Creating an Extension with a Menu Command

本文内容

[Prerequisites](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/creating-an-extension-with-a-menu-command" \l "prerequisites)

[Creating a Menu Command](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/creating-an-extension-with-a-menu-command" \l "creating-a-menu-command)

[Changing the Menu Command Handler](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/creating-an-extension-with-a-menu-command" \l "changing-the-menu-command-handler)

[Cleaning up the Experimental Environment](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/creating-an-extension-with-a-menu-command" \l "cleaning-up-the-experimental-environment)

[Deploying your extension](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/creating-an-extension-with-a-menu-command" \l "deploying-your-extension)

[Next Steps](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/creating-an-extension-with-a-menu-command" \l "next-steps)

This walkthrough shows how to create an extension with a menu command that launches Notepad.

Prerequisites

Starting in Visual Studio 2015, you do not install the Visual Studio SDK from the download center. It is included as an optional feature in Visual Studio setup. You can also install the VS SDK later on. For more information, see [Installing the Visual Studio SDK](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/installing-the-visual-studio-sdk).

Creating a Menu Command

**1.** Create a VSIX project named **FirstMenuCommand**. You can find the VSIX project template in the **New Project** dialog under **Visual C# / Extensibility**.

**2.** When the project opens, add a custom command item template named **FirstCommand**. In the **Solution Explorer**, right-click the project node and select **Add / New Item**. In the **Add New Item** dialog, go to **Visual C# / Extensibility** and select Custom Command. In the **Name** field at the bottom of the window, change the command file name to **FirstCommand.cs**.

**3.** Build the project and start debugging.

The experimental instance of Visual Studio appears. For more information about the experimental instance, see [The Experimental Instance](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/the-experimental-instance).

**4.** In the experimental instance, open the **Tools / Extensions and Updates** window. You should see the **FirstMenuCommand** extension here. (If you open **Extensions and Updates** in your working instance of Visual Studio, you won't see **FirstMenuCommand**).

Now go to the **Tools** menu in the experimental instance. You should see **Invoke FirstCommand** command. At this point it just brings up a message box that says "FirstCommandPackage Inside FirstMenuCommand.FirstCommand.MenuItemCallback()". We'll see how to actually start Notepad from this command in the next section.

Changing the Menu Command Handler

Now let's update the command handler to start Notepad.

**1.** Stop debugging and go back to your working instance of Visual Studio. Open the FirstCommand.cs file and add the following using statement:

using System.Diagnostics;

**2.** Find the private FirstCommand constructor. This is where the command is hooked up to the command service and the command handler is specified. Change the name of the command handler to StartNotepad, as follows:

private FirstCommand(Package package)

{

if (package == null)

{

throw new ArgumentNullException(nameof(package));

}

this.package = package;

OleMenuCommandService commandService = this.ServiceProvider.GetService(typeof(IMenuC ommandService)) as OleMenuCommandService;

if (commandService != null)

{

CommandID menuCommandID = new CommandID(CommandSet, CommandId);

// Change to StartNotepad handler.

MenuCommand menuItem = new MenuCommand(this.StartNotepad, menuCommandID);

commandService.AddCommand(menuItem);

}

}

**3.** Remove the MenuItemCallback method and add a StartNotepad method which will just start Notepad:

private void StartNotepad(object sender, EventArgs e)

{

Process proc = new Process();

proc.StartInfo.FileName = "notepad.exe";

proc.Start();

}

**4.** Now try it out. When you start debugging the project and click **Tools / Invoke FirstCommand**, you should see an instance of Notepad come up.

You can use an instance of the **Process** class to run any executable, not just Notepad. Try it with calc.exe, for example.

Cleaning up the Experimental Environment

If you are developing multiple extensions, or just exploring outcomes with different versions of your extension code, your experimental environment may stop working the way it should. In this case, you should run the reset script. It's called **Reset the Visual Studio 2015 Experimental Instance**, and it ships as part of the Visual Studio SDK. This script removes all references to your extensions from the experimental environment, so you can start from scratch.

You can get to this script in one of two ways:

**1.** From the desktop, find **Reset the Visual Studio 2015 Experimental Instance**.

**2.** From the command line, run the following:

<VSSDK installation>\VisualStudioIntegration\Tools\Bin\CreateExpInstance.exe /Reset /VSInstance=14.0 /RootSuffix=Exp && PAUSE

Deploying your extension

Now that you have your tool extension running the way you want, it's time to think about sharing it with your friends and colleagues. That's easy, as long as they have Visual Studio 2015 installed. All you have to do is send them the .vsix file you built. (Be sure to build it in Release mode.)

You can find the .vsix file for this extension in the FirstMenuCommand bin directory. Specifically, assuming you have built the Release configuration, it will be in:

**<code directory>\FirstMenuCommand\FirstMenuCommand\bin\Release\ FirstMenuCommand.vsix**

To install the extension, your friend needs to close all open instances of Visual Studio, then double-click the .vsix file, which brings up the **VSIX Installer**. The files are copied to the **%LocalAppData%\Microsoft\VisualStudio\14.0\Extensions** directory.

When your friend brings up Visual Studio again, he'll find the FirstMenuCommand extension in **Tools / Extensions and Updates**. He can go to **Extensions and Updates** to uninstall or disable the extension, too.

Next Steps

This walkthrough has shown you only a small part of what you can do with a Visual Studio extension. Here's a short list of other (reasonably easy) things you can do with Visual Studio extensions:

**1.** You can do many more things with a simple menu command:

**a.** Add your own icon: [Adding Icons to Menu Commands](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/adding-icons-to-menu-commands)

**b.** Change the text of the menu command: [Changing the Text of a Menu Command](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/changing-the-text-of-a-menu-command)

**c.** Add a menu shortcut to a command: [Binding Keyboard Shortcuts to Menu Items](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/binding-keyboard-shortcuts-to-menu-items)

**2.** Add different kinds of commands, menus, and toolbars: [Extending Menus and Commands](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/extending-menus-and-commands)

**3.** Add tool windows and extend the built-in Visual Studio tool windows: [Extending and Customizing Tool Windows](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/extending-and-customizing-tool-windows)

**4.** Add IntelliSense, code suggestions, and other features to existing code editors: [Extending the Editor and Language Services](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/extending-the-editor-and-language-services)

**5.** Add Options and Property pages and user settings to your extension: [Extending Properties and the Property Window](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/extending-properties-and-the-property-window) and [Extending User Settings and Options](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/extending-user-settings-and-options)

Other kinds of extensions require a little more work, such as creating a new type of project ([Extending Projects](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/extending-projects)), creating a new type of editor ([Creating Custom Editors and Designers](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/creating-custom-editors-and-designers)), or implementing your extension in an isolated shell: [Visual Studio Isolated Shell](https://docs.microsoft.com/zh-cn/visualstudio/extensibility/visual-studio-isolated-shell)