

Visual Studio Community 工具包

项目 • 2024/01/13

此工具包将帮助你编写成功的 Visual Studio 扩展，并且有一些内容同时适用于绝对初学者和专家。

以下视频介绍 Visual Studio 扩展性模型。

<https://www.microsoft.com/zh-cn/videoplayer/embed/RWPht8?postJsllMsg=true&autoCaptions=zh-cn>

选择起点

根据你的经验水平，选择开始位置。

入门指南

如果你是扩展开发新手，希望从头开始，以确保不会错过前面的任何信息。请前往[入门部分](#)。

典型场景演练

了解 Visual Studio 扩展性模型的基础知识后，便可探索要扩展的功能类型。为此，请查看[配方部分](#)，了解灵感和分步指南。

发布已完成的扩展

编写扩展后，便可进行发布。无论你是想与几个朋友、你的公司还是整个世界共享它，都需要查看[发布部分](#)。

提示和技巧

- [使用 Visual Studio 扩展中的生成](#)
- [使用 Visual Studio 扩展中的文件和文档](#)
- [使用 Visual Studio 扩展中的项目](#)
- [使用 Visual Studio 扩展中的解决方案](#)

Visual Studio 扩展工具包入门指南

项目 • 2024/01/13

入门指南的目标是让你基本了解 Visual Studio 扩展，并获取编写成功扩展的正确路径。

```
var doc = |
```

获取工具

该旅程首先[将正确的工具安装到](#) Visual Studio。

你的首个扩展

接下来，我们将[编写一个简单的扩展](#)作为将来自定义项的起点。

扩展剖析

现在可以仔细检查[扩展的所有移动部分](#)了。这使你可以更广泛地了解扩展性模型。

有用的资源

若要充分利用此工具包和 Visual Studio 扩展性，请查看[有用的资源](#)。这是让自己为将来取得成功做好准备的快速方法。

获取编写 Visual Studio 扩展所需的工具

项目 • 2024/01/13

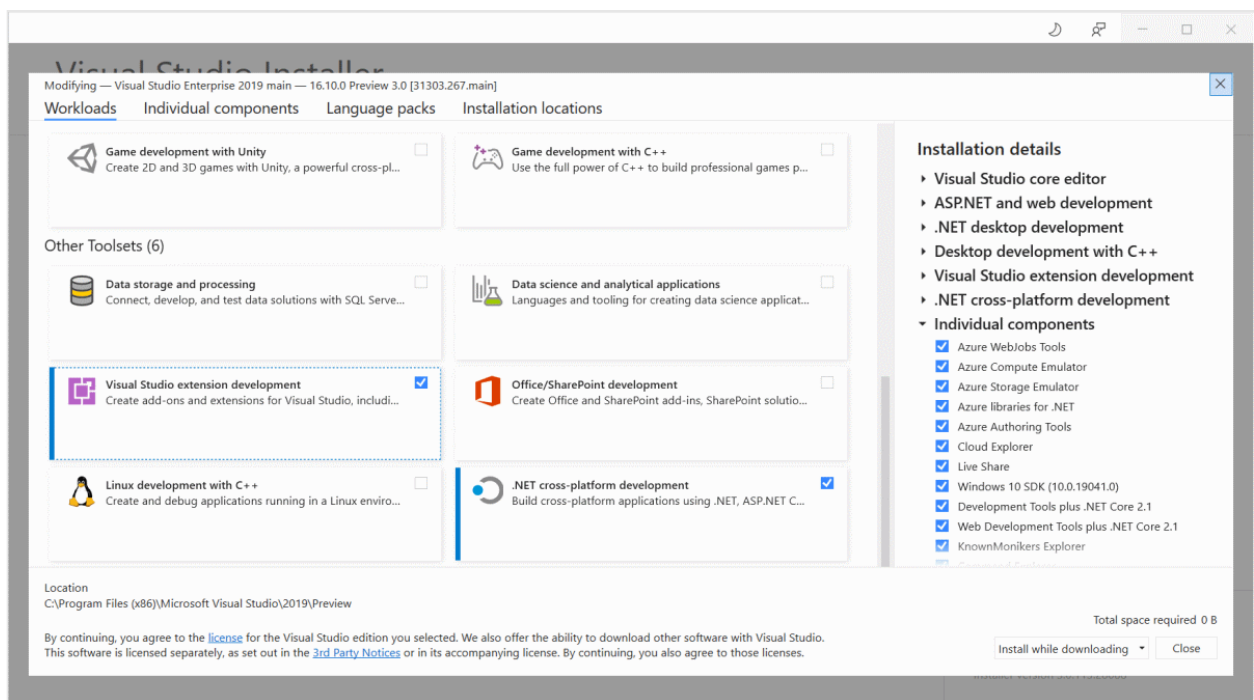
若要编写扩展，必须安装扩展性工作负荷。从技术上讲，这是你所需要的，但这组文档使用名为 *Extensibility Essentials* 的社区驱动扩展。每个版本的 Visual Studio 都有自己的版本：[Extensibility Essentials 2019](#) 或 [Extensibility Essentials 2022](#)。

以下视频介绍了所需的工具。

<https://www.microsoft.com/zh-cn/videoplayer/embed/RWPhT?postJsllMsg=true&autoCaptions=zh-cn>

安装扩展性工作负荷

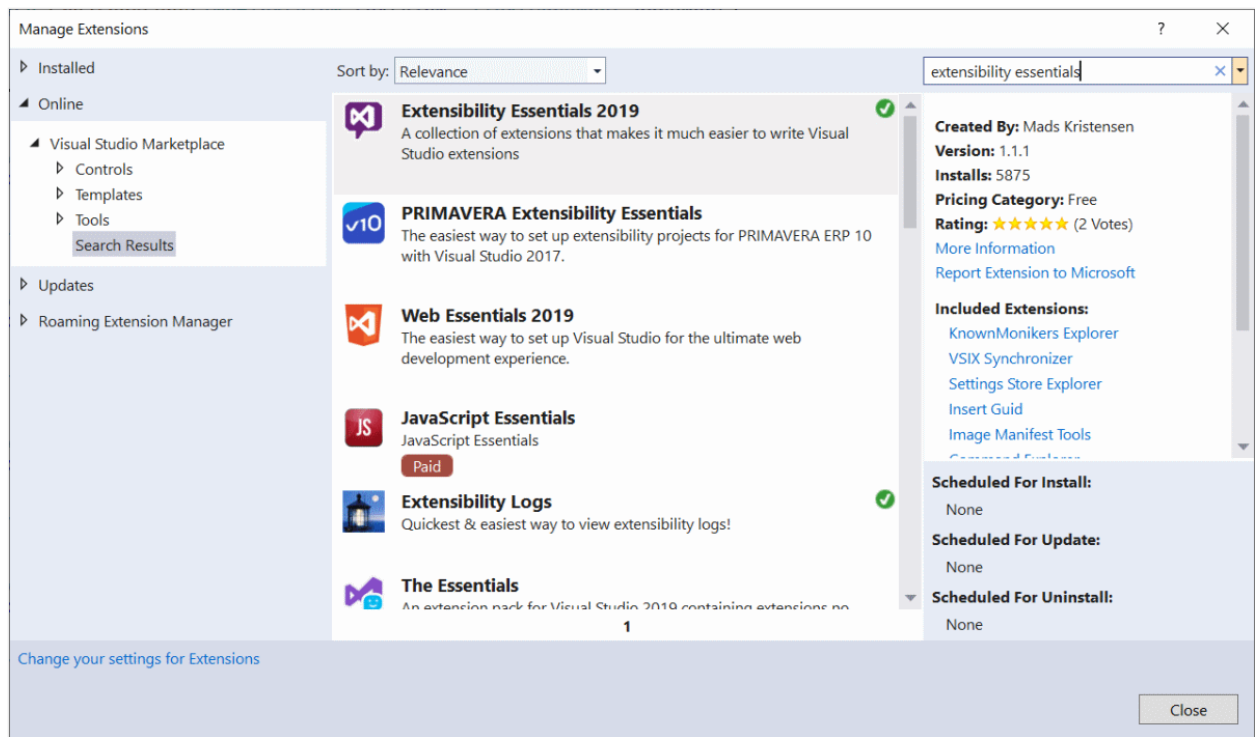
若要打开 Visual Studio 安装程序，请选择“工具”，然后选择“获取工具和功能...”。然后安装 Visual Studio 扩展开发工作负荷。



安装 Extensibility Essentials

若要安装 Extensibility Essentials，请选择“扩展”，选择“管理扩展”，然后搜索 扩展性。

- 对于 Visual Studio 2019，请安装 [Extensibility Essentials 2019](#)。
- 对于 Visual Studio 2022，请安装 [Extensibility Essentials 2022](#)。



就是这样，你现在可以开始开发 [第一个扩展](#)了。

Visual Studio 扩展剖析

项目 • 2024/07/12

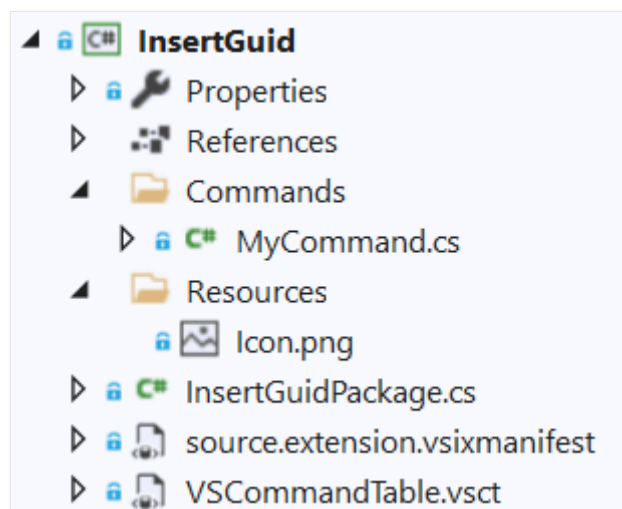
VSIX 包是一个 .vsix 文件，其中包含一个或多个 Visual Studio 扩展以及 Visual Studio 用于分类和安装扩展的元数据。VSIX 包格式遵循开放打包约定 (OPC) 标准，这意味着它可以由任何可以打开 ZIP 文件的工具打开。

扩展项目是一个 C# 项目，其中包含一些使其独特的附加功能。以下视频探讨了扩展项目，以更好地了解扩展项目的工作原理：

<https://www.microsoft.com/zh-cn/videoplayer/embed/RWPhtJ?postJsllMsg=true&autoCaptions=zh-cn>

文件结构

使用 VSIX Project w/Command (Community) 模板创建新扩展时，文件结构如下所示：



.vsixmanifest 文件是主文件。它是一个 XML 文件，其中包含有关 Visual Studio 使用的扩展的信息。扩展的所有组件都在 .vsixmanifest 文件中注册。它是 VSIX 项目中唯一必需的文件。

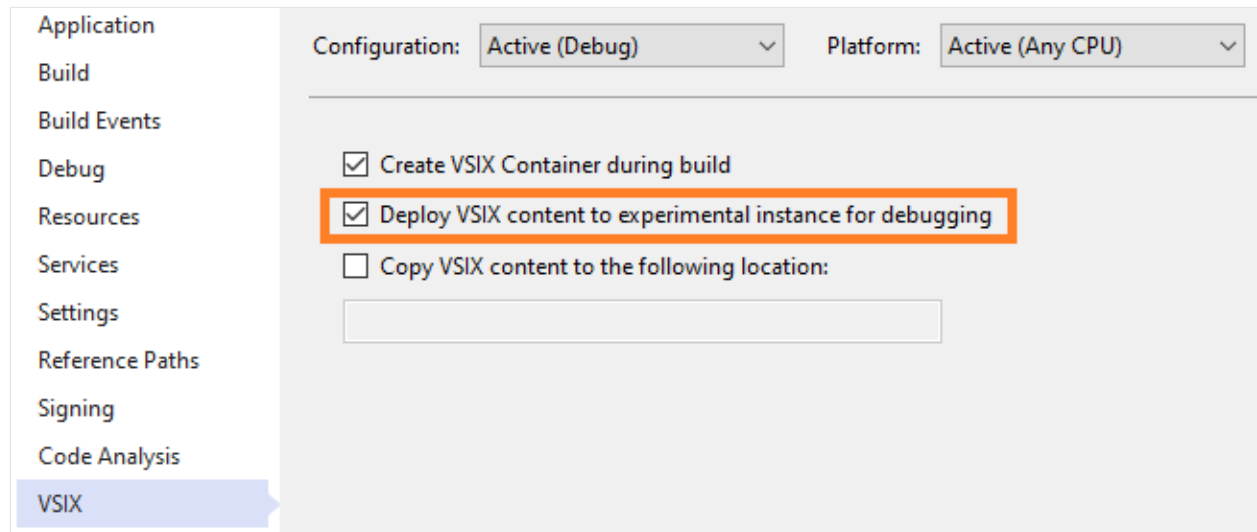
VSCommandTable.vsct 文件是声明命令的位置。它是一个 XML 文件，包含按钮命令、菜单、键盘快捷方式绑定等的定义。该文件将其内容编译为 output.dll 中的 blob，Visual Studio 使用该 blob 来构造其整个命令表菜单结构。此文件仅声明命令表中的组件；而不处理任何命令调用。

*Package.cs 文件是 Package 类，这是大多数扩展的入口点。在此处，通常会找到已注册的命令处理程序、工具窗口、选项页、服务和其他组件。

编译

项目编译为位于 `/bin/debug` 或 `/bin/release` 文件夹中的 `.vsix` 文件，具体取决于当前的解决方案生成配置。Visual Studio 扩展开发[工作负载](#)提供专用的 MSBuild 目标和任务来处理 VSIX 项目风格。

当 VSIX 项目生成时，它会自动自行部署到实验实例。这可以在 VSIX 项目设置中进行控制：



实验实例

为了保护 Visual Studio 开发环境免受可能更改它的未经测试的应用程序的影响，VS SDK 提供了可用于实验的实验空间。可以像往常一样使用 Visual Studio 开发新的应用程序，但可以使用此实验实例来运行它们。

每个具有 VSIX 包的应用程序都会在调试模式下启动 Visual Studio 实验实例。

如果要在特定解决方案外部启动 Visual Studio 的实验实例，请在命令窗口中运行以下命令：

```
shell
```

```
devenv.exe /RootSuffix Exp
```

有关更多扩展性概念，请查看[有用的资源](#)，这些资源将在学习本工具包时派上用场。

你的第一个 Visual Studio 扩展

项目 • 2024/01/13

本文将引导你完成一些简单的步骤，以便启动并运行第一个 Visual Studio 扩展。Visual Studio 扩展是使用 .NET Framework 和 C# 编写的。如果你已经是 .NET 开发人员，你会发现编写扩展类似于编写大多数其他 .NET 程序和库。

今天要编写的扩展会添加一个命令，该命令在执行时将新 GUID 插入文本编辑器中。它非常简单、有用，并很好地介绍了扩展开发的各个方面。

如果你是视觉学习者，请检查本教程后面的某人的此简短视频。

<https://www.microsoft.com/zh-cn/videoplayer/embed/RWPmjK?postJsllMsg=true&autoCaptions=zh-cn>

在开始编写第一个 Visual Studio 扩展（这很容易，我承诺！）之前，请确保你已获得 [所需的工具](#)。

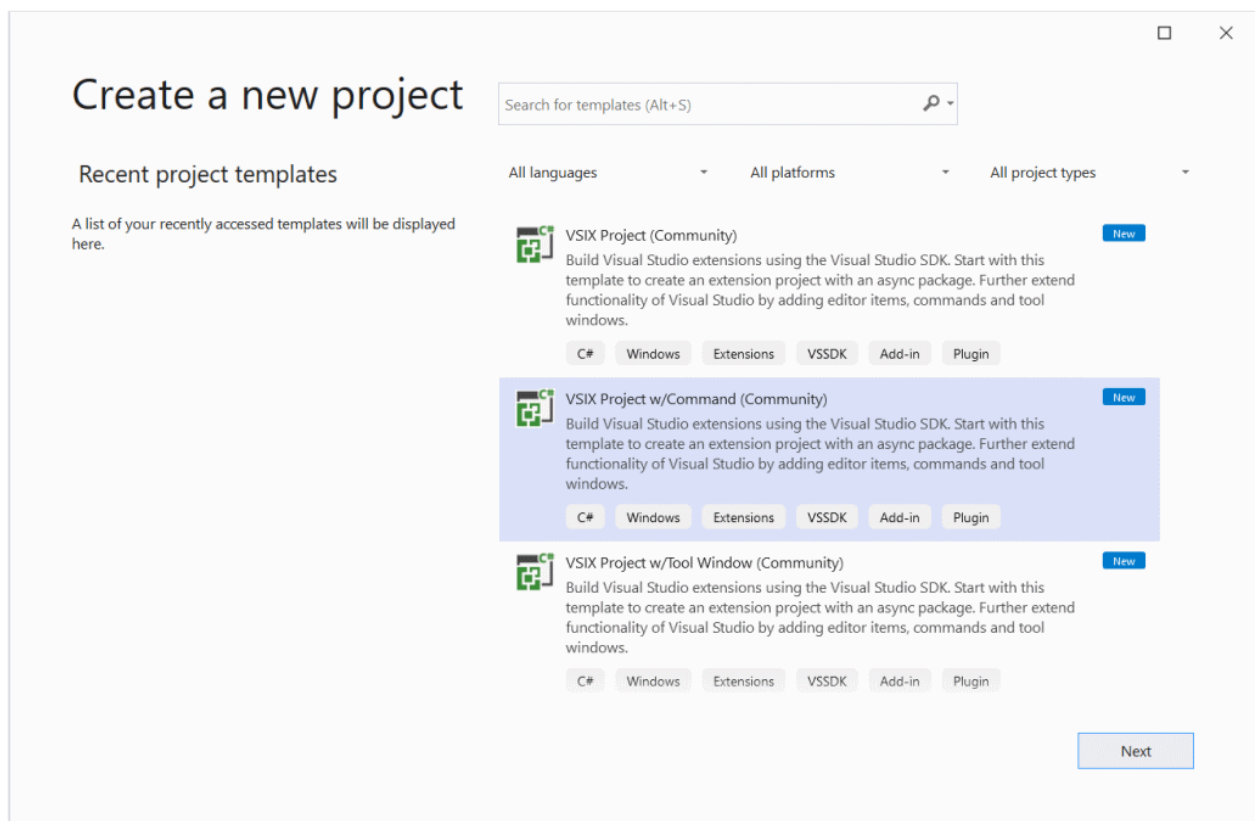
创建项目

有几个项目模板可供选择，因此你想要做出正确的选择。此社区工具包中使用的模板名称中都有名字对象（社区）。

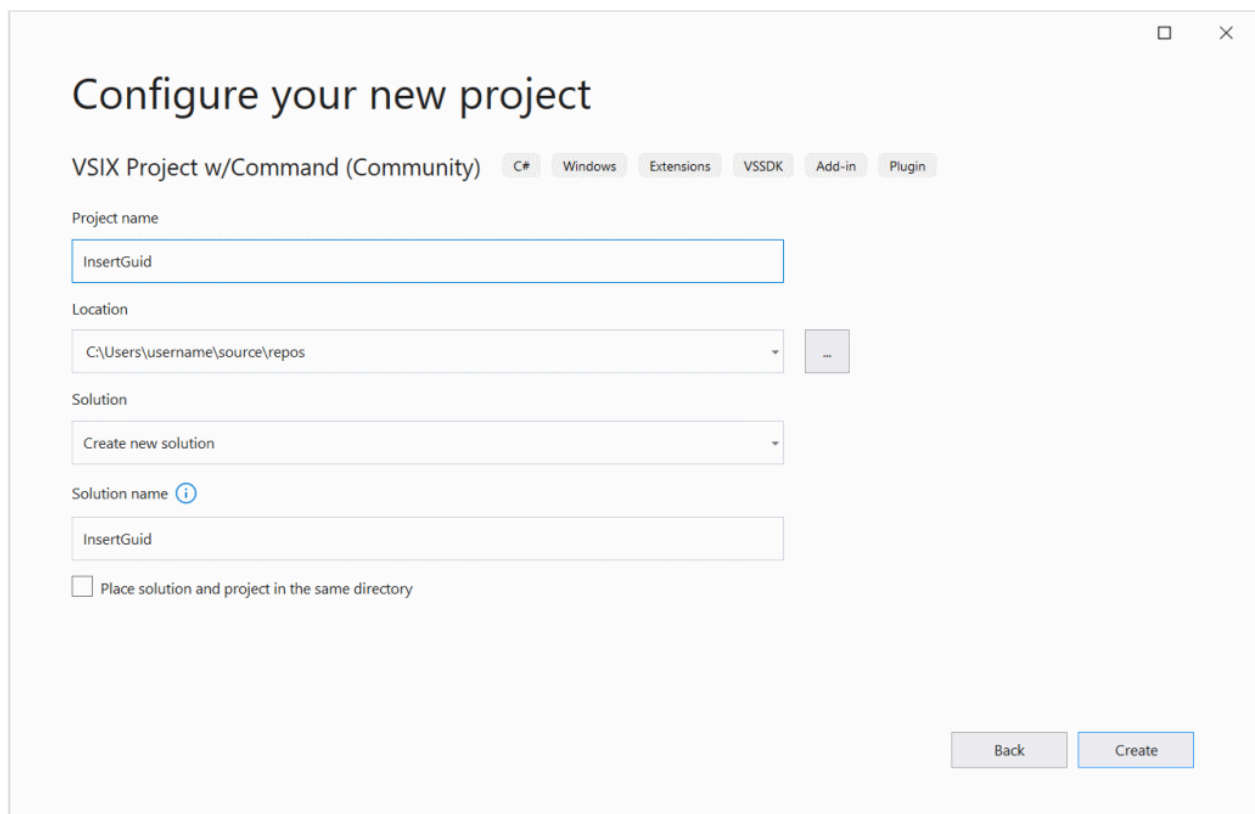
VSIX 项目 w/Command (Community) 模板附带了一个已挂钩的命令，因此可以轻松地从那里开始。对于大多数扩展而言，这是一个很好的起点。如果知道需要工具窗口，请使用 **VSIX 项目 w/Tool Window (社区)** 模板。它还具有用于打开工具窗口的命令。

将空 VSIX 项目 (社区) 或 **VSIX 项目 (社区)** 模板用于仅限 MEF 的扩展或其他高级方案。

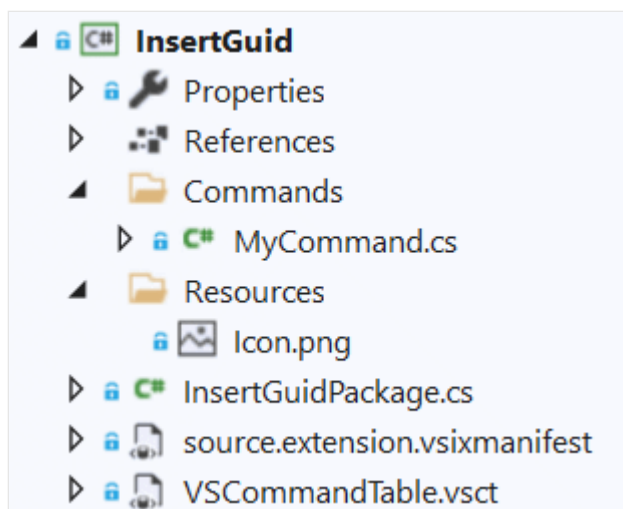
这次，你将选择 **VSIX 项目 w/Command (Community)** 模板，如下面的屏幕截图所示。



选择项目模板后，需要为项目命名。将其命名为 **InsertGuid**。



点击“**创建**”按钮后，最终应会看到如下所示的基本 VSIX 项目：



重要文件

我们来看看最重要的文件。

`InsertGuidPackage.cs` 称为 Package 类。Visual Studio 调用其 `InitializeAsync(...)` 方法来初始化扩展。从此处可以添加事件侦听器并注册命令、工具窗口、设置和其他内容。

`source.extension.vsixmanifest` 是扩展的清单文件。它包含元数据，如标题和说明，但也包含有关扩展包含的内容的信息。

`VSCommandTable.vsct` 是一个 XML 文件，其中以声明方式定义命令和键绑定，以便它们可以注册到 Visual Studio。

`Commands/MyCommand.cs` 是 `VSCommandTable.vsct` 文件中定义的命令的命令处理程序。它通过单击按钮来控制命令执行时会发生什么情况。

修改命令

首先，你需要确保命令在 Visual Studio 菜单系统中具有正确的名称、图标和位置。

打开 `VSCommandTable.vsct` 文件，找到一个 `<Group>` 和一个 `<Button>`。请注意该按钮如何将组指定为其父组，并且该组的父级是内置的 `VSMainMenu/Tools` 菜单。

对于扩展，需要“插入 GUID”命令按钮位于“编辑”主菜单下，因此需要将组重新父级到“编辑”菜单。将工具替换为“编辑”，就像在以下代码片段中一样：








XML

```
<Group guid="InsertGuid" id="MyMenuGroup" priority="0x0600">
  <Parent guid="VSMainMenu" id="Edit"/>
```

```
</Group>
```

你可以获得完整的 IntelliSense 位置，以便轻松找到合适的位置。

```
<Group guid="InsertGuid" id="MyMenuGroup" priority="0x0600">  
  <Parent guid="VSMainMenu" id="Edit" />  
</Group>
```

	CodeWindow.TextEditGroup	<Group>
	Edit	<Menu>
	Edit.CutCopyGroup	<Group>
	Edit.FindGroup	<Group>
	Edit.GoToGroup	<Group>
	Edit.ObjectsGroup	<Group>
	Edit.PasteGroup	<Group>

还需要 `<Button>` 更新。通过将元素的属性 `<Icon>` 更新 `id` 为 `PasteAppend` 来为其提供一个新图标。使用 `<ButtonText>` 良好的描述性名称更新文本，并使用命令的技术名称进行更新 `<LocCanonicalName>` - 这是当用户在“**工具 > 选项 > 环境 > 键盘**”对话框中为命令分配自定义键盘快捷方式时向用户显示的名称。

XML


```
<Button guid="InsertGuid" id="MyCommand" priority="0x0100" type="Button">  
  <Parent guid="InsertGuid" id="MyMenuGroup" />  
  <Icon guid="ImageCatalogGuid" id="PasteAppend" />  
  <CommandFlag>IconIsMoniker</CommandFlag>  
  <Strings>  
    <ButtonText>Insert GUID</ButtonText>  
    <LocCanonicalName>.Edit.InsertGuid</LocCanonicalName>  
  </Strings>  
</Button>
```

❗ 备注

始终以 `<LocCanonicalName>` 点字符开头。它确保不会自动预写其他文本，并且不会显示点。

可以使用 Visual Studio 图像库中提供的数千个图标，甚至可以获取 IntelliSense 中显示的预览：

```
<Button guid="InsertGuid" id="MyCommand" priority="0x0100" type="Button">
  <Parent guid="InsertGuid" id="MyMenuGroup" />
  <Icon guid="ImageCatalogGuid" id="paste" />
  <CommandFlag>IconIsMoniker</CommandFlag>
  <Strings>
    <ButtonText>My Command</ButtonText>
    <LocCanonicalName>.InsertGuid.MyCommand</LocCanonicalName>
  </Strings>
</Button>
```



Paste	Image
PasteAppend	Image
PasteReplace	Image
PasteTable	Image

现在，你已更新命令的名称、图标和位置，现在可以编写一些代码以将 guid 插入文本编辑器中。

打开 `/Commands/MyCommand.cs` 文件并将其修改为在执行时插入新的 guid：

C#

```
using System;
using Community.VisualStudio.Toolkit;
using EnvDTE;
using Microsoft.VisualStudio.Shell;
using Task = System.Threading.Tasks.Task;

namespace InsertGuid
{
    [Command(PackageIds.MyCommand)]
    internal sealed class MyCommand : BaseCommand<MyCommand>
    {
        protected override async Task ExecuteAsync(OleMenuCmdEventArgs e)
        {
            await Package.JoinableTaskFactory.SwitchToMainThreadAsync();
            DocumentView docView = await
VS.Documents.GetActiveDocumentViewAsync();
            if (docView?.TextView == null) return;
            SnapshotPoint position =
docView.TextView.Caret.Position.BufferPosition;
            docView.TextBuffer?.Insert(position, Guid.NewGuid().ToString());
        }
    }
}
```

你正在使用该 `vs` 对象获取活动编辑器文本视图，然后将 guid 插入到其文本缓冲区的插入点位置。

❗ 备注

你将在社区工具包的很多位置看到和 `ThreadHelper.ThrowIfNotOnUIThread()` 显示 `await JoinableTaskFactory.SwitchToMainThreadAsync()`。它们处理线程切换最佳做

法，目前不需要知道何时以及如何使用这些做法 - 使用代码修复（灯泡）的编译器警告使这一点变得非常简单。

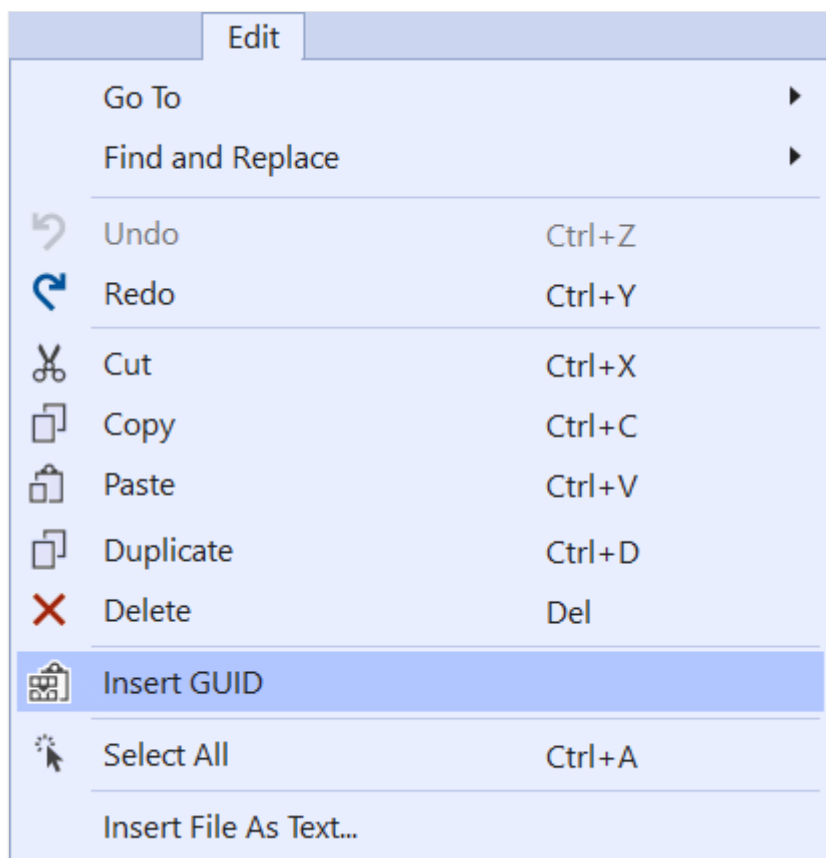
扩展的第一个草稿现已完成，是时候测试它了。

运行和调试

运行扩展与运行任何其他 .NET 项目一样简单。只需按 *F5* 即可在附加调试器的情况下运行，或者按 *Ctrl+F5* 运行而不运行。

这样做将启动已安装扩展的 Visual Studio 实验实例。实验实例是 Visual Studio 的常规版本，但安装了单独的设置和扩展。它有助于使事情保持分开。

实验实例启动时，应在“编辑”主菜单中看到“插入 GUID”命令。



打开任何基于文本的文件并执行命令以插入新的 guid。就这么简单！

总结

现已创建第一个扩展，该扩展将命令按钮添加到主菜单，并在执行时与文本编辑器交互。

祝贺！！




可以在示例存储库[中找到](#)此扩展的代码。

相关内容

- [扩展剖析](#)
- [菜单和命令](#)
- [最佳做法检查列表](#)

Visual Studio 扩展上的有用资源

项目 • 2023/10/10

适用范围:  Visual Studio  Visual Studio for Mac  Visual Studio Code

这些资源可帮助你更好地导航 Visual Studio 扩展性的世界。

以下视频介绍了适用于 Visual Studio 扩展作者的有用资源。

<https://www.microsoft.com/zh-cn/videoplayer/embed/RWP8Kw?postJsllMsg=true&autoCaptions=zh-cn>

资源

下面是一些有用的资源，可以帮助你在扩展旅程中。

- [GitHub 上的 VSIX 社区](#)
- [VSIX 社区示例存储库](#)
- [官方 VS SDK 文档](#)
- [VS SDK 示例存储库](#)
- [Gitter.im 上的扩展性聊天室](#)

了解如何搜索帮助

编写扩展有点利基活动，因此在线搜索帮助并不总是返回相关结果。但是，我们可以通过多种方式优化搜索词以生成更好的结果。

- 使用精确的接口和类名作为搜索词的一部分。
- 尝试将 VSIX、VSSDK 或 *Visual Studio* 单词添加到搜索词。
- 尽可能直接在 GitHub 上搜索，而不是 Google/必应。
- 向 Gitter.im [聊天室中的其他](#) 扩展程序提问。

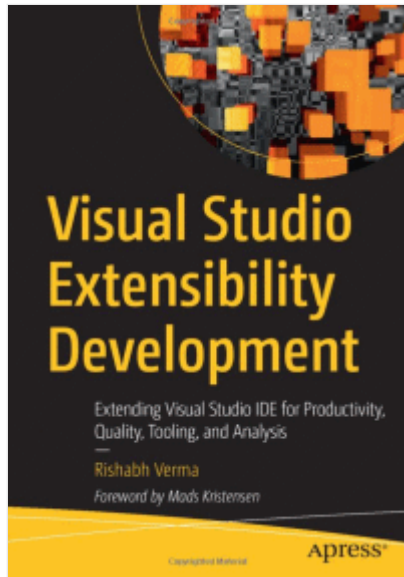
使用开放源代码作为学习工具

你可能对希望扩展执行的操作及其工作原理有想法。但是应使用哪些 API 以及如何正确连接 API？这些都是困难的问题，很多人放弃时，这些不回答。

一种好方法是在市场中查找执行类似操作的扩展，或使用与要执行的操作类似的元素。然后找到这些扩展的源代码，并查看它们执行的操作以及它们使用过的 API，然后从那里获取。

预订

若要开始学习 Visual Studio 扩展性模型，请考虑 [Rishabh Verma](#) 的 [Visual Studio 扩展性开发](#) 书籍。



这是最好的书可供学习。

术语表

为了更好地了解此社区工具包并能够在线搜索帮助，具有扩展性术语的共享词汇至关重要。下面是一个字母顺序的概念和字词列表，这些概念和字词对于扩展程序而言非常重要。

DTE

EnvDTE 是一个程序集包装的 COM 库，其中包含 Visual Studio 核心自动化的对象和成员。或者，一个易于使用的界面，用于与 Visual Studio 交互。

卖场

[Visual Studio Marketplace](#) 是扩展程序用来与世界共享其扩展的公共扩展存储。它由 Microsoft 拥有和维护，是唯一的官方扩展市场。

MEF

托管扩展性框架由 Visual Studio 中的多个组件使用，主要是编辑器。注册扩展点的方法与包不同。

包

有时称为 *Package* 类。Visual Studio 调用其 `InitializeAsync(...)` 方法来初始化扩展。从此处可以添加事件侦听器并注册命令、工具窗口、设置和其他内容。在编译期间，*Package* 类上的属性用于生成 .pkgdef 文件，该文件会自动添加到扩展中。

.pkgdef

这是一个包，其中包含要添加到 Visual Studio 的专用注册表中的键和值。可以从 *Package* 类自动生成此文件，也可以手动创建 .pkgdef 文件，并将其作为 `<Asset>` .vsixmanifest 文件中包含。

VSCT

Visual Studio 命令表文件。这是声明菜单、命令和键绑定的位置。

VSIX

指 Visual Studio 扩展 (.vsix) 的文件扩展名，还指 Visual Studio 扩展性的假名。

VSSDK

这是 Visual Studio SDK 的缩写，即构成公共表面的类、服务和组件是 Visual Studio 的扩展性 API。它通常用于引用 [Microsoft.VisualStudio.SDK](#) NuGet 包。

在 [Visual Studio SDK 术语表](#)中查找详细信息。

将菜单和命令添加到 Visual Studio 扩展

项目 • 2024/01/13

本文将指导你完成将菜单和命令添加到 Visual Studio 扩展的步骤。命令最常用作 Visual Studio 周围菜单中的按钮。若要创建命令，需要执行两个步骤：

1. 定义命令
2. 处理单击/调用

定义命令

每个菜单中的每个按钮都是一个命令。若要将命令添加到扩展，必须先在 .vsct 文件中定义它。它可能如下所示：

XML

```
<Buttons>
  <Button guid="MyPackage" id="MyCommand" priority="0x0105" type="Button">
    <Parent guid="VSMainMenu"
id="View.DevWindowsGroup.OtherWindows.Group1"/>
    <Icon guid="ImageCatalogGuid" id="StatusInformation" />
    <CommandFlag>IconIsMoniker</CommandFlag>
    <Strings>
      <ButtonText>R&unner Window</ButtonText>
    </Strings>
  </Button>
</Buttons>
```

此按钮位于元素中指定的“查看>其他 Windows”**菜单中的 Parent 父组中**。

现在可以立即运行扩展，以查看命令是否显示在正确的位置和菜单中。

处理单击/调用

定义按钮后，我们需要处理调用按钮时发生的情况。我们在如下所示的 C# 类中执行此操作：

C#

```
[Command("489ba882-f600-4c8b-89db-eb366a4ee3b3", 0x0100)]
public class MyCommand : BaseCommand<TestCommand>
{
    protected override Task ExecuteAsync(OleMenuCmdEventArgs e)
    {
```

```
        // Do something
    }
}
```

请确保从 `Package` 类 `InitializeAsync` 的方法调用它。

C#

```
protected override async Task InitializeAsync(CancellationTokencancellationTokencancellationToken, IProgress<ServiceProgressData> progress)
{
    await this.RegisterCommandsAsync();
}
```

命令 Guid 和 ID 必须与 .vsct 文件中元素的 guid/id 对 `Button` 匹配

相关内容

- [Visual Studio 命令表格 \(.Vsct\) 文件](#)

生成自定义工具窗口

项目 • 2024/01/13

自定义工具窗口是向 Visual Studio 添加复杂 UI 的绝佳选项。

工具窗口是 Visual Studio 中的核心 UI 概念，以下视频将演示如何添加自定义窗口。

<https://www.microsoft.com/zh-cn/videoplayer/embed/RWPhtK?postJsllMsg=true&autoCaptions=zh-cn>

工具窗口是一个窗口，可以像解决方案资源管理器、错误列表和其他已知工具窗口一样移动和停靠。工具窗口由 Visual Studio 提供的外部外壳和自定义内部 UI 控件（通常是扩展提供的 XAML `<usercontrol>`）组成。

① 备注

若要使用工具窗口创建新扩展，请使用 **VSIX 项目 w/Tool Window (社区)** 模板创建新项目，并跳过此食谱的其余部分。有关详细信息，[请参阅入门](#)。

将工具窗口添加到现有扩展需要 4 个简单的步骤：

1. 创建工具窗口外部 shell 类。
2. 将 XAML `<usercontrol>` 添加到工具窗口。
3. 注册工具窗口。
4. 创建用于显示工具窗口的命令。

让我们从步骤 1 开始。

创建工具窗口

`BaseToolWindow<T>` 使用泛型基类时，系统会要求我们提供一些基本信息。我们必须指定工具窗口的标题，创建并返回 XAML 用户控件，并设置 Visual Studio 用于创建窗口外壳的实际 `ToolWindowPane` 类。

C#

```
using System;
using System.Runtime.InteropServices;
using System.Threading;
using System.Threading.Tasks;
using System.Windows;
using Community.VisualStudio.Toolkit;
using EnvDTE80;
using Microsoft.VisualStudio.Imaging;
```

```

using Microsoft.VisualStudio.Shell;

public class MyToolWindow : BaseToolWindow<MyToolWindow>
{
    public override string GetTitle(int toolWindowId) => "My Tool Window";

    public override Type PaneType => typeof(Pane);

    public override async Task<FrameworkElement> CreateAsync(int
toolWindowId, CancellationToken cancellationToken)
    {
        await Task.Delay(2000); // Long running async task
        return new MyUserControl();
    }

    // Give this a new unique guid
    [Guid("d3b3ebd9-87d1-41cd-bf84-268d88953417")]
    internal class Pane : ToolWindowPane
    {
        public Pane()
        {
            // Set an image icon for the tool window
            BitmapImageMoniker = KnownMonikers.StatusInformation;
        }
    }
}

```

必须从 `CreateAsync(int, CancellationToken)` 方法创建自定义用户控件的实例，然后在 Visual Studio 创建自定义用户控件时自动传递给工具窗口 shell。

但首先，必须创建用户控件。

添加 XAML 用户控件

它可以是任何 XAML 及其代码隐藏类，因此下面是包含单个按钮的 `<usercontrol>` 简单示例：

XML

```

<UserControl x:Class="MyUserControl"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:toolkit="clr-
namespace:Community.VisualStudio.Toolkit;assembly=Community.VisualStudio.Too
lkit"
    mc:Ignorable="d"

```

```

        toolkit:Themes.UseVsTheme="True"
        d:DesignHeight="300" d:DesignWidth="300"
        Name="MyToolWindow">
    <Grid>
        <StackPanel Orientation="Vertical">
            <Label Margin="10" HorizontalAlignment="Center">My
Window</Label>
            <Button Content="Click me!" Click="button1_Click" Width="120"
Height="80" Name="button1"/>
        </StackPanel>
    </Grid>
</UserControl>

```

现在，我们有了返回自定义控件的工具窗口类。下一步是向 Visual Studio 注册工具窗口。

注册工具窗口

注册工具窗口意味着我们告诉 Visual Studio 它是否存在以及如何实例化它。我们使用特性从包类 `[ProvideToolWindow]` 执行此操作。

C#

```

[ProvideToolWindow(typeof(MyToolWindow.Pane))]
public sealed class MyPackage : ToolkitPackage
{
    protected override async Task InitializeAsync(CancellationTok
cancellationToken, IProgress<ServiceProgressData> progress)
    {
        this.RegisterToolWindows();
    }
}

```

❗ 备注

请注意，包类必须继承自 `ToolkitPackage` 或从 `Package` 中 `AsyncPackage` 继承。

可以指定工具窗口应具有的风格以及默认情况下应显示的位置。以下示例显示工具窗口应放置在链接风格中的解决方案资源管理器相同的停靠容器中。

C#

```

[ProvideToolWindow(typeof(MyToolWindow.Pane), Style = VsDockStyle.Linked,
Window = WindowGuids.SolutionExplorer)]

```

若要使工具窗口默认可见，可以使用该属性在不同的 UI 上下文 `[ProvideToolWindowVisibility]` 中指定其可见性。

C#

```
[ProvideToolWindowVisibility(typeof(MyToolWindow.Pane),  
VSConstants.UICONTEXT.NoSolution_string)]
```

用于显示工具窗口的命令

这与任何其他命令相同，你可以看到如何在菜单和命令食谱[中添加](#)一个命令。

显示工具窗口的命令处理程序类将如下所示：

C#

```
using Community.VisualStudio.Toolkit;  
using Microsoft.VisualStudio.Shell;  
using Task = System.Threading.Tasks.Task;  
  
[Command(PackageIds.RunnerWindow)]  
internal sealed class MyToolWindowCommand : BaseCommand<MyToolWindowCommand>  
{  
    protected override async Task ExecuteAsync(OleMenuCmdEventArgs e) =>  
        await MyToolWindow.ShowAsync();  
}
```

工具窗口的命令放置通常位于 **视图 -> 主菜单中的其他 Windows** 下。

这就可以了。恭喜，现已创建自定义工具窗口。

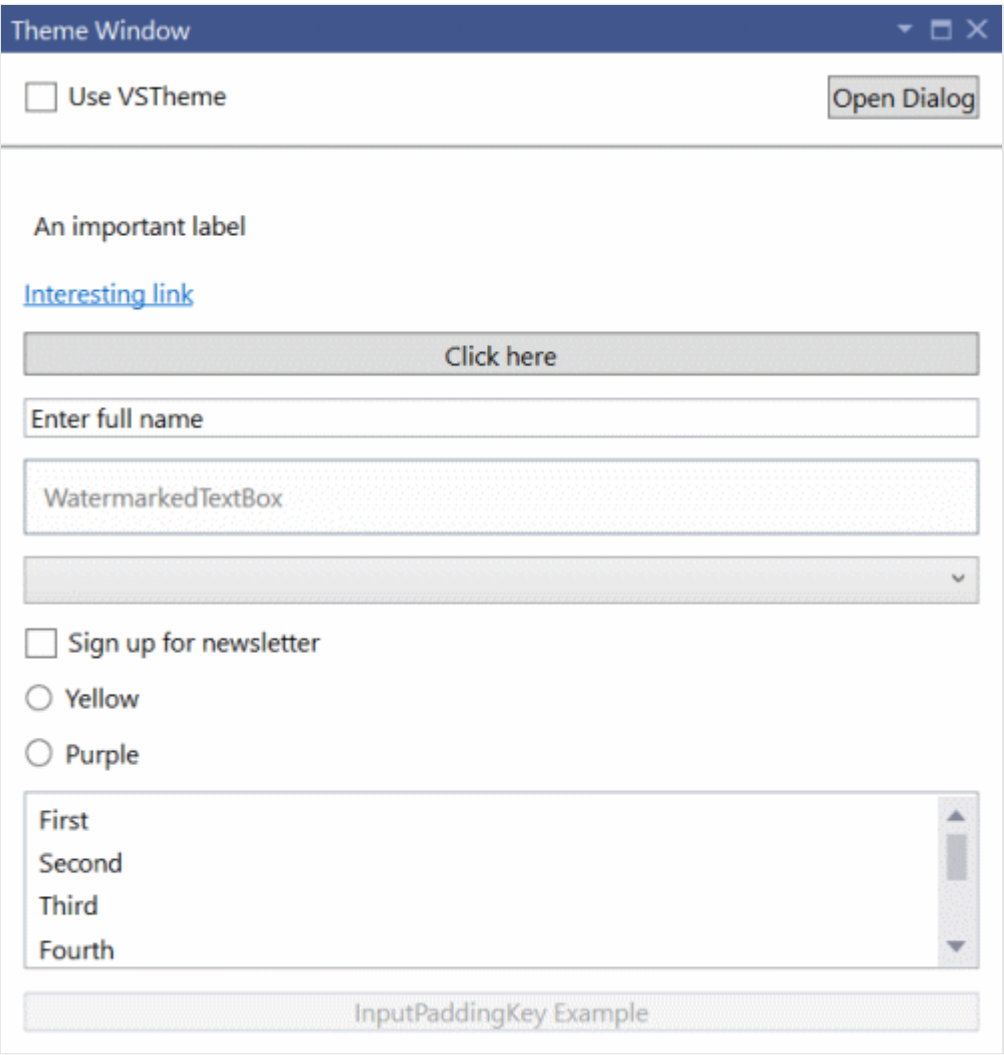
获取源代码

可以在示例存储库[中找到此配方的](#) [源代码](#)。

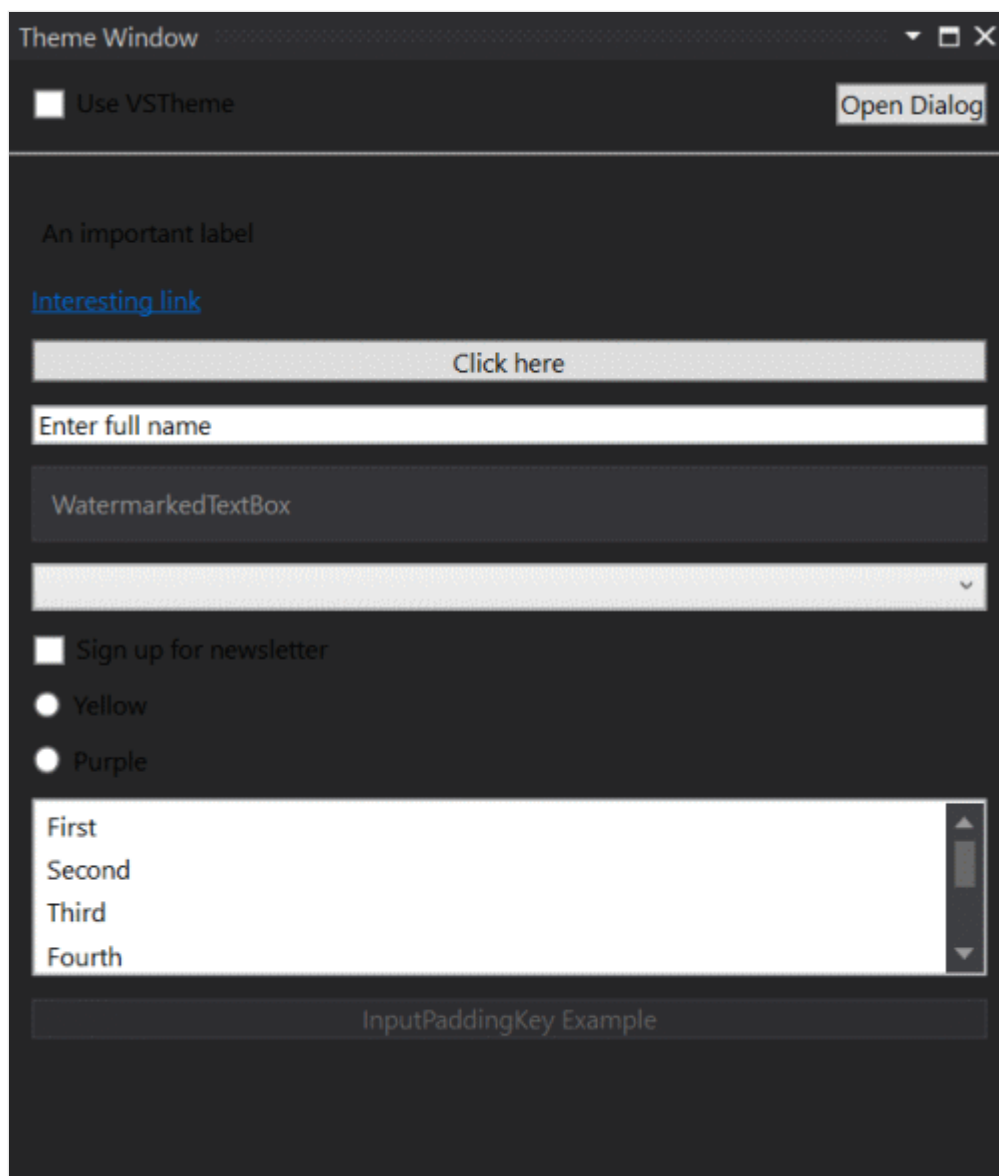
匹配 Visual Studio 扩展中的 Visual Studio 主题

项目 • 2024/01/13

每当使用 WPF 生成任何自定义 UI 时，都需要确保它与 Visual Studio 主题匹配。这样一来，UI 会看起来是本机的，感觉更像是 Visual Studio 的自然部分。否则，工具窗口和对话框最终可能会在浅色主题中如下所示：



请注意文本框和按钮的填充看起来如何不正确？ 深色主题中的情况会变得更糟：



现在，文本和背景色几乎无法阅读。不好。

有一种简单的方法可以确保 UI 的背景色、按钮样式等与 Visual Studio 的背景色、按钮样式等匹配 Visual Studio 的背景色和简单的小技巧。这样，在浅色主题中，相同的 UI 如下所示：

Theme Window

☒ Use VSTheme

Open Dialog

An important label

Interesting link

Click here

Enter full name

WatermarkedTextBox

☐ Sign up for newsletter

☐ Yellow

☐ Purple

First

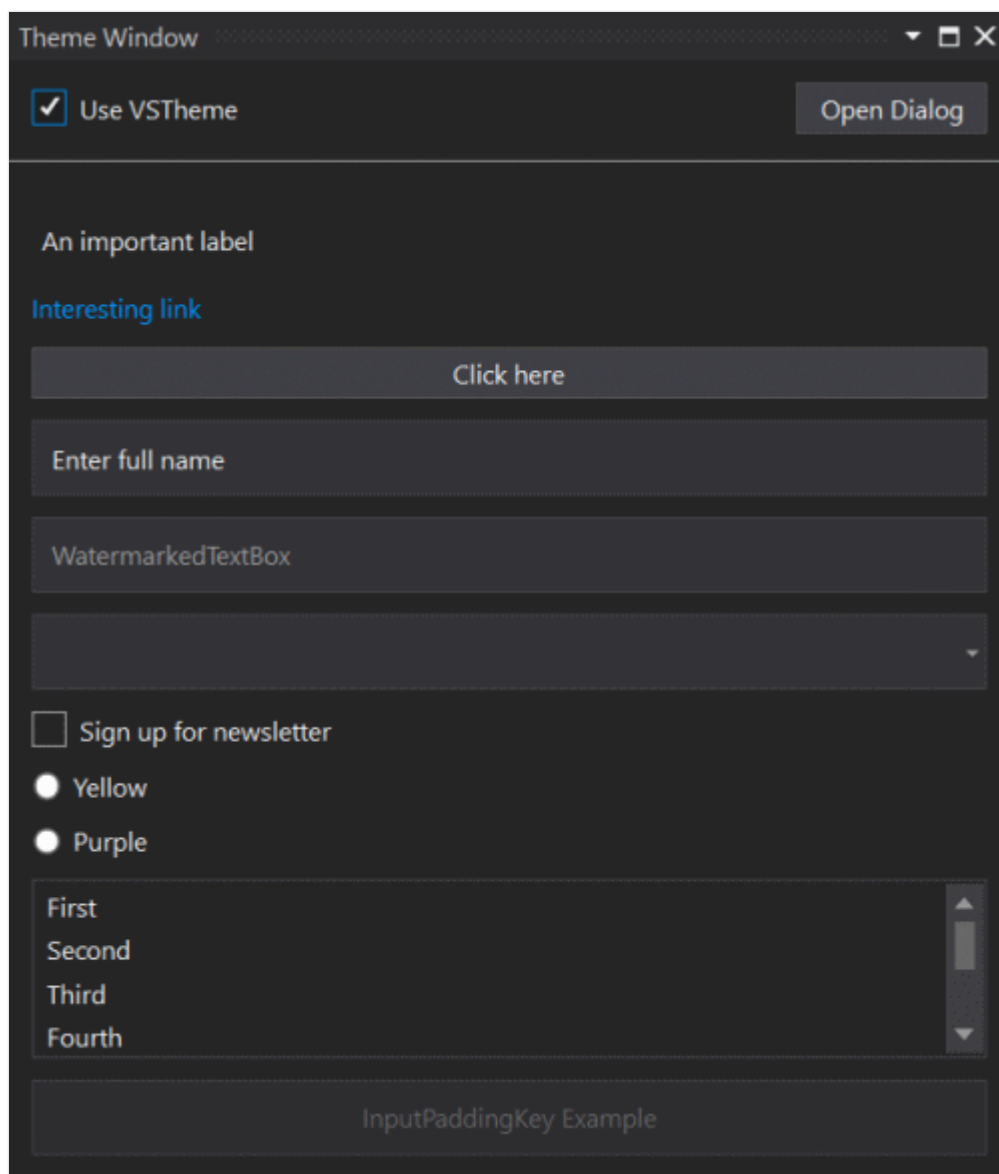
Second

Third

Fourth

InputPaddingKey Example

或在深色主题中：



这看起来好多了。 让我们看看如何主题化 UI。

WPF UserControl

下面是可在工具窗口中直接使用的 WPF `<UserControl>` 示例。

XML

```
<UserControl x:Class="TestExtension.RunnerWindowControl"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:toolkit="clr-
namespace:Community.VisualStudio.Toolkit;assembly=Community.VisualStudio.Too
lkit"
    toolkit:Themes.UseVsTheme="True">
```

请注意导入的 `xmlns:toolkit` 命名空间和 `toolkit:Themes.UseVsTheme="True"` 属性。它们将自动为 Visual Studio 使用自己的 WPF 控件应用官方样式。我们不必执行任何其他操作即可将样式应用于整个 `<UserControl>` 样式。简单！

另一个好处是，当用户将颜色主题从浅色更改为深色时，我们的 UI 也会立即切换，而无需重新加载。

DialogWindow 控件

Visual Studio 附带了可用于自定义窗口的控件，即控件 `DialogWindow`。建议将其用于任何对话框窗口，但也可以在工具窗口中使用。

它与其他 XAML 窗口类型非常相似。

XML

```
<platform:DialogWindow
    x:Class="TestExtension.ThemWindowDialog"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:platform="clr-
namespace:Microsoft.VisualStudio.PlatformUI;assembly=Microsoft.VisualStudio.
Shell.15.0"
    xmlns:toolkit="clr-
namespace:Community.VisualStudio.Toolkit;assembly=Community.VisualStudio.Too
lkit"
    toolkit:Themes.UseVsTheme="True">
```

请注意工具包和平台和属性 `toolkit:Themes.UseVsTheme="True"` 的导入命名空间。

就这么简单。现在，对话框窗口使用 Visual Studio 颜色和样式主题。

获取源代码

可以在 Community Toolkit 测试项目中[找到此扩展的](#) [源代码](#)。

相关内容

详细了解这些资源的 Visual Studio 颜色。

- [Visual Studio 的颜色和样式](#)
- [Visual Studio 的共享颜色](#)
- [颜色值参考](#)

Visual Studio 扩展中的自定义设置和选项

项目 • 2024/01/13

存储和检索设置是许多扩展的必备项。让我们探讨如何使用以下目标使用设置：

- 提供自定义选项的简单方法。
- 公开“工具>选项”对话框中的选项。
- 访问和修改设置的线程安全方法。
- 同步和异步支持。
- 无需加载包，以便设置进行初始化。

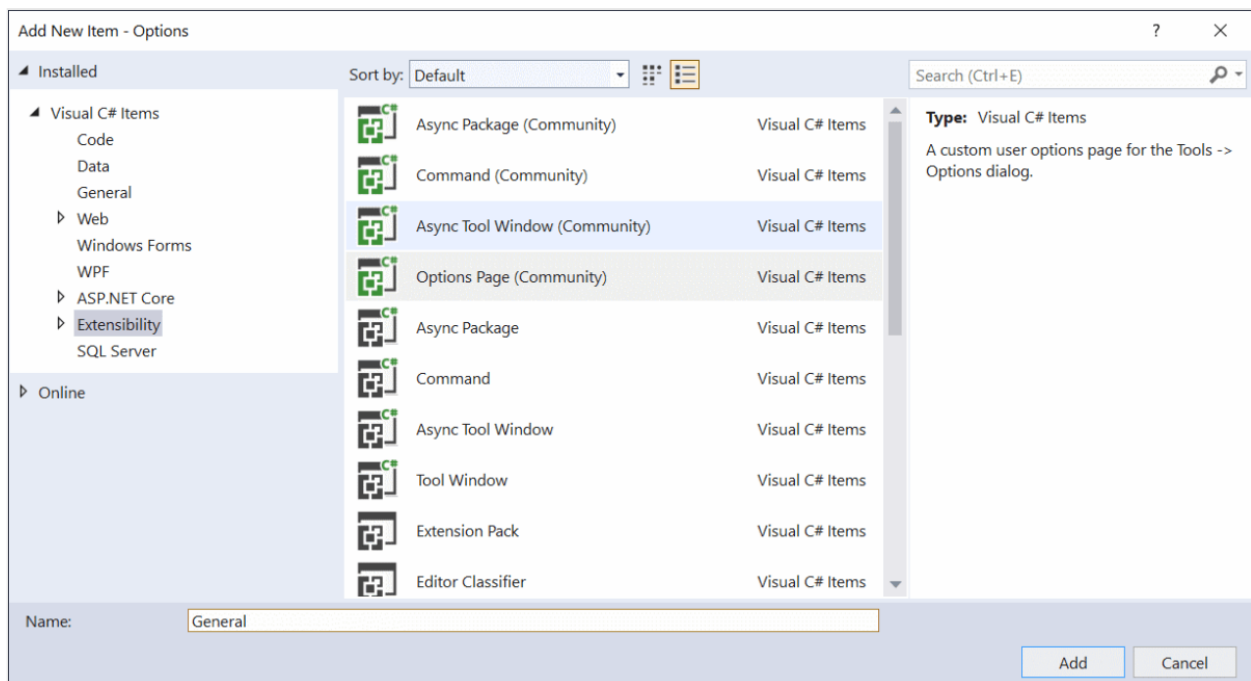
以下视频演示了如何向扩展添加选项。

<https://www.microsoft.com/zh-cn/videoplayer/embed/RWPmjL?postJsllMsg=true&autoCaptions=zh-cn>

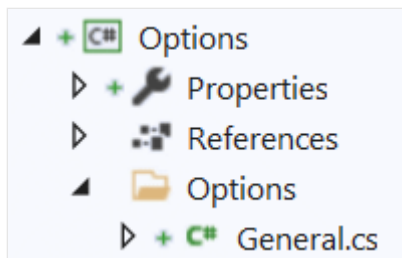
下面是“工具>选项”对话框中的外观。

“添加选项”页

右键单击项目并选择“添加新 > 项...”以显示可用的模板。然后选择左侧的“扩展性”类别，然后选择“选项页”（社区）模板。在下面的名称字段中，编写“常规”。



这将在项目的根目录中创建 `/Options/General.cs`。



以下是 General.cs 文件的内容：

```
C#

internal partial class OptionsProvider
{
    // Register the options with these attributes on your package class:
    // [ProvideOptionPage(typeof(OptionsProvider.GeneralOptions),
    "MyExtension", "General", 0, 0, true)]
    // [ProvideProfile(typeof(OptionsProvider.GeneralOptions),
    "MyExtension", "General", 0, 0, true)]
    public class GeneralOptions : BaseOptionPage<General> { }
}

public class General : BaseOptionModel<General>
{
    [Category("My category")]
    [DisplayName("My Option")]
    [Description("An informative description.")]
    [DefaultValue(true)]
    public bool MyOption { get; set; } = true;
}
```

这是简短和简单的，我们将介绍细节。但首先，我们必须注册“选项”页。

注册“选项”页

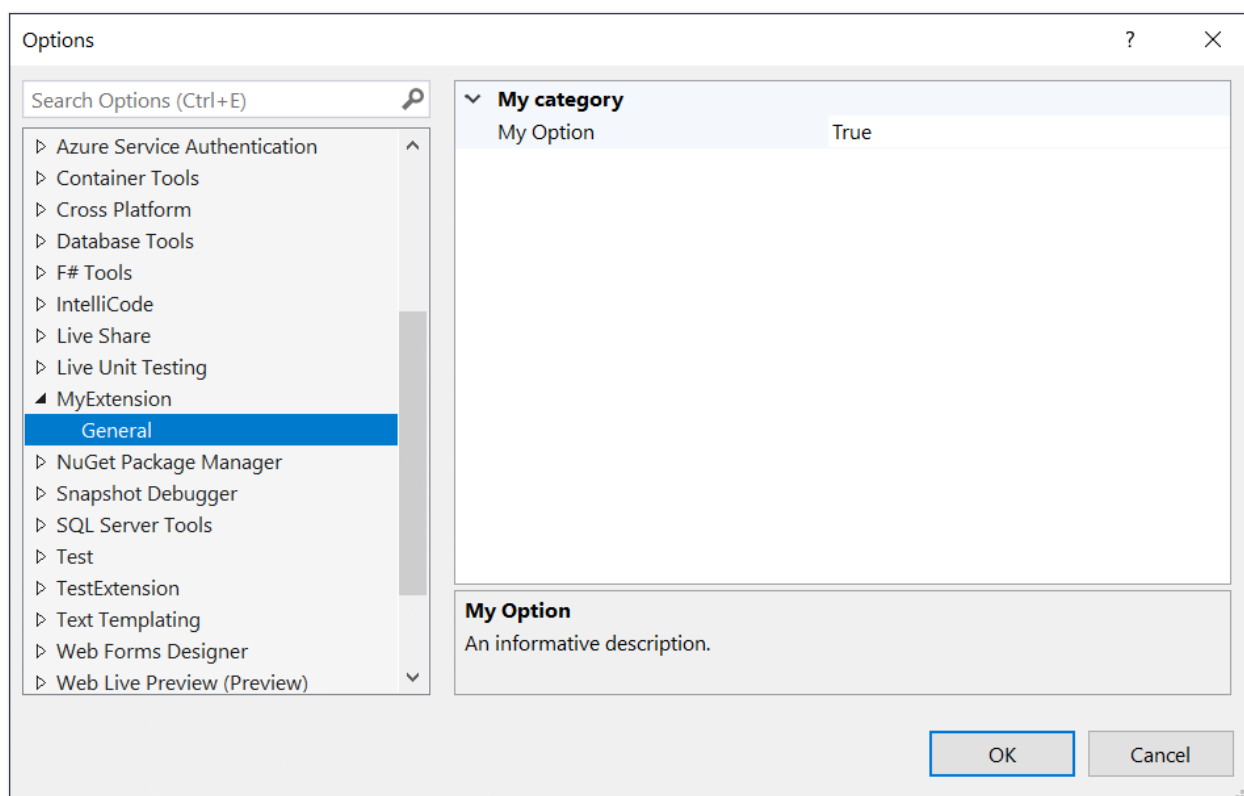
在 General.cs 文件中的代码注释中，说明如何注册“选项”页。

我们只需将这两个属性复制到 Package 类中。这看起来可能如下所示：

```
C#

[ProvideOptionPage(typeof(OptionsProvider.GeneralOptions), "MyExtension",
"General", 0, 0, true)]
[ProvideProfile(typeof(OptionsProvider.GeneralOptions), "MyExtension",
"General", 0, 0, true)]
public sealed class OptionsPackage : ToolkitPackage
{
    ...
}
```

运行扩展后，现在应会看到“工具>选项”对话框中显示的“MyExtension/常规选项”页。



这两个属性非常相似，但处理不同的方案。

该 `ProvideOptionsPage` 属性使“选项”页显示在“工具 > 选项”对话框中。如果不希望用户看到选项页，可以省略此属性。

`ProvideProfile` 在漫游配置文件上注册选项，这意味着各个设置将在实例和 Visual Studio 安装之间与用户帐户一起漫游。它还启用 Visual Studio 的导入/导出设置功能。此属性是可选的。

各个选项

在 `General.cs` 文件中，可以看到各个选项的修饰方式不仅仅是用属性修饰的简单 C# 属性。

C#

```
[Category("My category")]
[DisplayName("My Option")]
[Description("An informative description.")]
[DefaultValue(true)]
public bool MyOption { get; set; } = true;
```

简单数据类型（例如 `string`，`bool`）`int` 都现成支持，涵盖大多数用例。对于其他数据类型，必须使用类型转换器。有些内置于 Visual Studio 中，例如 `EnumConverter`。

请考虑以下枚举：

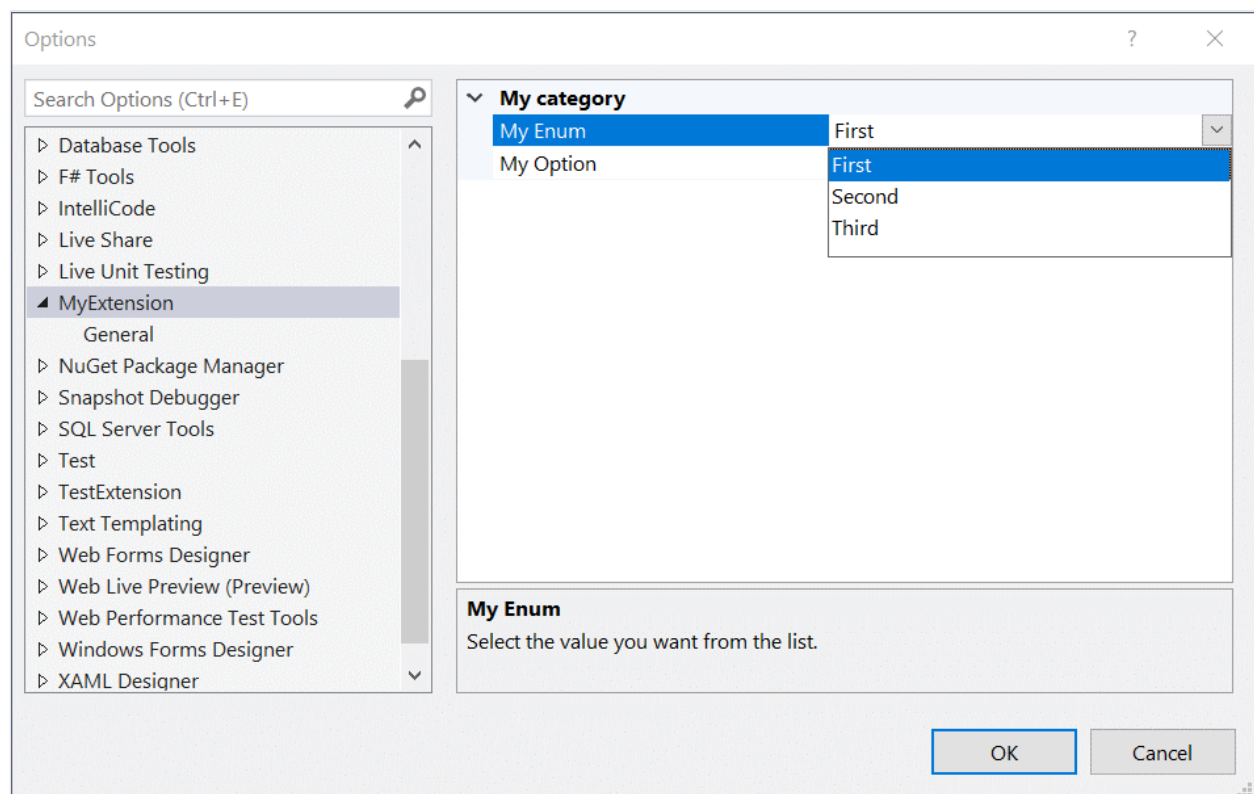
C#

```
public enum Numbers
{
    First,
    Second,
    Third,
}
```

可以通过声明 `TypeConverter` 如下所示，将这些值公开为下拉列表中的选项：

C#

```
[Category("My category")]
[DisplayName("My Enum")]
[Description("Select the value you want from the list.")]
[DefaultValue(Numbers.First)]
[TypeConverter(typeof(EnumConverter))]
public Numbers MyEnum { get; set; } = Numbers.First;
```



读取和写入选项

现在，你已经注册了允许用户更改其值的选项，接下来可以读取这些值以在我们的扩展中使用。

可以使用同步上下文和异步上下文中的设置。 让我们从同步开始：

```
C#

// read settings
var number = General.Instance.MyEnum;

// write settings
General.Instance.MyEnum = Numbers.Second;
General.Instance.Save();
```

用于读取和写入设置的 API 非常简单和直接。

在异步上下文中工作时，API 看起来非常相似。

```
C#

// read settings
var general = await General.GetLiveInstanceAsync();
var number = general.MyEnum;

// write settings
general.MyEnum = Numbers.Second;
await general.SaveAsync();
```

事件

保存设置后，将触发静态事件 `General.Saved` 。 可以像 .NET 中的任何其他事件一样订阅该事件，如下所示：

```
C#

General.Saved += OnSettingsSaved;

...

private void OnSettingsSaved(object sender, General e)
{
}

}
```

获取源代码

可以在示例存储库[中找到此扩展的](#) [源代码](#)。

相关内容

阅读有关这些方案的所有详细信息的文档，但请注意，虽然它们确实提供了更详细的文档，但它们不会遵循此示例中概述的最佳做法。它们也不使用社区工具包，使使用设置变得容易得多。

- [创建选项页](#)
- [使用设置存储](#)
- [写入用户设置存储](#)

在 Visual Studio 扩展中显示通知

项目 • 2024/01/13

有几个机制可用于向扩展的用户显示通知。选择正确的选项可能很有挑战性，因此让我们看看这些选项。

- 状态栏
- 信息栏
- 消息框
- “输出”窗口

它们用于不同的目的，并且对用户的关注具有不同级别的需求。

使用状态栏向用户通知不需要用户的任何操作或输入的事件。如果他们错过了状态栏中的通知，那么没关系 - 看到通知并不重要。

如果想要引起用户的注意，并向他们显示一些要执行的操作，请使用 **信息栏**。他们不必马上做，他们可以等到他们做完他们正在做的事情。通知很重要，但并不重要。

当通知必须阻止当前用户继续执行其操作时，请使用 **消息框**。这是一个阻止和关键通知。

若要通知用户非严重错误，请使用 **输出窗口**。如果要确保用户看到该窗口，则可以将焦点置于输出窗口，但建议不要这样做。

状态栏

状态栏是主窗口底部的一个区域，它显示有关当前窗口状态的信息（例如正在查看的内容和方式）、后台任务（如打印、扫描和格式）或其他上下文信息（如选择和键盘状态）。



当不需要充分关注用户时，请使用状态栏，但仍会向他们提供信息。

设置文本

这会将状态栏中的文本设置为任何字符串。

```
// call it from an async context
await VS.StatusBar.ShowMessageAsync("My text");

// or from a synchronous method:
VS.StatusBar.ShowMessageAsync("My text").FireAndForget();
```

动画图标

将动画图标添加到状态栏非常简单。



只需指定要使用的动画图标。

```
C#

// call it from an async context
await VS.StatusBar.StartAnimationAsync(StatusAnimation.Sync);

// or from a synchronous method:
VS.StatusBar.StartAnimationAsync(StatusAnimation.Sync).FireAndForget();
```

通过调用 `EndStatusbarAnimationAsync` 再次停止动画。

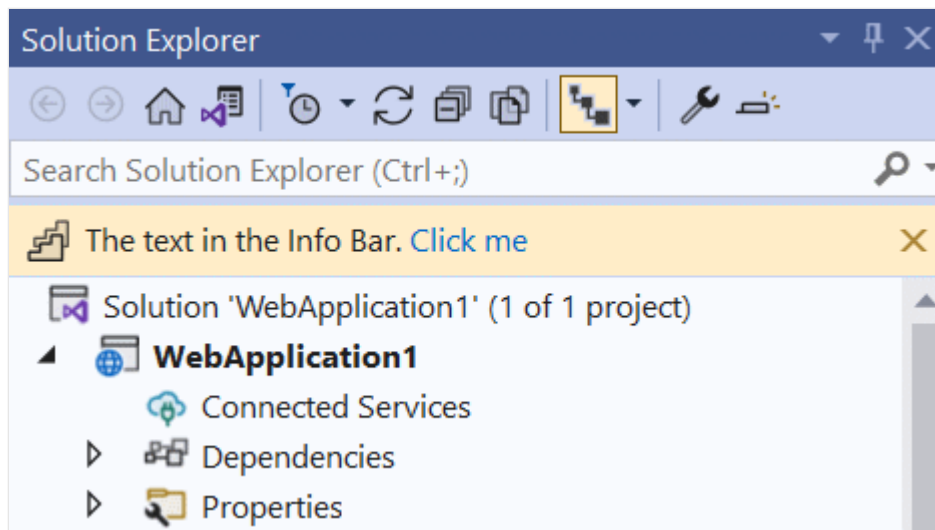
```
C#

// call it from an async context
await VS.StatusBar.EndAnimationAsync(StatusAnimation.Sync);

// or from a synchronous method:
VS.StatusBar.EndAnimationAsync(StatusAnimation.Sync).FireAndForget();
```

信息栏

信息栏是工具窗口或文档窗口顶部的黄色条。它可用于在不阻止用户的情况下吸引用户的注意力。信息栏可以包含图标、文本和多个超链接。



下面介绍如何将信息栏添加到解决方案资源管理器工具窗口。

C#

```
var model = new InfoBarModel(
    new[] {
        new InfoBarTextSpan("The text in the Info Bar. "),
        new InfoBarHyperlink("Click me")
    },
    KnownMonikers.PlayStepGroup,
    true);

InfoBar infoBar =
    VS.InfoBar.CreateInfoBar(ToolWindowGuids80.SolutionExplorer, model);
infoBar.ActionItemClicked += InfoBar_ActionItemClicked;
await infoBar.TryShowInfoBarUIAsync();

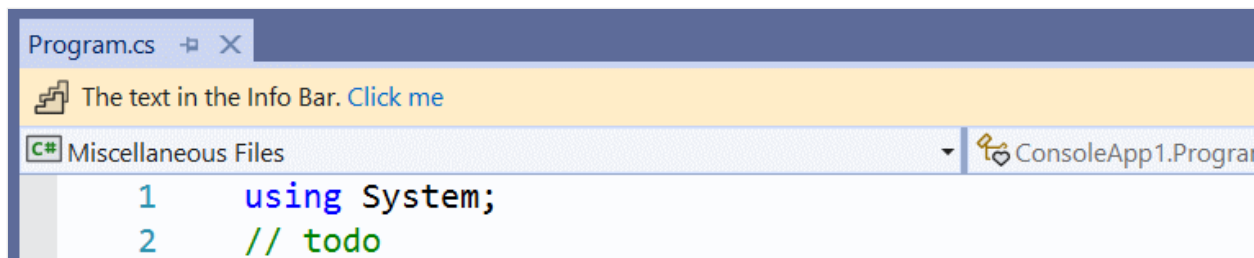
...

private void InfoBar_ActionItemClicked(object sender,
    InfoBarActionItemEventArgs e)
{
    ThreadHelper.ThrowIfNotOnUIThread();

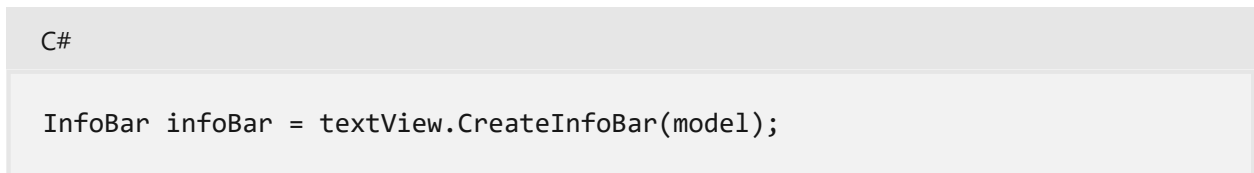
    if (e.ActionItem.Text == "Click me")
    {
        // do something
    }
}
```

若要向文档窗口添加信息栏，只需将打开的文档的文件名传递给

`VS.Notifications.CreateInfoBar(fileName, model)` 该方法。

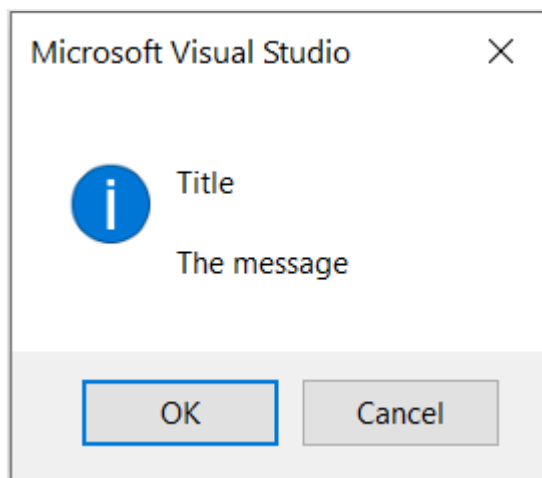


如果想要将信息栏直接转到某个 `ITextView` 信息栏，则可以使用此方便的扩展方法执行此操作：



消息框

可通过多种方式使用 .NET 显示消息框。例如，通过 Windows 窗体 或 WPF。它们会导致 Visual Studio 扩展中针对主窗口正确进行父级的一些问题，因此建议使用 Visual Studio 自己的消息框。

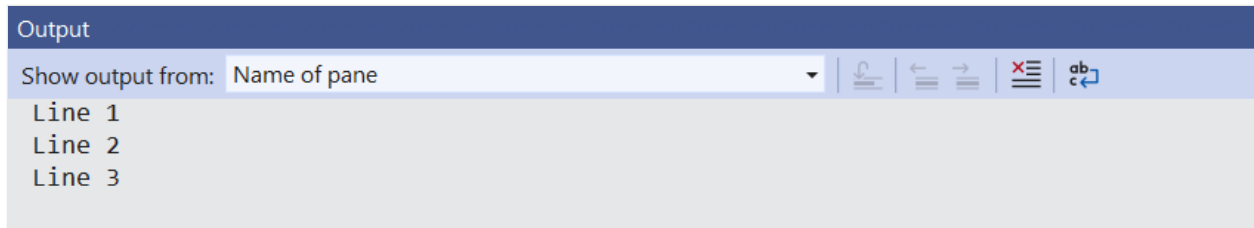


需要阻止 UI 以充分关注用户时，请使用消息框。



输出窗口

使用输出窗口显示有关异常和其他文本信息的信息。



使用该方法时 `VS.Windows.CreateOutputWindowPaneAsync`，创建自定义输出窗口窗格并直接写入该窗格。

C#

```
OutputWindowPane pane = await VS.Windows.CreateOutputWindowPaneAsync("Name  
of pane");  
await pane.WriteLineAsync("Line 1");  
await pane.WriteLineAsync("Line 2");  
await pane.WriteLineAsync("Line 3");
```

有关日志记录异常的详细信息，[请参阅错误处理方案](#)。

Visual Studio 扩展中的错误处理

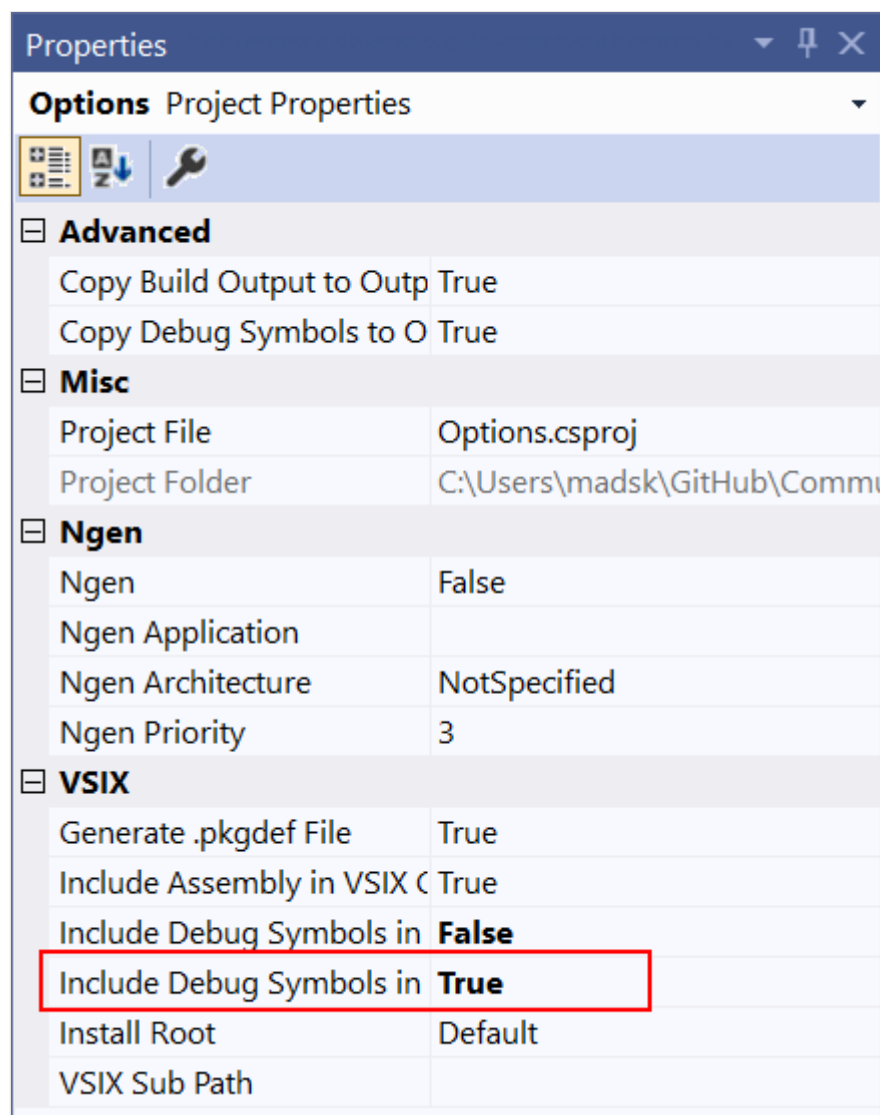
项目 • 2024/01/13

任何程序（包括 Visual Studio 扩展）中都可能出现异常。让我们确保正确处理它们的方式，以优化用户体验时发生的方式。

如果异常是不应发生的异常，则希望以修复异常的方式记录异常的详细信息。根据问题的严重性，你可能还希望让用户知道该问题。

包括符号

为了确保你拥有有关所发生异常的最准确信息，请确保在扩展名中包含 .pdb 文件。默认情况下启用此功能，但请确保通过显示项目的属性（F4）来检查。



请确保将 VSIX 容器中的“包括调试符号”属性设置为 True。

这会设置收集所需的信息。让我们看看一些策略来执行此操作。

自动遥测

可以使用可在扩展中的任何 .NET 应用程序中使用的任何遥测机制。常用选项包括 [Application Insights](#)、[Raygun](#)、[Google Analytics](#) 等。

必须使用这些遥测系统 API 手动报告异常详细信息。此选项是保持用户计算机上出现任何问题并让你提前主动解决问题的好方法。

使用此选项时，请确保在隐私声明中提及，因为某些用户不喜欢报告遥测数据。

记录到输出窗口

使用 `try/catch` 块处理异常时，让用户知道出了问题可能很有利。这样，他们就可以轻松地以易于修复的方式向你报告问题。

最佳方法之一是将异常详细信息输出到输出窗口。这样，用户可以看到发生异常，并向他们提供堆栈跟踪，以在 bug 报告中向你发送。

借助社区工具包，可以轻松执行此操作。在同步上下文中，只需在任何上下文 `Exception` 上使用 `Log()` 扩展方法。

```
C#  
  
try  
{  
    // Do work;  
}  
catch (Exception ex)  
{  
    ex.Log();  
}
```

对于异步上下文，等待扩展方法执行 `LogAsync()` 相同的操作。

```
C#  
  
try  
{  
    // Do work;  
}  
catch (Exception ex)  
{  
    await ex.LogAsync();  
}
```


通知用户

如果异常对用户严重性较低，则没有理由中断这些异常。 请考虑使用状态栏来显示发生错误。

如果异常严重并导致用户流中断，请考虑使用消息框让用户知道错误。 确保仍通过遥测和/或输出窗口记录异常，如前所述。

若要详细了解如何使用状态栏和消息框，请参阅 [“通知”食谱](#)。

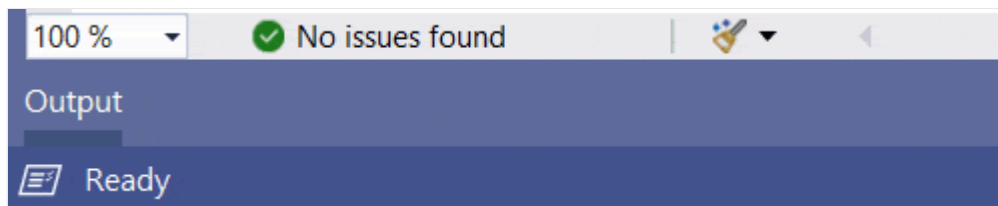
Visual Studio 扩展中后台任务的进度栏

项目 • 2024/01/13

可通过多种方式在 Visual Studio 中显示正在运行的后台任务的进度。下面介绍如何使用你自己的扩展中的进度栏。

状态栏

状态栏具有自己的进度指示器，它是向用户显示进度的最简单位置。它可见，但不要求用户注意或阻止其当前任务。对于大多数后台任务，状态栏是一个不错的选择。



使用 API 显示状态栏进度非常简单：

C#

```
await Task.Delay(1000); // long running task
await VS.StatusBar.ShowProgressAsync("Step 1/3", 1, 3);

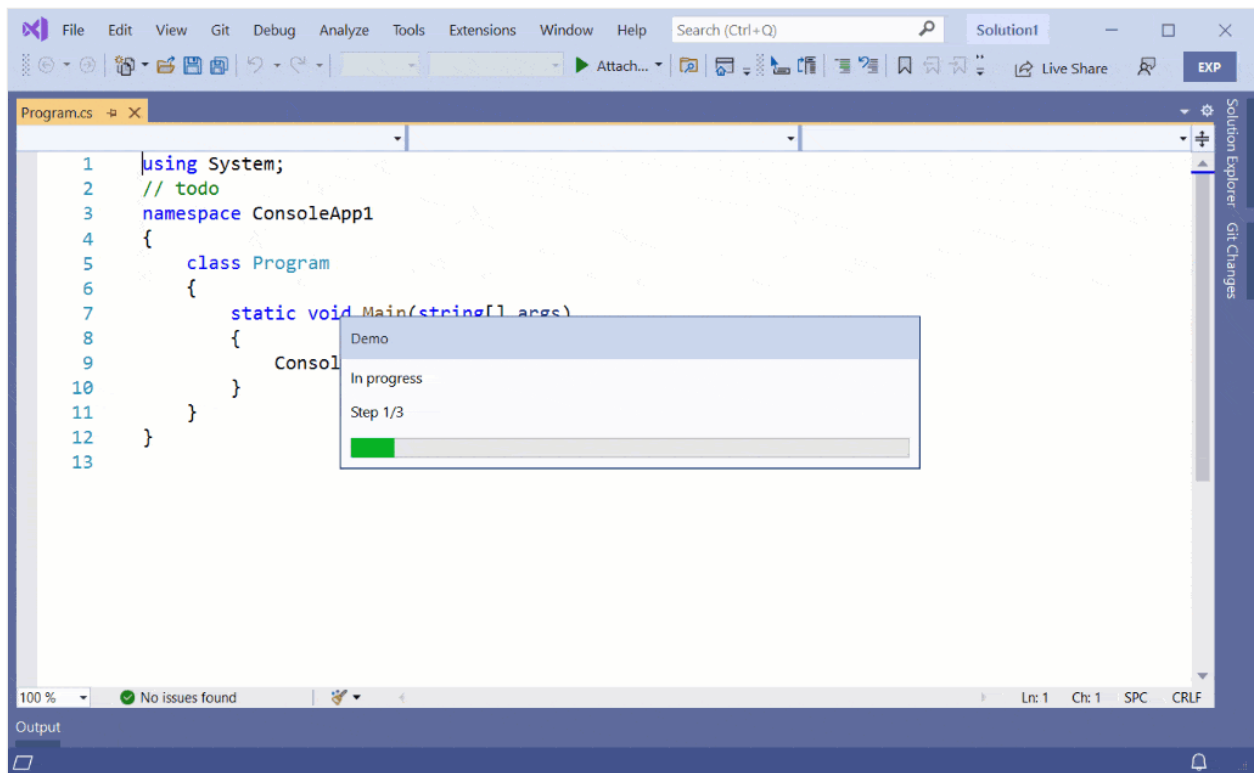
await Task.Delay(1000); // long running task
await VS.StatusBar.ShowProgressAsync("Step 2/3", 2, 3);

await Task.Delay(1000); // long running task
await VS.StatusBar.ShowProgressAsync("Step 3/3", 3, 3);

// Progress ends as current step (3) and total steps (3) are equal
```

线程等待对话框

线程等待对话框 (XP) 是一个进度指示器对话框，仅当后台任务花费的时间超过扩展程序指定的 x 秒数时才会弹出。它会一整天将进度写入状态栏，但在运行 x 秒数后会显示对话框。



每当需要时，都无法显示该网。它由 Visual Studio 管理，因此不会过多干扰最终用户。使用它所需的代码不仅仅是使用状态栏，但它仍然是一个相对简单的 API 才能使用：

C#

```
var fac = await VS.Services.GetThreadedWaitDialogAsync() as
IVsThreadedWaitDialogFactory;
IVsThreadedWaitDialog4 twd = fac.CreateInstance();

twd.StartWaitDialog("Demo", "Working on it...", "", null, "", 1, true,
true);

var totalSteps = 3;

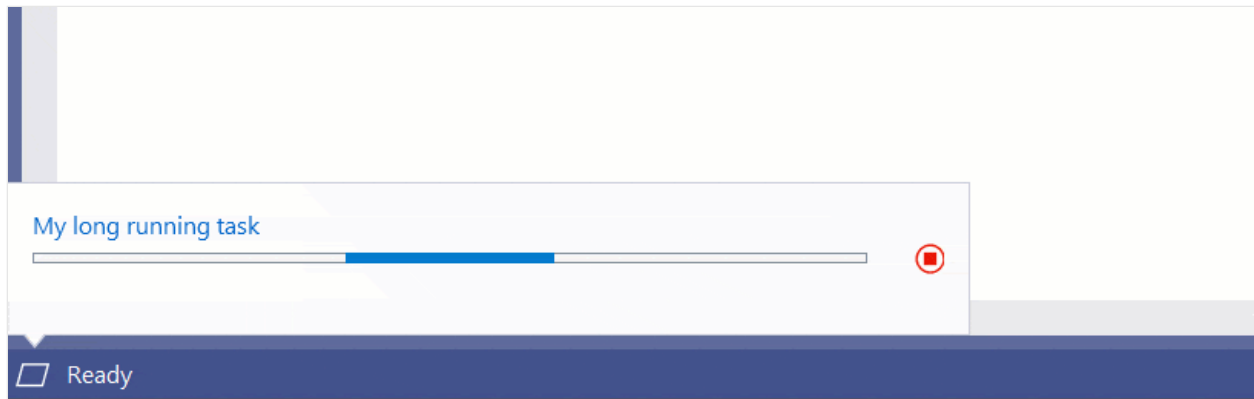
for (var currentStep = 1; currentStep <= totalSteps; currentStep++)
{
    var text = $"Step {currentStep}/{totalSteps}";
    twd.UpdateProgress("In progress", text, text, currentStep, totalSteps,
true, out _);

    await Task.Delay(1000); // long running task

    if (currentStep == totalSteps)
    {
        // Dismisses the dialog
        (twd as IDisposable).Dispose();
    }
}
```

任务状态中心

状态栏左下角是任务状态中心（TSC）。它用于报告长时间运行的后台任务的进度，并由 Visual Studio 中的许多服务使用。



API 具有比使用其他显示进度的方法更多的概念。

C#

```
private async Task StartAsync()
{
    IVsTaskStatusCenterService tsc = await
    VS.Services.GetTaskStatusCenterAsync();

    var options = default(TaskHandlerOptions);
    options.Title = "My long running task";
    options.ActionsAfterCompletion = CompletionActions.None;

    TaskProgressData data = default;
    data.CanBeCanceled = true;

    ITaskHandler handler = tsc.PreRegister(options, data);
    Task task = LongRunningTaskAsync(data, handler);
    handler.RegisterTask(task);
}

private async Task LongRunningTaskAsync(TaskProgressData data, ITaskHandler
handler)
{
    float totalSteps = 3;

    for (float currentStep = 1; currentStep <= totalSteps; currentStep++)
    {
        await Task.Delay(1000);

        data.PercentComplete = (int)(currentStep / totalSteps * 100);
        data.ProgressText = $"Step {currentStep} of {totalSteps} completed";
        handler.Progress.Report(data);
    }
}
```

相关内容

- [任务状态中心 API 参考](#)

使用 Visual Studio 扩展中的文件和文档

项目 • 2024/01/13

下面是一系列关于处理文件和文档的不同方法的小代码示例。

获取活动文本视图

获取当前活动文本视图以操作其文本缓冲区文本。

C#

```
DocumentView docView = await VS.Documents.GetActiveDocumentViewAsync();
if (docView?.TextView == null) return; //not a text window
SnapshotPoint position = docView.TextView.Caret.Position.BufferPosition;
docView.TextBuffer?.Insert(position, "some text"); // Inserts text at the
caret
```

文件图标关联

若要将图标与 解决方案资源管理器 中的文件扩展名相关联，请将该 `[ProvideFileIcon()]` 属性添加到包类。

C#

```
[ProvideFileIcon(".abc", "KnownMonikers.Reference")]
public sealed class MyPackage : ToolkitPackage
{
    ...
}
```

使用 KnownMonikers 资源管理器工具窗口查看集合中的 `KnownMonikers` 数千个可用图标。在主菜单中的“查看>其他 Windows”下找到它。

打开文件

`Microsoft.VisualStudio.Shell.VsShellUtilities` 使用帮助程序类。

C#

```
string fileName = "c:\\file.txt";
await VS.Document.OpenAsync(fileName);
```

通过项目打开文件

打开的文件是解决方案的一部分时，请使用此方法。

C#

```
string fileName = "c:\\file.txt";  
await VS.Documents.OpenViaProjectAsync(fileName);
```

在“预览”选项卡中打开文件

“预览”选项卡（也称为“临时”选项卡）是在文档右侧打开的临时选项卡。在“预览”选项卡中打开任何文件，如下所示：

C#

```
string fileName = "c:\\file.txt";  
await VS.Documents.OpenInPreviewTabAsync(fileName);
```

从 ITextBuffer 获取文件名

使用位于命名空间中的 `Microsoft.VisualStudio.Text` 扩展方法 `buffer.GetFileName()`。

C#

```
string fileName = buffer.GetFileName();
```

来自文件的 SolutionItem

`SolutionItem` 从绝对文件路径中查找。

C#

```
string fileName = "c:\\file.txt";  
PhysicalFile item = await PhysicalFile.FromFileAsync(fileName);
```

使用 Visual Studio 扩展中的项目

项目 • 2024/01/13

下面是一系列关于处理项目的不同方法的小型代码示例。

从包含的文件获取项目

这是如何从其中一个项目的文件获取项目。

C#

```
string fileName = "c:\\file\\in\\project.txt";
PhysicalFile item = await PhysicalFile.FromFileAsync(fileName);
Project project = item.ContainingProject;
```

将文件添加到项目

下面介绍如何将文件从磁盘添加到项目。

C#

```
Project project = await VS.Solutions.GetActiveProjectAsync();

var file1 = "c:\\file\\in\\project\\1.txt";
var file2 = "c:\\file\\in\\project\\2.txt";
var file3 = "c:\\file\\in\\project\\3.txt";

await project.AddExistingFilesAsync(file1, file2, file3);
```

查找项目类型

了解要处理的项目类型。

C#

```
bool isCsharp = await project.IsKindAsync(ProjectTypes.CSHARP);
```


使用 Visual Studio 扩展中的解决方案

项目 • 2024/01/13

下面是一系列关于使用解决方案的不同方法的小型代码示例。

解决方案事件

侦听任何解决方案事件。

```
C#

VS.Events.SolutionEvents.OnAfterOpenProject += OnAfterOpenProject;

...

private void OnAfterOpenProject(Project obj)
{
    // Handle the event
}
```

解决方案是否处于打开状态？

检查解决方案当前是否处于打开状态或打开状态。

```
C#

bool isOpen = await VS.Solutions.IsOpenAsync();
bool isOpening = await VS.Solutions.IsOpeningAsync();
```

获取解决方案中的所有项目

获取解决方案中所有项目的列表。

```
C#

var projects = await VS.Solutions.GetAllProjectsAsync();
```

使用 Visual Studio 扩展中的生成

项目 • 2024/01/13

下面是一系列关于使用生成的不同方法的小型代码示例。

生成解决方案

若要生成整个解决方案，请调用 `BuildAsync()` 该方法。

C#

```
bool buildStarted = await VS.Build.BuildSolutionAsync(BuildAction.Build);
```

生成项目

可以通过将项目传递给方法来生成任何项目。

C#

```
Project project = await VS.Solutions.GetActiveProjectAsync();  
await project.BuildAsync(BuildAction.Rebuild);
```

设置生成属性

演示如何在项目上设置生成属性。

C#

```
Project project = await VS.Solutions.GetActiveProjectAsync();  
bool succeeded = await project.TrySetAttributeAsync("propertyName",  
"value");
```

获取生成属性

演示如何获取任何项目或项目项的生成属性。

C#

```
Project item = await VS.Solutions.GetActiveProjectAsync();
```

```
string value = await item.GetAttributeAsync("propertyName");
```

发布 Visual Studio 扩展

项目 • 2024/01/13

本部分可帮助你让扩展准备好与你的团队或整个 Visual Studio 用户世界共享。

检查检查列表

在与任何人共享扩展之前，请确保检查[检查列表](#)，以确保扩展遵循最佳做法。

发布到市场

以下视频演示如何通过上传新扩展并将其发布到市场来共享新扩展。




<https://www.microsoft.com/zh-cn/videoplayer/embed/RWP8KB?postJsllMsg=true&autoCaptions=zh-cn> [↗](#)

在专用库上发布

还可以在任何 Web 主机上在组织内部或公共中托管自己的扩展库。

用于发布 Visual Studio 扩展的最佳做法检查列表

项目 • 2023/10/10

适用范围：  Visual Studio  Visual Studio for Mac  Visual Studio Code

下面是在发布 Visual Studio 扩展之前确保记住的事项列表。

以下视频介绍了最佳做法，以确保扩展是最佳扩展。

<https://www.microsoft.com/zh-cn/videoplayer/embed/RWPmjM?postJsllMsg=true&autoCaptions=zh-cn>

遵循线程规则

将 [Microsoft.VisualStudio.SDK.Analyzers](#) NuGet 包添加到 VSIX 项目，这有助于发现和修复线程处理上常见最佳做法的冲突。

添加高质量图标

所有扩展都应具有与之关联的图标。确保图标是一个高质量的 .png 文件，大小为 90x90 像素（以 96 DPI 或更高版本为单位）。将图标添加到 VSIX 项目后，在 .vsixmanifest 文件中将其注册为图标和预览图像。

名称和说明

研究表明，具有简短和描述性名称的扩展和准确的说明更有可能由用户安装。确保名称反映扩展的本质。 .vsixmanifest 文件中的简短说明应将预期设置为扩展的作用。因此，简要提及它所解决的问题及其具有的主要功能。

编写良好的市场说明

这是使扩展成功时应执行的最重要操作之一。良好的描述包括：

- 扩展添加的 UI 的屏幕截图/动画 GIF。
- 各个功能的详细说明。
- 指向更多详细信息的链接（如果适用）。

添加许可证

此许可证将显示在市场、VSIX 安装程序和“**扩展**”和“**汇报...**”对话框中。应始终指定许可证来设置用户的期望。使用 choosealicense.com 来帮助查找适合你的许可证。许可证对于帮助删除任何问题和歧义非常重要，这对许多 Visual Studio 用户来说非常重要。

添加隐私声明

如果扩展收集遥测等数据，或者以任何其他方式与远程终结点通信，请在说明中添加有关该数据的说明。

尽可能使用 KnownMonikers

Visual Studio 附带了 KnownMonikers 集中提供的数千个图标。将图标添加到命令按钮时，查看是否可以使用现有的 KnownMonikers 图标，因为它们是 Visual Studio 用户熟悉的设计语言的一部分。下面是 KnownMonikers 的完整列表，并获取 [KnownMonikers Explorer](#) 扩展以查找适合你的方案。

使它感觉本机到 VS

遵循 Visual Studio 本身使用的相同设计模式和原则，使扩展对用户感觉自然。它还减少了设计不善的 UI 造成的干扰。确保所有按钮、菜单、工具栏和工具窗口在用户处于正确的上下文中才能使用它们时默认可见。有一些经验规则要遵循：

- 请勿添加新的顶级菜单（文件、编辑、...旁）。
- 在它们不适用的上下文中，不应显示任何按钮、菜单和工具栏。
- 如果需要 [自动加载](#)（可能不是），请尽可能晚执行。
- 使用 [VisibilityConstraints](#) 切换命令的可见性，而不是依赖于自动加载。

使用适当的版本范围

一直支持 Visual Studio 2010 版本，以确保每个人都可以使用新的扩展，这很诱人。问题在于，通过这样做，不再可以使用扩展支持的最低版本之后引入的任何 API。通常，这些新 API 非常重要，有助于提高扩展和 Visual Studio 本身的性能和可靠性。

下面是用于决定支持哪些版本的 Visual Studio 的建议：

- 仅支持早期版本和当前版本的 Visual Studio - 如果可能，则不支持旧版本。
- 例如，`[16.0,)` 不要指定开放式版本范围。详细了解 [版本范围](#)。

创建扩展包

项目 • 2024/01/13

本文介绍如何创建扩展包。扩展包是一组可以一起安装的扩展。借助扩展包，可以轻松地将其他用户喜欢的扩展，或将一组扩展捆绑到一起，以用于特定场景。

以下视频介绍了如何创建扩展包。

<https://www.microsoft.com/zh-cn/videoplayer/embed/RWP8KA?postJsllMsg=true&autoCaptions=zh-cn>

从项目模板创建

扩展包项目模板创建一个扩展包，其中包含一组可以一起安装的扩展。

在“**新建项目**”对话框中，搜索 *扩展* 并选择“**扩展包**”。对于 **项目** 名称，请输入 *测试扩展包*。选择**创建**。

Visual Studio 在 解决方案资源管理器 中打开项目，并在编辑器中打开文件扩展名.vsix。

JSON

```
{
  "version": "1.0.0.0",
  "extensions": [
    {
      "vsixId": "OneDarkPro.e1e706e2-05d3-4da9-8754-652cd8ab65f4",
      "name": "One Dark Pro"
    },
    {
      "vsixId": "7fa839e2-b938-4b1c-9277-edaeb6fdeb5",
      "name": "Winter is Coming"
    }
  ]
}
```




添加到现有扩展

在解决方案资源管理器中，右键单击项目节点，然后选择“**添加新>项**”。转到“Visual C# 扩展性”节点，然后选择“扩展包”。保留默认文件名 (ExtensionPack1.cs)。

项目的根目录中的 .vsix 文件是将项目转换为扩展包的内容。只需确保其“生成操作”设置为“内容”，VSIX 中的“包含”设置为 *True*，如下所示。

Properties

Extensions.vsex File Properties

☒ **Advanced**

Build Action	Content
Copy to Output Directory	Do not copy
Custom Tool	
Custom Tool Namespace	

☒ **Misc**

File Name	Extensions.vsex
Full Path	C:\Users\madsk\source\repos\I

☒ **VSIX**

Include in VSIX	True
Install Root	Default
Target Path	
VSIX Sub Path	