pasta **minhas imagens**. A maioria dos computadores que executam o sistema operacional Windows inclui esse diretório. Isso também permite que os usuários com níveis mínimos de acesso do sistema executem o aplicativo com segurança. O exemplo de código a seguir requer que você já tenha um formulário com um `PictureBox` controle adicionado.

```
// Replace the image named below with your own icon.  
// Note the escape character used (@) when specifying the path.  
pictureBox1.Image = Image.FromFile  
    (System.Environment.GetFolderPath  
    (System.Environment.SpecialFolder.MyPictures)  
    + @"\Image.gif");
```

```
' Replace the image named below with your own icon.  
PictureBox1.Image = Image.FromFile _  
    (System.Environment.GetFolderPath _  
    (System.Environment.SpecialFolder.MyPictures) _  
    & "\Image.gif")
```

Confira também

- [System.Drawing.Image.FromFile](#)
- [System.Drawing.Image](#)
- [System.Windows.Forms.Control.BackgroundImage](#)

Visão geral do uso do teclado (Windows Forms .NET)

14/05/2021 • 7 minutes to read

No Windows Forms, a entrada do usuário é enviada aos aplicativos na forma de [mensagens do Windows](#). Uma série de métodos substituíveis processam essas mensagens no nível do aplicativo, formulário e controle. Quando esses métodos recebem mensagens do teclado, eles geram eventos que podem ser tratados para obter informações sobre a entrada do teclado. Em muitos casos, os aplicativos do Windows Forms serão capazes de processar todas as entradas do usuário simplesmente manipulando esses eventos. Em outros casos, um aplicativo pode precisar substituir um dos métodos que processam mensagens para interceptar uma mensagem específica antes de ela ser recebida pelo aplicativo, formulário ou controle.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Eventos de teclado

Todos os controles do Windows Forms herdam um conjunto de eventos relacionados às entradas de mouse e teclado. Por exemplo, um controle pode manipular o [KeyPress](#) evento para determinar o código de caractere de uma chave que foi pressionada. Para obter mais informações, consulte [usando eventos de teclado](#).

Métodos que processam mensagens de entrada do usuário

Formulários e controles têm acesso à [IMessageFilter](#) interface e um conjunto de métodos substituíveis que processam mensagens do Windows em pontos diferentes na fila de mensagens. Todos esses métodos têm um [Message](#) parâmetro, que encapsula os detalhes de baixo nível das mensagens do Windows. É possível implementar ou substituir esses métodos para analisar a mensagem e, então, consumi-la ou passá-la para o próximo consumidor na fila de mensagens. A tabela a seguir apresenta os métodos que processam todas as mensagens do Windows no Windows Forms.

MÉTODO	OBSERVAÇÕES
PreFilterMessage	Este método intercepta mensagens do Windows que estão na fila (também chamadas de postadas) no nível do aplicativo.
PreProcessMessage	Este método intercepta as mensagens do Windows no nível do formulário e do controle antes que elas sejam processadas.
WndProc	Este método processa mensagens do Windows no nível do formulário e do controle.
DefWndProc	Esse método realiza o processamento padrão de mensagens do Windows no nível do formulário e do controle. Isso fornece a funcionalidade mínima de uma janela.

MÉTODO	OBSERVAÇÕES
OnNotifyMessage	Este método intercepta as mensagens do Windows no nível do formulário e do controle após o seu processamento. O EnableNotifyMessage bit de estilo deve ser definido para que este método seja chamado.

Mensagens de teclado e mouse também são processadas por um conjunto adicional de métodos substituíveis específicos para esses tipos de mensagens. Para obter mais informações, consulte a seção [chaves de pré-processamento](#) . .

Tipos de chaves

Windows Forms identifica a entrada de teclado como códigos de chave virtual que são representados pela enumeração bit-de bits [Keys](#) . Com a [Keys](#) enumeração, você pode combinar uma série de teclas pressionadas para resultar em um único valor. Esses valores correspondem aos valores que acompanham o **WM_KEYDOWN** e **WM_SYSKEYDOWN** mensagens do Windows. Você pode detectar a maioria dos pressionamentos de chave física manipulando os [KeyDown](#) [KeyUp](#) eventos ou. As chaves de caractere são um subconjunto da [Keys](#) enumeração e correspondem aos valores que acompanham o **WM_CHAR** e **WM_SYSCHAR** mensagens do Windows. Se a combinação de teclas pressionadas resultar em um caractere, você poderá detectar o caractere manipulando o [KeyPress](#) evento.

Ordem dos eventos de teclado

Conforme listado anteriormente, existem três eventos relacionados que podem ocorrer em um controle. A sequência a seguir mostra a ordem geral dos eventos:

1. O usuário envia por push a chave "a", a chave é pré-processado, despachada e [KeyDown](#) ocorre um evento.
2. O usuário mantém a chave "a", a chave é pré-processado, despachada e um [KeyPress](#) evento ocorre. Esse evento ocorre várias vezes enquanto o usuário pressiona uma tecla.
3. O usuário libera a chave "a", a chave é pré-processado, despachada e [KeyUp](#) ocorre um evento.

Chaves de pré-processamento

Assim como outras mensagens, as mensagens do teclado são processadas no [WndProc](#) método de um formulário ou controle. No entanto, antes que as mensagens do teclado sejam processadas, o [PreProcessMessage](#) método chama um ou mais métodos que podem ser substituídos para manipular chaves de caracteres especiais e chaves físicas. Você pode substituir esses métodos para detectar e filtrar certas teclas antes que as mensagens sejam processadas pelo controle. A tabela a seguir mostra a ação que está sendo executada e o método relacionado que ocorre, na ordem em que o método ocorre.

Pré-processamento de um evento KeyDown

AÇÃO	MÉTODO RELACIONADO	OBSERVAÇÕES
Verifique se há uma chave de comando como um acelerador ou um atalho de menu.	ProcessCmdKey	Este método processa uma chave de comando, que tem precedência sobre teclas regulares. Se esse método retornar <code>true</code> , a mensagem principal não será enviada e um evento de tecla não ocorrerá. Se retornar <code>false</code> , IsInputKey será chamado .

AÇÃO	MÉTODO RELACIONADO	OBSERVAÇÕES
Verifique se há uma chave especial que requer pré-processamento ou uma chave de caractere normal que deva gerar um KeyDown evento e ser expedido para um controle.	IsInputKey	Se o método retornar <code>true</code> , significa que o controle é um caractere regular e um KeyDown evento é gerado. Se <code>false</code> , ProcessDialogKey for chamado. Observação: Para garantir que um controle obtenha uma chave ou uma combinação de chaves, você pode manipular o PreviewKeyDown evento e o conjunto IsInputKey de PreviewKeyDownEventArgs para <code>true</code> para a chave ou as chaves desejadas.
Verifique se há uma tecla de navegação (teclas ESC, TAB, Enter ou teclas de seta).	ProcessDialogKey	Este método processa uma tecla física que utiliza uma funcionalidade especial dentro do controle, como alternar o foco entre o controle e seu pai. Se o controle imediato não tratar a chave, o ProcessDialogKey será chamado no controle pai e assim por diante para o controle de nível superior na hierarquia. Se esse método retornar <code>true</code> , o pré-processamento estará completo e um evento de tecla não será gerado. Se ele retornar <code>false</code> , KeyDown ocorrerá um evento.

Pré-processamento de um evento de pressionamento de tecla

AÇÃO	MÉTODO RELACIONADO	OBSERVAÇÕES
Verifique se a chave é um caractere normal que deve ser processado pelo controle	IsInputChar	Se o caractere for um caractere normal, esse método retornará <code>true</code> , o KeyPress evento será gerado e nenhum outro pré-processamento ocorrerá. Caso contrário, ProcessDialogChar será chamado.
Verifique se o caractere é um mnemônico (como &OK em um botão)	ProcessDialogChar	Esse método, semelhante a ProcessDialogKey , será chamado de hierarquia de controle. Se o controle for um controle de contêiner, ele verificará mnemônicos chamando ProcessMnemonic a si mesmo e seus controles filho. Se ProcessDialogChar retornar <code>true</code> , um KeyPress evento não ocorrerá.

Processando mensagens do teclado

Depois que as mensagens do teclado atingem o [WndProc](#) método de um formulário ou controle, elas são processadas por um conjunto de métodos que podem ser substituídos. Cada um desses métodos retorna um [Boolean](#) valor que especifica se a mensagem do teclado foi processada e consumida pelo controle. Se um dos métodos retornar `true`, então a mensagem será considerada tratada e ela não será passada para a base ou pai do controle para processamento adicional. Caso contrário, a mensagem permanecerá na fila e poderá ser processada em outro método da base ou pai do controle. A tabela a seguir apresenta os métodos que

processam mensagens do teclado.

MÉTODO	OBSERVAÇÕES
ProcessKeyMessage	Esse método processa todas as mensagens de teclado que são recebidas pelo WndProc método do controle.
ProcessKeyPreview	Esse método envia a mensagem do teclado para o pai do controle. Se ProcessKeyPreview retornar <code>true</code> , nenhum evento de chave será gerado; caso contrário, ProcessKeyEventArgs será chamado.
ProcessKeyEventArgs	Esse método gera os KeyDown KeyPress eventos, e KeyUp , conforme apropriado.

Substituindo métodos de teclado

Há vários métodos disponíveis para substituição quando uma mensagem do teclado é pré-processada e processada. No entanto, alguns métodos são opções muito melhores do que outros. A tabela a seguir mostra tarefas que você talvez queira realizar e a melhor maneira de substituir os métodos do teclado. Para obter mais informações sobre como substituir métodos, consulte [herança \(guia de programação C#\)](#) ou [herança \(Visual Basic\)](#)

TAREFA	MÉTODO
Interceptar uma chave de navegação e gerar um KeyDown evento. Por exemplo, você deseja que as teclas TAB e Enter sejam identificadas em uma caixa de texto.	Substitua IsInputKey . Observação: Como alternativa, você pode manipular o PreviewKeyDown evento e o conjunto IsInputKey de PreviewKeyDownEventArgs para <code>true</code> para a chave ou as chaves desejadas.
Execute tratamento de navegação ou entrada especial em um controle. Por exemplo, você deseja usar as teclas de seta no controle de lista para alterar o item selecionado.	Substituição ProcessDialogKey
Interceptar uma chave de navegação e gerar um KeyPress evento. Por exemplo, em um controle de caixa de rotação você deseja o pressionamento de várias teclas de seta para acelerar a progressão pelos itens.	Substitua IsInputChar .
Execute a manipulação especial de entrada ou navegação durante um KeyPress evento. Por exemplo, em uma controle de lista, manter pressionada a tecla "r" ignora os itens que começam com a letra r.	Substituição ProcessDialogChar
Execute o tratamento mnemônico personalizado. Por exemplo, você deseja manipular mnemônicos em botões desenhados pelo proprietário contidos em uma barra de ferramentas.	Substitua ProcessMnemonic .

Confira também

- [Keys](#)
- [WndProc](#)
- [PreProcessMessage](#)
- [Usando eventos de teclado \(Windows Forms .NET\)](#)

- [Como modificar eventos de tecla de teclado \(Windows Forms .NET\)](#)
- [Como verificar se há pressionamentos de tecla modificadores \(Windows Forms .NET\)](#)
- [Como simular eventos de teclado \(Windows Forms .NET\)](#)
- [Como lidar com mensagens de entrada de teclado no formulário \(Windows Forms .NET\)](#)
- [Adicionar um controle \(Windows Forms .NET\)](#)

Usando eventos de teclado (Windows Forms .NET)

14/05/2021 • 2 minutes to read

A maioria dos programas do Windows Forms processa a entrada do teclado tratando eventos de teclado. Este artigo fornece uma visão geral dos eventos de teclado, incluindo detalhes sobre quando usar cada evento e os dados que são fornecidos para cada evento. Para obter mais informações sobre eventos em geral, consulte [visão geral de eventos \(Windows Forms .net\)](#).

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Eventos de teclado

O Windows Forms fornece dois eventos que ocorrem quando um usuário pressiona uma tecla do teclado e um evento quando um usuário libera uma tecla do teclado:

- O [KeyDown](#) evento ocorre uma vez.
- O [KeyPress](#) evento, que pode ocorrer várias vezes quando um usuário mantém a mesma chave.
- O [KeyUp](#) evento ocorre uma vez quando um usuário libera uma chave.

Quando um usuário pressiona uma tecla, o Windows Forms determina qual evento deve ser gerado com base em se a mensagem do teclado especifica uma tecla de caractere ou uma tecla física. Para obter mais informações sobre caracteres e chaves físicas, consulte [visão geral do teclado, eventos de teclado](#).

A tabela a seguir descreve os três eventos de teclado.

EVENTO DE TECLADO	DESCRIÇÃO	RESULTADOS
-------------------	-----------	------------

EVENTO DE TECLADO	DESCRIÇÃO	RESULTADOS
<p>KeyDown</p>	<p>Esse evento é gerado quando um usuário pressiona uma tecla física.</p>	<p>O manipulador para KeyDown recebe:</p> <ul style="list-style-type: none"> • Um KeyEventArgs parâmetro, que fornece a KeyCode Propriedade (que especifica um botão de teclado físico). • A Modifiers Propriedade (Shift, CTRL ou ALT). • A KeyData Propriedade (que combina o código de chave e o modificador). O KeyEventArgs parâmetro também fornece: <ul style="list-style-type: none"> ◦ A Handled propriedade, que pode ser definida para impedir que o controle subjacente receba a chave. ◦ A SuppressKeyPress propriedade, que pode ser usada para suprimir KeyPress os KeyUp eventos e para esse pressionamento de tecla.
<p>KeyPress</p>	<p>Esse evento é gerado quando as teclas são pressionadas resultam em um caractere. Por exemplo, um usuário pressiona as teclas SHIFT e a letra "a" minúscula, o que resulta em uma letra "A" maiúscula.</p>	<p>KeyPress é gerado após KeyDown .</p> <ul style="list-style-type: none"> • O manipulador para KeyPress recebe: • Um KeyPressEventArgs parâmetro, que contém o código de caractere da chave que foi pressionada. Esse código de caractere é exclusivo para cada combinação de uma tecla de caractere e tecla modificadora. <p>Por exemplo, a tecla "A" gerará:</p> <ul style="list-style-type: none"> ◦ O código de caractere 65, se ela estiver pressionada com a tecla SHIFT ◦ Ou a tecla CAPS LOCK, 97 se ela for pressionada sozinha, ◦ E 1 se ela estiver pressionada com a tecla CTRL.

EVENTO DE TECLADO	DESCRIÇÃO	RESULTADOS
KeyUp	Esse evento é gerado quando um usuário libera uma tecla física.	<p>O manipulador para KeyUp recebe:</p> <ul style="list-style-type: none">• Um KeyEventArgs parâmetro:<ul style="list-style-type: none">◦ Que fornece a KeyCode Propriedade (que especifica um botão de teclado físico).◦ A Modifiers Propriedade (Shift, CTRL ou ALT).◦ A KeyData Propriedade (que combina o código de chave e o modificador).

Confira também

- [Visão geral do uso do teclado \(Windows Forms .NET\)](#)
- [Como modificar eventos de tecla de teclado \(Windows Forms .NET\)](#)
- [Como verificar se há pressionamentos de tecla modificadores \(Windows Forms .NET\)](#)
- [Como simular eventos de teclado \(Windows Forms .NET\)](#)
- [Como lidar com mensagens de entrada de teclado no formulário \(Windows Forms .NET\)](#)

Visão geral de como validar a entrada do usuário (Windows Forms .NET)

14/05/2021 • 6 minutes to read

Quando os usuários inserem dados em seu aplicativo, convém verificar se os dados são válidos antes de seu aplicativo utilizá-los. Você pode exigir que determinados campos de texto não tenham comprimento zero, que um campo seja formatado como um número de telefone, ou que uma cadeia de caracteres não contenha caracteres inválidos. O Windows Forms fornece várias maneiras para validar a entrada em seu aplicativo.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Controle MaskedTextBox

Se você precisar exigir que os usuários insiram dados em um formato bem definido, como um número de telefone ou um número de peça, poderá fazer isso rapidamente e com o código mínimo usando o [MaskedTextBox](#) controle. Uma *máscara* é uma cadeia de caracteres composta de caracteres de uma linguagem de mascaramento que especifica quais caracteres podem ser inseridos em qualquer posição determinada de uma caixa de texto. O controle exibe um conjunto de avisos para o usuário. Se o usuário digita uma entrada incorreta, como digitar uma letra quando é necessário um dígito, o controle rejeitará automaticamente a entrada.

A linguagem de mascaramento usada pelo [MaskedTextBox](#) é flexível. Ela permite que você especifique os caracteres exigidos, caracteres opcionais, caracteres literais como hifens e parênteses, caracteres de moeda e separadores de data. O controle também funciona bem quando associado a uma fonte de dados. O [Format](#) evento em uma ligação de dados pode ser usado para reformatar dados de entrada para obedecer à máscara e o [Parse](#) evento pode ser usado para reformatar dados de saída para atender às especificações do campo de dados.

Validação controlada por evento

Se você quiser controle programático completo sobre a validação ou precisar de verificações de validação complexas, use os eventos de validação que são criados na maioria dos controles de Windows Forms. Cada controle que aceita entrada de usuário de forma livre tem um [Validating](#) evento que ocorrerá sempre que o controle exigir validação de dados. No [Validating](#) método de manipulação de eventos, você pode validar a entrada do usuário de várias maneiras. Por exemplo, se você tiver uma caixa de texto que deve conter um CEP, poderá fazer a validação das seguintes maneiras:

- Se o CEP deve pertencer a um grupo específico de CEPs, você pode fazer uma comparação de cadeia de caracteres na entrada para validar os dados inseridos pelo usuário. Por exemplo, se o CEP precisar estar no conjunto `{10001, 10002, 10003}`, você poderá usar uma comparação de cadeia de caracteres para validar os dados.
- Se o CEP precisar estar em um formulário específico, você poderá usar expressões regulares para validar os dados inseridos pelo usuário. Por exemplo, para validar a forma `#####` ou `#####-####`, você pode usar a expressão regular `^(\\d{5})(-\\d{4})?$`. Para validar a forma `A#A #A#`, você pode usar a expressão regular `[A-Z]\\d[A-Z] \\d[A-Z]\\d`. Para obter mais informações sobre expressões regulares, consulte expressões regulares do [.net](#) e exemplos de expressões [regulares](#).

- Se o código postal deve ser um CEP válido dos Estados Unidos, você poderia chamar um serviço Web de código postal para validar os dados inseridos pelo usuário.

O **Validating** evento é fornecido como um objeto do tipo **CancelEventArgs**. Se você determinar que os dados do controle não são válidos, cancele o **Validating** evento definindo a propriedade desse objeto **Cancel** como `true`. Se você não definir a **Cancel** propriedade, Windows Forms assumirá que a validação foi bem-sucedida para esse controle e gerará o **Validated** evento.

Para obter um exemplo de código que valida um endereço de email em um **TextBox**, consulte a **Validating** referência do evento.

Controles ligados a dados de validação orientada por evento

A validação é útil quando você vincula os controles a uma fonte de dados, como uma tabela de banco de dado. Usando a validação, você pode garantir que os dados do controle satisfaçam o formato exigido pela fonte de dados e que ele não contenha caracteres especiais, como aspas e barras invertidas que possam não ser seguras.

Quando você usa a vinculação de dados, os dados em seu controle são sincronizados com a fonte de dados durante a execução do **Validating** evento. Se você cancelar o **Validating** evento, os dados não serão sincronizados com a fonte de dados.

IMPORTANT

Se você tiver validação personalizada que ocorre após o **Validating** evento, ela não afetará a associação de dados. Por exemplo, se você tiver um código em um **Validated** evento que tente cancelar a vinculação de dados, a vinculação de dados ainda ocorrerá. Nesse caso, para executar a validação no **Validated** evento, altere a propriedade do controle **Binding.DataSourceUpdateMode** de **DataSourceUpdateMode.OnValidation** para **DataSourceUpdateMode.Never**. Adicione `your-control.DataBindings["field-name"].WriteValue()` ao seu código de validação.

Validação implícita e explícita

Então quando os dados de um controle são validados? Isso cabe a você, o desenvolvedor. Você pode usar a validação implícita ou explícita, dependendo das necessidades do seu aplicativo.

Validação implícita

A abordagem de validação implícita valida os dados enquanto o usuário os digita. Valide os dados lendo as chaves à medida que elas forem pressionadas ou mais comumente sempre que o usuário levar o foco de entrada para fora do controle. Essa abordagem é útil quando você deseja dar ao usuário comentários imediatos sobre os dados enquanto eles estão trabalhando.

Se você quiser usar a validação implícita para um controle, deverá definir a propriedade desse controle **AutoValidate** como **EnablePreventFocusChange** or **EnableAllowFocusChange**. Se você cancelar o **Validating** evento, o comportamento do controle será determinado pelo valor que você atribuiu a **AutoValidate**. Se você tiver atribuído **EnablePreventFocusChange**, cancelar o evento fará com que o **Validated** evento não ocorra. O foco de entrada permanecerá no controle atual até que o usuário altere os dados para um formato válido. Se você tiver atribuído **EnableAllowFocusChange**, o **Validated** evento não ocorrerá quando você cancelar o evento, mas o foco ainda será alterado para o próximo controle.

A atribuição **Disable** à **AutoValidate** Propriedade impede completamente a validação implícita. Para validar seus controles, você precisará usar a validação explícita.

Validação explícita

A abordagem de validação explícita valida os dados de uma só vez. Você pode validar os dados em resposta a uma ação do usuário, como clicar em um botão **salvar** ou em um link **Avançar**. Quando a ação do usuário ocorre, você pode disparar a validação explícita de uma das seguintes maneiras:

- Chame [Validate](#) para validar o último controle para ter o foco perdido.
- Chame [ValidateChildren](#) para validar todos os controles filho em um formulário ou controle de contêiner.
- Chamar um método personalizado para validar manualmente os dados nos controles.

Comportamento padrão de validação implícita para controles

Diferentes Windows Forms controles têm padrões diferentes para sua [AutoValidate](#) propriedade. A tabela a seguir mostra os controles mais comuns e seus valores padrão.

CONTROL	COMPORTAMENTO DE VALIDAÇÃO PADRÃO
ContainerControl	Inherit
Form	EnableAllowFocusChange
PropertyGrid	Propriedade não exposta no Visual Studio
ToolStripContainer	Propriedade não exposta no Visual Studio
SplitContainer	Inherit
UserControl	EnableAllowFocusChange

Fechando o formulário e substituindo a validação

Quando um controle mantém o foco porque os dados que ele contém são inválidos, é impossível fechar o formulário pai de uma das maneiras usuais:

- Clicando no botão **Fechar**.
- Selecionando o > menu **Fechar** sistema.
- Chamando o [Close](#) método programaticamente.

No entanto, em alguns casos, você talvez queira permitir que o usuário feche o formulário independentemente se os valores nos controles são válidos. Você pode substituir a validação e fechar um formulário que ainda contém dados inválidos criando um manipulador para o [FormClosing](#) evento do formulário. No evento, defina a [Cancel](#) propriedade como `false`. Isso força o formulário a fechar. Para obter mais informações e um exemplo, consulte [Form.FormClosing](#).

NOTE

Se você forçar o formulário a fechar desta maneira, todos os dados nos controles do formulário que ainda não tiverem sido salvos serão perdidos. Além disso, formulários modais não validam o conteúdo dos controles quando eles são fechados. Você ainda pode usar a validação de controle para bloquear o foco para um controle, mas não precisa se preocupar com o comportamento associado ao fechamento do formulário.

Confira também

- [Usando eventos de teclado \(Windows Forms .NET\)](#)
- [Control.Validating](#)
- [Form.FormClosing](#)
- [System.Windows.Forms.FormClosingEventArgs](#)

Como modificar eventos de tecla de teclado (Windows Forms .NET)

14/05/2021 • 3 minutes to read

O Windows Forms fornece a capacidade de consumir e modificar as entradas do teclado. O consumo de uma chave refere-se à manipulação de uma chave dentro de um método ou manipulador de eventos para que outros métodos e eventos mais adiante na fila de mensagens não recebam o valor da chave. E modificar uma chave refere-se à modificação do valor de uma chave para que os métodos e manipuladores de eventos que estejam mais abaixo na fila de mensagens recebam um valor de chave diferente. Este artigo mostra como realizar essas tarefas.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Consumir uma chave

Em um [KeyPress](#) manipulador de eventos, defina a [Handled](#) propriedade da [KeyPressEventArgs](#) classe como

```
true
```

-ou-

Em um [KeyDown](#) manipulador de eventos, defina a [Handled](#) propriedade da [KeyEventArgs](#) classe como

```
true
```

.

NOTE

Definir a [Handled](#) propriedade no [KeyDown](#) manipulador de eventos não impede que os [KeyPress](#) eventos e [KeyUp](#) sejam gerados para o pressionamento de teclas atual. Use a [SuppressKeyPress](#) propriedade para essa finalidade.

O exemplo a seguir trata o [KeyPress](#) evento para consumir `A` as `a` teclas de caracteres e. Essas chaves não podem ser digitadas na caixa de texto:

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar == 'a' || e.KeyChar == 'A')
        e.Handled = true;
}
```

```
Private Sub TextBox1_KeyPress(sender As Object, e As KeyPressEventArgs)
    If e.KeyChar = "a"c Or e.KeyChar = "A"c Then
        e.Handled = True
    End If
End Sub
```

Modificar uma chave de caractere padrão

Em um [KeyPress](#) manipulador de eventos, defina a [KeyChar](#) propriedade da [KeyPressEventArgs](#) classe como o valor da nova chave de caractere.

O exemplo a seguir trata o [KeyPress](#) evento para alterar qualquer uma A a das chaves de caractere e para ! :

```
private void textBox2_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar == 'a' || e.KeyChar == 'A')
    {
        e.KeyChar = '!';
        e.Handled = false;
    }
}
```

```
Private Sub TextBox2_KeyPress(sender As Object, e As KeyPressEventArgs)
    If e.KeyChar = "a"c Or e.KeyChar = "A"c Then
        e.KeyChar = "!"c
        e.Handled = False
    End If
End Sub
```

Modificar uma chave que não seja de caractere

Você só pode modificar pressionamentos de tecla não caractere, herdando do controle e substituindo o [PreProcessMessage](#) método. Como a entrada [Message](#) é enviada para o controle, ela é processada antes de o controle gerar eventos. Você pode interceptar essas mensagens para modificá-las ou bloqueá-las.

O exemplo de código a seguir demonstra como usar a [WParam](#) Propriedade do [Message](#) parâmetro para alterar a tecla pressionada. Esse código detecta uma chave de F1 a F10 e converte a chave em uma chave numérica que varia de 0 a 9 (em que F10 é mapeado para 0).

```
public override bool PreProcessMessage(ref Message m)
{
    const int WM_KEYDOWN = 0x100;

    if (m.Msg == WM_KEYDOWN)
    {
        Keys keyCode = (Keys)m.WParam & Keys.KeyCode;

        // Detect F1 through F9.
        m.WParam = keyCode switch
        {
            Keys.F1 => (IntPtr)Keys.D1,
            Keys.F2 => (IntPtr)Keys.D2,
            Keys.F3 => (IntPtr)Keys.D3,
            Keys.F4 => (IntPtr)Keys.D4,
            Keys.F5 => (IntPtr)Keys.D5,
            Keys.F6 => (IntPtr)Keys.D6,
            Keys.F7 => (IntPtr)Keys.D7,
            Keys.F8 => (IntPtr)Keys.D8,
            Keys.F9 => (IntPtr)Keys.D9,
            Keys.F10 => (IntPtr)Keys.D0,
            _ => m.WParam
        };
    }

    // Send all other messages to the base method.
    return base.PreProcessMessage(ref m);
}
```



```

Public Overrides Function PreProcessMessage(ByRef m As Message) As Boolean

    Const WM_KEYDOWN = &H100

    If m.Msg = WM_KEYDOWN Then
        Dim keyCode As Keys = CType(m.WParam, Keys) And Keys.KeyCode

        Select Case keyCode
            Case Keys.F1 : m.WParam = CType(Keys.D1, IntPtr)
            Case Keys.F2 : m.WParam = CType(Keys.D2, IntPtr)
            Case Keys.F3 : m.WParam = CType(Keys.D3, IntPtr)
            Case Keys.F4 : m.WParam = CType(Keys.D4, IntPtr)
            Case Keys.F5 : m.WParam = CType(Keys.D5, IntPtr)
            Case Keys.F6 : m.WParam = CType(Keys.D6, IntPtr)
            Case Keys.F7 : m.WParam = CType(Keys.D7, IntPtr)
            Case Keys.F8 : m.WParam = CType(Keys.D8, IntPtr)
            Case Keys.F9 : m.WParam = CType(Keys.D9, IntPtr)
            Case Keys.F10 : m.WParam = CType(Keys.D0, IntPtr)
        End Select
    End If

    Return MyBase.PreProcessMessage(m)
End Function

```

Confira também

- [Visão geral do uso do teclado \(Windows Forms .NET\)](#)
- [Usando eventos de teclado \(Windows Forms .NET\)](#)
- [Keys](#)
- [KeyDown](#)
- [KeyPress](#)

Como verificar se há pressionamentos de tecla modificadores (Windows Forms .NET)

14/05/2021 • 2 minutes to read

Como o usuário digita chaves em seu aplicativo, você pode monitorar as teclasificadoras pressionadas, como Shift, ALT e Ctrl. Quando uma tecla modificadora é pressionada em combinação com outras chaves ou até mesmo um clique do mouse, seu aplicativo pode responder adequadamente. Por exemplo, pressionar a tecla S pode fazer com que um "S" apareça na tela. Se as teclas Ctrl + S forem pressionadas, em vez disso, o documento atual poderá ser salvo.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Se você manipular o [KeyDown](#) evento, a [KeyEventArgs.Modifiers](#) Propriedade recebida pelo manipulador de eventos especifica quais teclasificadoras são pressionadas. Além disso, a [KeyEventArgs.KeyData](#) propriedade especifica o caractere que foi pressionado junto com quaisquer chaves de modificador combinadas com um OR-bit or.

Se você estiver manipulando o [KeyPress](#) evento ou um evento do mouse, o manipulador de eventos não receberá essas informações. Use a [ModifierKeys](#) propriedade da [Control](#) classe para detectar um modificador de chave. Em ambos os casos, você deve executar uma e/ou um [Keys](#) valor apropriado e o valor que você está testando. A [Keys](#) Enumeração oferece variações de cada chave de modificador, portanto, é importante que você faça o bit e verifique com o valor correto.

Por exemplo, a tecla Shift é representada pelos seguintes valores de chave:

- [Keys.Shift](#)
- [Keys.ShiftKey](#)
- [Keys.RShiftKey](#)
- [Keys.LShiftKey](#)

O valor correto para testar o deslocamento como uma chave de modificador é [Keys.Shift](#). Da mesma forma, para testar Ctrl e ALT como modificadores, você deve usar os [Keys.Control](#) [Keys.Alt](#) valores e, respectivamente.

Detectar chave de modificador

Detecte se uma tecla modificadora é pressionada, comparando a [ModifierKeys](#) propriedade e o [Keys](#) valor de enumeração com um operador and bit a bit.

O exemplo de código a seguir mostra como determinar se a tecla Shift é pressionada dentro dos [KeyPress](#) [KeyDown](#) manipuladores de eventos e.

```
// Event only raised when non-modifier key is pressed
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    if ((Control.ModifierKeys & Keys.Shift) == Keys.Shift)
        MessageBox.Show("KeyPress " + Keys.Shift);
}

// Event raised as soon as shift is pressed
private void textBox1_KeyDown(object sender, KeyEventArgs e)
{
    if ((Control.ModifierKeys & Keys.Shift) == Keys.Shift)
        MessageBox.Show("KeyDown " + Keys.Shift);
}
```

```
' Event only raised when non-modifier key is pressed
Private Sub TextBox1_KeyPress(sender As Object, e As KeyPressEventArgs)
    If ((Control.ModifierKeys And Keys.Shift) = Keys.Shift) Then
        MessageBox.Show("KeyPress " & [Enum].GetName(GetType(Keys), Keys.Shift))
    End If
End Sub

' Event raised as soon as shift is pressed
Private Sub TextBox1_KeyDown(sender As Object, e As KeyEventArgs)
    If ((Control.ModifierKeys And Keys.Shift) = Keys.Shift) Then
        MessageBox.Show("KeyPress " & [Enum].GetName(GetType(Keys), Keys.Shift))
    End If
End Sub
```

Confira também

- [Visão geral do uso do teclado \(Windows Forms .NET\)](#)
- [Usando eventos de teclado \(Windows Forms .NET\)](#)
- [Keys](#)
- [ModifierKeys](#)
- [KeyDown](#)
- [KeyPress](#)

Como lidar com mensagens de entrada de teclado no formulário (Windows Forms .NET)

14/05/2021 • 2 minutes to read

Os Windows Forms proporcionam a capacidade de manipular mensagens de teclado no nível do formulário, antes que as mensagens cheguem a um controle. Este artigo mostra como realizar essa tarefa.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Manipular uma mensagem do teclado

Manipule o [KeyPress](#) [KeyDown](#) evento ou do formulário ativo e defina a [KeyPreview](#) Propriedade do formulário como `true`. Essa propriedade faz com que o teclado seja recebido pelo formulário antes que eles atinjam qualquer controle no formulário. O exemplo de código a seguir trata o [KeyPress](#) evento detectando todas as chaves de número e consumindo 1, 4 e 7.

```
// Detect all numeric characters at the form level and consume 1,4, and 7.
// Form.KeyPreview must be set to true for this event handler to be called.
void Form1_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar >= 48 && e.KeyChar <= 57)
    {
        MessageBox.Show($"Form.KeyPress: '{e.KeyChar}' pressed.");

        switch (e.KeyChar)
        {
            case (char)49:
            case (char)52:
            case (char)55:
                MessageBox.Show($"Form.KeyPress: '{e.KeyChar}' consumed.");
                e.Handled = true;
                break;
        }
    }
}
```

```
' Detect all numeric characters at the form level and consume 1,4, and 7.
' Form.KeyPreview must be set to true for this event handler to be called.
Private Sub Form1_KeyPress(sender As Object, e As KeyPressEventArgs)
    If e.KeyChar >= Chr(48) And e.KeyChar <= Chr(57) Then
        MessageBox.Show($"Form.KeyPress: '{e.KeyChar}' pressed.")

        Select Case e.KeyChar
            Case Chr(49), Chr(52), Chr(55)
                MessageBox.Show($"Form.KeyPress: '{e.KeyChar}' consumed.")
                e.Handled = True
        End Select
    End If
End Sub
```

Confira também

- [Visão geral do uso do teclado \(Windows Forms .NET\)](#)
- [Usando eventos de teclado \(Windows Forms .NET\)](#)
- [Keys](#)
- [ModifierKeys](#)
- [KeyDown](#)
- [KeyPress](#)

Como simular eventos de teclado (Windows Forms .NET)

14/05/2021 • 3 minutes to read

Windows Forms fornece algumas opções para simular de forma programática a entrada do teclado. Este artigo fornece uma visão geral dessas opções.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Usar SendKeys

Windows Forms fornece a [System.Windows.Forms.SendKeys](#) classe para enviar pressionamentos de teclas para o aplicativo ativo. Há dois métodos para enviar pressionamentos de teclas para um aplicativo: [SendKeys.Send](#) e [SendKeys.SendWait](#). A diferença entre os dois métodos é que `SendWait` o bloqueia o thread atual quando o pressionamento de tecla é enviado, aguardando uma resposta, embora `Send` não. Para obter mais informações sobre `SendWait`, consulte [para enviar um pressionamento de tecla para um aplicativo diferente](#).

Caution

Se seu aplicativo for destinado ao uso internacional com uma variedade de teclados, o uso do [SendKeys.Send](#) pode gerar resultados imprevisíveis e deve ser evitado.

Nos bastidores, o `SendKeys` usa uma implementação mais antiga do Windows para enviar entrada, o que pode falhar em janelas modernas, em que é esperado que o aplicativo não esteja sendo executado com direitos administrativos. Se a implementação mais antiga falhar, o código tentará automaticamente a implementação mais recente do Windows para enviar a entrada. Além disso, quando a [SendKeys](#) classe usa a nova implementação, o [SendWait](#) método não bloqueia mais o thread atual ao enviar pressionamentos de teclas para outro aplicativo.

IMPORTANT

Se seu aplicativo depender de um comportamento consistente independentemente do sistema operacional, você poderá forçar a [SendKeys](#) a usar a nova implementação adicionando a seguinte configuração de aplicativo ao arquivo de `app.config`.

```
<appSettings>
  <add key="SendKeys" value="SendInput"/>
</appSettings>
```

Para forçar a [SendKeys](#) classe a usar *apenas* a implementação anterior, use o valor `"JournalHook"` em vez disso.

Para enviar um pressionamento de tecla para o mesmo aplicativo

Chame o [SendKeys.Send](#) ou [SendKeys.SendWait](#) método ou da [SendKeys](#) classe. Os pressionamentos de teclas especificados serão recebidos pelo controle ativo do aplicativo.

O exemplo de código a seguir usa `Send` para simular o pressionamento das teclas ALT e seta para baixo. Esses pressionamentos de teclas fazem com que o [ComboBox](#) controle exiba seu menu suspenso. Este exemplo pressupõe um [Form](#) com [Button](#) e [ComboBox](#).

```
private void button1_Click(object sender, EventArgs e)
{
    comboBox1.Focus();
    SendKeys.Send("%+{DOWN}");
}
```

```
Private Sub Button1_Click(sender As Object, e As EventArgs)
    ComboBox1.Focus()
    SendKeys.Send("%+{DOWN}")
End Sub
```

Para enviar um pressionamento de tecla para um aplicativo diferente

Os [SendKeys.Send](#), [SendKeys.SendWait](#) métodos e enviam pressionamentos de teclas para o aplicativo ativo, que geralmente é o aplicativo do qual você está enviando pressionamentos de teclas. Para enviar pressionamentos de teclas para outro aplicativo, primeiro você precisa ativá-lo. Como não há nenhum método gerenciado para ativar outro aplicativo, você deve usar métodos nativos do Windows para concentrar o outro aplicativo. O exemplo de código a seguir usa a invocação de plataforma para chamar os `FindWindow` `SetForegroundWindow` métodos e para ativar a janela do aplicativo de calculadora e, em seguida, chama `Send` para emitir uma série de cálculos para o aplicativo de calculadora.

O exemplo de código a seguir usa `Send` para simular o pressionamento de teclas no aplicativo Calculadora do Windows 10. Ele primeiro pesquisa uma janela de aplicativo com o título `Calculator` e, em seguida, ativa-a. Depois de ativado, os pressionamentos de tecla são enviados para calcular 10 mais 10.

```
[DllImport("USER32.DLL", CharSet = CharSet.Unicode)]
public static extern IntPtr FindWindow(string lpClassName, string lpWindowName);

[DllImport("USER32.DLL")]
public static extern bool SetForegroundWindow(IntPtr hWnd);

private void button1_Click(object sender, EventArgs e)
{
    IntPtr calcWindow = FindWindow(null, "Calculator");

    if (SetForegroundWindow(calcWindow))
        SendKeys.Send("10{+}10=");
}
```

```
<Runtime.InteropServices.DllImport("USER32.DLL", CharSet:=Runtime.InteropServices.CharSet.Unicode)>
Public Shared Function FindWindow(lpClassName As String, lpWindowName As String) As IntPtr : End Function

<Runtime.InteropServices.DllImport("USER32.DLL")>
Public Shared Function SetForegroundWindow(hWnd As IntPtr) As Boolean : End Function

Private Sub Button1_Click(sender As Object, e As EventArgs)
    Dim hCalcWindow As IntPtr = FindWindow(Nothing, "Calculator")

    If SetForegroundWindow(hCalcWindow) Then
        SendKeys.Send("10{+}10=")
    End If
End Sub
```

Usar métodos OnEventName

A maneira mais fácil de simular eventos de teclado é chamar um método no objeto que gera o evento. A maioria dos eventos tem um método correspondente que os invoca, nomeado no padrão de `On` seguido por

`EventName` , como `OnKeyPress` . Essa opção só é possível em controles ou formulários personalizados, pois esses métodos são protegidos e não podem ser acessados de fora do contexto do controle ou do formulário.

Esses métodos protegidos estão disponíveis para simular eventos de teclado.

- `OnKeyDown`
- `OnKeyPress`
- `OnKeyUp`

Para obter mais informações sobre esses eventos, consulte [usando eventos de teclado \(Windows Forms .net\)](#).

Confira também

- [Visão geral do uso do teclado \(Windows Forms .NET\)](#)
- [Usando eventos de teclado \(Windows Forms .NET\)](#)
- [Usando eventos de mouse \(Windows Forms .NET\)](#)
- [SendKeys](#)
- [Keys](#)
- [KeyDown](#)
- [KeyPress](#)

Visão geral do uso do mouse (Windows Forms .NET)

14/05/2021 • 4 minutes to read

Receber e manipular entradas de mouse é uma parte importante qualquer aplicativo do Windows. Você pode manipular eventos do mouse para executar uma ação em seu aplicativo ou usar as informações de local do mouse para executar testes de clique ou outras ações. Além disso, você pode alterar a maneira como os controles em seu aplicativo manipulam a entrada do mouse. Este artigo descreve detalhadamente esses eventos de mouse e como obter e alterar as configurações do sistema para o mouse.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

No Windows Forms, a entrada do usuário é enviada aos aplicativos na forma de [mensagens do Windows](#). Uma série de métodos substituíveis processam essas mensagens no nível do aplicativo, formulário e controle. Quando esses métodos recebem mensagens do mouse, eles geram eventos que podem ser tratados para obter informações sobre a entrada do mouse. Em muitos casos, Windows Forms aplicativos podem processar toda a entrada do usuário simplesmente tratando esses eventos. Em outros casos, um aplicativo pode substituir um dos métodos que processam mensagens para interceptar uma mensagem específica antes de ele ser recebido pelo aplicativo, formulário ou controle.

Eventos de mouse

Todos os controles do Windows Forms herdam um conjunto de eventos relacionados às entradas de mouse e teclado. Por exemplo, um controle pode manipular o [MouseDown](#) evento para determinar o local de um clique do mouse. Para obter mais informações sobre os eventos de mouse, consulte [usando eventos de mouse](#).

Localização do mouse e teste de colisão

Quando o usuário move o mouse, o sistema operacional move o ponteiro do mouse. O ponteiro do mouse contém um único pixel, chamado de ponto de acesso, que o sistema operacional rastreia e reconhece como a posição do ponteiro. Quando o usuário move o mouse ou pressiona um botão do mouse, o [Control](#) que contém o [HotSpot](#) gera o evento de mouse apropriado.

Você pode obter a posição atual do mouse com a [Location](#) Propriedade do [MouseEventArgs](#) ao manipular um evento do mouse ou usando a [Position](#) propriedade da [Cursor](#) classe. Você pode usar as informações de local do mouse para realizar testes de colisão e, em seguida, executar uma ação com base no local do mouse. O recurso de teste de colisão é integrado a vários controles em Windows Forms, como [ListView](#) os [TreeView](#) [MonthCalendar](#) controles, e [DataGridView](#).

Usado com o evento de mouse apropriado, [MouseHover](#) por exemplo, o teste de colisão é muito útil para determinar quando o aplicativo deve realizar uma ação específica.

Alterando as configurações de entrada do mouse

Você pode detectar e alterar a maneira como um controle manipula a entrada do mouse derivando do controle e usando [GetStyle](#) os [SetStyle](#) métodos e. O [SetStyle](#) método usa uma combinação de valores bit-a-bit [ControlStyles](#) para determinar se o controle terá um clique padrão, um comportamento de clique duplo ou se o

controle manipulará seu próprio processamento de mouse. Além disso, a [SystemInformation](#) classe inclui propriedades que descrevem os recursos do mouse e especificam como o mouse interage com o sistema operacional. A tabela a seguir resume essas propriedades.

PROPRIEDADE	DESCRIÇÃO
DoubleClickSize	Obtém as dimensões, em pixels, da área em que o usuário deve clicar duas vezes para que o sistema operacional considere os dois cliques um clique duplo.
DoubleClickTime	Obtém o número máximo de milissegundos que podem decorrer entre um primeiro clique e um segundo clique para que a ação do mouse seja considerada um clique duplo.
MouseButtons	Obtém o número de botões do mouse.
MouseButtonsSwapped	Obtém um valor que indica se as funções dos botões esquerdo e direito do mouse foram trocadas.
MouseHoverSize	Obtém as dimensões, em pixels, do retângulo no qual o ponteiro do mouse deve permanecer pelo tempo de foco do mouse antes que uma mensagem de foco do mouse seja gerada.
MouseHoverTime	Obtém o tempo, em milissegundos, que o ponteiro do mouse deve permanecer no retângulo de foco antes que uma mensagem de foco do mouse seja gerada.
MousePresent	Obtém um valor que indica se um mouse está instalado.
MouseSpeed	Obtém um valor que indica a velocidade atual do mouse, de 1 a 20.
MouseWheelPresent	Obtém um valor que indica se um mouse com botão de rolagem está instalado.
MouseWheelScrollDelta	Obtém o valor delta do incremento de uma única rotação da roda do mouse.
MouseWheelScrollLines	Obtém o número de linhas a rolar quando o botão de rolagem do mouse é girado.

Métodos que processam mensagens de entrada do usuário

Formulários e controles têm acesso à [IMessageFilter](#) interface e um conjunto de métodos substituíveis que processam mensagens do Windows em pontos diferentes na fila de mensagens. Todos esses métodos têm um [Message](#) parâmetro, que encapsula os detalhes de baixo nível das mensagens do Windows. É possível implementar ou substituir esses métodos para analisar a mensagem e, então, consumi-la ou passá-la para o próximo consumidor na fila de mensagens. A tabela a seguir apresenta os métodos que processam todas as mensagens do Windows no Windows Forms.

MÉTODO	OBSERVAÇÕES
--------	-------------

MÉTODO	OBSERVAÇÕES
PreFilterMessage	Este método intercepta mensagens do Windows que estão na fila (também chamadas de postadas) no nível do aplicativo.
PreProcessMessage	Este método intercepta as mensagens do Windows no nível do formulário e do controle antes que elas sejam processadas.
WndProc	Este método processa mensagens do Windows no nível do formulário e do controle.
DefWndProc	Esse método realiza o processamento padrão de mensagens do Windows no nível do formulário e do controle. Isso fornece a funcionalidade mínima de uma janela.
OnNotifyMessage	Esse método intercepta mensagens no nível de formulário e de controle, depois que elas tiverem sido processadas. O EnableNotifyMessage bit de estilo deve ser definido para que este método seja chamado.

Confira também

- [Usando eventos de mouse \(Windows Forms .NET\)](#)
- [Visão geral do comportamento do mouse do tipo "arrastar e soltar" \(Windows Forms .NET\)](#)
- [Gerenciar ponteiros do mouse \(Windows Forms .NET\)](#)

Usando eventos de mouse (Windows Forms .NET)

14/05/2021 • 7 minutes to read

A maioria dos programas Windows Forms processa a entrada do mouse manipulando os eventos do mouse. Este artigo fornece uma visão geral dos eventos do mouse, incluindo detalhes sobre quando usar cada evento e os dados que são fornecidos para cada evento. Para obter mais informações sobre eventos em geral, consulte [visão geral de eventos \(Windows Forms .net\)](#).

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Eventos de mouse

A principal maneira de responder a entradas de mouse é manipular eventos de mouse. A tabela a seguir mostra os eventos de mouse e descreve quando eles são gerados.

EVENTOS DE MOUSE	DESCRIÇÃO
Click	Esse evento ocorre quando o botão do mouse é liberado, normalmente antes do MouseDown evento. O manipulador para esse evento recebe um argumento do tipo EventArgs . Manipule este evento quando você só precisar determinar quando um clique ocorrer.
MouseDown	Este evento ocorre quando o usuário clica no controle com o mouse. O manipulador para esse evento recebe um argumento do tipo MouseEventArgs . Manipule este evento quando você precisar obter informações sobre o mouse quando um clique ocorrer.
DoubleClick	Este evento ocorre quando um usuário clica duas vezes no controle. O manipulador para esse evento recebe um argumento do tipo EventArgs . Manipule este evento quando você só precisar determinar quando um clique duplo ocorrer.
MouseDoubleClick	Este evento ocorre quando o usuário clica duas vezes no controle com o mouse. O manipulador para esse evento recebe um argumento do tipo MouseEventArgs . Manipule este evento quando você precisar obter informações sobre o mouse quando um clique duplo ocorrer.
MouseDown	Este evento ocorre quando o ponteiro do mouse está sobre o controle e um botão do mouse é pressionado. O manipulador para esse evento recebe um argumento do tipo MouseEventArgs .
MouseEnter	Este evento ocorre quando o ponteiro do mouse entra na borda ou na área de cliente do controle, dependendo do tipo de controle. O manipulador para esse evento recebe um argumento do tipo EventArgs .

EVENTOS DE MOUSE	DESCRIÇÃO
MouseHover	Este evento ocorre quando o ponteiro do mouse para e permanece sobre o controle. O manipulador para esse evento recebe um argumento do tipo EventArgs .
MouseLeave	Este evento ocorre quando o ponteiro do mouse deixa a borda ou a área de cliente do controle, dependendo do tipo de controle. O manipulador para esse evento recebe um argumento do tipo EventArgs .
MouseMove	Este evento ocorre quando o ponteiro do mouse se move enquanto está sobre um controle. O manipulador para esse evento recebe um argumento do tipo MouseEventArgs .
MouseUp	Este evento ocorre quando o ponteiro do mouse está sobre o controle e o usuário libera um botão do mouse. O manipulador para esse evento recebe um argumento do tipo MouseEventArgs .
MouseWheel	Este evento ocorre quando o usuário gira o botão de rolagem do mouse enquanto o controle está em foco. O manipulador para esse evento recebe um argumento do tipo MouseEventArgs . Você pode usar a Delta propriedade de MouseEventArgs para determinar a distância com que o mouse foi rolado.

Informações do mouse

Um [MouseEventArgs](#) é enviado aos manipuladores de eventos do mouse relacionados ao clique de um botão do mouse e do controle dos movimentos do mouse. [MouseEventArgs](#) fornece informações sobre o estado atual do mouse, incluindo o local do ponteiro do mouse em coordenadas do cliente, quais botões do mouse são pressionados e se a roda do mouse foi rolada. Vários eventos de mouse, como aqueles que são gerados quando o ponteiro do mouse digita ou deixava os limites de um controle, enviam um [EventArgs](#) para o manipulador de eventos sem informações adicionais.

Se você quiser saber o estado atual dos botões do mouse ou o local do ponteiro do mouse e deseja evitar manipular um evento do mouse, também poderá usar as [MouseButtons](#) [MousePosition](#) Propriedades e da [Control](#) classe. [MouseButtons](#) Retorna informações sobre quais botões do mouse estão pressionados no momento. O [MousePosition](#) retorna as coordenadas de tela do ponteiro do mouse e é equivalente ao valor retornado por [Position](#).

Convertendo entre coordenadas de cliente e da tela

Como algumas informações de localização do mouse estão em coordenadas de cliente e algumas estão em coordenadas de tela, talvez seja necessário converter um ponto de um sistema de coordenadas para outro. Você pode fazer isso facilmente usando os [PointToClient](#) métodos e [PointToScreen](#) disponíveis na [Control](#) classe.

Comportamento de evento de clique padrão

Se quiser manipular eventos de clique do mouse na ordem correta, você precisará conhecer a ordem em que os eventos de clique são gerados em controles dos Windows Forms. Todos os controles de Windows Forms geram eventos de clique na mesma ordem quando qualquer botão do mouse com suporte é pressionado e liberado, exceto quando indicado na lista a seguir para controles individuais. A lista a seguir mostra a ordem dos eventos gerados por um único clique do botão do mouse:

1. [MouseDown](#) .
2. [Click](#) .
3. [MouseClicked](#) .
4. [MouseUp](#) .

A seguir está a ordem dos eventos gerados para um clique duplo com botão do mouse:

1. [MouseDown](#) .
2. [Click](#) .
3. [MouseClicked](#) .
4. [MouseUp](#) .
5. [MouseDown](#) .
6. [DoubleClick](#) .

Isso pode variar, dependendo se o controle em questão tem o bit de [StandardDoubleClick](#) estilo definido como `true` . Para obter mais informações sobre como definir um [ControlStyles](#) bit, consulte o [SetStyle](#) método.

7. [MouseDoubleClick](#) .
8. [MouseUp](#) .

Controles individuais

Os seguintes controles não estão em conformidade com o comportamento de evento de clique padrão do mouse:

- [Button](#)
- [CheckBox](#)
- [ComboBox](#)
- [RadioButton](#)

NOTE

Para o [ComboBox](#) controle, o comportamento de evento detalhado posteriormente ocorrerá se o usuário clicar no campo de edição, no botão ou em um item dentro da lista.

- Clique com o botão esquerdo: [Click](#) , [MouseClicked](#)
- Clique com o botão direito do mouse: nenhum evento de clique gerado
- Clique duas vezes com o botão esquerdo: [Click](#) , [MouseClicked](#) ; [Click](#) , [MouseClicked](#)
- Duplo clique com o botão direito: nenhum evento de clique gerado
- [TextBox](#) controles,,, [RichTextBox](#) [ListBox](#) [MaskedTextBox](#) e [CheckedListBox](#)

NOTE

O comportamento do evento detalhado a seguir ocorre quando o usuário clica em qualquer lugar desses controles.

- Clique com o botão esquerdo: [Click](#) , [MouseClicked](#)
- Clique com o botão direito do mouse: nenhum evento de clique gerado

- Clique duas vezes com o botão esquerdo: [Click](#) ,, [MouseClicked](#) [DoubleClick](#) , [MouseDoubleClick](#)
- Duplo clique com o botão direito: nenhum evento de clique gerado
- Controle [ListView](#)

NOTE

O comportamento de evento detalhado posteriormente ocorre somente quando o usuário clica nos itens no [ListView](#) controle. Nenhum evento é gerado para cliques em qualquer outro lugar no controle. Além dos eventos descritos posteriormente, há os [BeforeLabelEdit](#) [AfterLabelEdit](#) eventos e, que podem ser de seu interesse se você quiser usar a validação com o [ListView](#) controle.

- Clique com o botão esquerdo: [Click](#) , [MouseClicked](#)
- Clique com o botão direito do mouse em [Click](#) : [MouseClicked](#)
- Clique duas vezes com o botão esquerdo: [Click](#) , [MouseClicked](#) ; [DoubleClick](#) , [MouseDoubleClick](#)
- Clique duas vezes com o botão direito do mouse: [Click](#) , [MouseClicked](#) ; [DoubleClick](#) , [MouseDoubleClick](#)
- Controle [TreeView](#)

NOTE

O comportamento de evento detalhado posteriormente ocorre somente quando o usuário clica nos itens em si ou à direita dos itens no [TreeView](#) controle. Nenhum evento é gerado para cliques em qualquer outro lugar no controle. Além daqueles descritos posteriormente, há os [BeforeCheck](#) eventos,,, [BeforeSelect](#) , [BeforeLabelEdit](#) [AfterSelect](#) [AfterCheck](#) e [AfterLabelEdit](#) , que podem ser de seu interesse se você quiser usar a validação com o [TreeView](#) controle.

- Clique com o botão esquerdo: [Click](#) , [MouseClicked](#)
- Clique com o botão direito do mouse em [Click](#) : [MouseClicked](#)
- Clique duas vezes com o botão esquerdo: [Click](#) , [MouseClicked](#) ; [DoubleClick](#) , [MouseDoubleClick](#)
- Clique duas vezes com o botão direito do mouse: [Click](#) , [MouseClicked](#) ; [DoubleClick](#) , [MouseDoubleClick](#)

Comportamento de pintura de controles de alternância

Os controles de alternância, como os controles derivados da [ButtonBase](#) classe, têm o seguinte comportamento de pintura distinto em combinação com eventos de clique do mouse:

1. O usuário pressiona o botão do mouse.
2. O controle pinta no estado pressionado.
3. O [MouseDown](#) evento é gerado.
4. O usuário libera o botão do mouse.
5. O controle pinta no estado elevado.
6. O [Click](#) evento é gerado.
7. O [MouseClicked](#) evento é gerado.
8. O [MouseUp](#) evento é gerado.

NOTE

Se o usuário mover o ponteiro para fora do controle de alternância enquanto o botão do mouse estiver inoperante (como mover o mouse para fora do [Button](#) controle enquanto ele é pressionado), o controle de alternância será pintado no estado gerado e apenas o [MouseUp](#) evento ocorrerá. Os [Click](#) [MouseClick](#) eventos ou não ocorrerão nessa situação.

Confira também

- [Visão geral do uso do mouse \(Windows Forms .NET\)](#)
- [Gerenciar ponteiros do mouse \(Windows Forms .NET\)](#)
- [Como simular eventos do mouse \(Windows Forms .NET\)](#)
- [System.Windows.Forms.Control](#)

Visão geral do comportamento do mouse do tipo "arrastar e soltar" (Windows Forms .NET)

14/05/2021 • 3 minutes to read

O Windows Forms incluem um conjunto de métodos, eventos e classes que implementam o comportamento do tipo "arrastar e soltar". Este tópico fornece uma visão geral do suporte a arrastar e soltar no Windows Forms.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Eventos de arrastar e soltar

Há duas categorias de eventos em uma operação de arrastar e soltar: os eventos que ocorrem no destino atual da operação do tipo "arrastar e soltar" e eventos que ocorrem na fonte da operação de arrastar e soltar. Para executar operações de arrastar e soltar, você deve lidar com esses eventos. Trabalhando com as informações disponíveis no evento argumentos desses eventos, você pode facilmente orientar as operações do tipo "arrastar e soltar".

Eventos no destino de soltura atual

A tabela a seguir mostra os eventos que ocorrem no destino atual de uma operação do tipo "arrastar e soltar".

EVENTOS DE MOUSE	DESCRIÇÃO
DragEnter	Esse evento ocorre quando um objeto é arrastado para os limites do controle. O manipulador para esse evento recebe um argumento do tipo DragEventArgs .
DragOver	Esse evento ocorre quando um objeto é arrastado enquanto o ponteiro do mouse está dentro dos limites do controle. O manipulador para esse evento recebe um argumento do tipo DragEventArgs .
DragDrop	Esse evento ocorre quando uma operação do tipo "arrastar e soltar" é concluída. O manipulador para esse evento recebe um argumento do tipo DragEventArgs .
DragLeave	Esse evento ocorre quando um objeto é arrastado para fora dos limites do controle. O manipulador para esse evento recebe um argumento do tipo EventArgs .

A [DragEventArgs](#) classe fornece o local do ponteiro do mouse, o estado atual dos botões do mouse e as teclas modificadoras do teclado, os dados que estão sendo arrastados e os [DragDropEffects](#) valores que especificam as operações permitidas pela origem do evento de arrastar e o efeito de soltar destino para a operação.

Eventos na fonte de soltar

A tabela a seguir mostra os eventos que ocorrem na fonte atual de uma operação do tipo "arrastar e soltar".

EVENTOS DE MOUSE	DESCRIÇÃO
GiveFeedback	Esse evento ocorre durante uma operação de arrastar. Ele fornece uma indicação visual para o usuário que a operação do tipo "arrastar e soltar" está ocorrendo, como uma alteração no ponteiro do mouse. O manipulador para esse evento recebe um argumento do tipo GiveFeedbackEventArgs .
QueryContinueDrag	Esse evento é gerado durante uma operação do tipo "arrastar e soltar" e permite que a fonte de arrastar determine se a operação do tipo "arrastar e soltar" deve ser cancelada. O manipulador para esse evento recebe um argumento do tipo QueryContinueDragEventArgs .

A [QueryContinueDragEventArgs](#) classe fornece o estado atual dos botões do mouse e as teclas de modificador do teclado, um valor que especifica se a tecla ESC foi pressionada e um [DragAction](#) valor que pode ser definido para especificar se a operação de arrastar e soltar deve continuar.

Executando o recurso de arrastar e soltar

As operações de arrastar e soltar sempre envolvem dois componentes, a **origem de arrastar** e o destino de **soltar**. Para iniciar uma operação de arrastar e soltar, designe um controle como a origem e manipule o [MouseDown](#) evento. No manipulador de eventos, chame o [DoDragDrop](#) método que fornece os dados associados com a queda e o [DragDropEffects](#) valor a.

Defina a propriedade do controle de destino [AllowDrop](#) definida como `true` para permitir que o controle aceite uma operação de arrastar e soltar. O destino lida com dois eventos, primeiro um evento em resposta à arraste que está sobre o controle, como [DragOver](#). E um segundo evento que é a própria ação de soltar, [DragDrop](#).

O exemplo a seguir demonstra um arraste de um [Label](#) controle para um [TextBox](#). Quando a operação de arrastar é concluída, o `TextBox` responde atribuindo o texto do rótulo a si mesmo.

```
// Initiate the drag
private void label1_MouseDown(object sender, MouseEventArgs e) =>
    DoDragDrop(((Label)sender).Text, DragDropEffects.All);

// Set the effect filter and allow the drop on this control
private void textBox1_DragOver(object sender, DragEventArgs e) =>
    e.Effect = DragDropEffects.All;

// React to the drop on this control
private void textBox1_DragDrop(object sender, DragEventArgs e) =>
    textBox1.Text = (string)e.Data.GetData(typeof(string));
```

```
' Initiate the drag
Private Sub Label1_MouseDown(sender As Object, e As MouseEventArgs)
    DoDragDrop(DirectCast(sender, Label).Text, DragDropEffects.All)
End Sub

' Set the effect filter and allow the drop on this control
Private Sub TextBox1_DragOver(sender As Object, e As DragEventArgs)
    e.Effect = DragDropEffects.All
End Sub

' React to the drop on this control
Private Sub TextBox1_DragDrop(sender As Object, e As DragEventArgs)
    TextBox1.Text = e.Data.GetData(GetType(String))
End Sub
```

Para obter mais informações sobre os efeitos de arrastar, consulte [Data](#) e [AllowedEffect](#) .

Confira também

- [Visão geral do uso do mouse \(Windows Forms .NET\)](#)
- [Control.DragDrop](#)
- [Control.DragEnter](#)
- [Control.DragLeave](#)
- [Control.DragOver](#)

Como distinguir entre cliques e cliques duplos (Windows Forms .NET)

14/05/2021 • 4 minutes to read

Normalmente, um único *clique* inicia uma ação de interface do usuário e um *clique duas vezes* estende a ação. Por exemplo, um clique normalmente seleciona um item e um clique duplo edita o item selecionado. No entanto, os eventos de clique Windows Forms não acomodam facilmente um cenário em que um clique e um clique duplo executam ações incompatíveis, porque uma ação vinculada ao [Click MouseClick](#) evento ou é executada antes da ação associada ao [DoubleClick MouseDoubleClick](#) evento ou. Este tópico demonstra duas soluções para esse problema.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Uma solução é manipular o evento de clique duplo e reverter as ações no tratamento de evento de clique. Em raras situações, talvez seja necessário simular o comportamento de clicar e clicar duas vezes manipulando o [MouseDown](#) evento e usando as [DoubleClickTime](#) [DoubleClickSize](#) Propriedades e da [SystemInformation](#) classe. Você mede o tempo entre cliques e, se um segundo clique ocorrer antes de o valor de [DoubleClickTime](#) ser atingido e o clique estiver dentro de um retângulo definido por [DoubleClickSize](#), execute a ação de clique duplo; caso contrário, execute a ação de clique.

Reverter uma ação de clique

Verifique se o controle em que você está trabalhando tem um comportamento de clique duplo padrão. Caso contrário, habilite o controle com o [SetStyle](#) método. Trate o evento de clique duplo e reverta a ação de clique, bem como a ação de clique duplo. O exemplo de código a seguir demonstra um como criar um botão personalizado com clique duplo habilitado, bem como reverter a ação de clique no código de tratamento de eventos de clique duplo.

Este exemplo de código usa um novo controle de botão que permite cliques duplos:

```
public partial class DoubleClickButton : Button
{
    public DoubleClickButton()
    {
        // Set the style so a double click event occurs.
        SetStyle(ControlStyles.StandardClick | ControlStyles.StandardDoubleClick, true);
    }
}
```

```
Public Class DoubleClickButton : Inherits Button

    Public Sub New()
        SetStyle(ControlStyles.StandardClick Or ControlStyles.StandardDoubleClick, True)
    End Sub

End Class
```

O código a seguir demonstra como um formulário altera o estilo de borda com base em um clique ou clique

duas vezes no novo controle de botão:

```
public partial class Form1 : Form
{
    private FormBorderStyle _initialStyle;
    private bool _isDoubleClicking;

    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        _initialStyle = this.FormBorderStyle;

        var button1 = new DoubleClickButton();
        button1.Location = new Point(50, 50);
        button1.Size = new Size(200, 23);
        button1.Text = "Click or Double Click";
        button1.Click += Button1_Click;
        button1.DoubleClick += Button1_DoubleClick;

        Controls.Add(button1);
    }

    private void Button1_DoubleClick(object sender, EventArgs e)
    {
        // This flag prevents the click handler logic from running
        // A double click raises the click event twice.
        _isDoubleClicking = true;
        FormBorderStyle = _initialStyle;
    }

    private void Button1_Click(object sender, EventArgs e)
    {
        if (_isDoubleClicking)
            _isDoubleClicking = false;
        else
            FormBorderStyle = FormBorderStyle.FixedToolWindow;
    }
}
```

```

Partial Public Class Form1

    Private _initialStyle As FormBorderStyle
    Private _isDoubleClicking As Boolean

    Public Sub New()
        InitializeComponent()
    End Sub

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Dim button1 As New DoubleClickButton

        _initialStyle = FormBorderStyle

        button1.Location = New Point(50, 50)
        button1.Size = New Size(200, 23)
        button1.Text = "Click or Double Click"

        AddHandler button1.Click, AddressOf Button1_Click
        AddHandler button1.DoubleClick, AddressOf Button1_DoubleClick

        Controls.Add(button1)

    End Sub

    Private Sub Button1_DoubleClick(sender As Object, e As EventArgs)
        ' This flag prevents the click handler logic from running
        ' A double click raises the click event twice.
        _isDoubleClicking = True
        FormBorderStyle = _initialStyle
    End Sub

    Private Sub Button1_Click(sender As Object, e As EventArgs)
        If _isDoubleClicking Then
            _isDoubleClicking = False
        Else
            FormBorderStyle = FormBorderStyle.FixedToolWindow
        End If
    End Sub

End Class

```

Para distinguir entre cliques

Manipule o [MouseDown](#) evento e determine o local e o intervalo de tempo entre cliques usando a [SystemInformation](#) propriedade e um [Timer](#) componente. Execute a ação apropriada dependendo se ocorreu um clique ou um clique duplo. O exemplo de código a seguir demonstra como fazer isso.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace project
{
    public partial class Form2 : Form
    {
        private DateTime _lastClick;
        private bool _inDoubleClick;
        private Rectangle _doubleClickArea;
        private TimeSpan _doubleClickMaxTime;
        private Action _doubleClickAction;
        private Action _singleClickAction;
        private Timer _clickTimer;
    }
}

```

```

public Form2()
{
    InitializeComponent();
    _doubleClickMaxTime = TimeSpan.FromMilliseconds(SystemInformation.DoubleClickTime);

    _clickTimer = new Timer();
    _clickTimer.Interval = SystemInformation.DoubleClickTime;
    _clickTimer.Tick += ClickTimer_Tick;

    _singleClickAction = () => MessageBox.Show("Single clicked");
    _doubleClickAction = () => MessageBox.Show("Double clicked");
}

private void Form2_MouseDown(object sender, MouseEventArgs e)
{
    if (_inDoubleClick)
    {
        _inDoubleClick = false;

        TimeSpan length = DateTime.Now - _lastClick;

        // If double click is valid, respond
        if (_doubleClickArea.Contains(e.Location) && length < _doubleClickMaxTime)
        {
            _clickTimer.Stop();
            _doubleClickAction();
        }

        return;
    }

    // Double click was invalid, restart
    _clickTimer.Stop();
    _clickTimer.Start();
    _lastClick = DateTime.Now;
    _inDoubleClick = true;
    _doubleClickArea = new Rectangle(e.Location - (SystemInformation.DoubleClickSize / 2),
                                     SystemInformation.DoubleClickSize);
}

private void ClickTimer_Tick(object sender, EventArgs e)
{
    // Clear double click watcher and timer
    _inDoubleClick = false;
    _clickTimer.Stop();

    _singleClickAction();
}
}

```

```
Imports System.Drawing
Imports System.Windows.Forms

Public Class Form2
    Private _lastClick As Date
    Private _inDoubleClick As Boolean
    Private _doubleClickArea As Rectangle
    Private _doubleClickMaxTime As TimeSpan
    Private _singleClickAction As Action
    Private _doubleClickAction As Action
    Private WithEvents _clickTimer As Timer

    Private Sub Form2_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        _doubleClickMaxTime = TimeSpan.FromMilliseconds(SystemInformation.DoubleClickTime)

        _clickTimer = New Timer()
        _clickTimer.Interval = SystemInformation.DoubleClickTime

        _singleClickAction = Sub()
            MessageBox.Show("Single click")
        End Sub

        _doubleClickAction = Sub()
            MessageBox.Show("Double click")
        End Sub
    End Sub

    Private Sub Form2_MouseDown(sender As Object, e As MouseEventArgs) Handles MyBase.MouseDown
        If _inDoubleClick Then

            _inDoubleClick = False

            Dim length As TimeSpan = Date.Now - _lastClick

            ' If double click is valid, respond
            If _doubleClickArea.Contains(e.Location) And length < _doubleClickMaxTime Then
                _clickTimer.Stop()
                Call _doubleClickAction()
            End If

            Return
        End If

        ' Double click was invalid, restart
        _clickTimer.Stop()
        _clickTimer.Start()
        _lastClick = Date.Now
        _inDoubleClick = True
        _doubleClickArea = New Rectangle(e.Location - (SystemInformation.DoubleClickSize / 2),
                                         SystemInformation.DoubleClickSize)
    End Sub

    Private Sub SingleClickTimer_Tick(sender As Object, e As EventArgs) Handles _clickTimer.Tick
        ' Clear double click watcher and timer
        _inDoubleClick = False
        _clickTimer.Stop()

        Call _singleClickAction()
    End Sub
End Class
```

Confira também

- [Visão geral do uso do mouse \(Windows Forms .NET\)](#)

- [Usando eventos de mouse \(Windows Forms .NET\)](#)
- [Gerenciar ponteiros do mouse \(Windows Forms .NET\)](#)
- [Como simular eventos do mouse \(Windows Forms .NET\)](#)
- [Control.Click](#)
- [Control.MouseDown](#)
- [Control.SetStyle](#)

Gerenciar ponteiros do mouse (Windows Forms .NET)

14/05/2021 • 2 minutes to read

O *ponteiro* do mouse, que às vezes é conhecido como o cursor, é um bitmap que especifica um ponto de foco na tela para entrada do usuário com o mouse. Este tópico fornece uma visão geral do ponteiro do mouse no Windows Forms e descreve algumas maneiras de modificar e controlar o ponteiro do mouse.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Acessando o ponteiro do mouse

O ponteiro do mouse é representado pela [Cursor](#) classe e cada [Control](#) tem uma [Control.Cursor](#) propriedade que especifica o ponteiro para esse controle. A [Cursor](#) classe contém propriedades que descrevem o ponteiro, como as [Position](#) Propriedades e [HotSpot](#), e os métodos que podem modificar a aparência do ponteiro, como os [Show](#) [Hide](#) métodos, e [DrawStretched](#).

O exemplo a seguir oculta o cursor quando o cursor está sobre um botão:

```
private void button1_MouseEnter(object sender, EventArgs e) =>
    Cursor.Hide();

private void button1_MouseLeave(object sender, EventArgs e) =>
    Cursor.Show();
```

```
Private Sub Button1_MouseEnter(sender As Object, e As EventArgs) Handles Button1.MouseEnter
    Cursor.Hide()
End Sub

Private Sub Button1_MouseLeave(sender As Object, e As EventArgs) Handles Button1.MouseLeave
    Cursor.Show()
End Sub
```

Controlando o ponteiro do mouse

Às vezes, pode ser recomendável limitar a área na qual o ponteiro do mouse pode ser usado ou alterar a posição do mouse. Você pode obter ou definir o local atual do mouse usando a [Position](#) Propriedade do [Cursor](#). Além disso, você pode limitar a área em que o ponteiro do mouse pode ser usado para definir a [Clip](#) propriedade. A área de recorte, por padrão, é a tela inteira.

O exemplo a seguir posiciona o ponteiro do mouse entre dois botões quando eles são clicados:

```
private void button1_Click(object sender, EventArgs e) =>
    Cursor.Position = PointToScreen(button2.Location);

private void button2_Click(object sender, EventArgs e) =>
    Cursor.Position = PointToScreen(button1.Location);
```

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    PointToScreen(Button2.Location)
End Sub

Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
    PointToScreen(Button1.Location)
End Sub
```

Alterando o ponteiro do mouse

Alterar o ponteiro do mouse é uma maneira importante de fornecer comentários para o usuário. Por exemplo, o ponteiro do mouse pode ser modificado nos manipuladores dos [MouseEnter](#) eventos e [MouseLeave](#) para informar ao usuário que as computações estão ocorrendo e limitar a interação do usuário no controle. Às vezes, o ponteiro do mouse será alterado devido a eventos de sistema, como quando seu aplicativo está envolvido em uma operação do tipo "arrastar e soltar".

A principal maneira de alterar o ponteiro do mouse é definindo a [Control.Cursor DefaultCursor](#) propriedade ou de um controle como um novo [Cursor](#). Para obter exemplos de alteração do ponteiro do mouse, consulte o exemplo de código na [Cursor](#) classe. Além disso, a [Cursors](#) classe expõe um conjunto de [Cursor](#) objetos para muitos tipos diferentes de ponteiros, como um ponteiro que se assemelha a uma mão.

O exemplo a seguir altera o cursor do ponteiro do mouse para um botão para uma mão:

```
button2.Cursor = System.Windows.Forms.Cursors.Hand;
```

```
Button2.Cursor = System.Windows.Forms.Cursors.Hand
```

Para exibir o ponteiro de espera, que se assemelha a uma ampulheta, sempre que o ponteiro do mouse estiver no controle, use a [UseWaitCursor](#) propriedade da [Control](#) classe.

Confira também

- [Visão geral do uso do mouse \(Windows Forms .NET\)](#)
- [Usando eventos de mouse \(Windows Forms .NET\)](#)
- [Como distinguir entre cliques e cliques duplos \(Windows Forms .NET\)](#)
- [Como simular eventos do mouse \(Windows Forms .NET\)](#)
- [System.Windows.Forms.Cursor](#)
- [Cursor.Position](#)

Como simular eventos do mouse (Windows Forms .NET)

14/05/2021 • 2 minutes to read

A simulação de eventos de mouse no Windows Forms não é tão simples quanto a simulação de eventos de teclado. Windows Forms não fornece uma classe auxiliar para mover o mouse e invocar ações de clique do mouse. A única opção para controlar o mouse é usar métodos nativos do Windows. Se você estiver trabalhando com um controle personalizado ou um formulário, poderá simular um evento do mouse, mas não poderá controlar diretamente o mouse.

IMPORTANT

A documentação do guia da área de trabalho do .NET 5 (e do .NET Core) está em construção.

Eventos

A maioria dos eventos tem um método correspondente que os invoca, nomeado no padrão de `On` seguido por `EventName`, como `OnMouseMove`. Essa opção só é possível em controles ou formulários personalizados, pois esses métodos são protegidos e não podem ser acessados de fora do contexto do controle ou do formulário. A desvantagem de usar um método como `OnMouseMove` é que ele não controla realmente o mouse ou interage com o controle, ele simplesmente gera o evento associado. Por exemplo, se você quisesse simular o passe do mouse sobre um item em um `ListBox`, `OnMouseMove` e o `ListBox` não reage visualmente com um item realçado sob o cursor.

Esses métodos protegidos estão disponíveis para simular eventos de mouse.

- `OnMouseDown`
- `OnMouseEnter`
- `OnMouseHover`
- `OnMouseLeave`
- `OnMouseMove`
- `OnMouseUp`
- `OnMouseWheel`
- `OnMouseClick`
- `OnMouseDoubleClick`

Para obter mais informações sobre esses eventos, consulte [usando eventos de mouse \(Windows Forms .net\)](#)

Invocar um clique

Considerando que a maioria dos controles faz algo quando clicado, como um botão que chama o código do usuário, ou a caixa de seleção altera seu estado de verificação, Windows Forms fornece uma maneira fácil de disparar o clique. Alguns controles, como uma `ComboBox`, não fazem nada de especial quando clicado e simulando um clique não tem efeito sobre o controle.

PerformClick

A `System.Windows.Forms.IButtonControl` interface fornece o `PerformClick` método que simula um clique no

controle. Os [System.Windows.Forms.Button](#) controles e [System.Windows.Forms.LinkLabel](#) implementam essa interface.

```
button1.PerformClick();
```

```
Button1.PerformClick()
```

InvokeClick

Com um formulário de controle personalizado, use o [InvokeOnClick](#) método para simular um clique do mouse. Esse é um método protegido que só pode ser chamado de dentro do formulário ou de um controle personalizado derivado.

Por exemplo, o código a seguir clica em uma caixa de seleção de `button1` .

```
private void button1_Click(object sender, EventArgs e)
{
    InvokeOnClick(checkBox1, EventArgs.Empty);
}
```

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    InvokeOnClick(CheckBox1, EventArgs.Empty)
End Sub
```

Usar métodos nativos do Windows

O Windows fornece métodos que você pode chamar para simular movimentos e cliques do mouse, como [User32.dll SendInput](#) e [User32.dll SetCursorPos](#) . O exemplo a seguir move o cursor do mouse para o centro de um controle:

```
[DllImport("user32.dll", EntryPoint = "SetCursorPos")]
[return: MarshalAs(UnmanagedType.Bool)]
private static extern bool SetCursorPos(int x, int y);

private void button1_Click(object sender, EventArgs e)
{
    Point position = PointToScreen(checkBox1.Location) + new Size(checkBox1.Width / 2, checkBox1.Height / 2);
    SetCursorPos(position.X, position.Y);
}
```

```
<Runtime.InteropServices.DllImport("USER32.DLL", EntryPoint:="SetCursorPos")>
Public Shared Function SetCursorPos(x As Integer, y As Integer) As Boolean : End Function

Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    Dim position As Point = PointToScreen(CheckBox1.Location) + New Size(CheckBox1.Width / 2,
    CheckBox1.Height / 2)
    SetCursorPos(position.X, position.Y)
End Sub
```

Confira também

- [Visão geral do uso do mouse \(Windows Forms .NET\)](#)
- [Usando eventos de mouse \(Windows Forms .NET\)](#)

- [Como distinguir entre cliques e cliques duplos \(Windows Forms .NET\)](#)
- [Gerenciar ponteiros do mouse \(Windows Forms .NET\)](#)