VISUAL STUDIO 2022 SUCCINCTLY

BY **ALESSANDRO DEL SOLE**

SUCCINCTLY EBOOK SERIES

Syncfusion®

Visual Studio 2022 Succinctly

Alessandro Del Sole

Foreword by Daniel Jebaraj

ISBN: 978-1-64200-232-4

Technical Reviewer: James McCaffrey
Copy Editor: Tres Watkins, VP of content, Syncfusion, Inc.
Acquisitions Coordinator: Tres Watkins, VP of content, Syncfusion, Inc.
Proofreader: Graham High, senior content producer, Syncfusion, Inc.

Table of Contents

The *Succinctly* Series of Books

Daniel Jebaraj
CEO of Syncfusion, Inc.

When we published our first *Succinctly* series book in 2012, *jQuery Succinctly*, our goal was to produce a series of concise technical books targeted at software developers working primarily on the Microsoft platform. We firmly believed then, as we do now, that most topics of interest can be translated into books that are about 100 pages in length.

We have since published over 200 books that have been downloaded millions of times. Reaching more than 2.7 million readers around the world, we have more than 70 authors who now cover a wider range of topics, such as Blazor, machine learning, and big data.

Each author is carefully chosen from a pool of talented experts who share our vision. The book before you and the others in this series are the result of our authors' tireless work. Within these pages, you will find original content that is guaranteed to get you up and running in about the time it takes to drink a few cups of coffee.

We are absolutely thrilled with the enthusiastic reception of our books. We believe the *Succinctly* series is the largest library of free technical books being actively published today. Truly exciting!

Our goal is to keep the information free and easily available so that anyone with a computing device and internet access can obtain concise information and benefit from it. The books will always be free. Any updates we publish will also be free.

Let us know what you think

If you have any topics of interest, thoughts, or feedback, please feel free to send them to us at succinctlyseries@syncfusion.com.

We sincerely hope you enjoy reading this book and that it helps you better understand the topic of study. Thank you for reading.

Please follow us on social media and help us spread the word about the *Succinctly* series!

About the Author

Alessandro Del Sole is a Xamarin Certified Mobile Developer and has been a Microsoft MVP since 2008. Awarded MVP of the Year in 2009, 2010, 2011, 2012, and 2014, he is internationally considered a Visual Studio expert and a .NET authority. Alessandro has authored many printed books and ebooks on programming with Visual Studio, including *Xamarin with Visual Studio*, *Visual Studio 2019 Succinctly*, *Visual Basic 2015 Unleashed*, and *Xamarin.Forms Succinctly*. He has written tons of technical articles about .NET, Visual Studio, and other Microsoft technologies in Italian and English for many developer portals, including MSDN Magazine and the Visual Basic Developer Center from Microsoft. He is a frequent speaker at Italian conferences and he has released a number of Windows Store apps. He has also produced instructional videos in both English and Italian. Alessandro works as a senior software engineer for Fresenius Medical Care, focusing on building mobile apps with Xamarin in the healthcare market. You can follow him on Twitter at @progalex and support him on Patreon.

Introduction

Visual Studio 2022 is the latest major release of the premier development environment from Microsoft. There are certainly many new features, as you would expect from a new version, but this time there are also very important changes in the IDE architecture. First, Visual Studio is now a 64-bit application. This is extremely important because now the IDE can fully leverage system resources, such as RAM instead of disk space, with significant performance improvements. In addition, Visual Studio 2022 focuses on supporting .NET 6 and its ecosystem, including the latest versions of .NET programming languages such as C# 10, providing both tools and code editor improvements that you need to work with the new development technologies and the most recent .NET APIs.

The release cycle of updates to Visual Studio makes it also ready to support future releases of important Microsoft development platforms, like .NET MAUI for cross-platform development. Of course, you will also find many productivity improvements, from the code editing experience, to enhanced debugging tools, to better integrated collaboration tools. In general, you will feel at home with Visual Studio 2022 since there are no particular changes in the user interface. You will rather find more powerful tools in every area of the development lifecycle.

In this book, I will use the terms Visual Studio 2022 and VS 2022 interchangeably. Continuing the tradition of the Visual Studio *Succinctly* books, the following chapters focus on what's new and updated in the integrated development environment, so new features of programming languages are excluded from the discussion unless required to describe new IDE features.

Note: Topics discussed in this ebook apply to all the editions of Visual Studio 2022 (Community, Professional, and Enterprise).

In the next sections, you will learn about installing Visual Studio 2022 and how to set it up for immediate productivity. This book is based on Visual Studio 2022 version 17.2, which is the latest available at the time of this writing.

Available editions

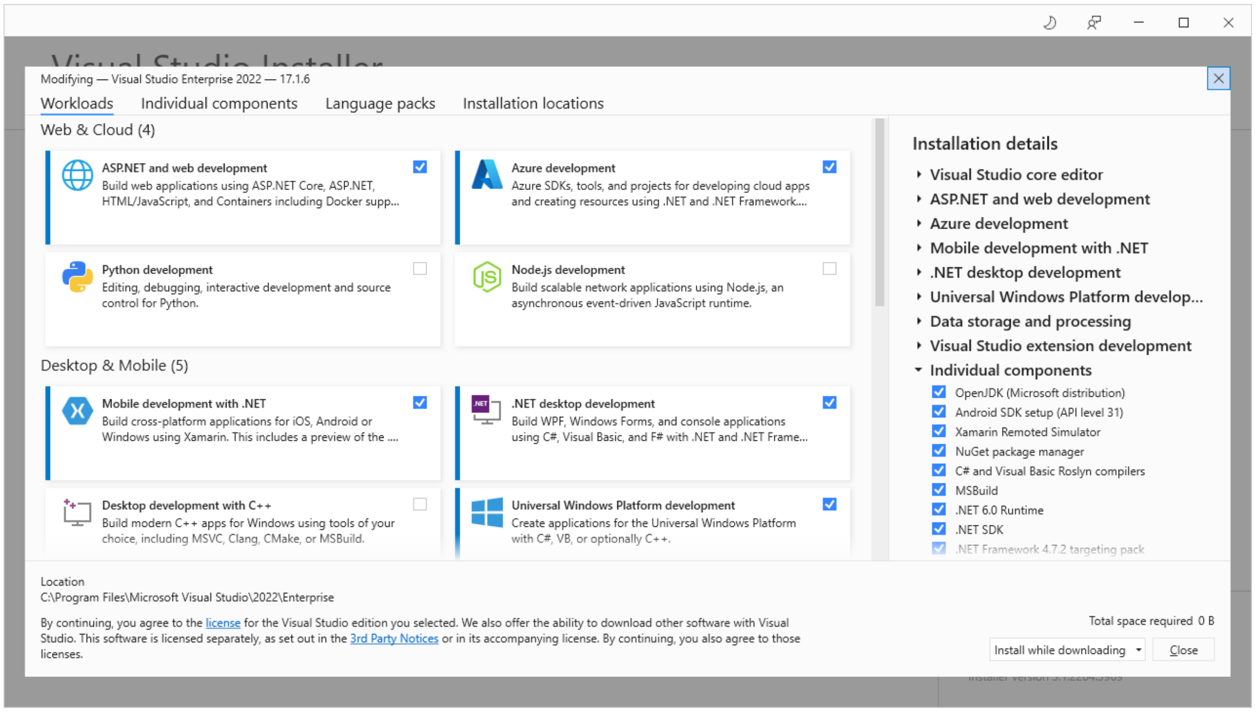
Like its predecessor, Visual Studio 2022 is available in the following editions:

- Community
- Professional
- Enterprise

The installer for each edition can be downloaded from the Visual Studio website and is called VisualStudioInstaller.exe. For the Professional and Enterprise editions, the installer offers a free trial, but you will be able to unlock the appropriate subscription level when you sign into Visual Studio with your Microsoft account.

Installing Visual Studio 2022

When you run the VisualStudioSetup.exe file, the Visual Studio Installer will be installed. This is the installation program for Visual Studio, and it was first introduced with Visual Studio 2017. The Visual Studio Installer in Visual Studio 2022 uses the same design principles as its predecessor and allows for installing sets of components, each targeting a specific development scenario. Each set of components is referred to as a workload. Workloads make installation and maintenance easier and allow developers to install what they actually need without unnecessary components, software development kits (SDKs), and tools. This way, you can save a lot of space on disk. You could even decide to install only the Visual Studio core editor without any additional workloads in order to get the basic coding environment. At startup, you will be able to select one or more workloads, as shown in Figure 1.



Modifying — Visual Studio Enterprise 2022 — 17.1.6

Web & Cloud (4)

ASP.NET and web development ✓
Build web applications using ASP.NET Core, ASP.NET, HTML/JavaScript, and Containers including Docker supp...

Azure development ✓
Azure SDKs, tools, and projects for developing cloud apps and creating resources using .NET and .NET Framework....

Python development ☐
Editing, debugging, interactive development and source control for Python.

Node.js development ☐
Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

Desktop & Mobile (5)

Mobile development with .NET ✓
Build cross-platform applications for iOS, Android or Windows using Xamarin. This includes a preview of the ....

.NET desktop development ✓
Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET and .NET Frame...

Desktop development with C++ ☐
Build modern C++ apps for Windows using tools of your choice, including MSVC, Clang, CMake, or MSBuild.

Universal Windows Platform development ✓
Create applications for the Universal Windows Platform with C#, VB, or optionally C++.

Installation details
- Visual Studio core editor
- ASP.NET and web development
- Azure development
- Mobile development with .NET
- .NET desktop development
- Universal Windows Platform develop...
- Data storage and processing
- Visual Studio extension development
- Individual components
  - ✓ OpenJDK (Microsoft distribution)
  - ✓ Android SDK setup (API level 31)
  - ✓ Xamarin Remoted Simulator
  - ✓ NuGet package manager
  - ✓ C# and Visual Basic Roslyn compilers
  - ✓ MSBuild
  - ✓ .NET 6.0 Runtime
  - ✓ .NET SDK
  - ✓ .NET Framework 4.7.2 targeting pack

Location
C:\Program Files\Microsoft Visual Studio\2022\Enterprise

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Total space required 0 B

Figure 1: Selecting Workloads in the Visual Studio Installer

Table 1 describes available workloads in the Visual Studio Installer.

Table 1: Workloads in the Visual Studio Installer

Workload	**Description**
ASP.NET and web development	Select this workload to develop web applications using ASP.NET and standards-based technologies like HTML, JavaScript, CSS, and JSON. It also enables you to quickly deploy your app to a web server or to Azure.
Azure development	This workload installs the latest Azure SDK for .NET and tools for Visual Studio 2019. This allows you to view resources in Cloud Explorer, create resources using Azure Resource Manager tools, and build applications and services ready to be hosted in Azure.
Python development	Select this workload to enable full Python support within Visual Studio, including editing, debugging, interactive windows, profiling, package management, and source control. It works with your existing Python install to help you develop cross-platform scripts, web apps, native extension modules, and IoT applications.
Node.js development	This workload adds everything you need to build apps for Node.js, including IntelliSense, local and remote debugging, profiling, npm integration, an interactive window, test runners, and Azure integration.
Mobile development with .NET	This workload installs Xamarin, the technology that allows you to create native iOS, Android, and Universal Windows Platform apps using a shared C# code base.
Universal Windows Platform Development	Select this workload if you want to write universal applications for Windows 10 and 11, including PC, tablet, smartphone, the HoloLens, Xbox, and IoT devices.
.NET desktop development	Select this workload if you want to build classic Windows desktop applications with WPF, Windows Forms, and console apps using .NET Framework.
Desktop development with C++	Select this workload if you wish to create, build, and debug native, classic desktop applications that run on versions ranging from Windows XP to the latest Windows 11 release using the C++ language and environment.
Mobile development with C++	Select this workload if you want to create cross-platform mobile apps using C++.
Game development with Unity	Select this workload if you want to develop cross-platform 2D and 3D games using the Unity framework and integrated tools for Visual Studio 2022.
Game development with C++	Select this workload if you want to create games using C++.

Workload	Description
Data storage and processing	This workload provides tools for accessing on-premises SQL Server databases, SQL databases on Azure, and Azure Data Lakes resources. It also provides support for U-SQL, Hive, and big data on Azure.
Data science and analytical applications	This workload installs languages such as Python, R, and F#, which are tied to building applications for data analysis.
Office/SharePoint development	This workload provides Office developer tools, which allow for creating Office and SharePoint add-ins and solutions.
Visual Studio extension development	This workload installs the Visual Studio SDK and allows you to write extensions such as new commands, tool windows, and templates.
Linux and embedded development with C++	This workload enables you to author C++ code for Linux servers, desktops, and devices from within Visual Studio 2022.

Note: When .NET MAUI ships in VS 2022 17.3, there will be a new workload called .NET Multi-Platform App UI Development.

For the instructional purposes of this ebook, the following workloads are required:

- ASP.NET and web development.
- .NET desktop development.
- Azure development.
- Mobile development with .NET.

You are not required to do the same—feel free to select only those you need. You can later install additional workloads as required.

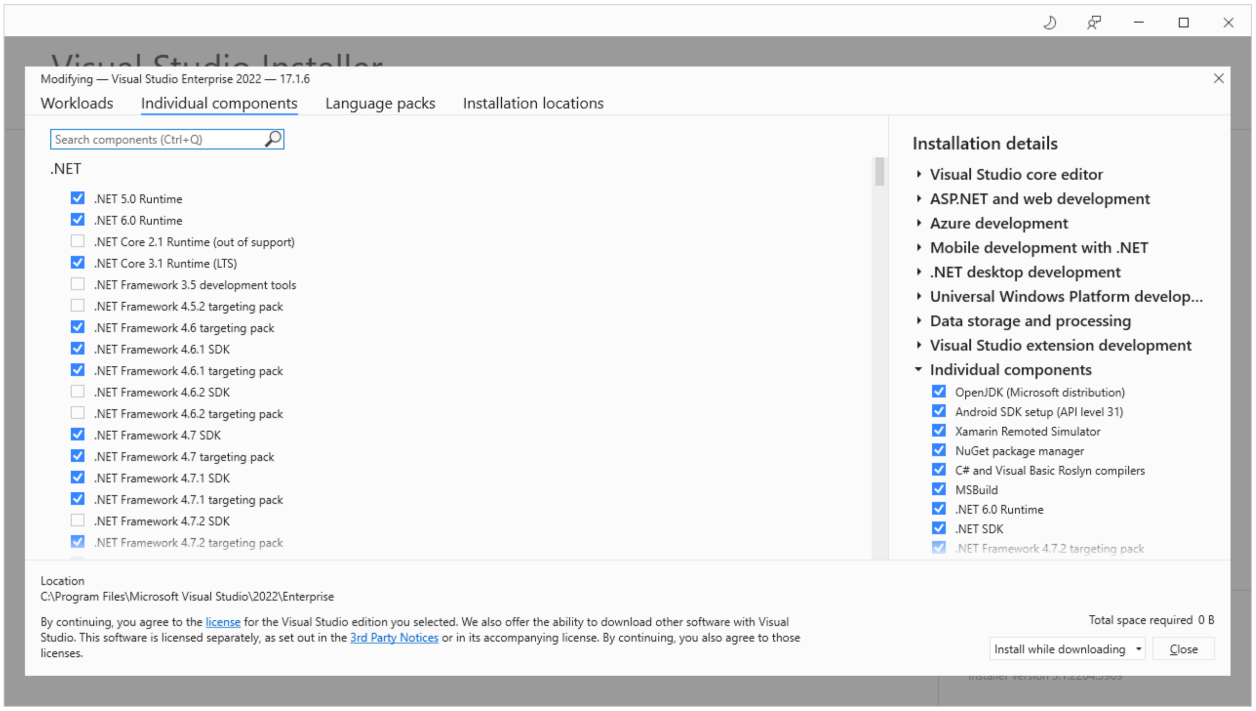
Note: There is one less workload than in Visual Studio 2019, which is .NET cross-platform development. This is because other workloads in VS 2022 already support this type of development with .NET Core.

Customizing the installation

As you would expect from an advanced setup tool, you can customize the Visual Studio installation in several ways. You can select additional individual components, and you can also select language packs and target folders on disk. This section will describe these customizations.

Installing individual components

In many cases, selecting workloads is not enough to get the tools you actually need. You will need to install additional individual components. As an example, the GitHub extension for Visual Studio 2022 is not installed by default, which means you might want to select this component if you plan to work with Git repositories on that popular service. You can click the **Individual components** tab to see the full list of available individual components. Figure 2 shows an example.



Visual Studio Installer

Modifying — Visual Studio Enterprise 2022 — 17.1.6

Workloads | Individual components | Language packs | Installation locations

Search components (Ctrl+Q)

.NET

- ✓ .NET 5.0 Runtime
- ✓ .NET 6.0 Runtime
- ☐ .NET Core 2.1 Runtime (out of support)
- ✓ .NET Core 3.1 Runtime (LTS)
- ☐ .NET Framework 3.5 development tools
- ☐ .NET Framework 4.5.2 targeting pack
- ✓ .NET Framework 4.6 targeting pack
- ✓ .NET Framework 4.6.1 SDK
- ✓ .NET Framework 4.6.1 targeting pack
- ☐ .NET Framework 4.6.2 SDK
- ☐ .NET Framework 4.6.2 targeting pack
- ✓ .NET Framework 4.7 SDK
- ✓ .NET Framework 4.7 targeting pack
- ✓ .NET Framework 4.7.1 SDK
- ✓ .NET Framework 4.7.1 targeting pack
- ☐ .NET Framework 4.7.2 SDK
- ✓ .NET Framework 4.7.2 targeting pack

Installation details

- Visual Studio core editor
- ASP.NET and web development
- Azure development
- Mobile development with .NET
- .NET desktop development
- Universal Windows Platform develop...
- Data storage and processing
- Visual Studio extension development
- Individual components
  - ✓ OpenJDK (Microsoft distribution)
  - ✓ Android SDK setup (API level 31)
  - ✓ Xamarin Remoted Simulator
  - ✓ NuGet package manager
  - ✓ C# and Visual Basic Roslyn compilers
  - ✓ MSBuild
  - ✓ .NET 6.0 Runtime
  - ✓ .NET SDK
  - ✓ .NET Framework 4.7.2 targeting pack

Location
C:\Program Files\Microsoft Visual Studio\2022\Enterprise

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Total space required 0 B

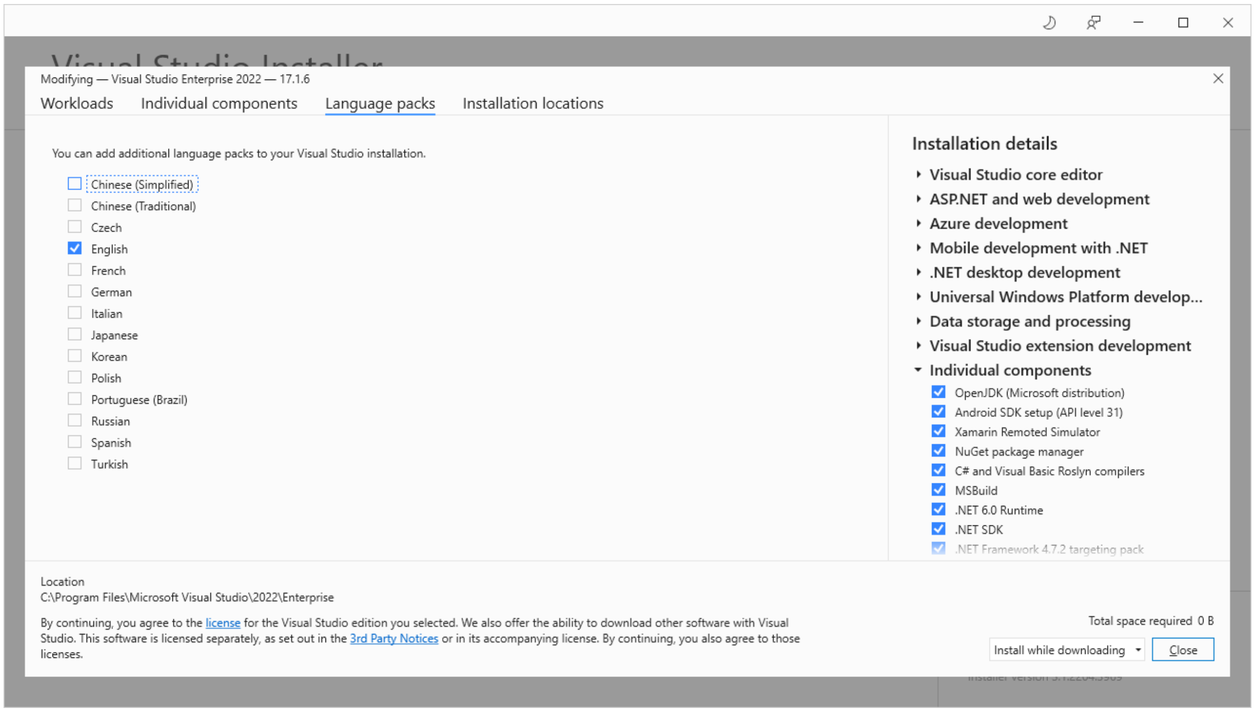
Install while downloading | Close

Figure 2: Installing Individual Components

Individual components are grouped by development areas. On the right side of the installer, you can see a summary of all the individual components currently included in the installation.

Installing language packs

You can use Visual Studio 2022 in the language of your choice by installing language packs. Simply click the **Language packs** tab and select one or more languages (see Figure 3).



Modifying — Visual Studio Enterprise 2022 — 17.1.6

You can add additional language packs to your Visual Studio installation.

- Chinese (Simplified)
- Chinese (Traditional)
- Czech
- [✓ English]
- French
- German
- Italian
- Japanese
- Korean
- Polish
- Portuguese (Brazil)
- Russian
- Spanish
- Turkish

Installation details

- Visual Studio core editor
- ASP.NET and web development
- Azure development
- Mobile development with .NET
- .NET desktop development
- Universal Windows Platform develop...
- Data storage and processing
- Visual Studio extension development
- Individual components
  - [✓ OpenJDK (Microsoft distribution)]
  - [✓ Android SDK setup (API level 31)]
  - [✓ Xamarin Remoted Simulator]
  - [✓ NuGet package manager]
  - [✓ C# and Visual Basic Roslyn compilers]
  - [✓ MSBuild]
  - [✓ .NET 6.0 Runtime]
  - [✓ .NET SDK]
  - [✓ .NET Framework 4.7.2 targeting pack]

Location
C:\Program Files\Microsoft Visual Studio\2022\Enterprise

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Total space required 0 B

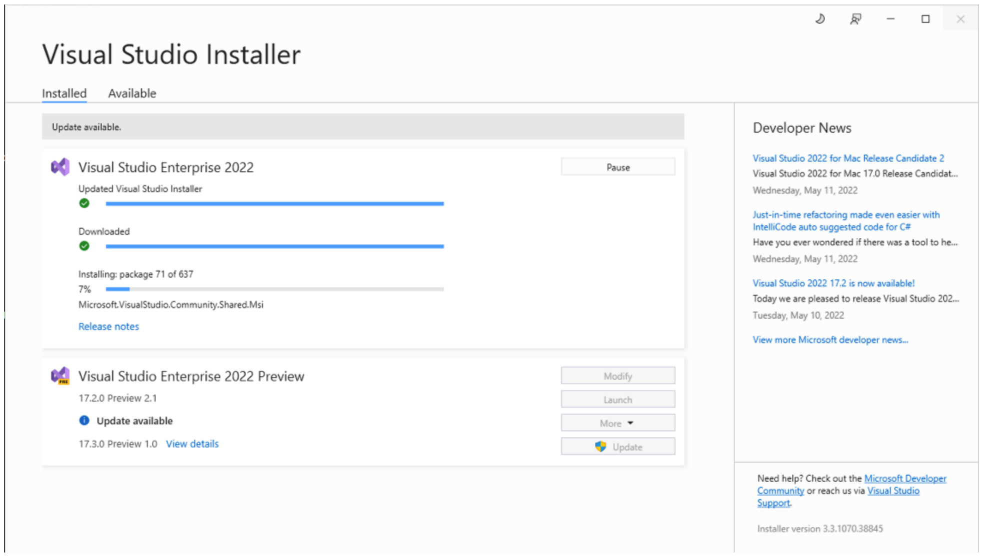
Figure 3: Installing Language Packs

Once Visual Studio 2022 is installed, you will be able to select a different language by selecting **Tools** > **Options** > **Environment** > **International Settings**.

Tip: You can install language packs at any time by launching the Visual Studio Installer even after the installation. It is also worth mentioning that changing the language requires Visual Studio to be restarted.

Supporting multiple versions and editions

Like Visual Studio 2019, you can install multiple editions side by side in Visual Studio 2022. For example, you might need to install both Professional and Enterprise editions on the same machine. Not limited to this, the Visual Studio Installer adds support for managing the installation of both Visual Studio 2019 and Visual Studio 2022, including previews. Figure 4 shows how the Installer makes it possible to handle multiple versions.


Visual Studio Installer

Installed Available

Update available.

Visual Studio Enterprise 2022
Updated Visual Studio Installer
Downloaded
Installing: package 71 of 637
7%
Microsoft.VisualStudio.Community.Shared.Msi
Release notes

Visual Studio Enterprise 2022 Preview
17.2.0 Preview 2.1
Update available
17.3.0 Preview 1.0 View details

Developer News

Visual Studio 2022 for Mac Release Candidate 2
Visual Studio 2022 for Mac 17.0 Release Candidat...
Wednesday, May 11, 2022

Just-in-time refactoring made even easier with IntelliCode auto suggested code for C#
Have you ever wondered if there was a tool to he...
Wednesday, May 11, 2022

Visual Studio 2022 17.2 is now available!
Today we are pleased to release Visual Studio 202...
Tuesday, May 10, 2022

View more Microsoft developer news...

Need help? Check out the Microsoft Developer Community or reach us via Visual Studio Support.

Installer version 3.3.1070.38845

Figure 4: Managing Multiple Versions of Visual Studio

In Figure 4, you can see that Visual Studio 2022 and Visual Studio 2022 Preview are installed side by side on the machine. However, you can only modify one installation at a time.

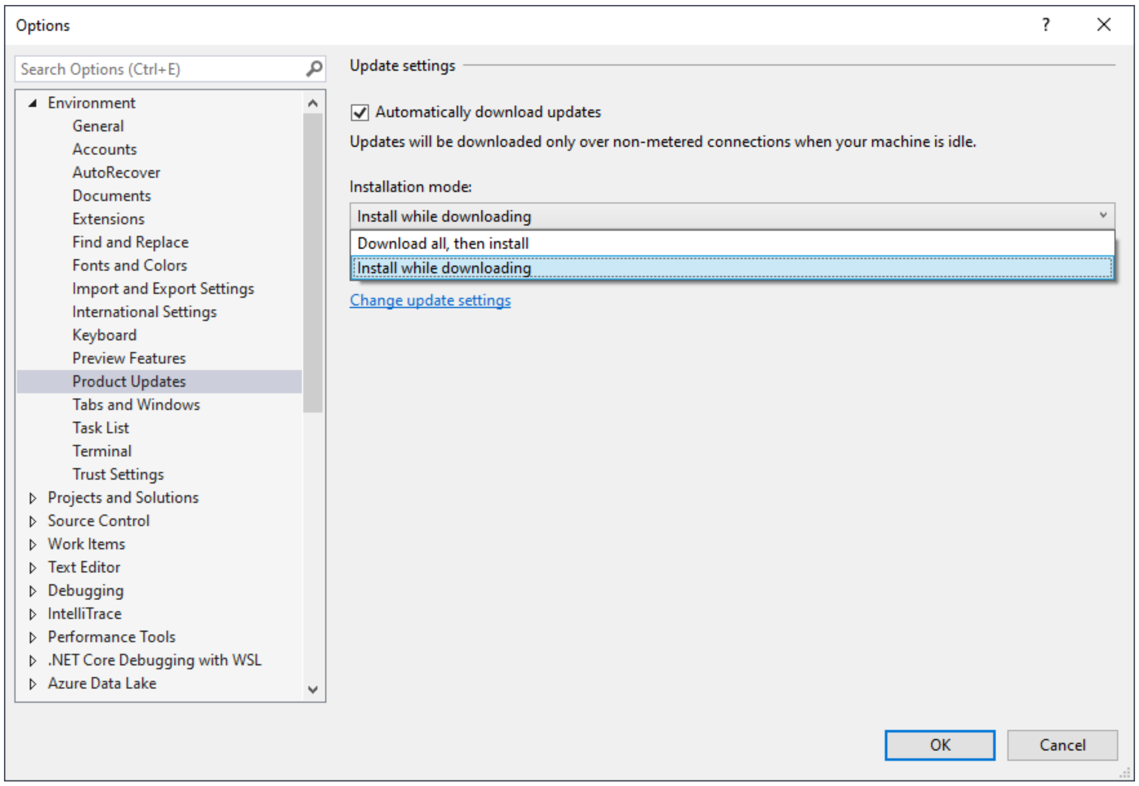
Trying out preview features

Together with the stable release, Microsoft offers preview builds of Visual Studio 2022 that you can use to have an early look at what will be included with the next updates. In fact, it is possible to download and install the so-called Visual Studio 2022 Preview from a dedicated download page. You can install Visual Studio 2022 Preview side by side with the stable release, which enables you to test not only upcoming features in the IDE, but also to preview builds of the various SDKs (such as .NET MAUI, as of this writing). Remember that this is pre-release software, so it is not supported, not to be used in production, and intended only for testing purposes.

Controlling Visual Studio updates

In Visual Studio 2022, you have an option to control how product updates are downloaded and installed. If you select **Tools** > **Options** > **General** > **Product Updates** (see Figure 5), you will be able to:

- Download updates automatically over non-metered connections and when the machine is idle.
- Decide whether to download updates and then install them or install while downloading.



Update settings

✓ Automatically download updates

Updates will be downloaded only over non-metered connections when your machine is idle.

Installation mode:

Install while downloading

Download all, then install

Install while downloading

Change update settings

Figure 5: Changing Update Options

These settings are sent to the Visual Studio Installer engine, which will manage product updates based on your decision. Figure 5 also shows the default settings: automatic download and installation while downloading.

Conclusions

The installation experience in Visual Studio 2022 is based on the Visual Studio Installer tool. In the new version, not only can you still select only the workloads you actually need for easier setup and maintenance, but you can now select language packs and change the installation folders. You can also manage multiple editions and versions in one place.

Chapter 1 Design Improvements

Visual Studio 2022 introduces several design improvements, including new iconography, new fonts, updated themes, and the option to have colored tabs. About icons and fonts, these are based on the popular open-source Microsoft Fluent design system. This chapter describes the design improvements in VS 2022, and it explains how to replace the default settings with custom ones.

Updated iconography

Icons in Visual Studio 2022 have been updated based on the following criteria: consistency, legibility, and familiarity. Consistency means that icons should represent predictable items. Legibility means using colors, fills, and contrast to improve accessibility for all users. Familiarity means that icon symbols with a long tradition should be kept, preserving colors when they help find items. Figure 6, taken from the blog post "We've upgraded the UI in Visual Studio 2022," compares some icons between Visual Studio 2019 (on the left) and Visual Studio 2022 (on the right), making the iconography changes clear.

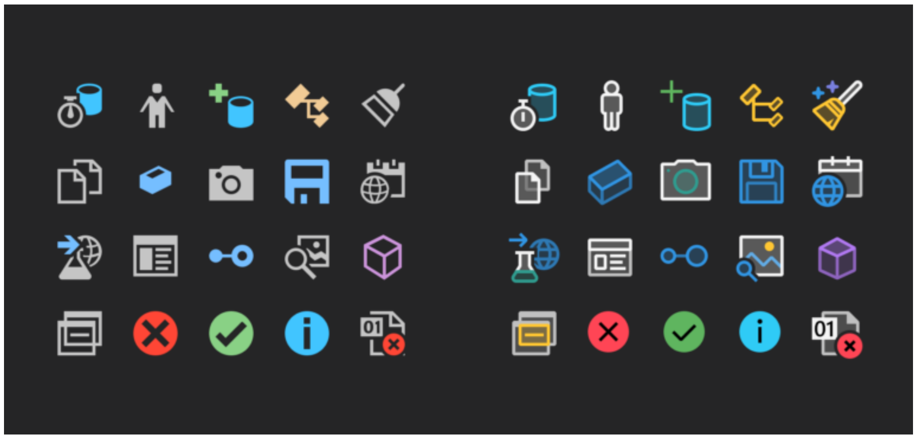


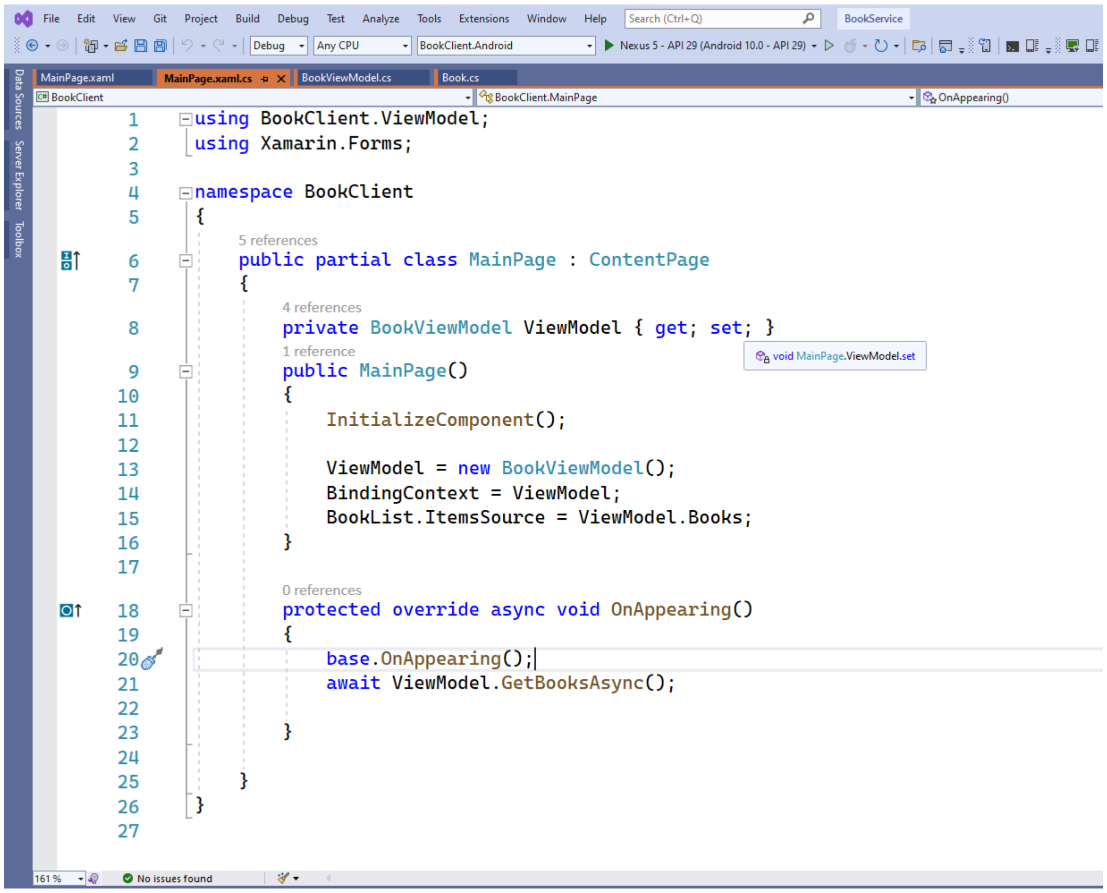
Figure 6: Iconography Comparison (Source: Microsoft)

The new icons, which also work much better with dark themes, still make commands and shortcuts easy to recognize but improve accessibility and contribute, together with the other improvements, to make your eyes less tired when spending many hours in front of a screen.

Microsoft has kept in mind additional principles when updating the icons. More specifically, a change is acceptable if it does not alter the original meaning of an icon; legibility works only with appropriate combinations of contrast and icon symbols; and colors should always make it easy to understand what an icon refers to. Icons are not the only update in terms of legibility, accessibility, and rest for your eyes. There is also some news about fonts, as discussed in the next section.

New fonts

Visual Studio 2022 introduces a new font called **Cascadia**, with two flavors: Mono and Code. By default, Cascadia Mono is enabled for both the user interface and the code editor. Cascadia is a new monospaced font that improves readability and accessibility especially for text editors and command-line tools. Figure 7 shows how the code editor looks with the Cascadia Mono font enabled.



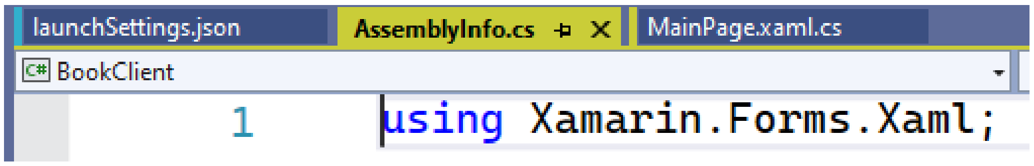
```
using BookClient.ViewModel;
using Xamarin.Forms;

namespace BookClient
{
    5 references
    public partial class MainPage : ContentPage
    {
        4 references
        private BookViewModel ViewModel { get; set; }
        1 reference
        public MainPage()
        {
            InitializeComponent();

            ViewModel = new BookViewModel();
            BindingContext = ViewModel;
            BookList.ItemsSource = ViewModel.Books;
        }

        0 references
        protected override async void OnAppearing()
        {
            base.OnAppearing();
            await ViewModel.GetBooksAsync();

        }

    }
}
```

Figure 7: The Cascadia Mono Font Improves Accessibility and Readability

You can always select a different font the usual way, by selecting **Tools** > **Options**, and then in the **Options** dialog locate the **Fonts and Colors** option under the **Environment** tab. The biggest difference between Cascadia Mono and Cascadia Code is that the latter better supports the addition of ligatures (glyphs used by many developers to improve readability).

Enabling color tabs

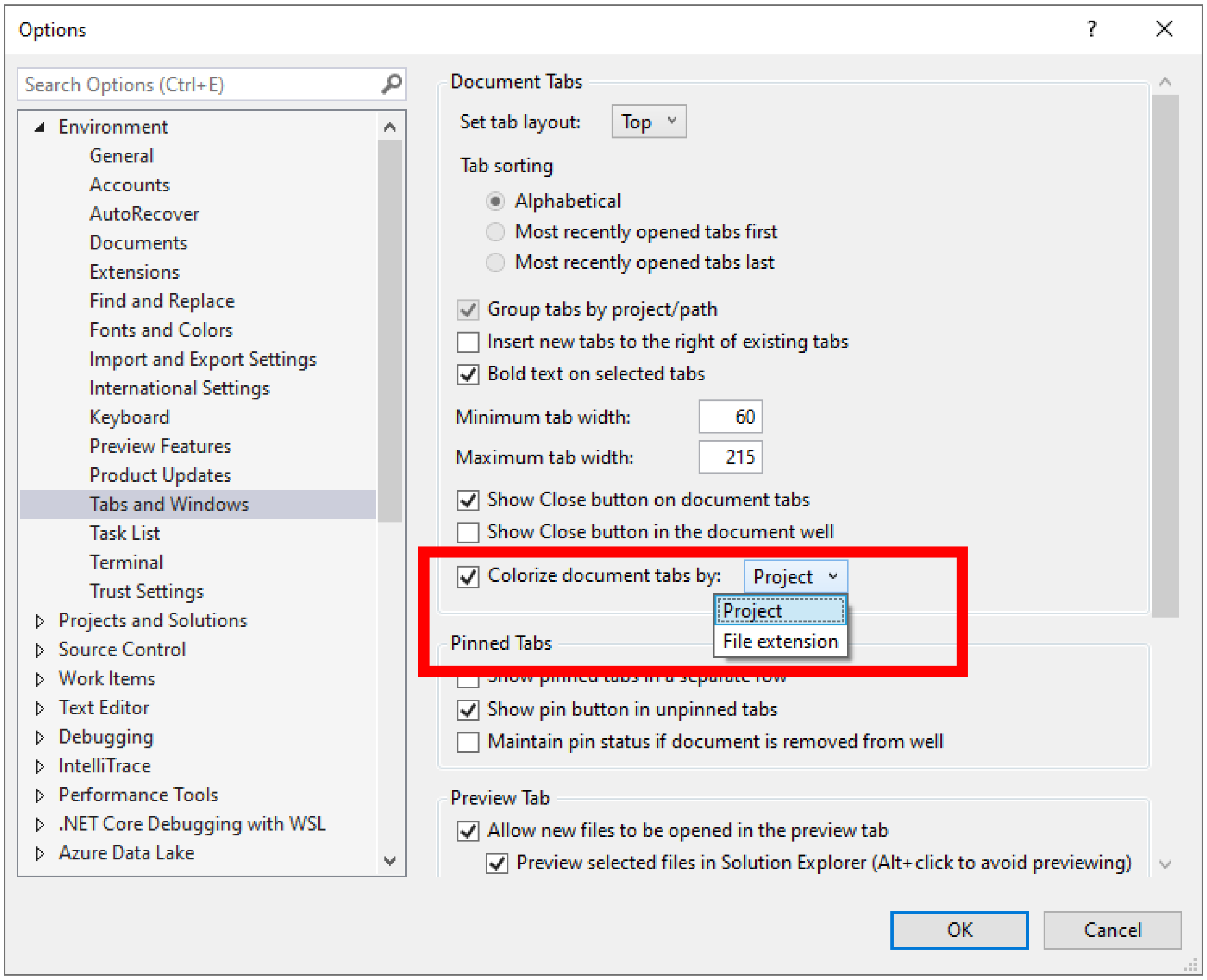
Another new feature in Visual Studio 2022 is color tabs. It enables coloring tabs that represent open files. Figure 8 shows an example where the tab of the currently open file is colored in volt and the inactive tabs are colored blue.



```
using Xamarin.Forms.Xaml;
```

Figure 8: Colored Tabs for Open and Closed Tabs

Colored tabs are not enabled by default, so you need to manually enable the feature in the **Options** dialog under the **Environment** > **Tabs and Windows** group, selecting the **Colorize documents tab by** option (see Figure 9). You can enable tab colorization by project or by file extension. If your solution contains several projects, it might be useful to have colored tabs by project so you can quickly determine which project a file belongs to.



Options

Document Tabs

Set tab layout: Top

Tab sorting

- [✓ Alphabetical]
- Most recently opened tabs first
- Most recently opened tabs last

- [✓] Group tabs by project/path
- [ ] Insert new tabs to the right of existing tabs
- [✓] Bold text on selected tabs

Minimum tab width: 60

Maximum tab width: 215

- [✓] Show Close button on document tabs
- [ ] Show Close button in the document well
- [✓] Colorize document tabs by: Project

Pinned Tabs

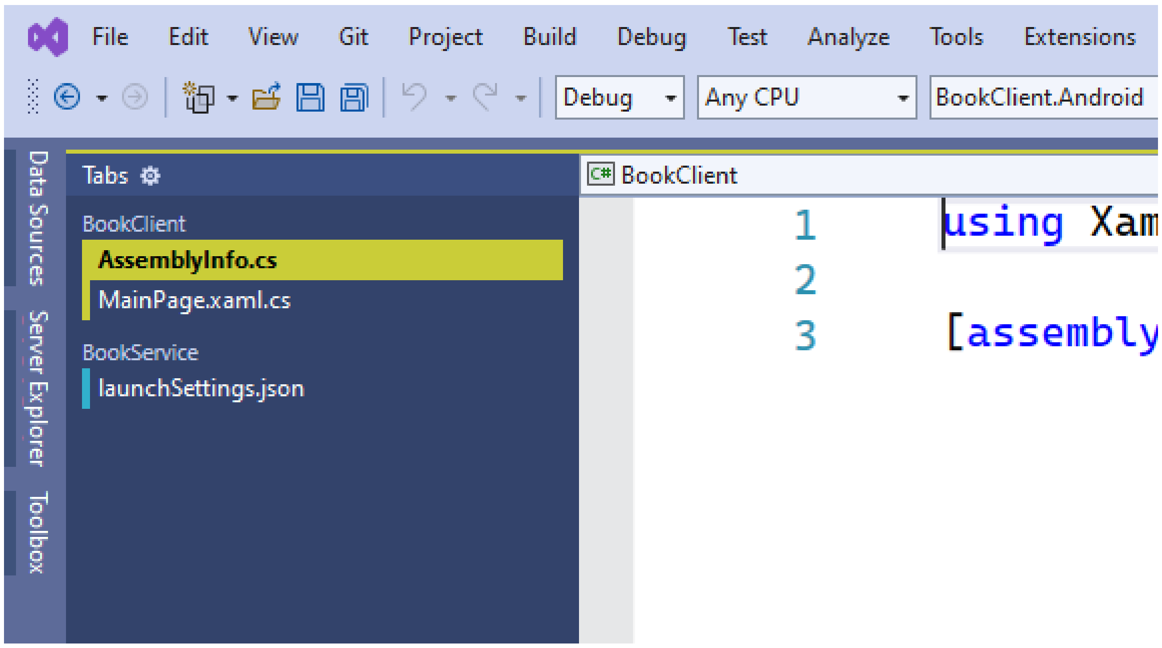
- [ ] Show pinned tabs in a separate row
- [✓] Show pin button in unpinned tabs
- [ ] Maintain pin status if document is removed from well

Preview Tab

- [✓] Allow new files to be opened in the preview tab
- [✓] Preview selected files in Solution Explorer (Alt+click to avoid previewing)

Figure 9: Enabling Tab Colorization

Every time you select a tab, it will be fully colored. You can also control the tab layout. If you right-click a tab and then select Set Tab Layout, you will be able to select the tab position (left, top, or right). Top is the default, like in Figure 8. Figure 10 shows how tabs appear when placed on the left side of the IDE.



```
using Xam

[assembly
```

Figure 10: Selecting a Tab Layout

Notice how tabs are grouped by project for easier reference. Visual Studio automatically chooses tab colors, but this is something you can customize by right-clicking a tab and then selecting **Set Tab Color**. You will be able to pick a color that will be applied to all the tabs of the same group.

Tip: Visual Studio detects if other instances of the IDE are running. In this case, every instance will show tabs with colors that are different from one instance to another to avoid confusion.

Further tab layout management

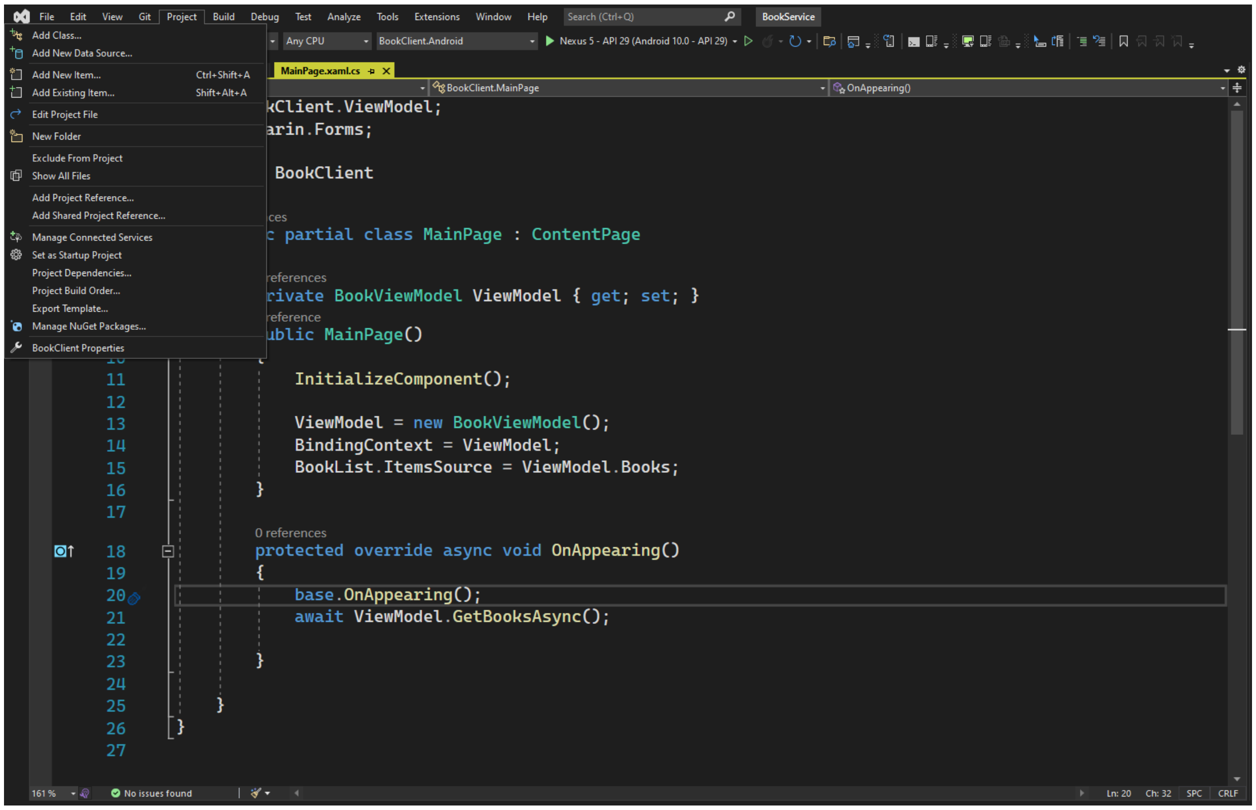
If you look at Figure 9, you can see options that allow for further tab customization:

- **Bold text on selected tabs** allows you to show the text of a tab in bold (see Figures 8 and 10 for examples).
- **Minimum tab width** and **Maximum tab width** allow you to respectively specify minimum and maximum values for the tab width. This is a new feature that can be helpful with long file names.

Customizing tabs is part of a strategy in Visual Studio 2022 that makes it easier to personalize the environment according to your needs, and this also involves themes, discussed in the next section.

Theme personalization

Several improvements have also been made to the graphical themes in Visual Studio. First, you can now access available themes directly from the Tools menu via the **Themes** submenu, which provides shortcuts to enable a theme plus a shortcut to the Visual Studio Marketplace to download additional themes specifically designed for VS 2022. The second improvement is related to the dark theme, which is one of the most popular themes. This theme has been upgraded to better support with the Microsoft Fluent design language and the new iconography discussed previously; therefore, it also better supports accessibility, and prevents your eyes from fatiguing. Figure 11 shows how the dark theme appears.



```
kClient.ViewModel;
arin.Forms;

 BookClient

ces
c partial class MainPage : ContentPage

references
rivate BookViewModel ViewModel { get; set; }
reference
ublic MainPage()
{
    InitializeComponent();

    ViewModel = new BookViewModel();
    BindingContext = ViewModel;
    BookList.ItemsSource = ViewModel.Books;
}

0 references
protected override async void OnAppearing()
{
    base.OnAppearing();
    await ViewModel.GetBooksAsync();

}

    }
}
```

Figure 11: Updated Dark Theme

Tip: If you still want to use the dark theme as it was in Visual Studio 2019, you can download it as an extension.

Another interesting option is the ability to use the current Windows theme. This is accomplished by selecting **Tools** > **Themes** > **Use system settings**. When you select this option, Visual Studio 2022 will synchronize its theme with the Windows theme, including specific settings such as ambient light. Finally, Microsoft also created an open-source tool called Visual Studio Theme Converter that allows for quickly converting themes from Visual Studio Code (you will need to compile the tool with Visual Studio on your own).

Chapter summary

Without a doubt, an environment where the user feels at home improves productivity. With this in mind, Microsoft has enhanced Visual Studio 2022 by introducing new and updated features that allow for better tab and theme personalization, plus new fonts and iconography that improve consistency, legibility, and accessibility for all developers. Productivity is key, and a comfortable environment is not only made of the look and feel of an IDE, but also the coding tools. This is the topic of the next chapter.

Chapter 2 .NET Productivity

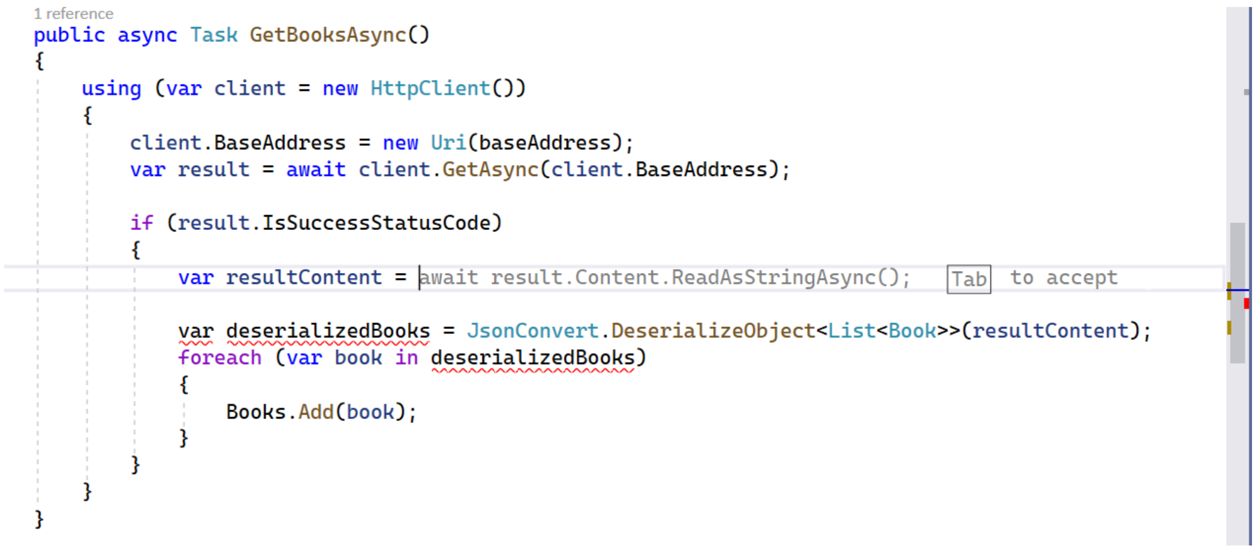
As you would expect from a new major release of Visual Studio, several productivity improvements are also available to the code editor since this is the place where you spend most of your developer life. This already powerful editor is enhanced with a new code completion engine, new code refactorings, and new visual tools that help you understand the structure of complex code while keeping your focus on what you are writing. This chapter describes all the new productivity features available in the code editor, with suggestions on how to get the most out of them.

IntelliCode Completions

IntelliCode is a technology from Microsoft that extends the popular IntelliSense code completion engine with a richer experience based on artificial intelligence and machine learning. In fact, IntelliCode really learns from your code patterns and from the way you write code to provide better suggestions based on the context and based on the type of code you are writing. In Visual Studio 2022, the code editor receives an important improvement called **IntelliCode Completions**, which consists of full, in-line suggestions as you type that you can simply accept or reject.

Note: At the time of this writing, the full IntelliCode Completions experience is available only for C#, though languages such as C++, JavaScript, and TypeScript also leverage this feature, especially with IntelliSense. Updates to IntelliSense based on IntelliCode are discussed shortly.

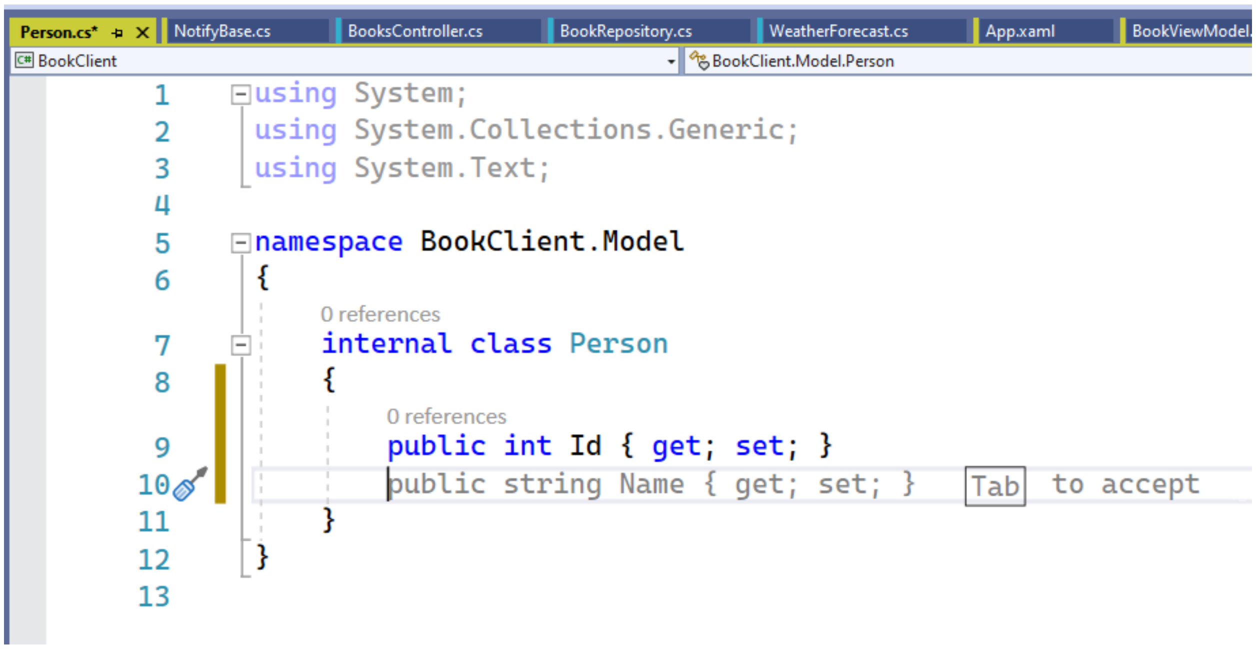
For a better understanding, consider Figure 12 where you can see some C# code that makes an API call to retrieve a list of hypothetical **`Book`** objects. If you look at the line where the **`resultContent`** variable is declared, you can see how Visual Studio 2022 suggests completing the whole line of code based on the current context, which is completely appropriate in this case because the code needs to read response content in the form of a string, and that's exactly the way you would need to complete the line.



```
1 reference
public async Task GetBooksAsync()
{
    using (var client = new HttpClient())
    {
        client.BaseAddress = new Uri(baseAddress);
        var result = await client.GetAsync(client.BaseAddress);

        if (result.IsSuccessStatusCode)
        {
            var resultContent = await result.Content.ReadAsStringAsync();   Tab  to accept

            var deserializedBooks = JsonConvert.DeserializeObject<List<Book>>(resultContent);
            foreach (var book in deserializedBooks)
            {
                Books.Add(book);
            }
        }
    }
}
```

Figure 12: IntelliCode Completions

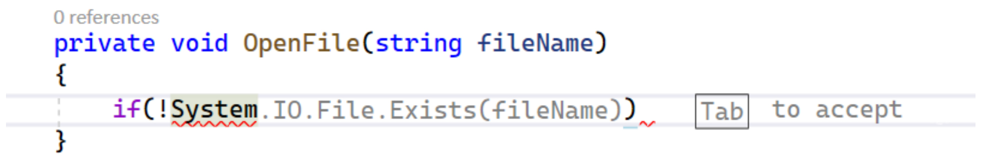
You can simply press **Tab** to insert the provided suggestion, press **Esc** to ignore the suggestion, or type your code manually. Figure 13 shows another example based on the definition of a **`Person`** class. Here, IntelliCode understands the context and suggests adding a property called **`Name`**, of type **`string`**, which would perfectly fit within a **`Person`** class.



```
using System;
using System.Collections.Generic;
using System.Text;

namespace BookClient.Model
{
    0 references
    internal class Person
    {
        0 references
        public int Id { get; set; }
        public string Name { get; set; }   Tab  to accept
    }
}
```

Figure 13: IntelliCode Suggesting Member Definition and Names

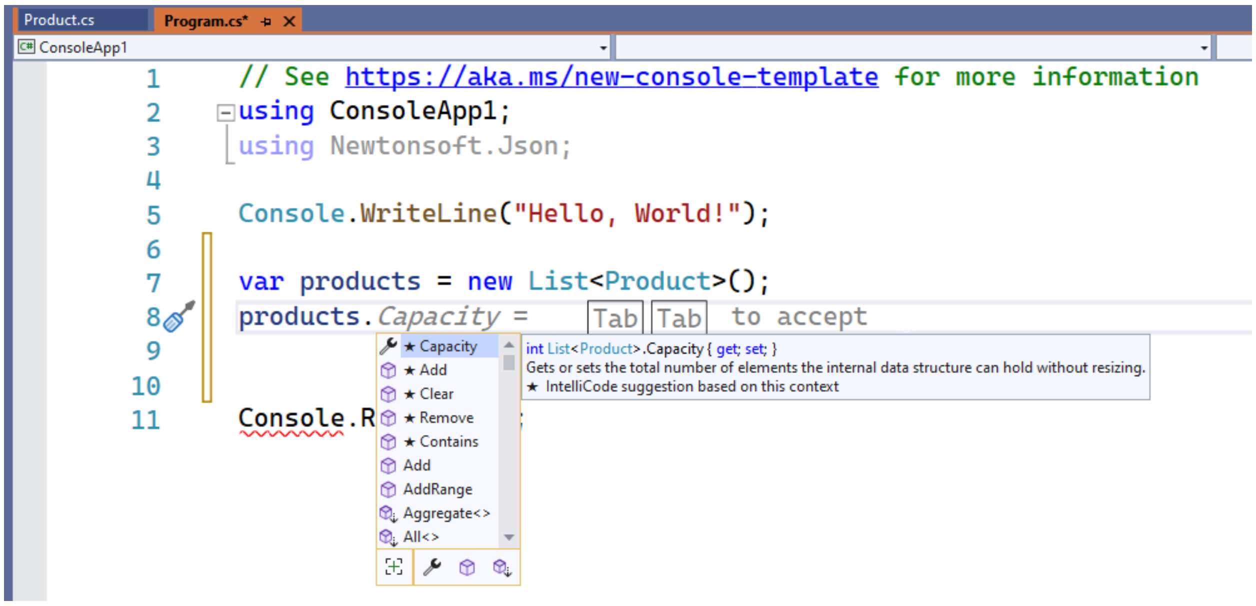
As a last example, consider Figure 14. There is the definition of a method that needs to open a file, and IntelliCode understands the intention of first checking if the file exists so it provides a suggestion on how to properly complete the line of code.



```
0 references
private void OpenFile(string fileName)
{
    if(!System.IO.File.Exists(fileName))   Tab  to accept
}
```

Figure 14: IntelliCode Context-Based Suggestion

As you can imagine, the number of suggestions that IntelliCode can provide is nearly infinite. Because it is built upon machine learning, it learns from your code more and more. For your curiosity, the IntelliCode engine has been trained with over half a million open-source repositories on GitHub (source) and, among the others, it takes into consideration the libraries you use, nearby code, and variable names. IntelliCode also improves the popular IntelliSense completion engine in several ways. First, IntelliSense now suggests, at the top of the list, completions that could satisfy a specific context. If you look at Figure 15, a hypothetical collection called **products**, of type **List<Product>**, is being instantiated and IntelliSense first suggests instance members that could fit in that context. A star icon on the left of the member represents an IntelliCode suggestion.



```
// See https://aka.ms/new-console-template for more information
using ConsoleApp1;
using Newtonsoft.Json;

Console.WriteLine("Hello, World!");

var products = new List<Product>();
products.Capacity =    Tab Tab  to accept


Console.R
```

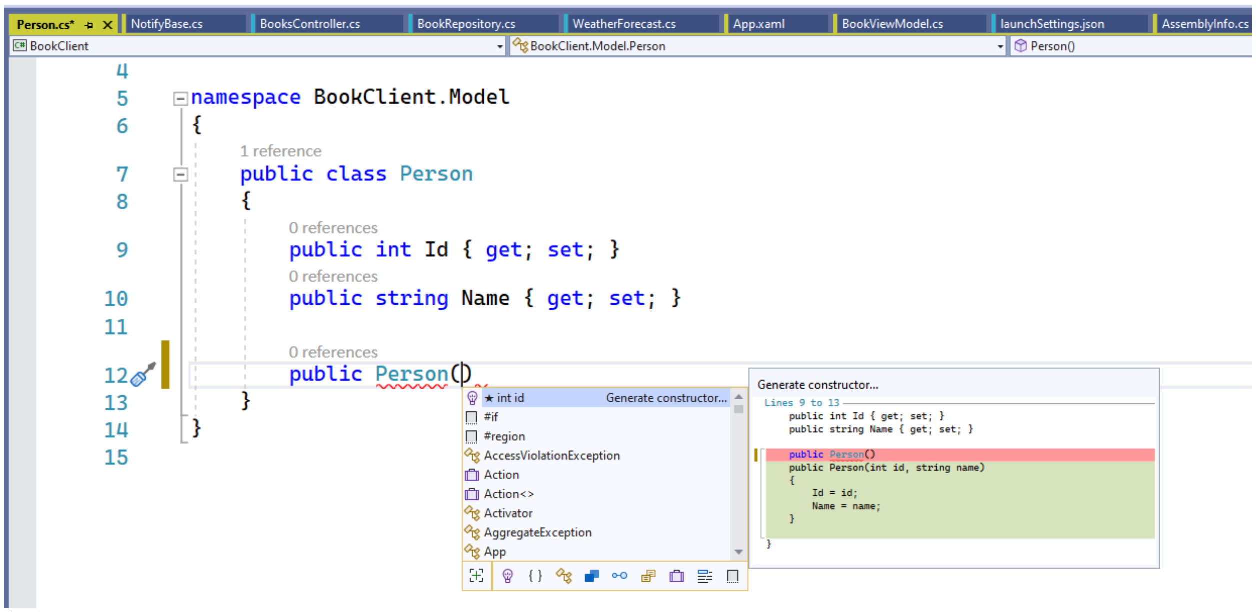
```
int List<Product>.Capacity { get; set; }
Gets or sets the total number of elements the internal data structure can hold without resizing.
★ IntelliCode suggestion based on this context
```

Figure 15: IntelliCode Empowering IntelliSense Suggestions

You can scroll to the member that you need to add and then press **Tab** twice to quickly insert the code. IntelliSense improvements based on IntelliCode also include suggestions as you type, which are discussed in the next section.

IntelliCode Completions as you type

In some situations, the code editor can provide IntelliCode suggestions that would normally be available as code refactorings, and that you would then manually show via the light bulb and screwdriver icons in the editor. If you look at Figure 16, you can see how a suggestion called **Generate constructor** appears directly in the IntelliSense list.



```
namespace BookClient.Model
{
    1 reference
    public class Person
    {
        0 references
        public int Id { get; set; }
        0 references
        public string Name { get; set; }

        0 references
        public Person()
    }
}
```

Generate constructor...

```
Lines 9 to 13
    public int Id { get; set; }
    public string Name { get; set; }

    public Person()
    public Person(int id, string name)
    {
        Id = id;
        Name = name;
    }
}
```

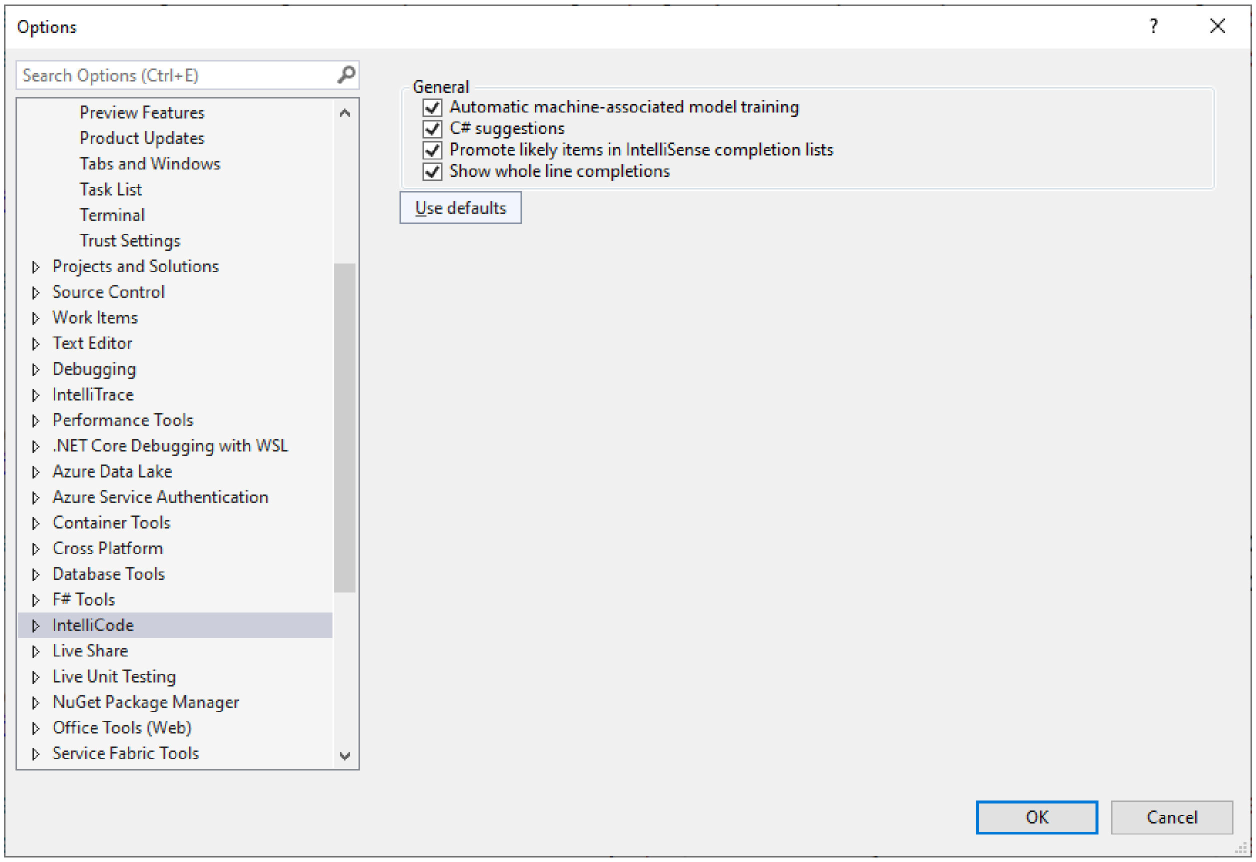
Figure 16: IntelliCode Completions as You Type

If you accept this suggestion, a parameterized constructor will be generated based on the number of properties defined in your class. The IntelliSense pop-up now also includes a new icon represented by a light bulb that you can click to filter by IntelliCode completions. In general, this feature is available when Visual Studio detects code for which a refactoring is available.

Note: At the time of this writing, this feature is available for a few scenarios only, such as constructor and property generation. It is reasonable to expect that it will be enhanced with the coming service releases.

Managing IntelliCode Completions

IntelliCode Completions are enabled by default. If you wish to disable this feature, you can open the **Options** window and locate the **IntelliCode** node (see Figure 17).


Options

Search Options (Ctrl+E)

- Preview Features
- Product Updates
- Tabs and Windows
- Task List
- Terminal
- Trust Settings
- Projects and Solutions
- Source Control
- Work Items
- Text Editor
- Debugging
- IntelliTrace
- Performance Tools
- .NET Core Debugging with WSL
- Azure Data Lake
- Azure Service Authentication
- Container Tools
- Cross Platform
- Database Tools
- F# Tools
- IntelliCode
- Live Share
- Live Unit Testing
- NuGet Package Manager
- Office Tools (Web)
- Service Fabric Tools

General
- [✓] Automatic machine-associated model training
- [✓] C# suggestions
- [✓] Promote likely items in IntelliSense completion lists
- [✓] Show whole line completions

Figure 17: Managing IntelliCode Completions Options

Table 2 summarizes the options and their descriptions.

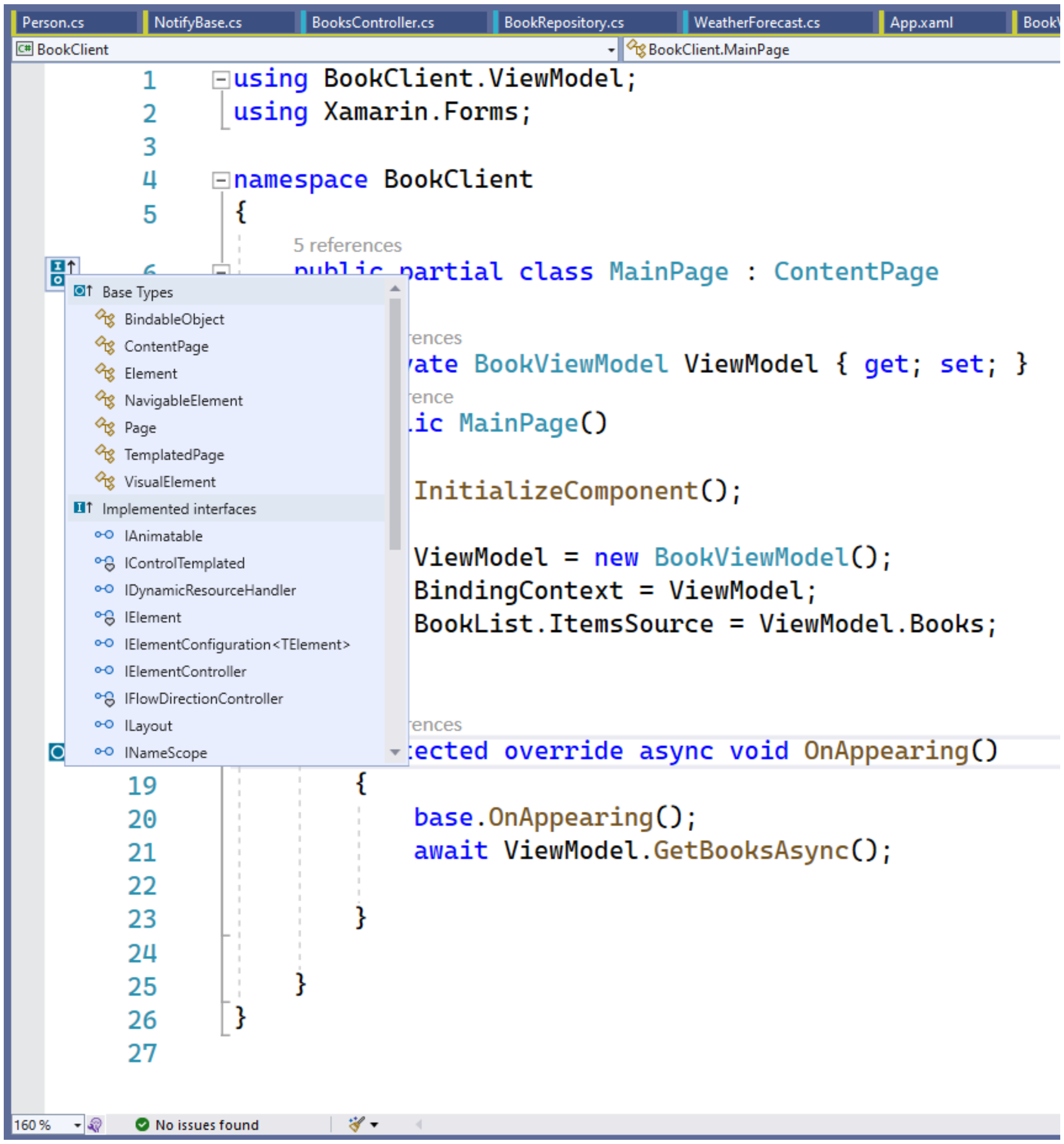
Table 2: IntelliCode Completions Options

Option	**Description**
Automatic machine-associated model training	By enabling this option, you give IntelliCode permission to learn from your coding styles and patterns so that suggestions will be improved and more personalized as the engine learns from you.
C# suggestions	Enables IntelliCode Completions as you type, discussed in the previous section.
Promote likely items in IntelliSense completion lists	When enabled, this option allows IntelliSense to display suggestions based on the context, powered by IntelliCode (see Figure 15 for an example).
Show whole line completions	This option controls the full line suggestion feature that was the first topic of this chapter.

Click **OK** when ready. This is probably one of the most interesting and powerful features of Visual Studio 2022, especially for .NET, so using it and getting familiar with it will bring your coding experience to the next level.

Inheritance margin

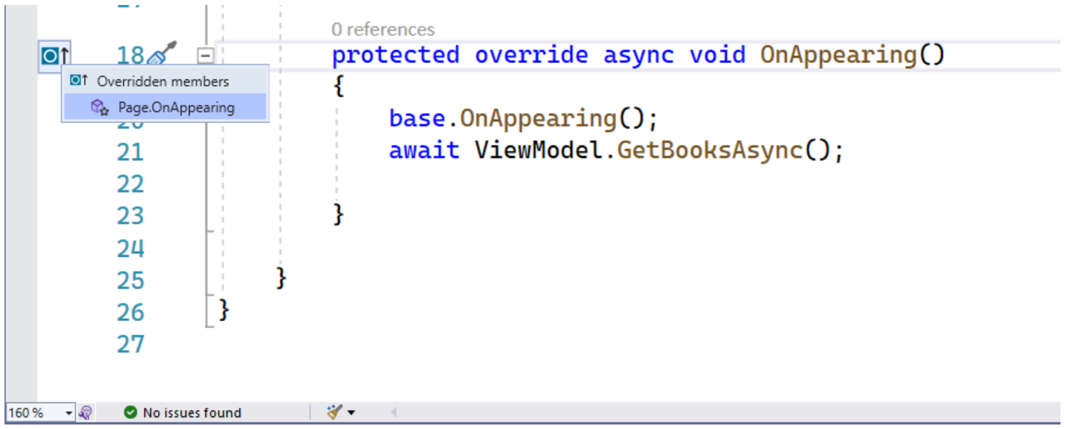
Inheritance margin was introduced with later updates of Visual Studio 2019 and improved over time, so it is a good idea to show how it works in Visual Studio 2022. This feature shows the base members and interfaces that a type is implementing through an icon displayed on the code editor margin. Figure 18 shows an example.



```
using BookClient.ViewModel;
using Xamarin.Forms;

namespace BookClient
{
        5 references
        public partial class MainPage : ContentPage

                ences
                ate BookViewModel ViewModel { get; set; }
                ence
                ic MainPage()

                InitializeComponent();

                ViewModel = new BookViewModel();
                BindingContext = ViewModel;
                BookList.ItemsSource = ViewModel.Books;


                ences
                ected override async void OnAppearing()
            {
                base.OnAppearing();
                await ViewModel.GetBooksAsync();

            }

        }
}
```

Figure 18: Displaying Base Types and Implemented Interfaces with Inheritance Margin

This feature also applies to individual members so that it is possible to know what base members have been overridden. Figure 19 shows an example based on a method called **`OnAppearing`** that overrides the same-named method from its base class.



```
0 references
protected override async void OnAppearing()
{
    base.OnAppearing();
    await ViewModel.GetBooksAsync();

}
        }
    }
```

Figure 19: Displaying Overridden Base Members

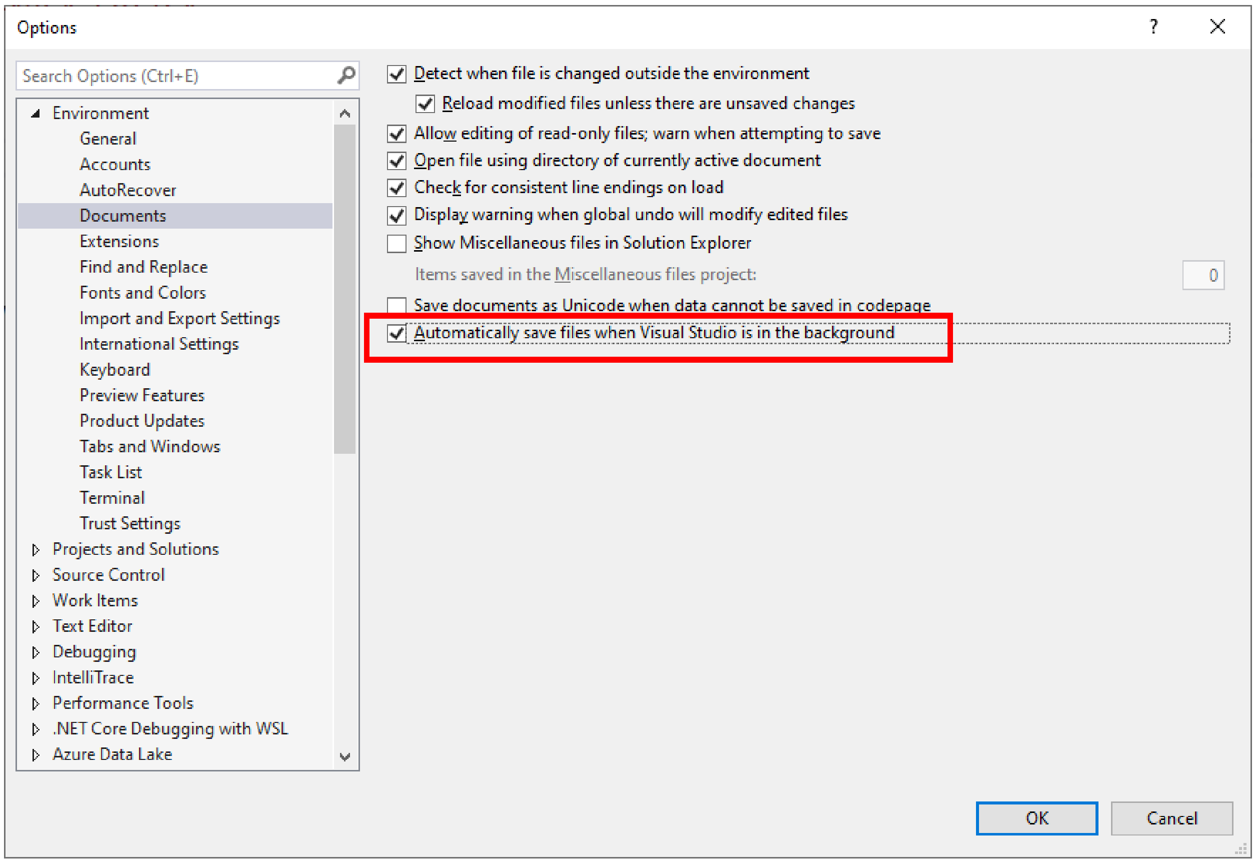
This feature can be useful when you have very complex class hierarchies and you want a visual representation of the code structure.

Improvements to code typing

Some minor yet useful improvements have been added to the coding experience and are summarized in the following paragraphs.

Document autosave

Visual Studio 2022 offers a new autosave feature that allows for automatically saving all open dirty documents when the IDE loses focus. Files are saved to disk if possible, otherwise they will need to be saved manually (the * indicator will still indicate dirty documents). This feature is not enabled by default, but you can enable it in the Options dialog under the Documents group of the Environment node. The option you need to enable is called **Automatically save files when Visual Studio is in the background** (see Figure 20).



- ✓ Detect when file is changed outside the environment
  - ✓ Reload modified files unless there are unsaved changes
- ✓ Allow editing of read-only files; warn when attempting to save
- ✓ Open file using directory of currently active document
- ✓ Check for consistent line endings on load
- ✓ Display warning when global undo will modify edited files
- ☐ Show Miscellaneous files in Solution Explorer

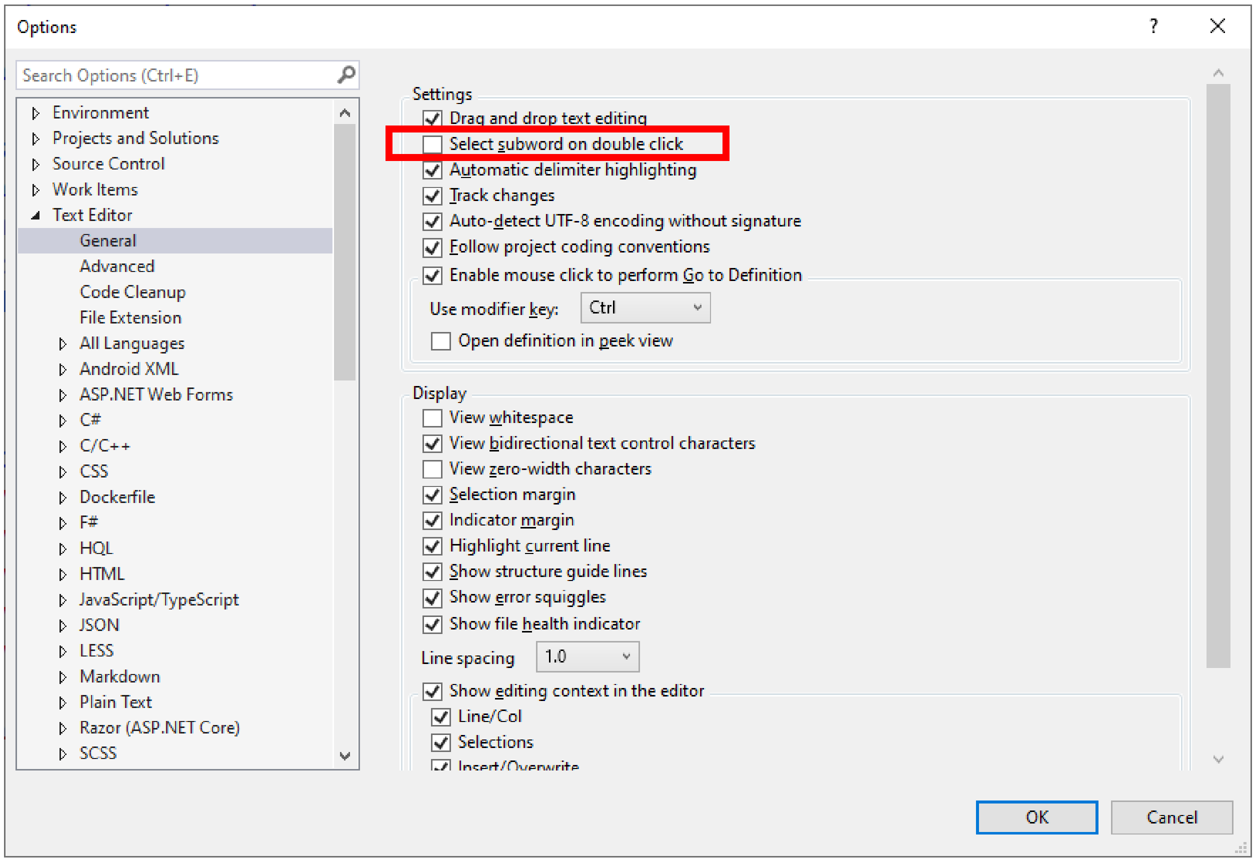
Items saved in the Miscellaneous files project: 0

- ☐ Save documents as Unicode when data cannot be saved in codepage
- ✓ Automatically save files when Visual Studio is in the background

Figure 20: Enabling Document Autosave

Subword navigation

Visual Studio 2022 introduces **subword navigation**. For example, if you have the **`FirstName`** literal, **`First`** and **`Name`** are recognized as individual words, so navigating with Ctrl+Alt+Left Arrow and Ctrl+Alt+Right Arrow will cause Visual Studio to highlight both individually. This feature is enabled by default, but you can go back to the classic behavior by disabling the **Select subword on double click** option in the Text Editor > General group of the Options window (see Figure 21).



Settings

- [x] Drag and drop text editing
- [ ] Select subword on double click
- [x] Automatic delimiter highlighting
- [x] Track changes
- [x] Auto-detect UTF-8 encoding without signature
- [x] Follow project coding conventions
- [x] Enable mouse click to perform Go to Definition

Use modifier key: Ctrl

- [ ] Open definition in peek view

Display

- [ ] View whitespace
- [x] View bidirectional text control characters
- [ ] View zero-width characters
- [x] Selection margin
- [x] Indicator margin
- [x] Highlight current line
- [x] Show structure guide lines
- [x] Show error squiggles
- [x] Show file health indicator

Line spacing 1.0

- [x] Show editing context in the editor
- [x] Line/Col
- [x] Selections
- [x] Insert/Overwrite

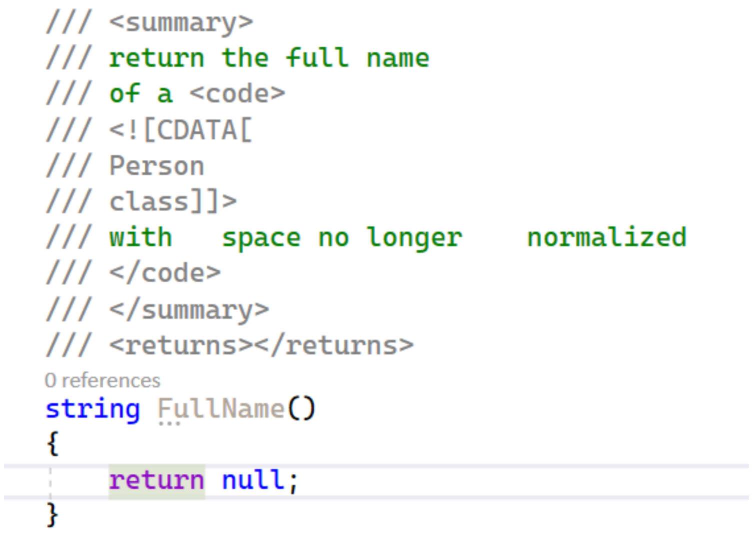
Figure 21: Controlling Subword Navigation

Multi-caret copy/paste improvements

As you may know, Visual Studio has offered a multi-caret developer experience for several years, but when you copied and pasted with multi-carets, the entire clipboard content was duplicated on each line. In Visual Studio 2022, this has improved. Now, when you paste over multiple carets, individual lines are pasted at the appropriate location.

Multiline XML comments

With XML comments, which you use to implement source code documentation, it is now possible to have multiline **`CData`** sections, and the white space will no longer be normalized. Figure 22 shows an example of how this works.



```
/// <summary>
/// return the full name
/// of a <code>
/// <![CDATA[
/// Person
/// class]]>
/// with   space no longer    normalized
/// </code>
/// </summary>
/// <returns></returns>
0 references
string FullName()
{
    return null;
}
```

Figure 22: Multiline CData Sections inside XML Comments

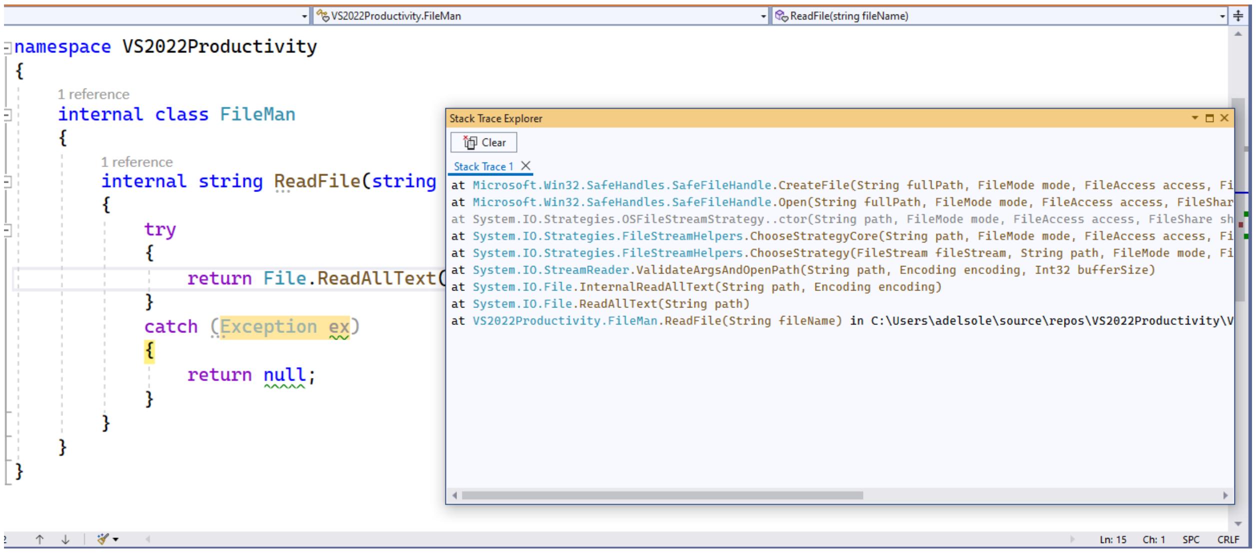
This can be useful when you need to add long **`CData`** sections and you want to keep indentation and spacing.

Synchronizing namespaces between objects

Visual Studio 2022 introduces a new feature that allows for synchronizing namespaces with the folder structure of a project. This is considered a code refactoring, and it is available by right-clicking a project name in Solution Explorer and then selecting **Sync Namespaces**. When finished, Visual Studio will show an informational dialog about the result of the operation. This feature can be very useful especially if you have added files under folders that, over time, you realize are no longer appropriate.

The Stack Trace Explorer window

A new tool window called Stack Trace Explorer has been added to Visual Studio 2022 in order to simplify the way you browse an exception's stack trace. It can be enabled by clicking the **Stack Trace Explorer** command in the **View** > **Other Windows** submenu. When an exception is thrown, you should be able to see the stack trace in the new window. If this does not happen, you can copy the stack trace text from the exception data tip and paste it into the tool window. Figure 23 shows an example based on a **`FileNotFoundException`**.



```
namespace VS2022Productivity
{
    1 reference
    internal class FileMan
    {
        1 reference
        internal string ReadFile(string
        {
            try
            {
                return File.ReadAllText(
            }
            catch (Exception ex)
            {
                return null;
            }
        }
    }
}
```

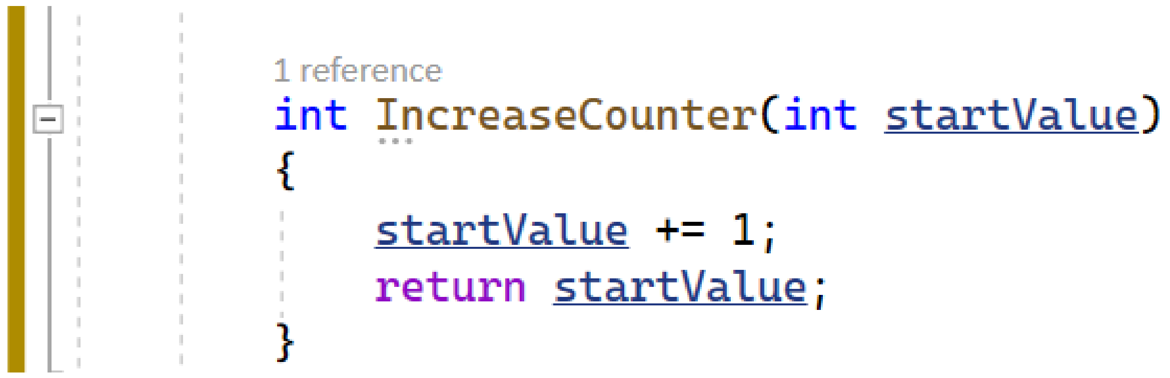
```
at Microsoft.Win32.SafeHandles.SafeFileHandle.CreateFile(String fullPath, FileMode mode, FileAccess access, Fi
at Microsoft.Win32.SafeHandles.SafeFileHandle.Open(String fullPath, FileMode mode, FileAccess access, FileShar
at System.IO.Strategies.OSFileStreamStrategy..ctor(String path, FileMode mode, FileAccess access, FileShare sh
at System.IO.Strategies.FileStreamHelpers.ChooseStrategyCore(String path, FileMode mode, FileAccess access, Fi
at System.IO.Strategies.FileStreamHelpers.ChooseStrategy(FileStream fileStream, String path, FileMode mode, Fi
at System.IO.StreamReader.ValidateArgsAndOpenPath(String path, Encoding encoding, Int32 bufferSize)
at System.IO.File.InternalReadAllText(String path, Encoding encoding)
at System.IO.File.ReadAllText(String path)
at VS2022Productivity.FileMan.ReadFile(String fileName) in C:\Users\adelsole\source\repos\VS2022Productivity\V
```

Figure 23: The Stack Trace Explorer Tool Window in Action

As you can see, the Stack Trace Explorer displays method calls with syntax colorization for improved legibility. In addition, you can click on user code to quickly move to a specific line in the stack trace. By following the same steps described previously, it is also possible to display multiple stack traces. You can click the **Clear** button to delete the content of the window.

Underlying reassigned variables

It is quite common to have variables that are reassigned with another value during their lifecycle. Visual Studio 2022 makes it easier to highlight variables that have been reassigned via an option that underlines the variable name and is specific to C# and Visual Basic. If you look at Figure 24, you can see how the **`IncreaseCounter`** method receives a value through the **`startValue`** variable, which is reassigned in the method body and is underlined.



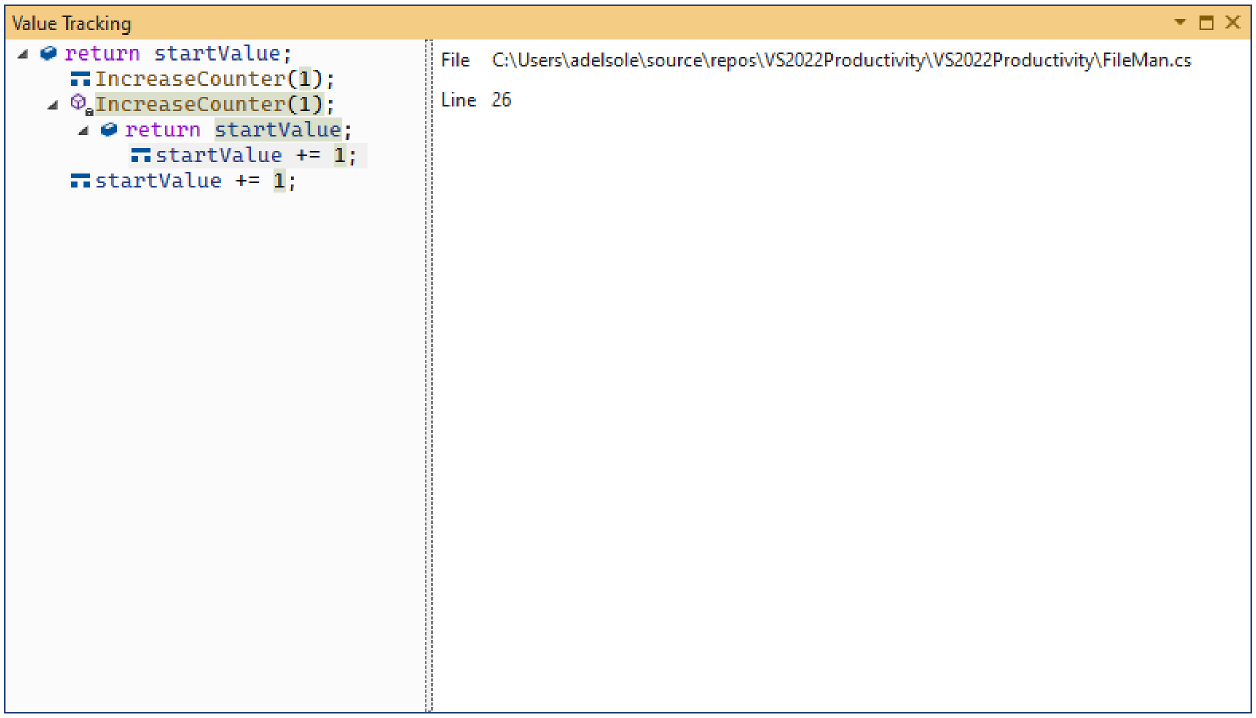
```
1 reference
int IncreaseCounter(int startValue)
{
    startValue += 1;
    return startValue;
}
```

Figure 24: Underlining Reassigned Variables

This feature is not enabled by default. You need to open the **Options** dialog and then enable the **Underline reassigned variables** option in the **Advanced** node of the C# and Visual Basic text editor options.

Tracking source values

Another new, useful feature in Visual Studio 2022 allows for tracking a variable's assignments through the code. To accomplish this, you right-click a variable name anywhere in the code and then select **Track Value Source**. Figure 25 shows an example based on a variable called **`startValue`**, and it shows a visual representation of all the assignments of this variable.



```
◢ return startValue;
    IncreaseCounter(1);
  ◢ IncreaseCounter(1);
    ◢ return startValue;
        startValue += 1;
    startValue += 1;
```

File C:\Users\adelsole\source\repos\VS2022Productivity\VS2022Productivity\FileMan.cs

Line 26

Figure 25: Tracking Value Sources

As you can see, the left side of the window shows the list of members that are manipulating the selected variable. Each member is identified with an icon that matches the way IntelliSense represents it. If you click on a member, you will see the code file and the line where the member is manipulating the variable. This tool window is very useful when you need to walk through a variable's assignments, which can be many.

Note: Rather than values, the track value source feature shows where a variable is manipulated.

Chapter summary

Productivity improvements are at the center of every major release of Visual Studio, and version 2022 makes no exception. In this new release, you can take advantage of the powerful IntelliCode code completion engine, which literally learns from your code patterns and suggests code depending on the context. You can now take advantage of visual features like inheritance margin, stack trace explorer, and track value source, which allow for investigating the behavior of types, exceptions, and variables without losing focus on the active file. All these improvements to .NET productivity also apply to developing for .NET 6, whose support in Visual Studio 2022 is discussed in the next chapter.

Chapter 3 Support for .NET 6

Together with Visual Studio 2022, Microsoft also released .NET 6, the new major release of the popular development technology. You can build applications based on .NET 6 only with Visual Studio 2022, since the previous versions do not have specific support for it. Obviously, the focus of this chapter will not go into the details of .NET 6 as a development platform, but an overview of the architecture will be provided with information about how Visual Studio 2022 allows for building apps on the most important release of .NET in recent years.

Overview of the .NET 6 architecture

Before .NET 6, you had to choose between .NET Framework, .NET Core, and Xamarin to build a specific type of application. All these technologies share many APIs, libraries, tools, and programming languages, though they target different scenarios.

The goal of Microsoft has been unifying all these platforms into one .NET technology, with one set of APIs, libraries, and tools. This unification goes under the name of .NET, without any additional descriptor like Framework or Core. The work of bringing all the platforms under one .NET started with .NET 5, released in October 2020, whose main goal was making Xamarin use .NET Core instead of Mono and have one code base for both. The next step was unifying the .NET Framework and .NET 5 into one shared set of APIs, which goes under the name of .NET 6, released on November 2021. Figure 26 shows the .NET 6 architecture through an illustration from Microsoft, available in the blog post titled “Announcing .NET 6.”

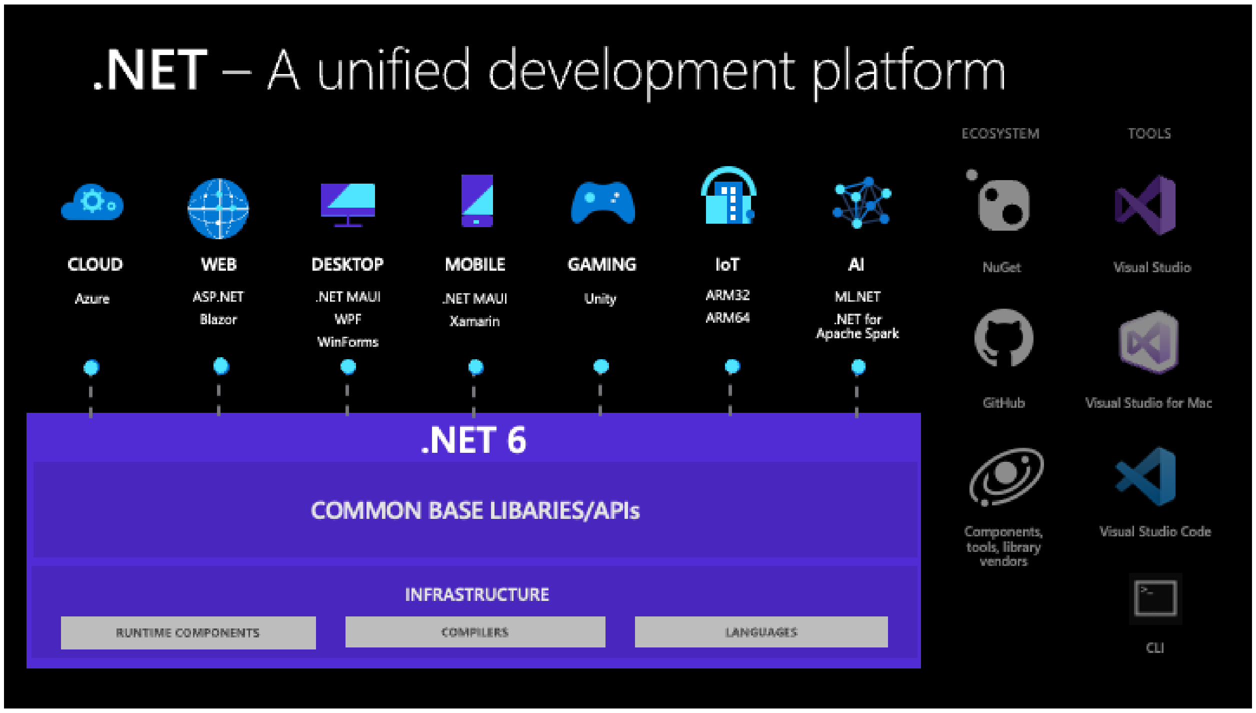
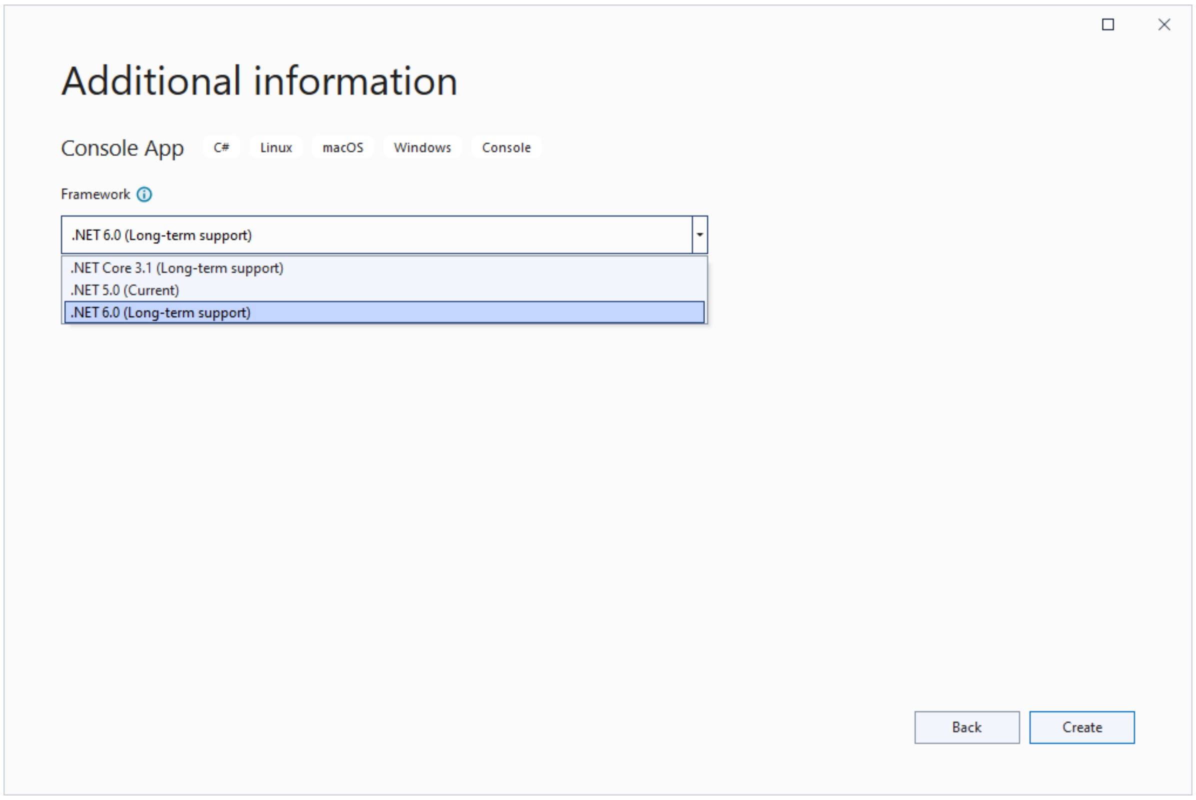


Figure 26: High-Level Architecture of .NET 6 (Source: Microsoft)

As you can see, all the high-level development platforms now rely on the same runtime, libraries, API, and tools. The most relevant addition to .NET 6 is probably the .NET Multi-platform Application User Interface, also referred to as .NET MAUI, the new mobile app development technology that can be considered an evolution of Xamarin.Forms and that allows for targeting mobile and desktop devices from one shared code base. At the time of this writing, .NET MAUI is available as Preview 12 and not yet in production, but later in this chapter you will see how to enable VS 2022 for MAUI. Obviously, .NET 6 not only improves productivity for developers who now have to deal with one platform, but it also provides general performance improvements and allows for using the new Hot Reload feature, described in Chapter 4, “Debugging Improvements.”

Creating projects for .NET 6

In Visual Studio 2022, you will be able to create projects that target .NET 6 from the New Project dialog. It is possible to target .NET 6 with console, ASP.NET Core, Windows Forms, and Windows Presentation Foundation projects. After specifying the project type, you will be asked to select the target .NET version. Figure 27 shows how this option appears for a console application, whereas Figure 28 shows how this option appears for an ASP.NET Core project.



Additional information

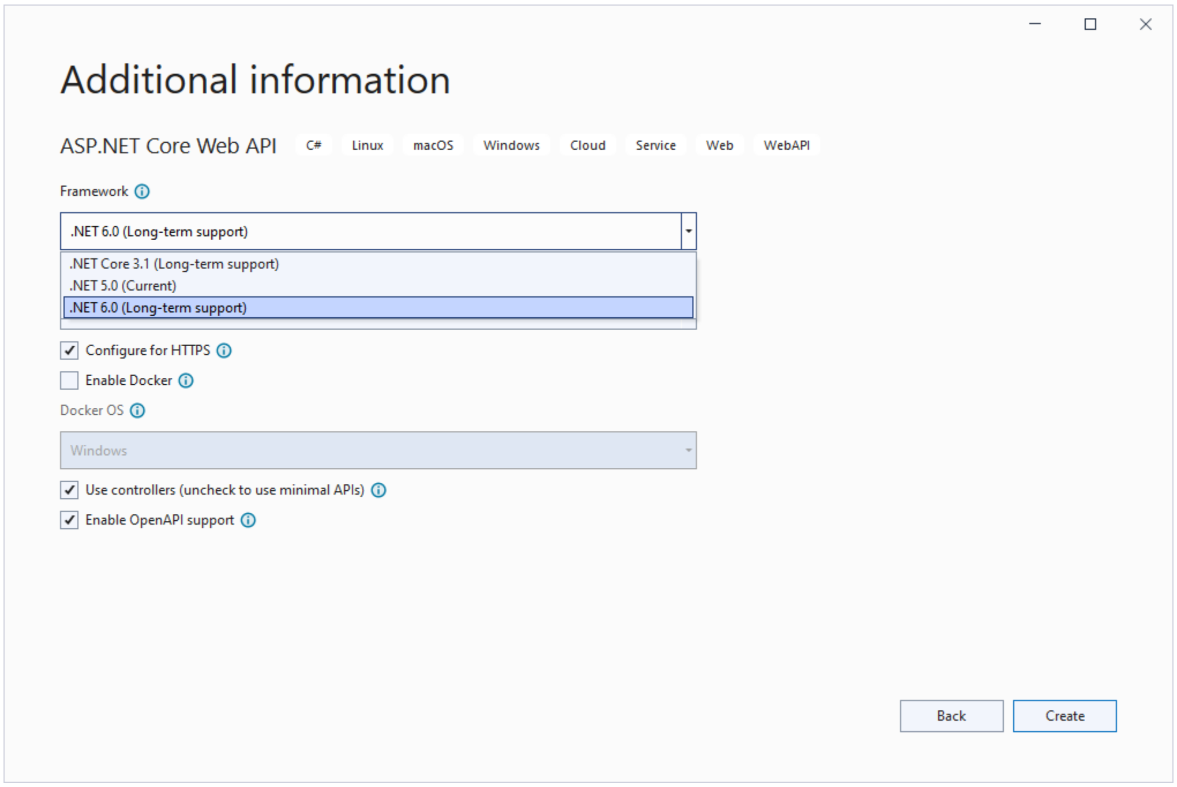
Console App C# Linux macOS Windows Console

Framework

.NET 6.0 (Long-term support)

.NET Core 3.1 (Long-term support)
.NET 5.0 (Current)
.NET 6.0 (Long-term support)

Figure 27: Targeting .NET 6 in a Console App



Additional information

ASP.NET Core Web API C# Linux macOS Windows Cloud Service Web WebAPI

Framework

.NET 6.0 (Long-term support)

.NET Core 3.1 (Long-term support)
.NET 5.0 (Current)
.NET 6.0 (Long-term support)

[✓] Configure for HTTPS
[ ] Enable Docker

Docker OS

Windows

[✓] Use controllers (uncheck to use minimal APIs)
[✓] Enable OpenAPI support

Figure 28: Targeting .NET 6 in an ASP.NET Project

As you can see, .NET 6 is identified as **Long-term support**, which means that it will be supported by Microsoft for three years. When you create a project based on .NET 6, you will immediately notice some changes in those project types that include a Program.cs file, such as console and ASP.NET Core. For a better understanding, consider Code Listing 1 which shows the content of the Program.cs file for a new ASP.NET Core project based on .NET 6.

Code Listing 1

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

Notice that there are no **`namespace`** or **`class Program`** blocks. Starting with C# 9, it is possible to consider the root elements of the application's entry point, such as the **`Program`** class, as implicit. This feature is known as top-level statements and is used by default in Visual Studio 2022. In addition, Visual Studio 2022 is implicitly declaring the following **`using`** directives:

- **`using System;`**
- **`using System.IO;`**
- **`using System.Collections.Generic;`**
- **`using System.Linq;`**
- **`using System.Net.Http;`**
- **`using System.Threading;`**
- **`using System.Threading.Tasks;`**

This allows for writing code that leverages types exposed by the aforementioned namespaces without declaring them explicitly. Top-level statements make it possible to write cleaner code, but you are still allowed to explicitly add the namespace and class definition. This can be either done manually or by using the old-style file template. If you want to do so, you can create a project based on .NET 5.0 and, when ready, change the target framework to .NET 6.0 in the project properties, as suggested by the Microsoft documentation.

Adding references to .NET 6 projects

Like .NET Core and .NET 5, .NET 6 makes it possible to distribute an application with only the subset of core libraries and dependencies it needs. This avoids having an entire runtime technology like the classic .NET Framework installed on the target machine, which might not also be possible on smaller devices like phones and tablets. Consequently, .NET 6 does not rely on a Global Assembly Cache like .NET Framework does, and dependencies are usually installed via NuGet packages. This also implies that the way you add a reference to another assembly has also changed in .NET 6 projects. If you look at Figure 29, you can see which options you now have to add a reference for in Visual Studio 2022.

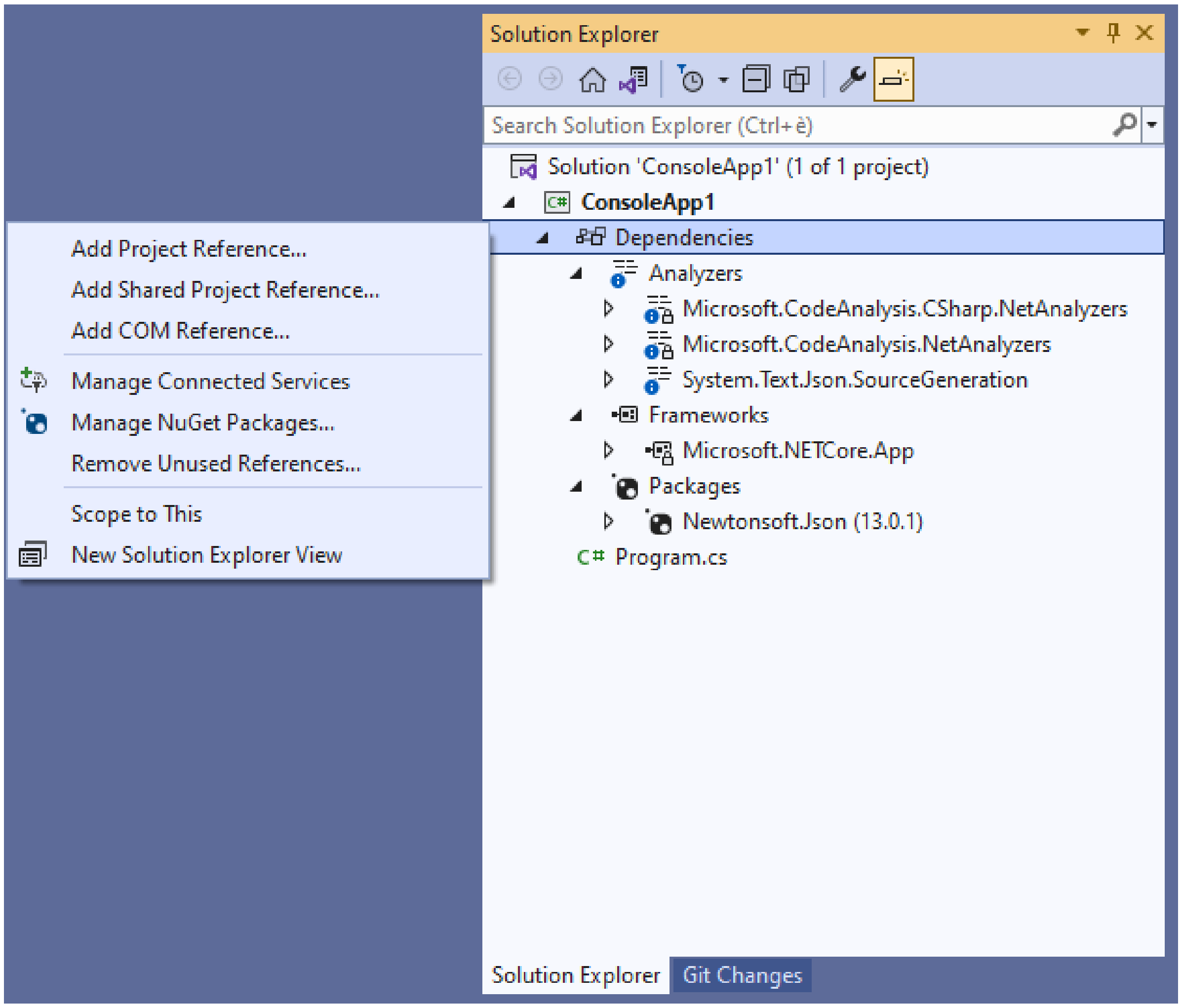

Solution Explorer
Search Solution Explorer (Ctrl+è)
Solution 'ConsoleApp1' (1 of 1 project)
ConsoleApp1
Dependencies
Analyzers
Microsoft.CodeAnalysis.CSharp.NetAnalyzers
Microsoft.CodeAnalysis.NetAnalyzers
System.Text.Json.SourceGeneration
Frameworks
Microsoft.NETCore.App
Packages
Newtonsoft.Json (13.0.1)
Program.cs
Add Project Reference...
Add Shared Project Reference...
Add COM Reference...
Manage Connected Services
Manage NuGet Packages...
Remove Unused References...
Scope to This
New Solution Explorer View
Solution Explorer
Git Changes

Figure 29: Adding References in Visual Studio 2022

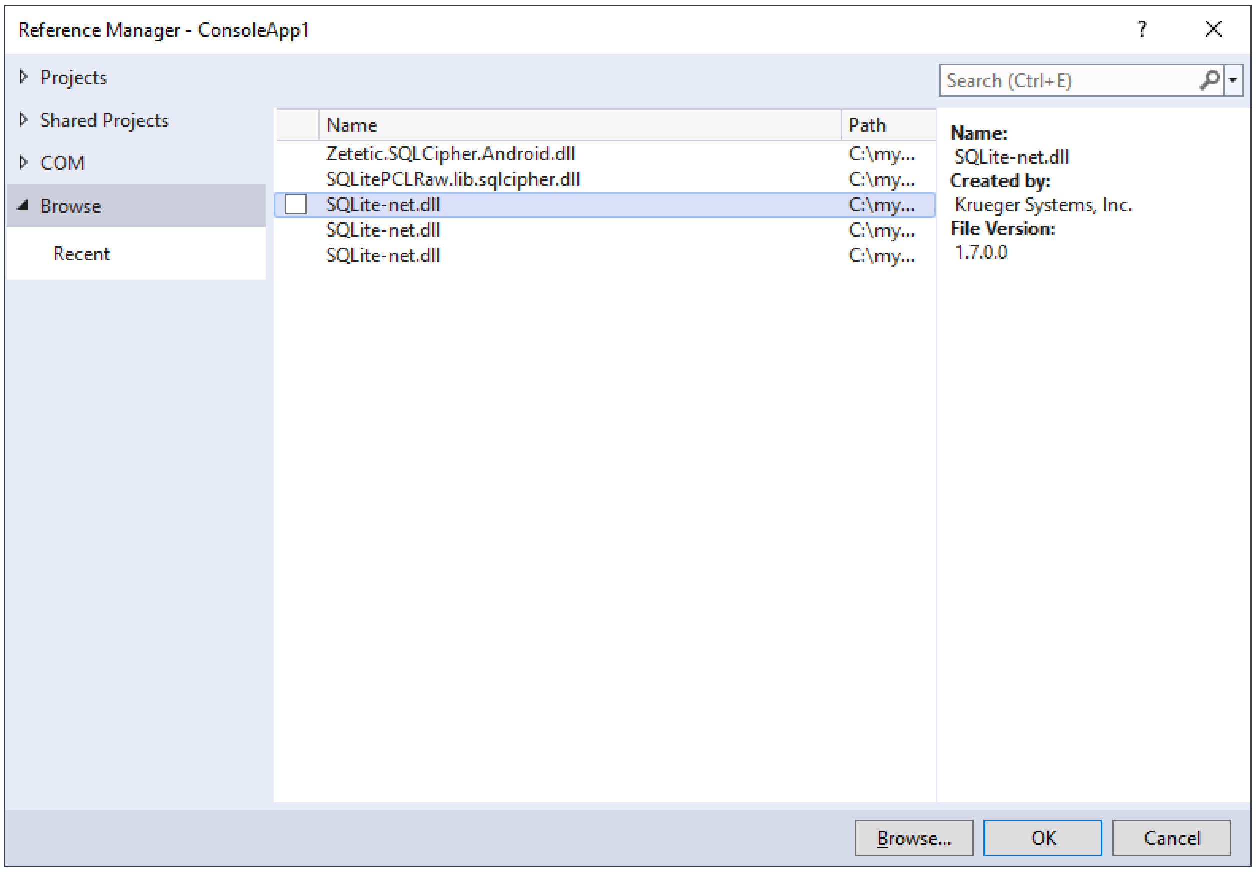
Note: The concepts described in this section do not apply to projects based on .NET Framework, which behave like they did in the past.

Here's a short list of the relevant points:

- The root node of referenced libraries is now called Dependencies, and it groups references by code analyzers, required Microsoft frameworks, and packages added via NuGet.
- The Add Reference command in the context menu is no longer available.
- Options to add a reference only appear if you right-click the Dependencies node. Right-clicking group names has no effect regarding this.

Assuming you need to add a reference to a local assembly, you can follow these steps:

1. Right-click the **Dependencies** node.
2. Select the **Add Project Reference** option.
3. When the Reference Manager dialog appears, click **Browse** and locate the assembly on disk. Figure 30 shows an example.



	Name	Path
	Zetetic.SQLCipher.Android.dll	C:\my...
	SQLitePCLRaw.lib.sqlcipher.dll	C:\my...
☐	SQLite-net.dll	C:\my...
	SQLite-net.dll	C:\my...
	SQLite-net.dll	C:\my...

Name:
SQLite-net.dll
Created by:
Krueger Systems, Inc.
File Version:
1.7.0.0

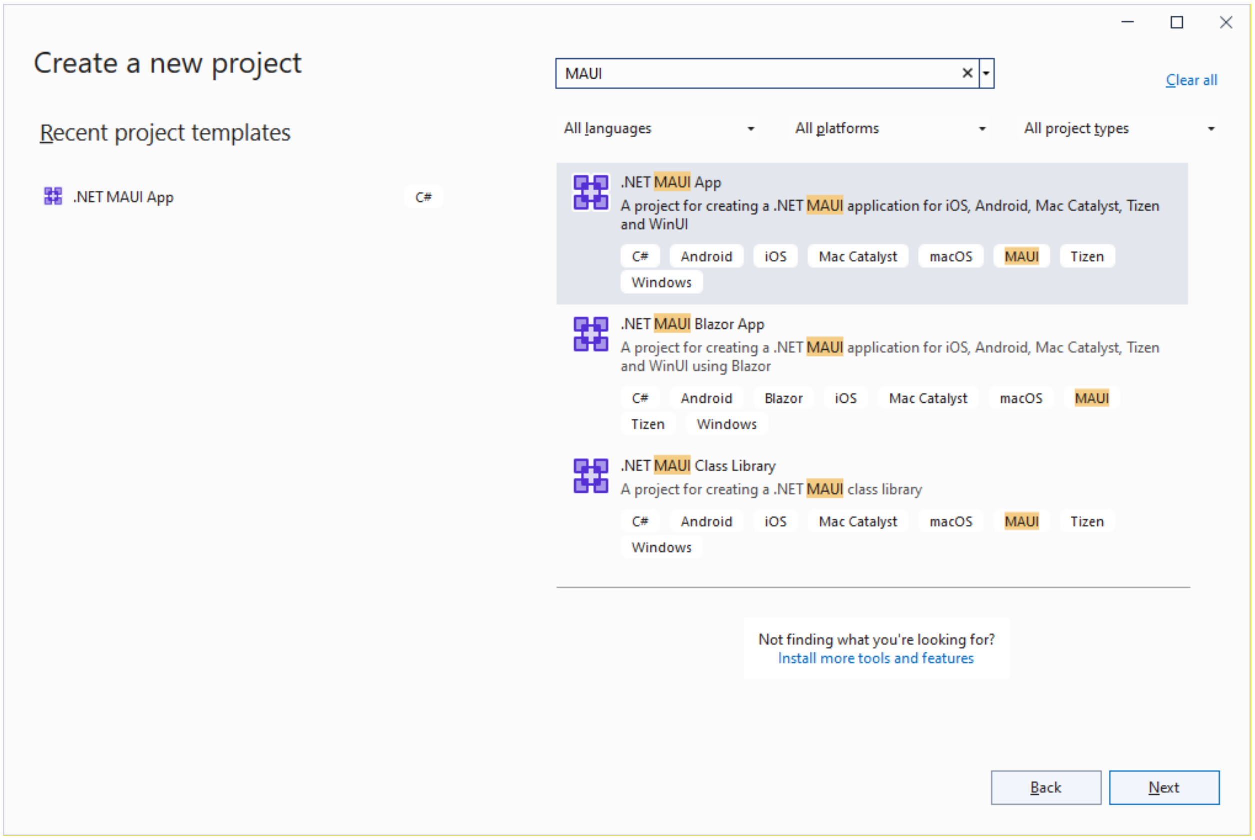
Figure 30: Adding References to Local Assemblies

At this point, the reference will be added and will appear under the Packages group.

Enabling .NET MAUI in Visual Studio 2022

The .NET Multi-platform Application UI framework can be considered the successor of Xamarin.Forms, with many improvements not only to supported platforms and package generation, but also to project development. Discussing .NET MAUI is outside the scope of this ebook, but it can be interesting for you to know how to enable .NET MAUI development in Visual Studio 2022. To enable .NET MAUI on your machine, you need to follow these steps:

1. Download the latest Visual Studio 2022 Preview. This can be installed side-by-side with the production version.
2. In the Visual Studio Preview Installer, select the **Mobile Development with .NET** workload. This is what you will also need to do with the final release.
3. When the installation is completed, open Visual Studio 2022 Preview and create a new project.
4. In the New Project dialog, search for the **.NET MAUI App** project template (see Figure 31).



.NET MAUI App
A project for creating a .NET MAUI application for iOS, Android, Mac Catalyst, Tizen and WinUI
C# Android iOS Mac Catalyst macOS MAUI Tizen Windows

.NET MAUI Blazor App
A project for creating a .NET MAUI application for iOS, Android, Mac Catalyst, Tizen and WinUI using Blazor
C# Android Blazor iOS Mac Catalyst macOS MAUI Tizen Windows

.NET MAUI Class Library
A project for creating a .NET MAUI class library
C# Android iOS Mac Catalyst macOS MAUI Tizen Windows

Not finding what you're looking for?
Install more tools and features

Figure 31: Creating a .NET MAUI Project

When the new project is ready, you will be able to develop your first .NET MAUI application, leveraging the new project structure and the dedicated Visual Studio tooling. You can refer to the official documentation to learn how to create your first app.

Chapter summary

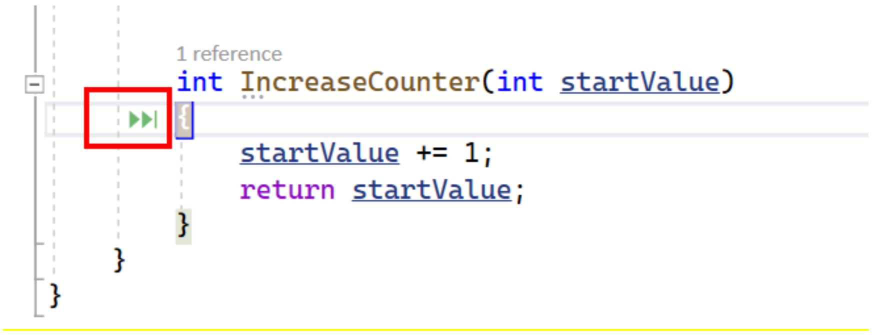
Visual Studio 2022 is the only version of the IDE that allows for creating projects based on .NET 6, and it offers all the tools you need to do so. Targeting .NET 6 is something you can do when creating a new project, and Visual Studio automatically takes advantage of the new C# 10 features, offering support for top-level statements. Because of the way apps are distributed, the way you add references has also changed accordingly, so you typically install libraries via NuGet, but you can still add references to local assemblies by browsing the disk. Visual Studio 2022 also adds support for .NET MAUI, which is based on .NET 6, with new project templates and simplified project structure. In its natural evolution, the new edition of Visual Studio adds important productivity tools to the debugging experience, which are discussed in the next chapter.

Chapter 4 Debugging Improvements

Debugging is the most crucial part of developing any kind of application; therefore, productivity in this area is extremely relevant. Microsoft adds new debugging features in every new major release of Visual Studio, and version 2022 is no exception. This chapter describes the debugging improvements you will find in VS 2022 that will boost your productivity in the most important part of the development lifecycle.

Forcing line execution

Visual Studio has always had a debugging command called **Run To Cursor**, which you can run by right-clicking somewhere in the code editor and then selecting the command from the context menu. This command allows for running all the code to the position of the cursor, but it will stop if any breakpoint is encountered. Visual Studio 2022 introduces a new command called **Force Run To Cursor**, which still executes all the code up to the line where the cursor is but ignores any preceding breakpoints. This command can be executed by right-clicking anywhere in the code editor and then selecting the item from the context menu. The latest versions of Visual Studio have also implemented a command called **Run To Click**, which simplifies the way you debug your code by clicking a play icon that appears when you hover over a line of code in debugging mode. Similar to Run To Cursor, this feature allows for executing all the code to the point where you select it, but it will stop if any breakpoint is encountered. In Visual Studio 2022, there is a new command called **Force Run To Click**, which can be enabled by pressing Shift and then hovering over the line of code you want to run code to. Figure 32 shows how the icon appears in the code editor.



```
1 reference
int IncreaseCounter(int startValue)
{
    startValue += 1;
    return startValue;
}
    }
}
```

Figure 32: Using Force Run To Click

When you click the icon, all the code will be executed to the current position and any breakpoints will be ignored. This feature can be very useful when you need to test the execution flow of some code but you do not want to remove any breakpoints that you plan to reuse later.

Breakpoint improvements

Visual Studio 2022 also extends breakpoints with new features. This section describes the new glyph and context menu, and the new dependent and temporary breakpoint features.

Breakpoint user interface

Visual Studio 2022 now indicates where a breakpoint can be added via a new dark glyph that appears when you hover over the editor margin (see Figure 33).

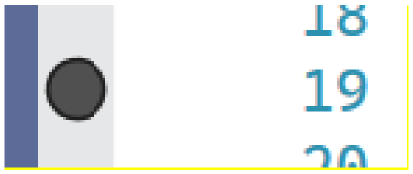


Figure 33: New Glyph Showing Where a Breakpoint is Supported

This will let you know immediately if a breakpoint can be added to a given location. Simply click to add a breakpoint, as usual. Another addition to the way you interact with breakpoints is a new context menu that appears if you right-click on the editor margin, as shown in Figure 34.

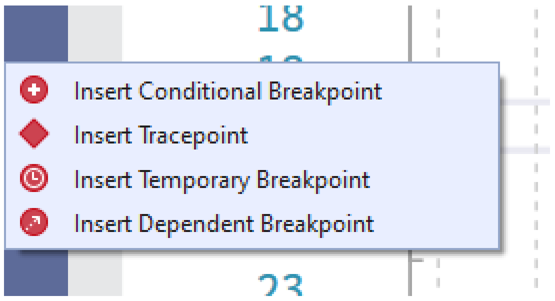

18
Insert Conditional Breakpoint
Insert Tracepoint
Insert Temporary Breakpoint
Insert Dependent Breakpoint
23

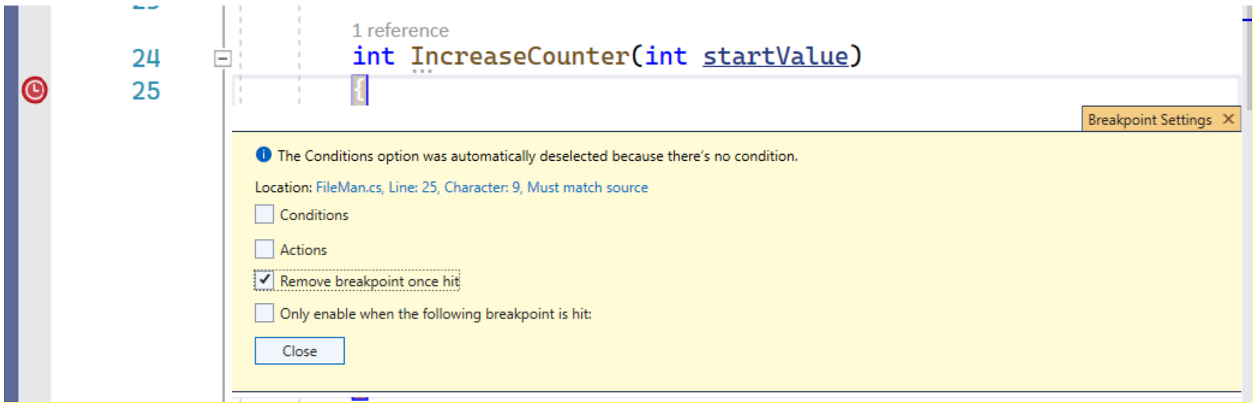
Figure 34: The New Context Menu for Breakpoints

As you might know, conditional breakpoints are hit only when the specified condition is met, whereas tracepoints display diagnostic messages in the Output window instead of breaking the execution. There are two new types of breakpoints, temporary breakpoints and dependent breakpoints, discussed in the next paragraphs.

Introducing temporary breakpoints

A temporary breakpoint is hit only once and is automatically removed once it has been hit. You have two ways to add a temporary breakpoint:

- Right-clicking the editor margin and selecting **Insert Temporary Breakpoint** (see Figure 34).
- Adding a breakpoint the usual way, then right-clicking the breakpoint and selecting **Conditions**, and finally enabling the **Remove breakpoint once hit** option in Breakpoint Settings (see Figure 35).



```
1 reference
int IncreaseCounter(int startValue)
{
```

ⓘ The Conditions option was automatically deselected because there's no condition.

Location: FileMan.cs, Line: 25, Character: 9, Must match source

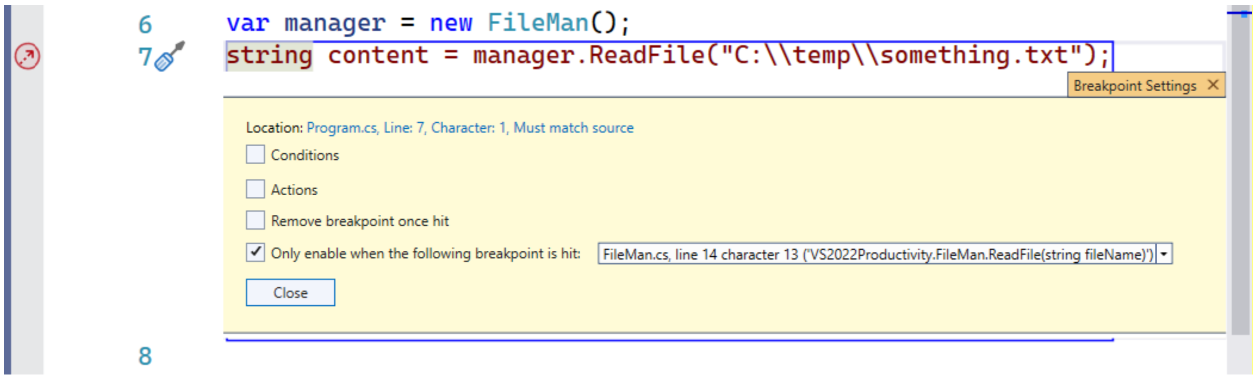
Breakpoint Settings
☐ Conditions
☐ Actions
☑ Remove breakpoint once hit
☐ Only enable when the following breakpoint is hit:
Close

Figure 35: Setting a Breakpoint as Temporary

Temporary breakpoints are useful when you need to test a condition or a code block only once, such as inside a `for` loop, without breaking the application execution all the time.

Introducing dependent breakpoints

Dependent breakpoints are hit only if another specified breakpoint is also hit, otherwise they will be skipped. You can insert a dependent breakpoint by right-clicking the editor margin and then selecting **Insert Dependent Breakpoint** (see Figure 34). This will add a breakpoint and will open the Breakpoint Settings pop-up, automatically enabling the option called **Only enable when the following breakpoint is hit**, as shown in Figure 36.



```
var manager = new FileMan();
string content = manager.ReadFile("C:\\temp\\something.txt");
```

Location: Program.cs, Line: 7, Character: 1, Must match source

Only enable when the following breakpoint is hit: FileMan.cs, line 14 character 13 ('VS2022Productivity.FileMan.ReadFile(string fileName)')

Breakpoint Settings
☐ Conditions
☐ Actions
☐ Remove breakpoint once hit
☑ Only enable when the following breakpoint is hit:
Close

Figure 36: Adding a Dependent Breakpoint

You will have a chance to specify which breakpoint must be hit first to make the new one dependent. When done, simply click Close. The glyph of a dependent breakpoint is an arrow that points up and to the right.

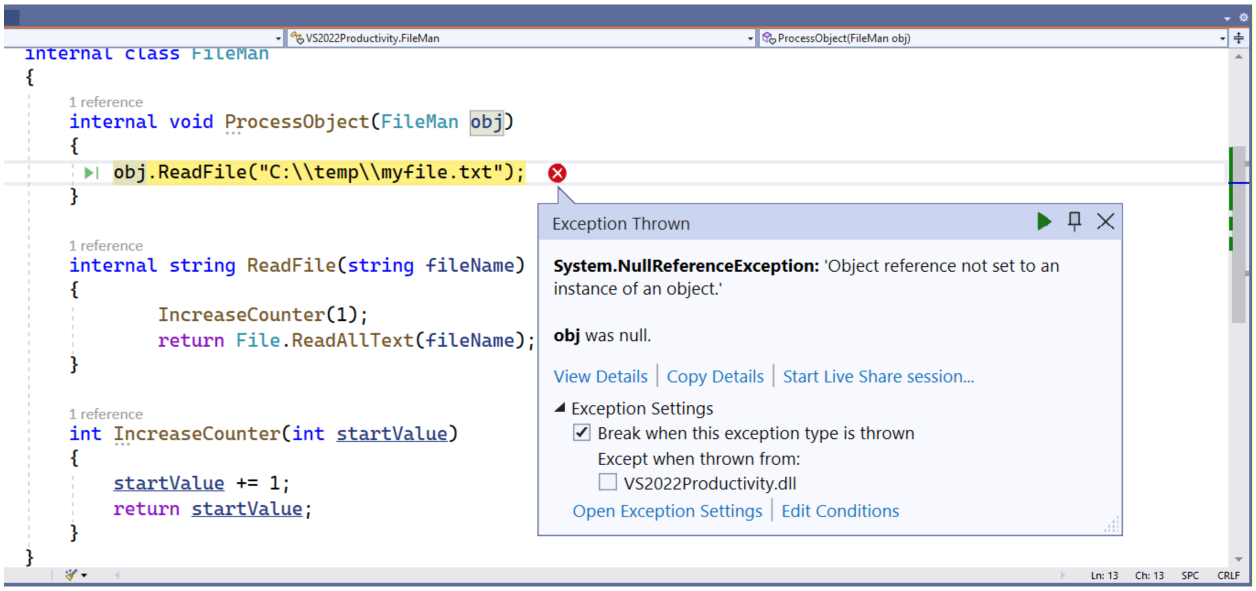
Tip: Both temporary and dependent breakpoints can be quickly removed by clicking them in the editor margin or via the appropriate commands in the Debug menu.

Breakpoint drag and drop

Breakpoints now support drag and drop, so you can quickly move a breakpoint from one location by simply clicking and dragging the breakpoint and releasing it to the desired line of code. This also works with breakpoints that include conditions and actions in their settings as long as the conditions and actions are also within the context of the new desired location.

Exception settings improvements

Visual Studio 2022 introduces an interesting improvement to the exception helper dialog that appears when a runtime exception is thrown. More specifically, a new Continue button has been added at the upper right corner of the dialog, as shown in Figure 37.



```
internal class FileMan
{
    1 reference
    internal void ProcessObject(FileMan obj)
    {
        obj.ReadFile("C:\\temp\\myfile.txt");
    }

    1 reference
    internal string ReadFile(string fileName)
    {
            IncreaseCounter(1);
            return File.ReadAllText(fileName);
    }

    1 reference
    int IncreaseCounter(int startValue)
    {
        startValue += 1;
        return startValue;
    }
}
```

Exception Thrown

System.NullReferenceException: 'Object reference not set to an instance of an object.'

obj was null.

View Details | Copy Details | Start Live Share session...

Exception Settings
- [✓] Break when this exception type is thrown
- Except when thrown from:
  - [ ] VS2022Productivity.dll

Open Exception Settings | Edit Conditions

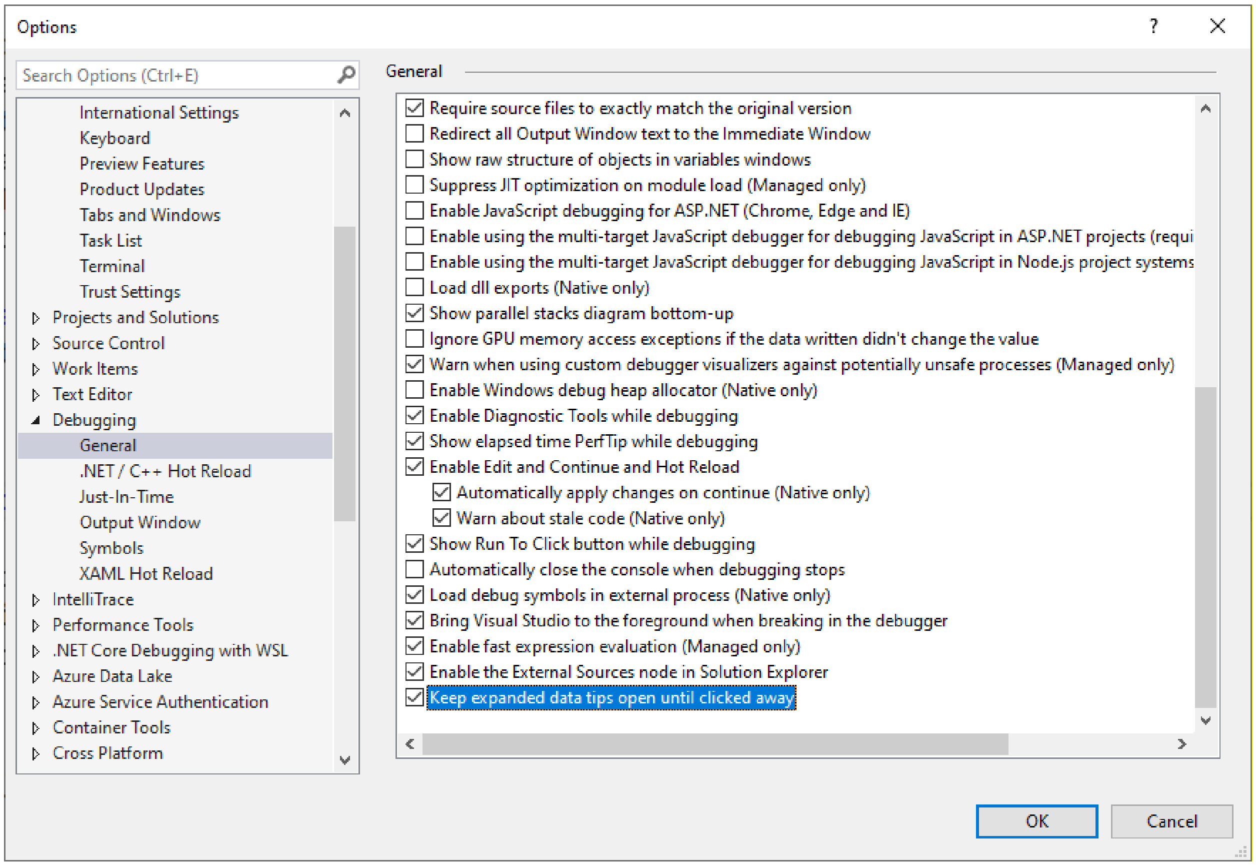
Figure 37: New Continue Button on Exception Helper Dialog

The new button is represented by a green play icon, and it works only if the specific context allows for resuming the application execution.

Note: Remember to avoid runtime exceptions using appropriate `try...catch` blocks. The code shown in Figure 37 is intentionally causing an exception to demonstrate the new feature, but it is not written properly because it is not checking for null references and not implementing a `try...catch` block.

Sticky data tips

As you probably know, when you hover over a variable at debugging time, the code editor shows the so-called **data tips**, small tooltips that allow you to investigate a variable's value. The default behavior of data tips is that they automatically disappear after a few seconds, but sometimes you do not have enough time to look at the content of a variable, especially with those that represent complex types. Visual Studio 2022 introduces **sticky data tips**, which leave a data tip expanded until you click outside it. You need to explicitly enable this feature in the **Debugging** section of the **Options** dialog, as shown in Figure 38.



General

- ☑ Require source files to exactly match the original version
- ☐ Redirect all Output Window text to the Immediate Window
- ☐ Show raw structure of objects in variables windows
- ☐ Suppress JIT optimization on module load (Managed only)
- ☐ Enable JavaScript debugging for ASP.NET (Chrome, Edge and IE)
- ☐ Enable using the multi-target JavaScript debugger for debugging JavaScript in ASP.NET projects (requi
- ☐ Enable using the multi-target JavaScript debugger for debugging JavaScript in Node.js project systems
- ☐ Load dll exports (Native only)
- ☑ Show parallel stacks diagram bottom-up
- ☐ Ignore GPU memory access exceptions if the data written didn't change the value
- ☑ Warn when using custom debugger visualizers against potentially unsafe processes (Managed only)
- ☐ Enable Windows debug heap allocator (Native only)
- ☑ Enable Diagnostic Tools while debugging
- ☑ Show elapsed time PerfTip while debugging
- ☑ Enable Edit and Continue and Hot Reload
    - ☑ Automatically apply changes on continue (Native only)
    - ☑ Warn about stale code (Native only)
- ☑ Show Run To Click button while debugging
- ☐ Automatically close the console when debugging stops
- ☑ Load debug symbols in external process (Native only)
- ☑ Bring Visual Studio to the foreground when breaking in the debugger
- ☑ Enable fast expression evaluation (Managed only)
- ☑ Enable the External Sources node in Solution Explorer
- ☑ Keep expanded data tips open until clicked away

Figure 38: Enabling Sticky Data Tips

You need to enable the option labeled **Keep expanded data tips open until clicked away**. Sticky data tips will be enabled at this point. If not, you might need to restart Visual Studio. The appearance of data tips will not change; they will just stay open until you click outside of them.

IEnumerable Debugger Visualizer

One of the biggest additions to the debugging experience in Visual Studio 2022 is the **IEnumerable Debugger Visualizer**, an integrated tool that allows you to debug collections with a structured view instead of investigating their values inside small data tips. This tool supports any collection that derives from `IEnumerable<T>`, and even arrays. Setting up a code example is a good idea to understand how it works, so create a new console application in C# (the new visualizer also supports Visual Basic). If you choose to target .NET 6, remember that the project will use top-level statements described in Chapter 3, "Support for .NET 6." The goal is creating and debugging a collection, so the first task is creating a data model represented by the **`Product`** class that you see in Code Listing 2.

Code Listing 2

```
public class Product
{
    public int Id { get; set; }
    public string ProductName { get; set; }
    public double QuantityInStock { get; set; }
    public string ProductCode { get; set; }
    public DateTime DateOfPurchase { get; set; }
}
```

Then you need a class that exposes a collection of products. Code Listing 3 shows an example of collection implementation, including the generation of sample data that can be used for local demonstration purposes.

Code Listing 3

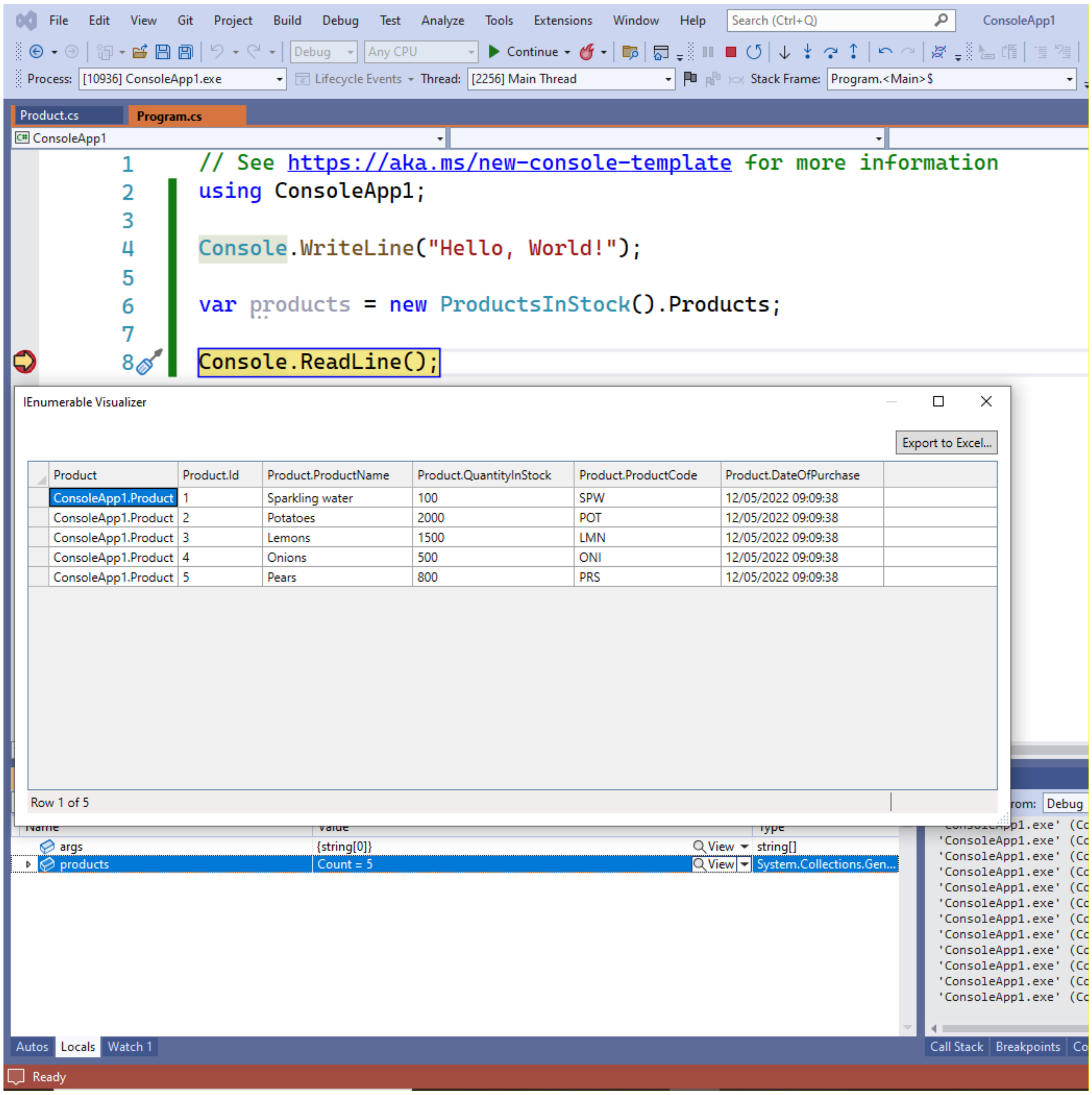
```
public class ProductsInStock
{
    public List<Product> Products { get; set; }

    public ProductsInStock()
    {
        Products = GenerateSampleProducts();
    }

    private List<Product> GenerateSampleProducts()
    {
        var product1 = new Product();
        product1.Id = 1;
        product1.ProductName = "Sparkling water";
        product1.ProductCode = "SPW";
        product1.DateOfPurchase = DateTime.Now;
        product1.QuantityInStock = 100;

        var product2 = new Product();
        product2.Id = 2;
        product2.ProductName = "Potatoes";
        product2.ProductCode = "POT";
        product2.DateOfPurchase = DateTime.Now;
        product2.QuantityInStock = 2000;

        var product3 = new Product();
```

```
            product3.Id = 3;
            product3.ProductName = "Lemons";
            product3.ProductCode = "LMN";
            product3.DateOfPurchase = DateTime.Now;
            product3.QuantityInStock = 1500;

            var product4 = new Product();
            product4.Id = 4;
            product4.ProductName = "Onions";
            product4.ProductCode = "ONI";
            product4.DateOfPurchase = DateTime.Now;
            product4.QuantityInStock = 500;

            var product5 = new Product();
            product5.Id = 5;
            product5.ProductName = "Pears";
            product5.ProductCode = "PRS";
            product5.DateOfPurchase = DateTime.Now;
            product5.QuantityInStock = 800;

            var products = new List<Product>()
                { product1, product2, product3, product4, product5 };
            return products;
        }
    }
```

In the **Program.cs** file, add the following two lines of code at the bottom of the file if you are targeting .NET 6, or at the bottom of the **`Main`** method if you are targeting .NET 5 and lower:

```
var products = new ProductsInStock().Products;
Console.ReadLine();
```

Add a breakpoint on the **`ReadLine`** statement and press **F5** to start debugging. When the application enters break mode, you will see the **`products`** variable listed in the **Locals** tool window. In this window, click the **View** button. At this point you will see the **IEnumerable Visualizer** window, which looks like Figure 39.



```
// See https://aka.ms/new-console-template for more information
using ConsoleApp1;

Console.WriteLine("Hello, World!");

var products = new ProductsInStock().Products;

Console.ReadLine();
```

IEnumerable Visualizer

Product	Product.Id	Product.ProductName	Product.QuantityInStock	Product.ProductCode	Product.DateOfPurchase
ConsoleApp1.Product	1	Sparkling water	100	SPW	12/05/2022 09:09:38
ConsoleApp1.Product	2	Potatoes	2000	POT	12/05/2022 09:09:38
ConsoleApp1.Product	3	Lemons	1500	LMN	12/05/2022 09:09:38
ConsoleApp1.Product	4	Onions	500	ONI	12/05/2022 09:09:38
ConsoleApp1.Product	5	Pears	800	PRS	12/05/2022 09:09:38

Row 1 of 5

Name	Value	Type
args	{string[0]}	string[]
products	Count = 5	System.Collections.Gen...

Figure 39: Debugging Collections with the IEnumerable Visualizer

As you can see, collection data is represented in a tabular view with columns and rows that show property values for each object instance in the collection. Notice that the first column of the table always shows the fully qualified object name. You also have an option to export data to an Excel file with the **Export to Excel** button, which is very useful with real, long data collections.

Note: The only way to display the IEnumerable Debugger Visualizer is to click the magnifier icon in the *Locals* ***window. If you click the magnifier icon when hovering over the collection variable name in the code editor, regular data tips will appear.***

This is a tremendous addition to the Visual Studio toolbox because it dramatically simplifies the way you debug and investigate collections, especially with real data that comes from a database or a web service. Do not throw away the sample project; it will be used in the next section.

Debugging external sources with Source Link

Applications built upon .NET certainly rely on the framework libraries, but also on external dependencies such as NuGet packages. During the development and debugging stages, you might get exceptions thrown by such dependencies, but normally you do not have an option to debug the code of the dependency itself. Visual Studio 2022 introduces a new feature called **Source Link** which allows for debugging and stepping through the source code of dependencies, such as framework libraries and NuGet packages, exactly like you would do with your own code, bringing your debugging experience to a higher level.

Source Link basically loads the public debugging symbols of each referenced library (also referred to as modules), so that Visual Studio can enable the usual debugging tools. In addition, Visual Studio will load the appropriate source code files of the dependency, which means that Source Link is available only with those dependencies whose code has been published to a Git repository. Visual Studio supports the majority of Git repositories with Source Link, such as Azure DevOps, GitHub, Bitbucket, GitLab, and Gitea.

Note: You can also enable your libraries and NuGet packages for Source Link. This is not covered in this ebook, but the Source Link project documentation provides guidance on how to do it.

Loading symbols for every dependency used in the application can have a huge impact on performance, especially the first time you start debugging an app. You need to be aware of this before you use this feature. Before looking at an example, you need to do a few preliminary steps to enable Source Link:

1. Open **Tools** > **Options** > **Debugging** and disable the **Enable Just My Code** option, so that Visual Studio can search for symbols outside of the solution.
2. Under the **Symbols** node of the **Debugging** options, select the **Microsoft Symbol Servers** and **NuGet.org Symbol Servers** options (see Figure 40).
3. At the bottom of the same node, ensure the **Load all modules, unless excluded** option is also selected (see Figure 40).

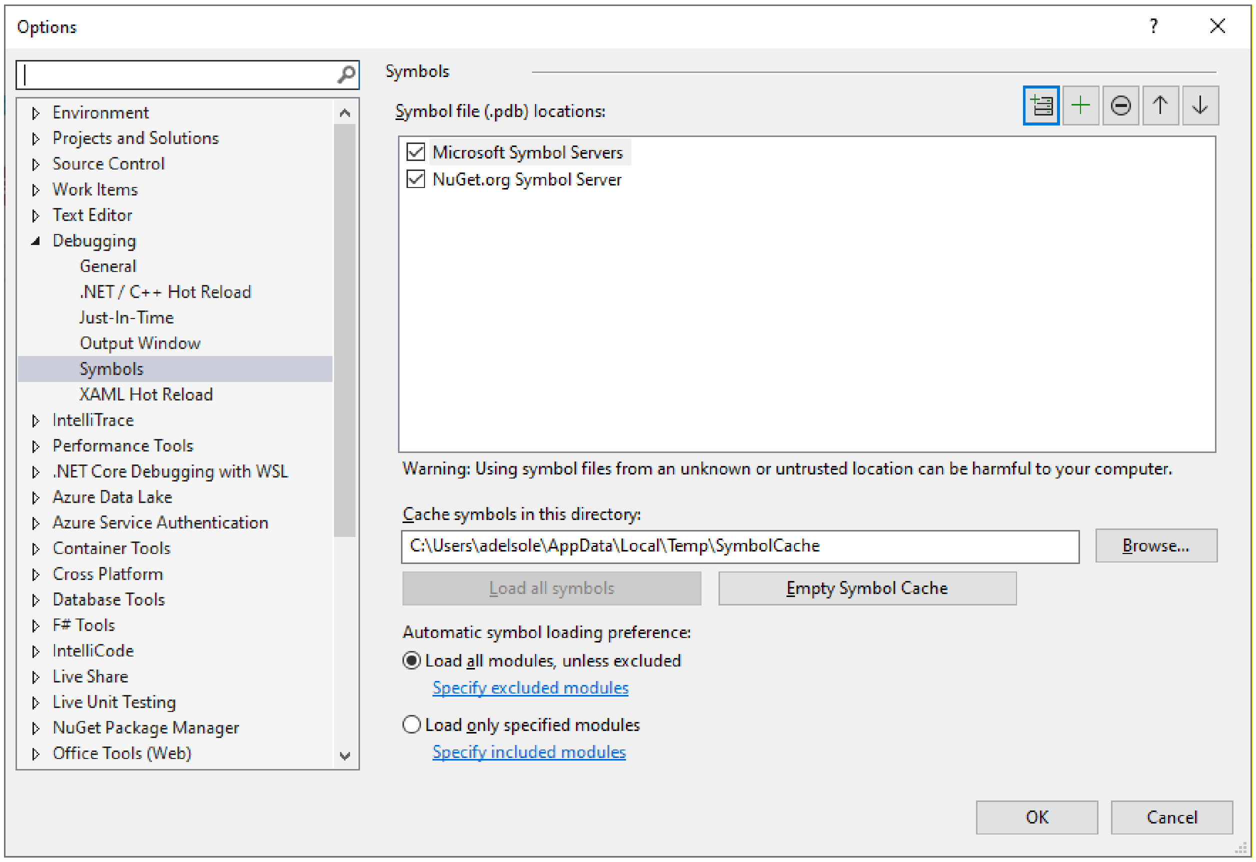


Figure 40: Setting Options to Enable Source Link

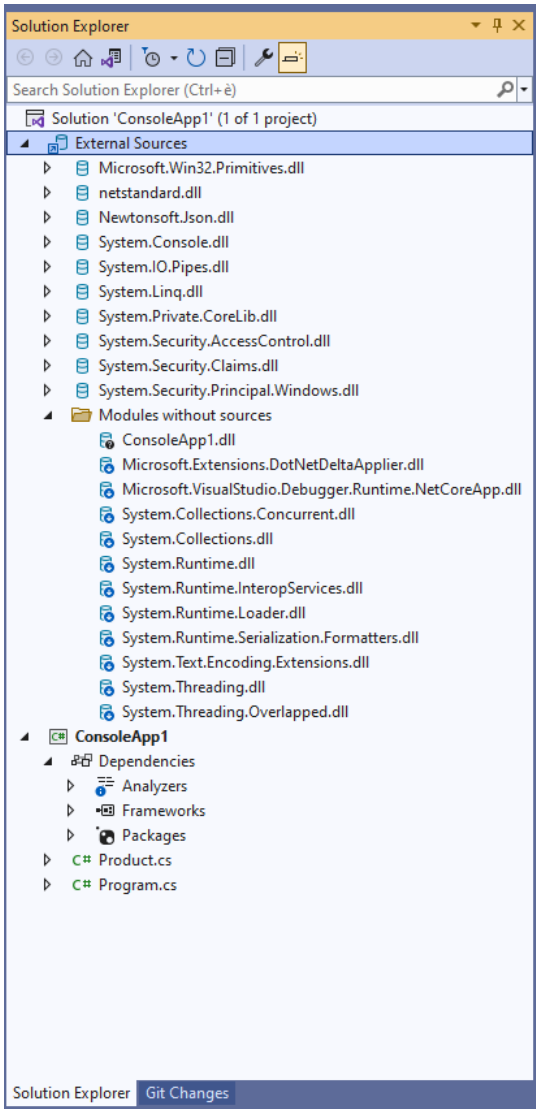
Click **OK** when done. Now suppose you want to implement JSON serialization for the **`Products`** collection discussed in the previous section about the IEnumerable Visualizer feature. First, add the **NewtonSoft.Json** NuGet package to the solution. You might know that now .NET 5 and 6 include native support for JSON, but without a doubt the NewtonSoft.Json library has been for years the *de facto* standard and millions of developers are using it, so debugging its code can be a common need. When the package has been installed, add a **`using Newtonsoft.Json;`** directive at the top of your **Program.cs** file, and then add the following line after the instance of the **`Products`** collection is created:

```
var serialized = JsonConvert.SerializeObject(products);
```

Place a breakpoint on this line and press **F5** to start debugging. At this point, you will notice that:

- A new node called **External Sources** appears in Solution Explorer.
- This node contains the list of modules that your app is referencing for which public debugging symbols were found on the server that can be debugged using Source Link.
- The **Modules without sources** folder contains the list of modules for which no public symbols were found on the server. These modules cannot be debugged with Source Link.

Figure 41 shows how the External Sources node appears.


Solution Explorer

Search Solution Explorer (Ctrl+è)

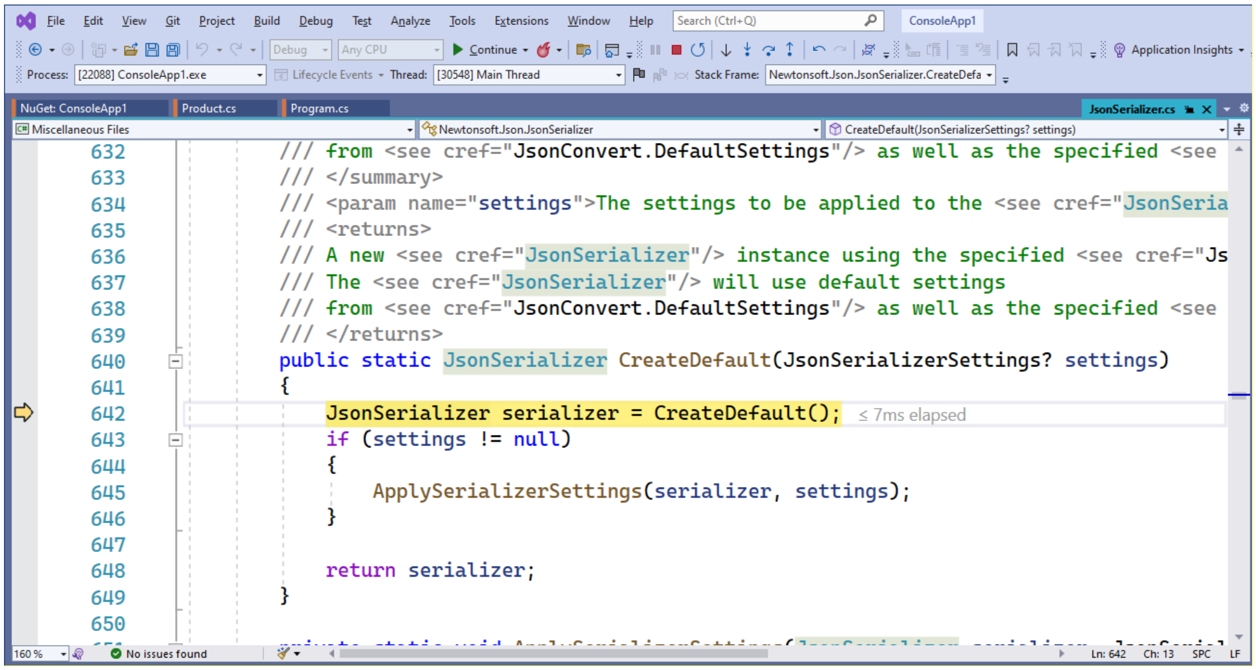
```
Solution 'ConsoleApp1' (1 of 1 project)
  External Sources
    Microsoft.Win32.Primitives.dll
    netstandard.dll
    Newtonsoft.Json.dll
    System.Console.dll
    System.IO.Pipes.dll
    System.Linq.dll
    System.Private.CoreLib.dll
    System.Security.AccessControl.dll
    System.Security.Claims.dll
    System.Security.Principal.Windows.dll
    Modules without sources
      ConsoleApp1.dll
      Microsoft.Extensions.DotNetDeltaApplier.dll
      Microsoft.VisualStudio.Debugger.Runtime.NetCoreApp.dll
      System.Collections.Concurrent.dll
      System.Collections.dll
      System.Runtime.dll
      System.Runtime.InteropServices.dll
      System.Runtime.Loader.dll
      System.Runtime.Serialization.Formatters.dll
      System.Text.Encoding.Extensions.dll
      System.Threading.dll
      System.Threading.Overlapped.dll
  ConsoleApp1
    Dependencies
      Analyzers
      Frameworks
      Packages
    Product.cs
    Program.cs
```

Figure 41: The External Sources Tool Window Displays External Modules

You will notice that the debugger takes more time to start because it needs to load symbols and reference modules. In general, it will take more time only the first time you use Source Link.

Note: Modules are incrementally loaded when a reference to them is required, so you will see the list of modules grow in the External Sources window as modules are referenced. This allows for minimizing the impact on the debugger performance when Source Link is enabled.

When symbols have been loaded, the debugger starts running the code under its control and it will reach the breakpoint you set up previously. Your goal is debugging the **`SerializeObject`** method from the **`Newtonsoft.Json.JsonConvert`** class, so press **F11** to step into the code. The availability of debugging symbols is not the only prerequisite to use Source Link, but also the availability of source code on a Git repository, so Visual Studio will search for and download the appropriate source files for debugging. You will see the download progress on the status bar. In this particular case, Visual Studio downloads the JsonSerializer.cs file from the NewtonSoft.Json repository, and the code editor will move to the appropriate line of code for debugging, as you can see in Figure 42.



```
/// from <see cref="JsonConvert.DefaultSettings"/> as well as the specified <see
/// </summary>
/// <param name="settings">The settings to be applied to the <see cref="JsonSeria
/// <returns>
/// A new <see cref="JsonSerializer"/> instance using the specified <see cref="Js
/// The <see cref="JsonSerializer"/> will use default settings
/// from <see cref="JsonConvert.DefaultSettings"/> as well as the specified <see
/// </returns>
public static JsonSerializer CreateDefault(JsonSerializerSettings? settings)
{
    JsonSerializer serializer = CreateDefault();
    if (settings != null)
    {
        ApplySerializerSettings(serializer, settings);
    }

    return serializer;
}
```

Figure 42: Stepping Through the Source Code of a Dependency

Now you can debug the code of the dependency as you would your own code. If the code you are debugging relies on other files, they will also be downloaded by Visual Studio, but only when they are required to minimize the impact on performance, and the debugger will be able to step through them. As you can understand, this feature is very powerful and extremely useful because it makes it possible to further investigate problems, exceptions, and variable values up to the root.

Easier live debugging with Hot Reload

Visual Studio 2022 enhances Hot Reload, a feature initially introduced with later service releases of Visual Studio 2019. Hot Reload is available to .NET and native C++ applications, and it allows adding or editing code while the application is running in debug mode, and also allows applying those changes without restarting the application. You may think of Hot Reload as the natural evolution of the Edit and Continue feature, with more supported scenarios and with a completely remade code injection engine. You can Hot Reload an application by clicking the **Hot Reload** button on the toolbar (see Figure 43).

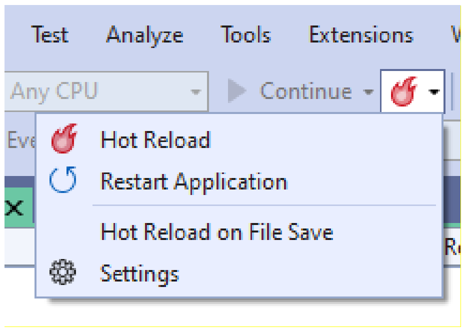


Figure 43: The Hot Reload Commands

If you do not see this button, you might need to enable Hot Reload. In the **Options** dialog, locate the **Debugging** > **.NET/C++ Hot Reload** node and make sure that the **Enable Hot Reload and Edit and Continue When Debugging** option is enabled. See the "Hot Reload settings" section later in this chapter for more details. The feature works silently, so you will not even notice that something has changed behind the scenes. Concerning .NET, Hot Reload applies to console, UWP, WPF, ASP.NET, Windows Forms, and MAUI projects. Xamarin.Forms is not supported in terms of C# code, but you can still use the XAML Hot Reload feature.

Tip: .NET 6 also allows for using Hot Reload without the debugger attached.

An example will help clarify how Hot Reload works. Create a blank WPF project that targets .NET 6. When ready, add the following simple button to the root **`Grid`** layout:

```
<Button x:Name="MainButton" Content="Click me!" Width="200" Height="40"
 Click="MainButton_Click"/>
```

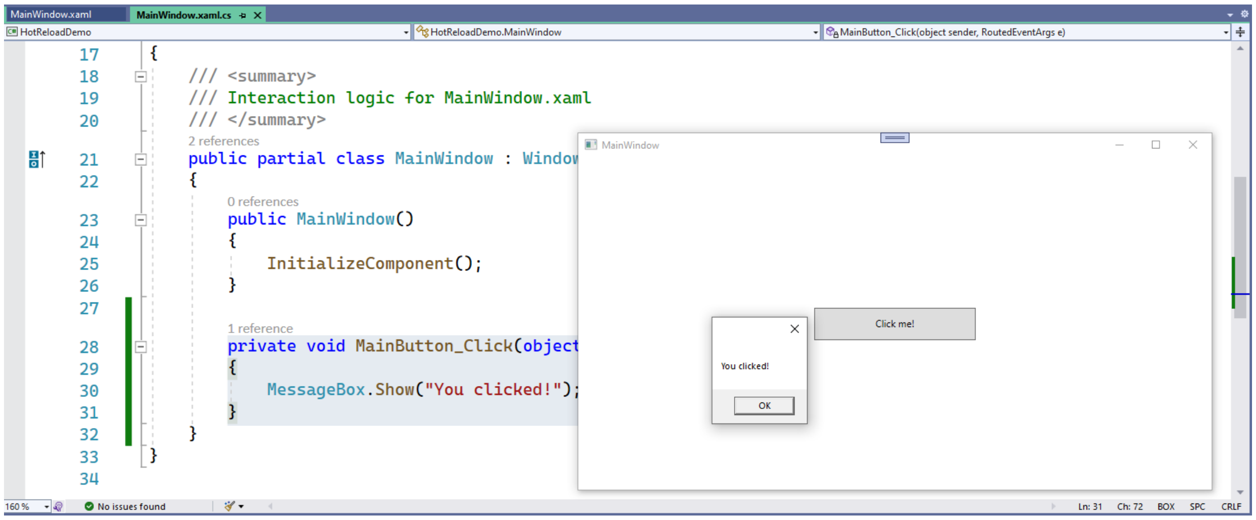
Make sure you add an empty **`Click`** event handler for the button as follows:

```
private void MainButton_Click(object sender, RoutedEventArgs e)
{
}
```

Now run the application in debug mode by pressing F5. As you can expect, clicking the button has no effect because the event handler is empty. With the application running, change the event handler as follows:

```
private void MainButton_Click(object sender, RoutedEventArgs e)

{

    MessageBox.Show("You clicked!");

}
```

If you now click the Hot Reload button in the toolbar, your edits will be applied without restarting the application, which means that you can click the button and see the message box appear (see Figure 44). A faster shortcut to enabling Hot Reload is described in the next paragraphs about Hot Reload settings.



```
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    2 references
    public partial class MainWindow : Windo
    {
        0 references
        public MainWindow()
        {
            InitializeComponent();
        }

        1 reference
        private void MainButton_Click(object
        {
            MessageBox.Show("You clicked!");
        }
    }
}
```

Figure 44: Code Changes After Hot Reload

Hot Reload is a good feature that improves productivity when debugging; however, it cannot be used in every situation. For example, an event handler is a perfect place because it gives enough time to the debugger to inject code and make it discoverable by the application. There are other specific circumstances in which Hot Reload can be used, as you will discover in the next section.

Scenarios supported by Hot Reload

At this writing, Hot Reload works with the following code edit types:

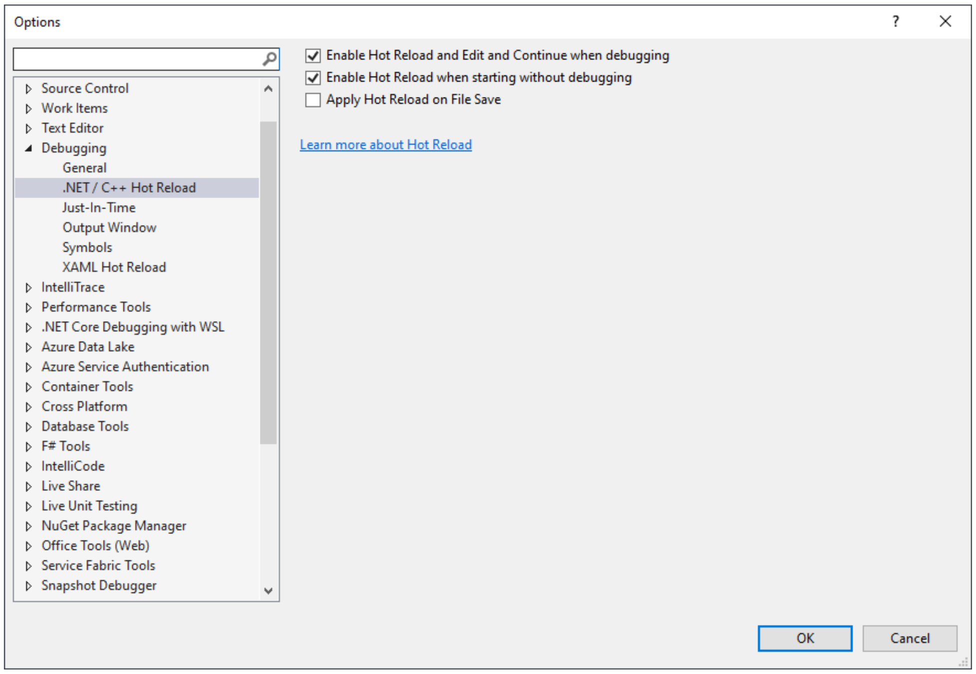
- Adding, editing, or removing custom **`Attribute`** objects.
- Adding or updating **`record struct`** blocks (available in C# 10).
- Adding or updating **`#line`** directives.
- Editing **`Switch`** expressions.
- Editing files with **`#line`** directives, including changes to the directive itself.
- Editing top-level statements (see Chapter 3, "Support for .NET 6" for information about top-level statements).
- Editing code that uses any of the new C# 10 features, such as global **`using`** directives, file-scoped namespaces, improved lambdas, and parameterless **`struct`** constructors.
- Renaming parameters of lambda expressions.
- Renaming parameters of existing methods.

However, Hot Reload is being continuously improved, so it is reasonable to expect support for additional coding scenarios in the next VS releases. Make sure you read the Visual Studio release notes when a new update is published so you can be aware of what's new with Hot Reload too.

Tip: If you make changes that are not supported and you try to Hot Reload the application, Visual Studio will show a dialog informing you that a full application restart is required, providing an option to do so.

Hot Reload settings

Hot Reload settings can be controlled in the Options dialog, more specifically by selecting the **Debugging** > **.NET/C++ Hot Reload** node (see Figure 45).



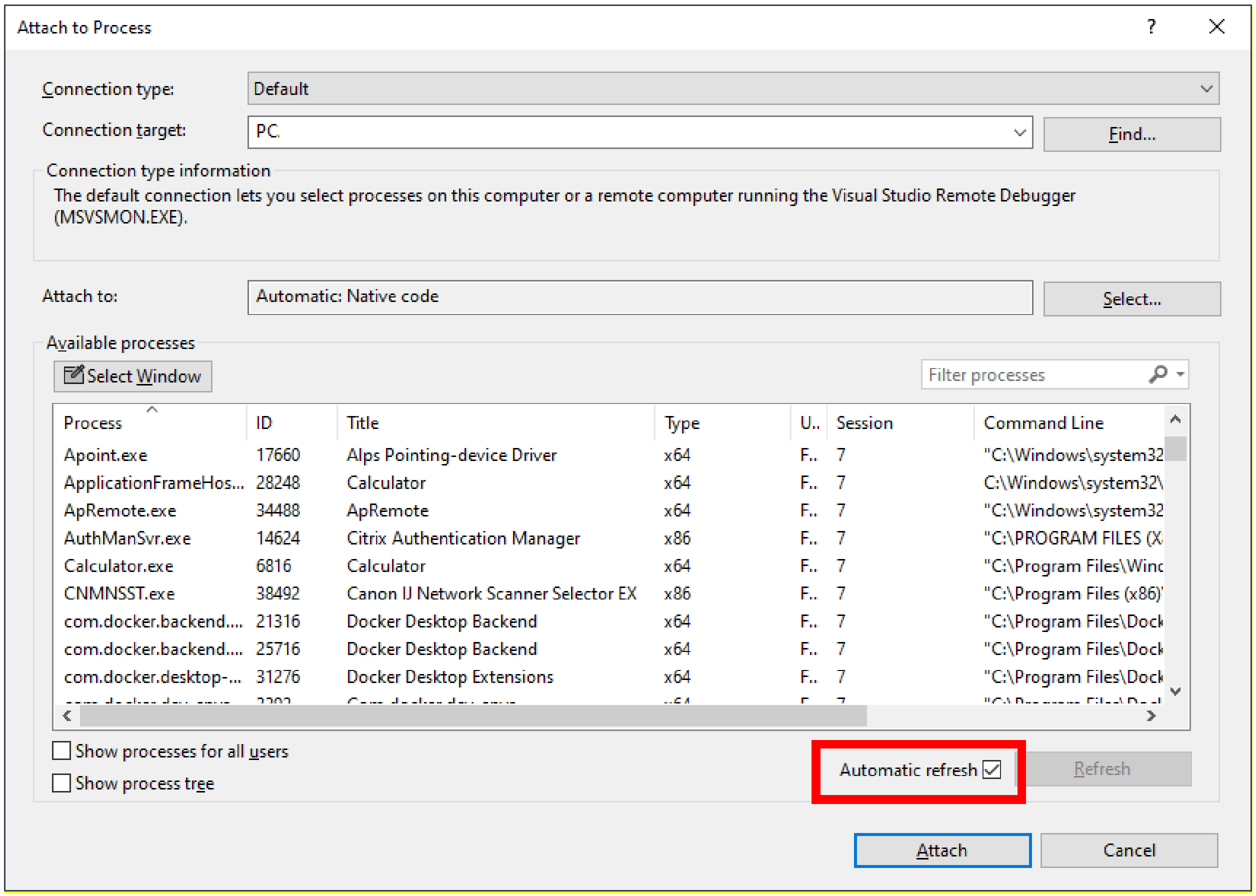
[✓] Enable Hot Reload and Edit and Continue when debugging
[✓] Enable Hot Reload when starting without debugging
[ ] Apply Hot Reload on File Save

Figure 45: Hot Reload Settings

From here you can enable and disable Hot Reload, and you can make Visual Studio invoke Hot Reload when you save files after making changes. This is accomplished by enabling the **Apply Hot Reload on File Save** option, or by selecting the **Hot Reload on File Save** command in the Hot Reload menu (see Figure 43). When this option is enabled, you can make changes to your code, save your file, and Hot Reload will be automatically invoked for you.

Improvements to the Attach to Process dialog

As you know, the Visual Studio debugger can be attached to external processes via the Attach to Process dialog, which you enable from the Debug menu. In VS 2022, there is a new option called **Automatic refresh**, which automatically refreshes the list of processes. Figure 46 shows how it appears.


Attach to Process

Connection type: Default

Connection target: PC

Connection type information

The default connection lets you select processes on this computer or a remote computer running the Visual Studio Remote Debugger (MSVSMON.EXE).

Attach to: Automatic: Native code

Available processes

Process	ID	Title	Type	U..	Session	Command Line
Apoint.exe	17660	Alps Pointing-device Driver	x64	F..	7	"C:\Windows\system32
ApplicationFrameHos...	28248	Calculator	x64	F..	7	C:\Windows\system32\
ApRemote.exe	34488	ApRemote	x64	F..	7	"C:\Windows\system32
AuthManSvr.exe	14624	Citrix Authentication Manager	x86	F..	7	"C:\PROGRAM FILES (X
Calculator.exe	6816	Calculator	x64	F..	7	"C:\Program Files\Winc
CNMNSST.exe	38492	Canon IJ Network Scanner Selector EX	x86	F..	7	"C:\Program Files (x86)'
com.docker.backend....	21316	Docker Desktop Backend	x64	F..	7	"C:\Program Files\Dock
com.docker.backend....	25716	Docker Desktop Backend	x64	F..	7	"C:\Program Files\Dock
com.docker.desktop-...	31276	Docker Desktop Extensions	x64	F..	7	"C:\Program Files\Dock

Show processes for all users: ☐

Show process tree: ☐

Automatic refresh ✓

Figure 46: The New Automatic Refresh Option in the Attach to Process Dialog

Chapter summary

Debugging is such a crucial activity in the application development lifecycle that Microsoft is continuously improving and adding productivity tools to the debugging experience in Visual Studio. VS 2022 offers tools that you can use in the code editor, such as temporary breakpoints, dependent breakpoints, and breakpoint drag-and-drop; and tools like Hot Reload that speed up the way you make changes to your code while debugging. The new and updated features you have seen in this chapter generally apply to C# and C++ code, but there are additional productivity features that are specifically added to the XAML editor, as you will discover in the next chapter.

Chapter 5 XAML Experience Improvements

XAML, the *eXtensible Application Markup Language*, is used to design user interfaces with a declarative mode in several development technologies, such as WPF, UWP, WinUI, and Xamarin.Forms. For this reason, the XAML code editor is always an important area of investment for Microsoft, and Visual Studio 2022 introduces interesting new features that are globally available to all XAML-based platforms, plus improvements that specifically target WPF. This chapter describes all the new features of the XAML code editor.

XAML Live Preview

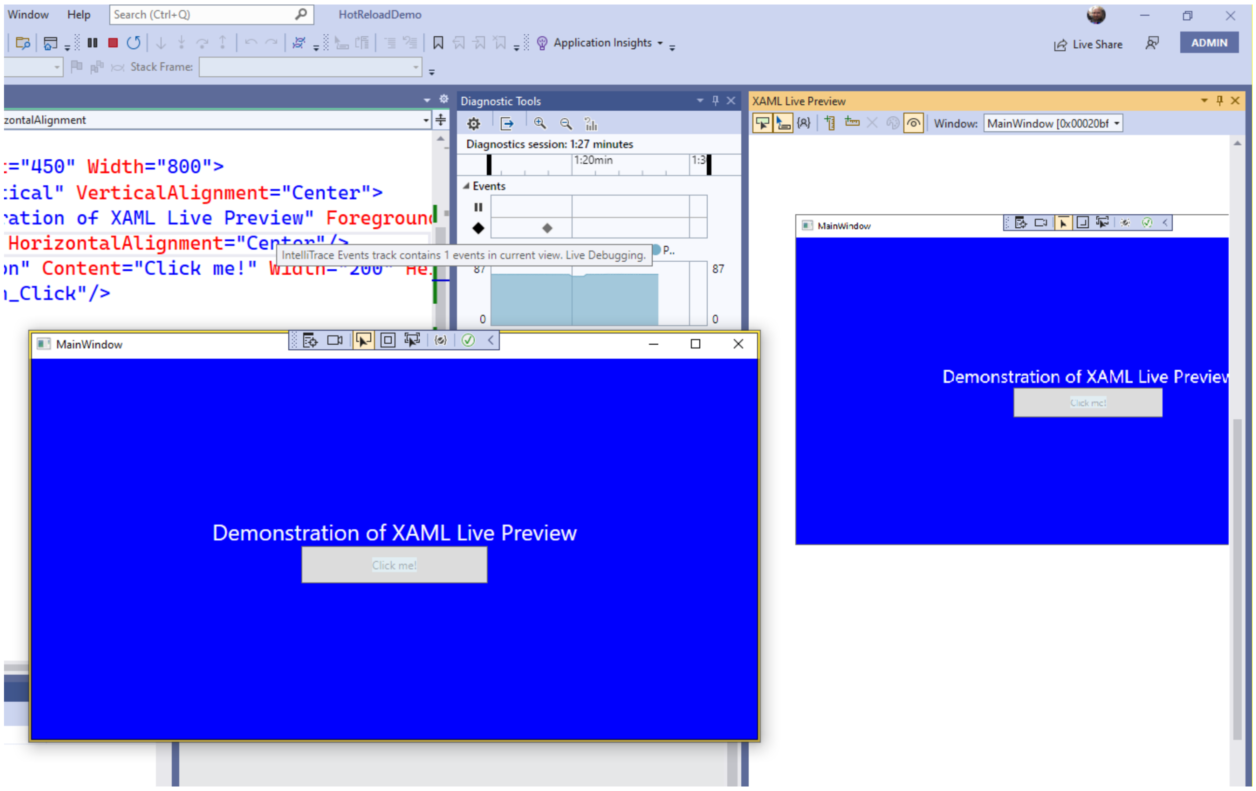
XAML Live Preview is a tool that works while debugging that allows for capturing the user interface of an application into an integrated window, making it easier to view the result of changes you apply via XAML Hot Reload. XAML Live Preview supports the following development technologies:

- Windows Presentation Foundation.
- Universal Windows Platform.
- .NET MAUI with the Android emulator.
- Xamarin.Forms 5.x with the Android emulator.
- WinUI 3.

Suppose you have a WPF application with the following, very basic code in the main window:

```
<Window x:Class="HotReloadDemo.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:local="clr-namespace:HotReloadDemo" Background="Blue"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <StackPanel Orientation="Vertical" VerticalAlignment="Center">
        <TextBlock Text="Demonstration of XAML Live Preview"
                   Foreground="White"
                   FontSize="24" HorizontalAlignment="Center"/>
        <Button x:Name="MainButton" Content="Click me!" Width="200"
                Height="40"
                Click="MainButton_Click"/>

    </StackPanel>
</Window>
```

When you start the application, you will be able to start XAML Live Preview by either clicking the **Show in XAML Live Preview** button on the debugging toolbar (with the camera icon) or by selecting **Debug** > **Windows** > **XAML Live Preview**. Figure 47 demonstrates this.



```
:="450" Width="800">
:ical" VerticalAlignment="Center">
ation of XAML Live Preview" Foreground
 HorizontalAlignment="Center"/>
n" Content="Click me!" Width="200" He
_Click"/>
```

Diagnostics session: 1:27 minutes

IntelliTrace Events track contains 1 events in current view. Live Debugging.

Figure 47: Previewing the UI with XAML Live Preview

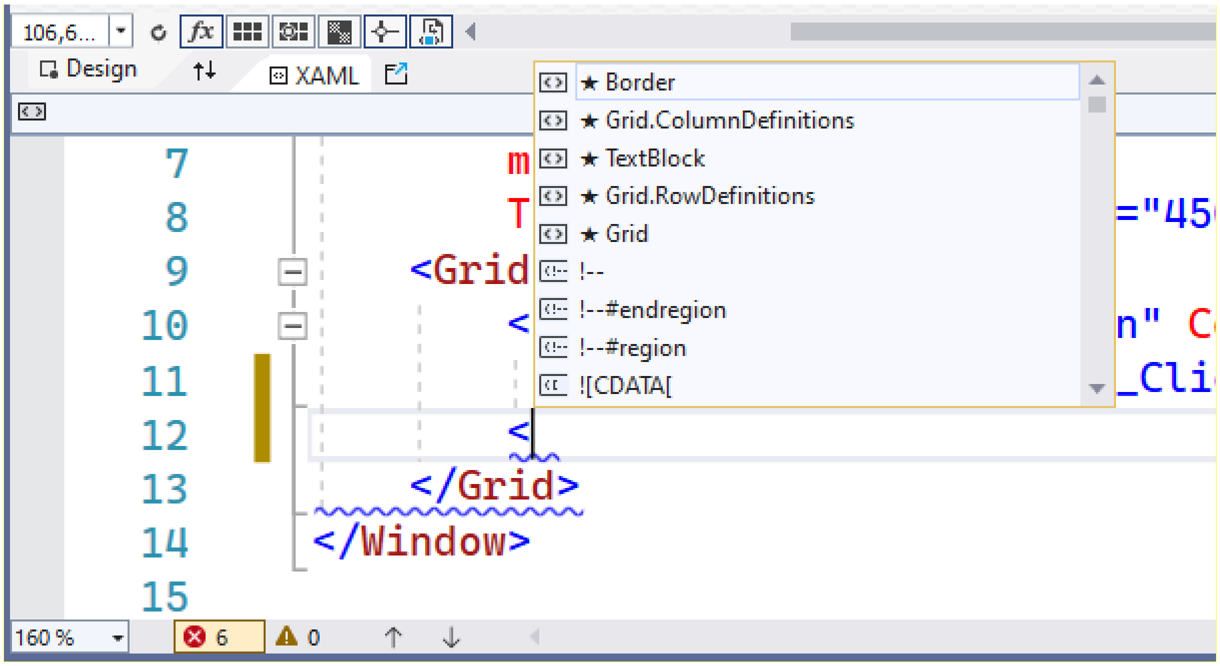
In the XAML Live Preview tool window, you can zoom the user interface (Ctrl+mouse wheel), enable rulers with the appropriate buttons on the toolbar, and get information on a visual element by first clicking the **Select** button (the leftmost one on the toolbar) and then hovering over a visual element. The **Show just my XAML** button will allow you to focus only on the markup added by you. If your application has multiple windows, you can use the **Window** dropdown to switch between windows.

Note: Visual element selection is only available with WPF and UWP. For the other platforms, XAML Live Preview only supports rulers and zooming.

This tool is a convenient addition that simplifies investigating the behavior of visual elements at runtime and makes it easier to see changes you apply via XAML Hot Reload. The new settings are discussed shortly in this chapter.

IntelliCode for XAML

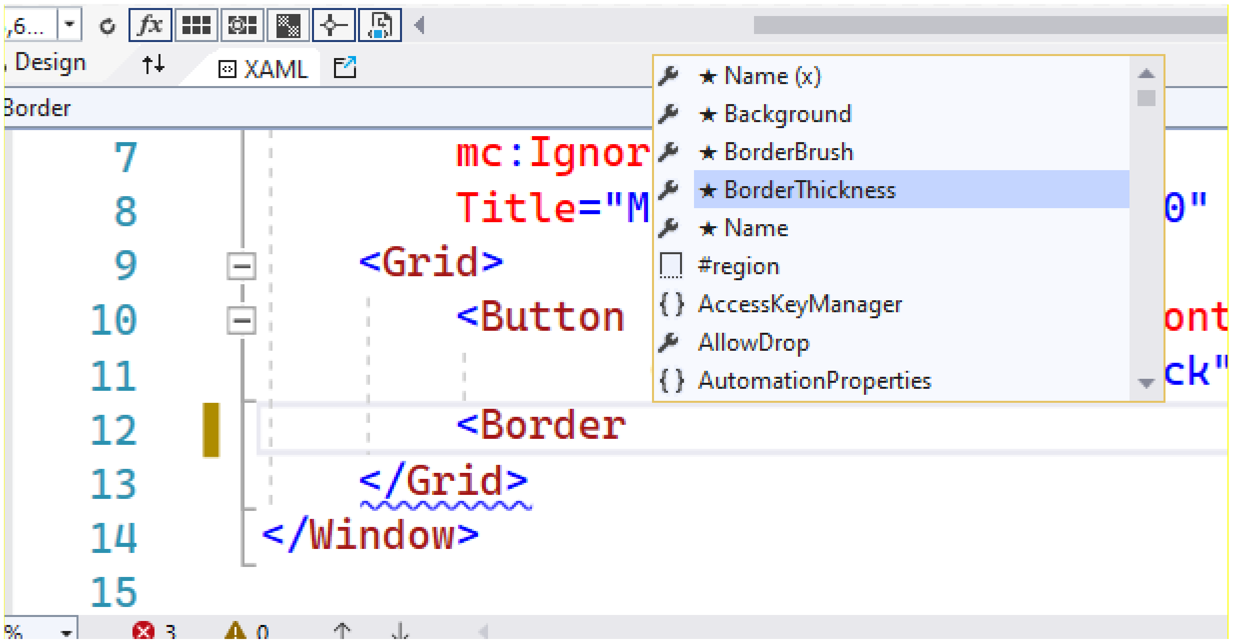
IntelliCode, the AI-powered code completion engine that now empowers IntelliSense, is now at the core of the XAML code editor. Due to the different editor purposes, the way suggestions work is not the same as for C#, but you still get suggestions based on context. More specifically, when you open an XML tag using the **<** character, IntelliSense will first show the IntelliCode suggestions that are easily recognizable because they are starred (see Figure 48).



```
                m
                T                    ="45
        <Grid
            <                        n" C
                                     _Cli
            <
        </Grid>
</Window>
```

Figure 48: IntelliCode Suggestions in the XAML Code Editor

Suggestions will continue as you type. For example, if you add a `Border`, IntelliCode suggestions will focus on the most common properties for this control, as shown in Figure 49.



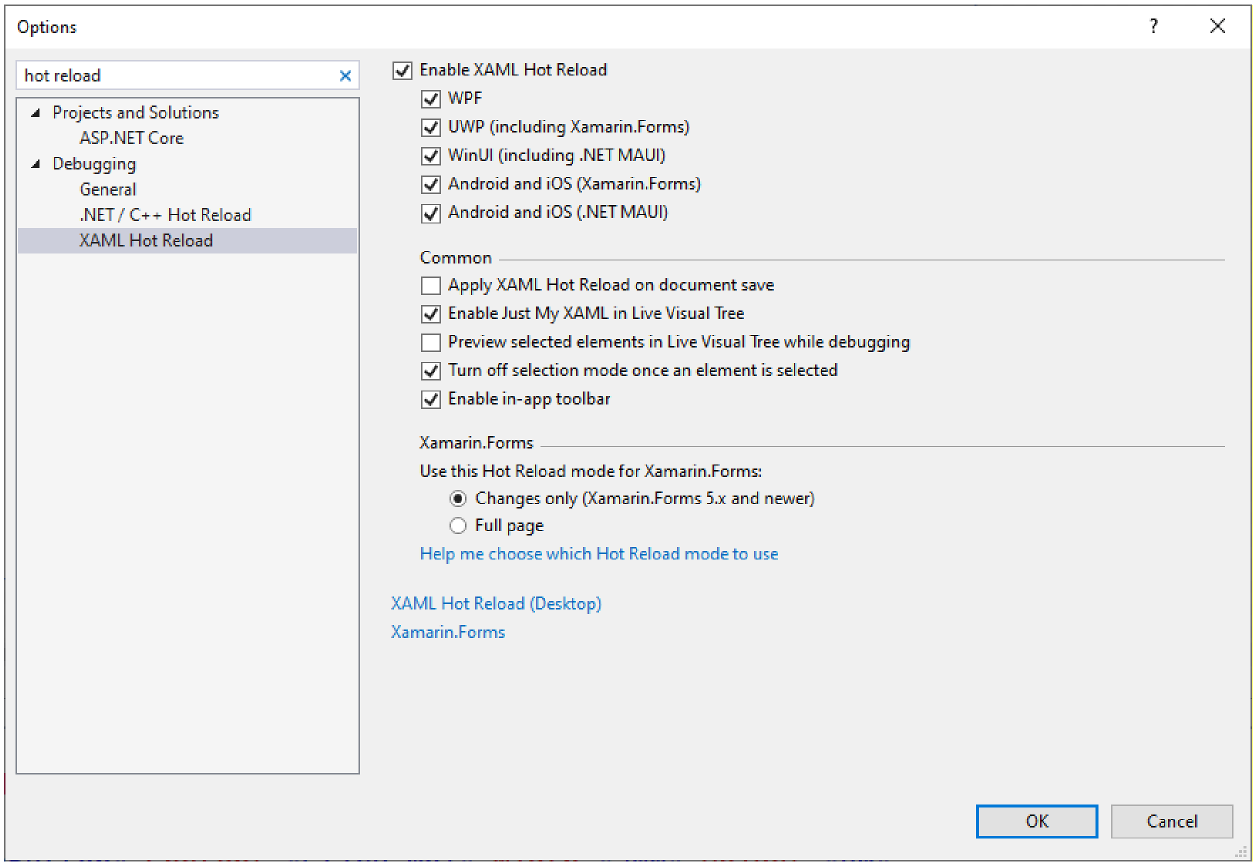
```
mc:Ignor
Title="M                                    0"
<Grid>
    <Button                                 ont
                                            ck"
        <Border
</Grid>
</Window>
```

Figure 49: IntelliCode Suggestions Continuing as You Type

The way IntelliCode suggests completions makes writing XAML markup faster, and this is a nice productivity feature, especially because developers do not typically use design tools, and they tend to type XAML markup manually.

XAML Hot Reload settings

XAML Hot Reload is a popular feature that allows for making changes to your XAML markup while debugging to see the results of your changes without the need to restart the application. This feature has been introduced in previous versions of Visual Studio, and in VS 2022 there is a new settings dialog that groups options in a different way. It can be accessed via **Tools** > **Options** > **Debugging** > **XAML Hot Reload**, and it looks like Figure 50.


Options

hot reload

- Projects and Solutions
  - ASP.NET Core
- Debugging
  - General
  - .NET / C++ Hot Reload
  - XAML Hot Reload

[✓] Enable XAML Hot Reload
- [✓] WPF
- [✓] UWP (including Xamarin.Forms)
- [✓] WinUI (including .NET MAUI)
- [✓] Android and iOS (Xamarin.Forms)
- [✓] Android and iOS (.NET MAUI)

Common
- [ ] Apply XAML Hot Reload on document save
- [✓] Enable Just My XAML in Live Visual Tree
- [ ] Preview selected elements in Live Visual Tree while debugging
- [✓] Turn off selection mode once an element is selected
- [✓] Enable in-app toolbar

Xamarin.Forms

Use this Hot Reload mode for Xamarin.Forms:
- [✓ Changes only (Xamarin.Forms 5.x and newer)]
- Full page

Help me choose which Hot Reload mode to use

XAML Hot Reload (Desktop)

Xamarin.Forms

OK Cancel

Figure 50: The New Settings Dialog for XAML Hot Reload

At the top of the dialog, you have the option to select the development platforms for which Hot Reload is enabled. In the **Common** group, you can manage settings that apply to all the supported development platforms and whose descriptions are provided in Table 3.

Table 3: XAML Hot Reload Common Settings

Setting	**Description**
Apply XAML Hot Reload on document save	Changes are applied only when the XAML file is saved to disk. It is disabled by default so that changes are applied as you type.
Enable Just My XAML in Live Visual Tree	When enabled (default), the Live Visual Tree tool window only shows visual elements added by the developer.
Preview selected elements in Live Visual Tree while debugging	When enabled, the Live Visual Tree tool window highlights visual elements as you interact with the application.

Setting	Description
Turn off selection mode once an element is selected	When enabled (default), it ensures that visual elements are not highlighted when they are focused and that only user interactions are preserved.
Enable in-app toolbar	When enabled (default), it shows the Hot Reload toolbar at the top of the application window. It does not apply to Xamarin.Forms emulators.

At the bottom of the dialog, the **Xamarin.Forms** group shows one option that is specific to Xamarin.Forms only, and that is **Use this Hot Reload mode for Xamarin.Forms**. You can choose between the two following settings:

- Changes only (Xamarin.Forms 5.x and newer).
- Full page.

The first option, which is the default, makes sure that only the changes you make to your XAML are rendered on the page, whereas the second option completely redraws the page. Obviously, applying only the changes provides the better performance, but it is only available with projects based on Xamarin.Forms 5.x. When lower versions are detected, the full-page option is applied by default.

WPF data-binding enhancements

Visual Studio 2022 introduces specific improvements to the data-binding experience in Windows Presentation Foundation, the premier technology from Microsoft to build desktop applications. This section describes such improvements, including a simplified way to add sample data with the so-called **item controls**.

Note: It is also worth mentioning that in Visual Studio 2022, the XAML designer for WPF projects based on the .NET Framework has been replaced with the XAML designer for WPF projects based on .NET Core.

Data context shortcut

For bindable properties of a control, the Properties window now shows a new cylinder icon that represents a shortcut to the Create Data Binding dialog. Figure 51 highlights this new icon.

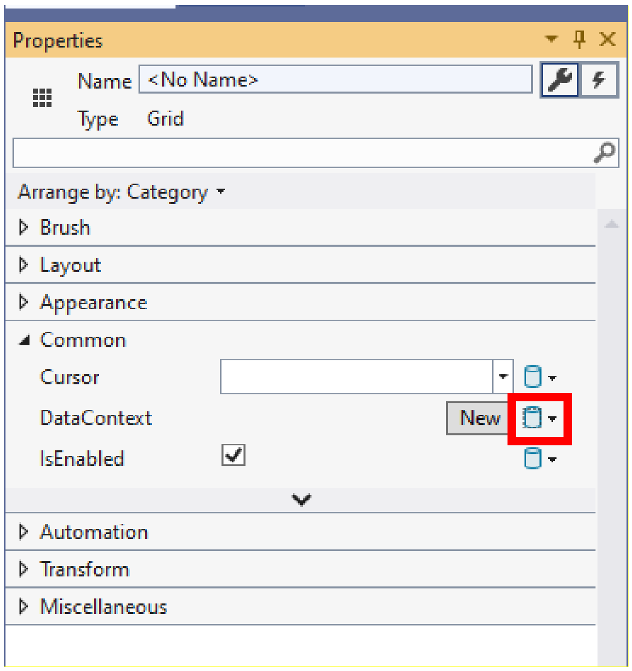


Figure 51: New Shortcut to Create Binding Expressions

Clicking this icon will launch the Create Data Binding dialog (see Figure 52) that you already know if you are familiar with WPF and its binding expressions.

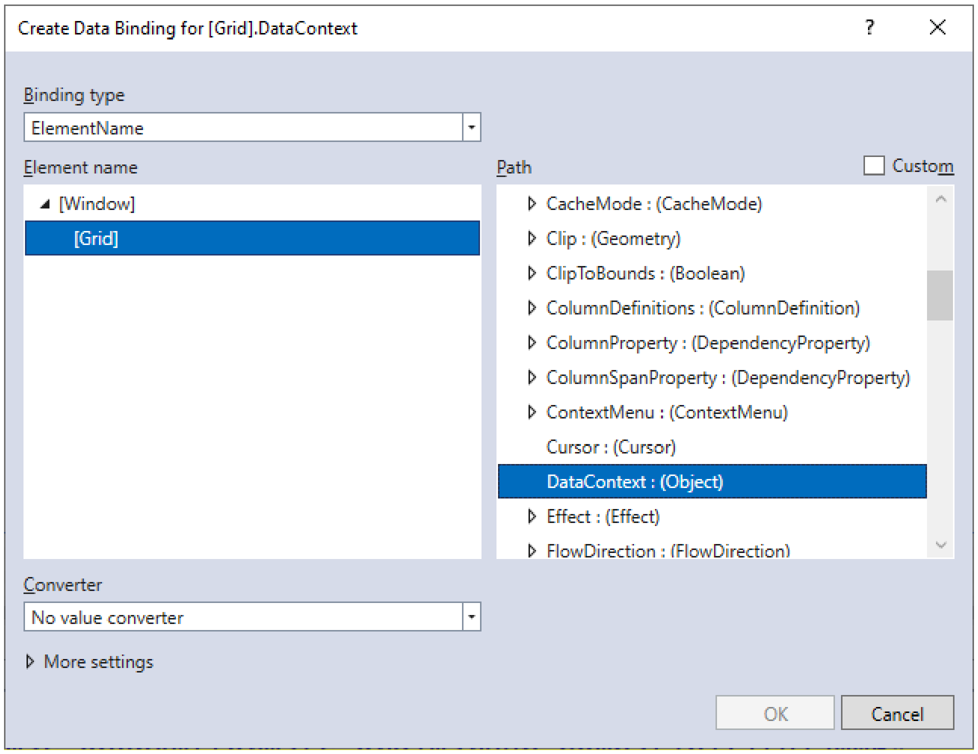


Figure 52: Creating Binding Expressions via User Interface

Obviously, a discussion about creating binding expressions in WPF would be out of scope here since the focus is on the new shortcut.

XAML Designer Quick Actions

Visual Studio 2019 introduced Quick Actions to the XAML Designer for WPF, which you access by selecting a visual element in the Designer and then clicking the light bulb icon. In Visual Studio 2022, quick actions have been extended with a new tab called **Binding**. Figure 53 shows how this looks.

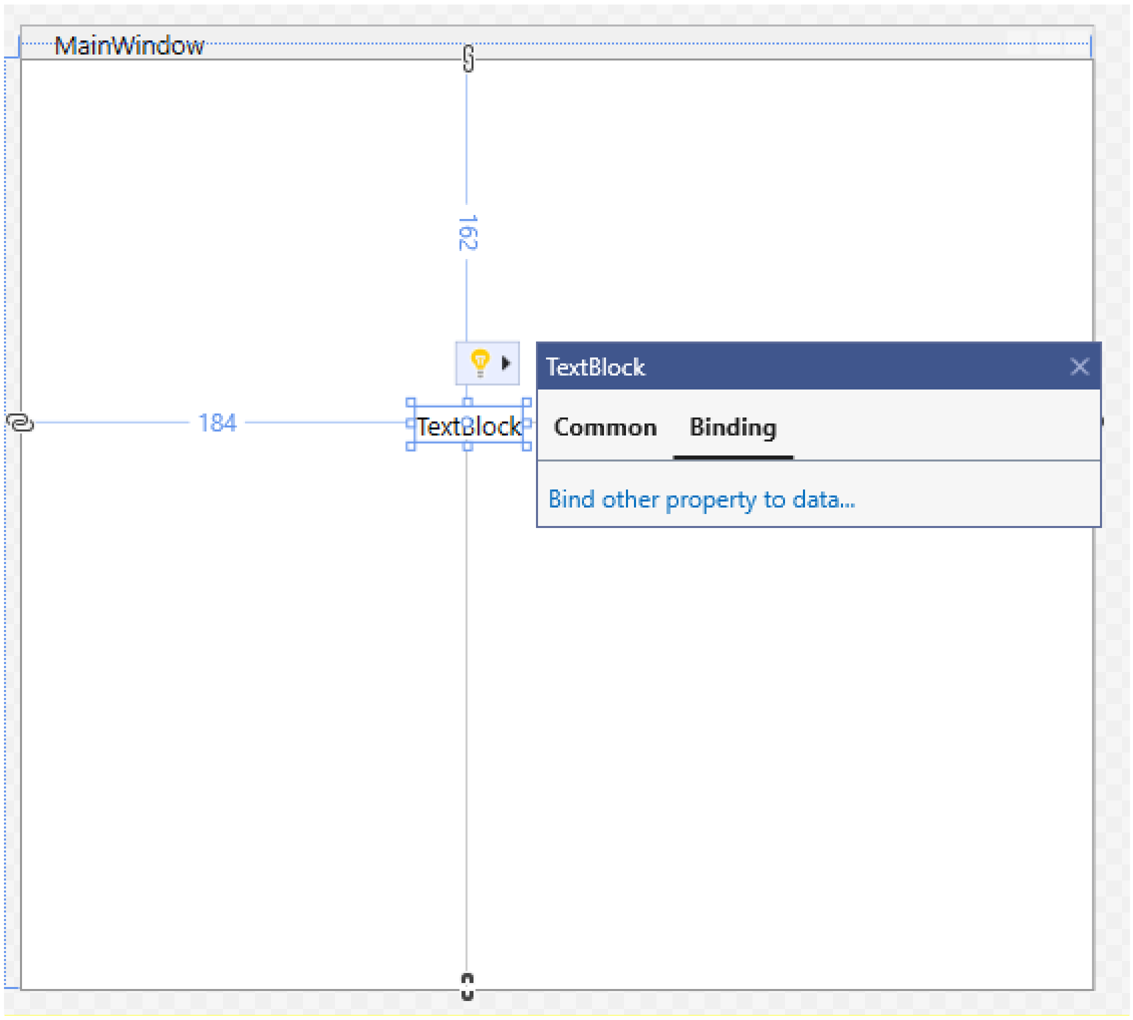


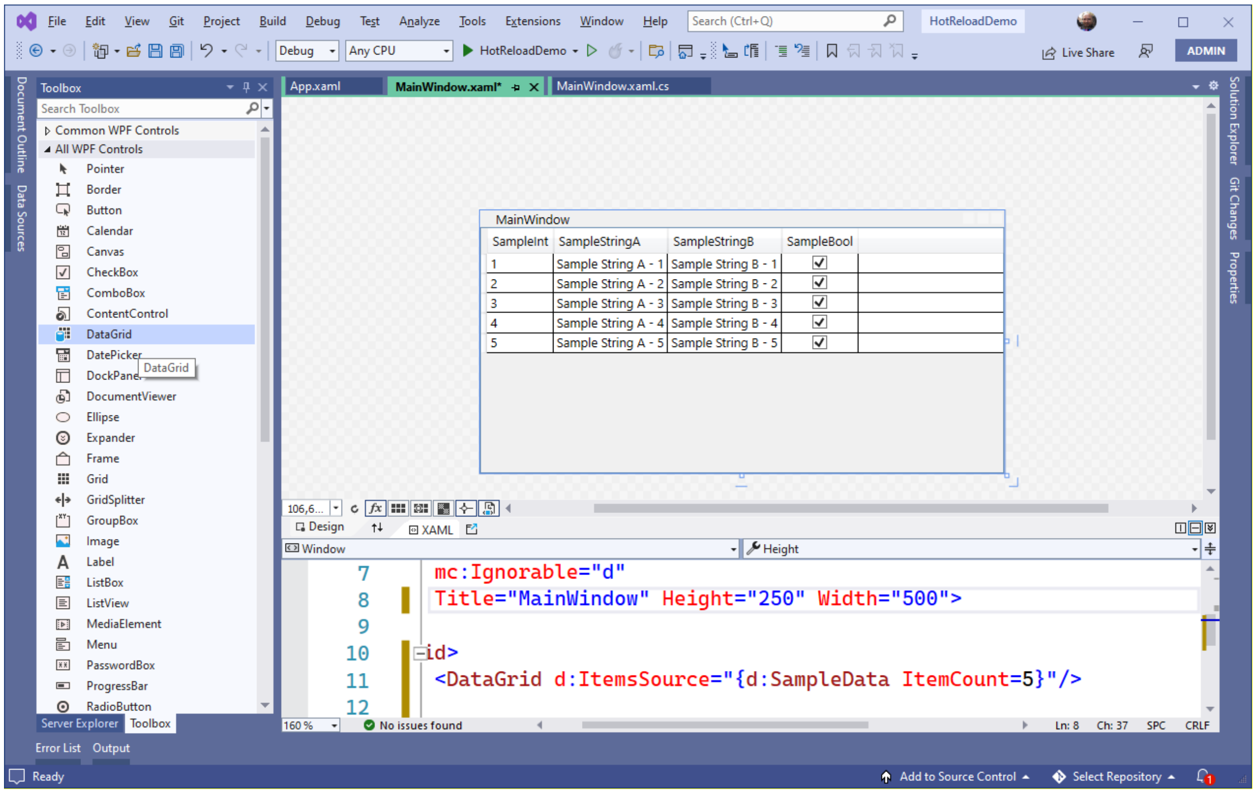
Figure 53: Accessing New Quick Actions

When you click the **Bind other property to data** shortcut, Visual Studio will open the Create Data Binding dialog (see Figure 52), where it also adds a new dropdown called **Target Property** at the top where you can select the property that needs to be data bound. The default property of the current control will be automatically selected. So, in the case of the `TextBlock`, the `Text` property will be automatically selected, and you will have an option to pick a different one.

Tip: Remember that these Quick Actions are only available in the XAML Designer and not in the code editor.

Automatic sample data

Another improvement to data binding in WPF is the automatic addition of sample data when you drag an item control from the Toolbox onto the designer surface. This feature is supported by the **`DataGrid`**, **`ListBox`**, and **`ListView`**. Figure 54 shows what happens when you drag a **`DataGrid`** onto the designer.



SampleInt	SampleStringA	SampleStringB	SampleBool
1	Sample String A - 1	Sample String B - 1	✓
2	Sample String A - 2	Sample String B - 2	✓
3	Sample String A - 3	Sample String B - 3	✓
4	Sample String A - 4	Sample String B - 4	✓
5	Sample String A - 5	Sample String B - 5	✓

```xml
        mc:Ignorable="d"
        Title="MainWindow" Height="250" Width="500">

    id>
        <DataGrid d:ItemsSource="{d:SampleData ItemCount=5}"/>
```

Figure 54: Automatic Sample Data

Tip: If you drag a control to the code editor, sample data is not added.

Sample data is exposed by the **`http://schemas.microsoft.com/expression/blend/2008`** XML namespace, represented by the **`d`** shortcut, which means that nothing is added to your source code and that it will be ignored at runtime. In fact, such an XML namespace is only intended to be used at design time, but at least you have something ready to use. You can also remove the **`d:ItemsSource`** assignment if you do not want to see sample data. The **`ItemCount`** property can be increased or decreased, depending on how many sample items you want to display.

Chapter summary

The eXtensible Application Markup Language (XAML) is at the core of the most important, modern development technologies from Microsoft, such as WPF, UWP, WinUI, and Xamarin.Forms. For this reason, Visual Studio 2022 delivers many improvements to the XAML designer and code editor. With XAML Live Preview, you can quickly investigate the XAML behavior at runtime; with a new IntelliSense engine powered by IntelliCode, you will get better suggestions as you type. Specifically for WPF, the improved support for data binding at design time will increase your productivity. But productivity is not just something that is about you individually; building applications is teamwork, and Visual Studio 2022 introduces a new feature for team collaboration on source code, as you will discover in the next chapter.

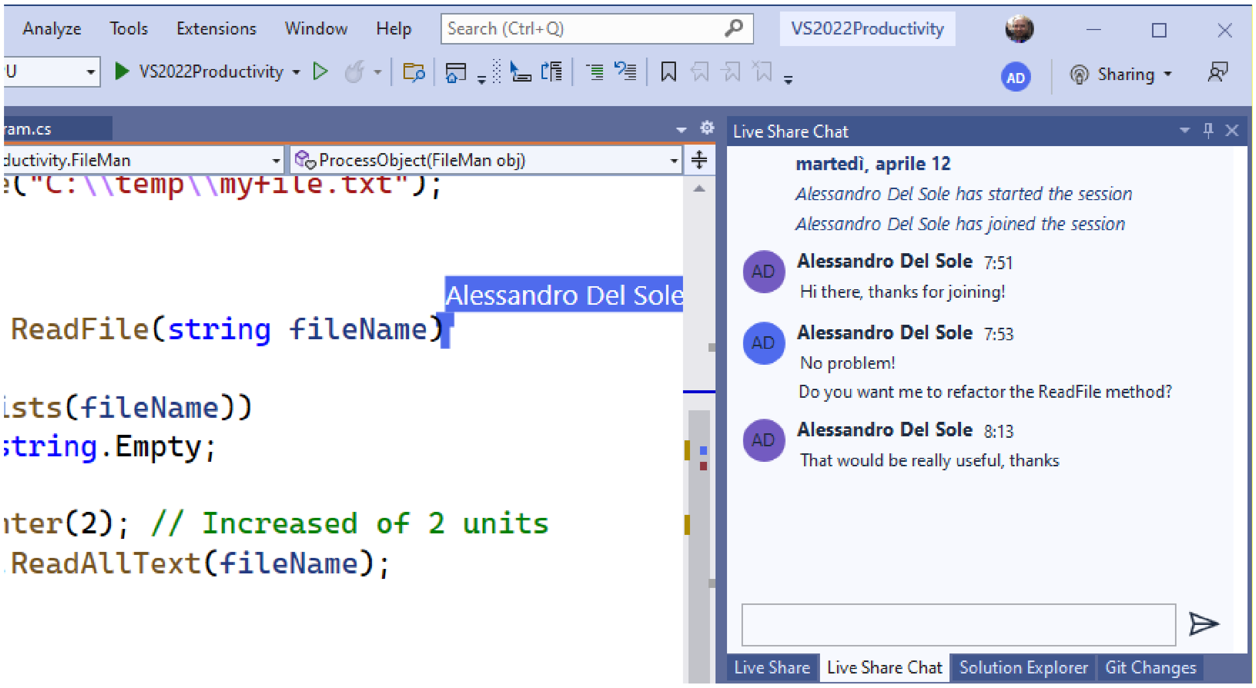
Chapter 6 Collaboration and Source Code Management

Collaboration in software development is crucial because it involves many people who may work remotely in different parts of the world and different time zones (as highlighted by the recent pandemic). Microsoft adds new collaboration tools to every major release of Visual Studio, and version 2022 is no exception. There are many additions, but certainly the most relevant ones are related to simplifying and enhancing the integrated tools to support Git repositories and collaboration on source code. In this chapter, you will see what’s new in VS 2022 from a collaboration point of view, and you will be able to elevate your Git productivity to the next level.

Note: The topics described in this chapter require that you are already familiar with Git and team collaboration in Visual Studio.

Live Share improvements

Live Share is a tool that has existed for several years in Visual Studio, and that allows for remote, real-time live collaboration over source code. It is available for Visual Studio and Visual Studio Code, and now through a new web client as well. Remote instances of Live Share can connect to a host instance (regardless of the development environment) and developers can write code and collaborate with one another. This is very useful with pair programming scenarios or when you need help from team members who are not in the same location. In Visual Studio 2022, Live Share introduces an integrated chat. Once you have started a hosting session, you can click the **Sharing** shortcut at the top right corner (see Figure 55) and then select **Live Share Chat**.



```
e("C:\\temp\\myfile.txt");

 ReadFile(string fileName)

ists(fileName))
string.Empty;

iter(2); // Increased of 2 units
 ReadAllText(fileName);
```

martedì, aprile 12

Alessandro Del Sole has started the session

Alessandro Del Sole has joined the session

Alessandro Del Sole 7:51
Hi there, thanks for joining!

Alessandro Del Sole 7:53
No problem!
Do you want me to refactor the ReadFile method?

Alessandro Del Sole 8:13
That would be really useful, thanks

Figure 55: Live Share Integrated Chat

Figure 55 shows an example of how developers can interact in the chat window during the sharing session. Opening a live chat is totally optional, but it can be useful if you need to communicate with other colleagues without using an external program.

Git tooling updates

Visual Studio 2022 introduces new support tools for working with Git repositories, and some existing tools have been modified. The first major change is that you no longer interact with source code changes using the Team Explorer tool window, which has been replaced by a new tool window called **Git Changes**. This is enabled by selecting **View** > **Git Changes**.

Tip: The Team Explorer tool window still exists, but it only shows a message saying that Git features have been moved. It explains where you can find them in VS 2022, such as in the Git menu.

Figure 56 shows an example of how the Git Changes window appears.

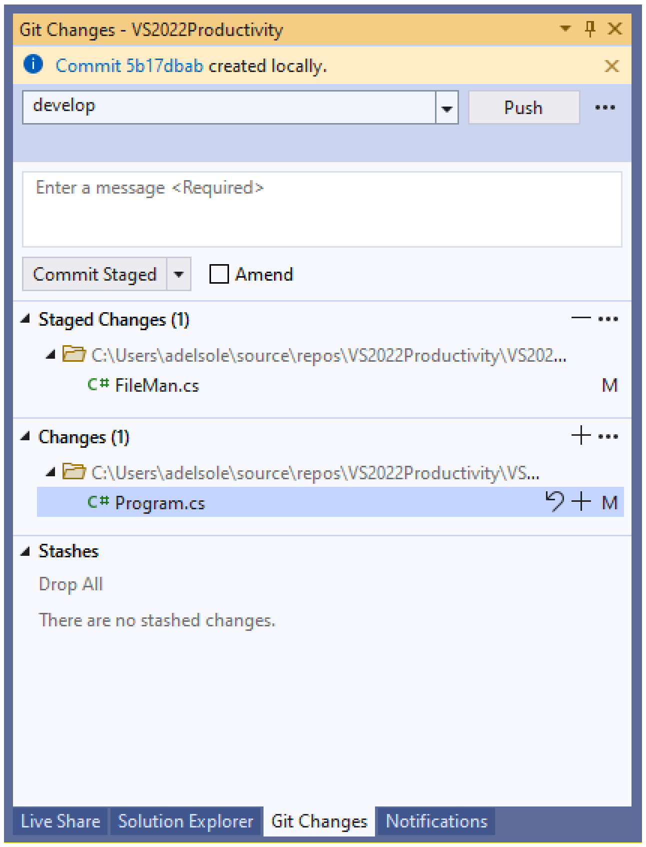

Git Changes - VS2022Productivity
Commit 5b17dbab created locally.
develop
Push
Enter a message <Required>
Commit Staged
Amend
Staged Changes (1)
C:\Users\adelsole\source\repos\VS2022Productivity\VS202...
FileMan.cs
M
Changes (1)
C:\Users\adelsole\source\repos\VS2022Productivity\VS...
Program.cs
M
Stashes
Drop All
There are no stashed changes.
Live Share
Solution Explorer
Git Changes
Notifications

Figure 56: New Git Changes Tool Window

The Git Changes tool window basically works like Team Explorer did in the past. You have a list of modified files grouped by changes, staged changes (changes that will go into the next commit), and stashed changes (changes that you keep aside but that you will not commit). You are still able to right-click files and folders to access shortcuts to common tasks, such as Undo Changes, View History, and Compare with Unmodified.

Comparing branches made easy

A new addition to the Git tooling in Visual Studio 2022 is the ability to compare branches with a few mouse clicks. If you open the list of branches in the current repository, by clicking the name of the current branch on the status bar, you can right-click a different branch and select the **Compare with Current Branch** command, as shown in Figure 57.

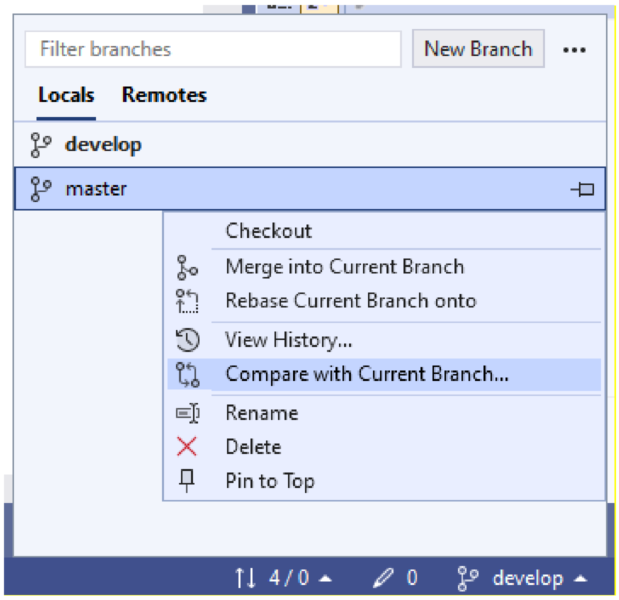
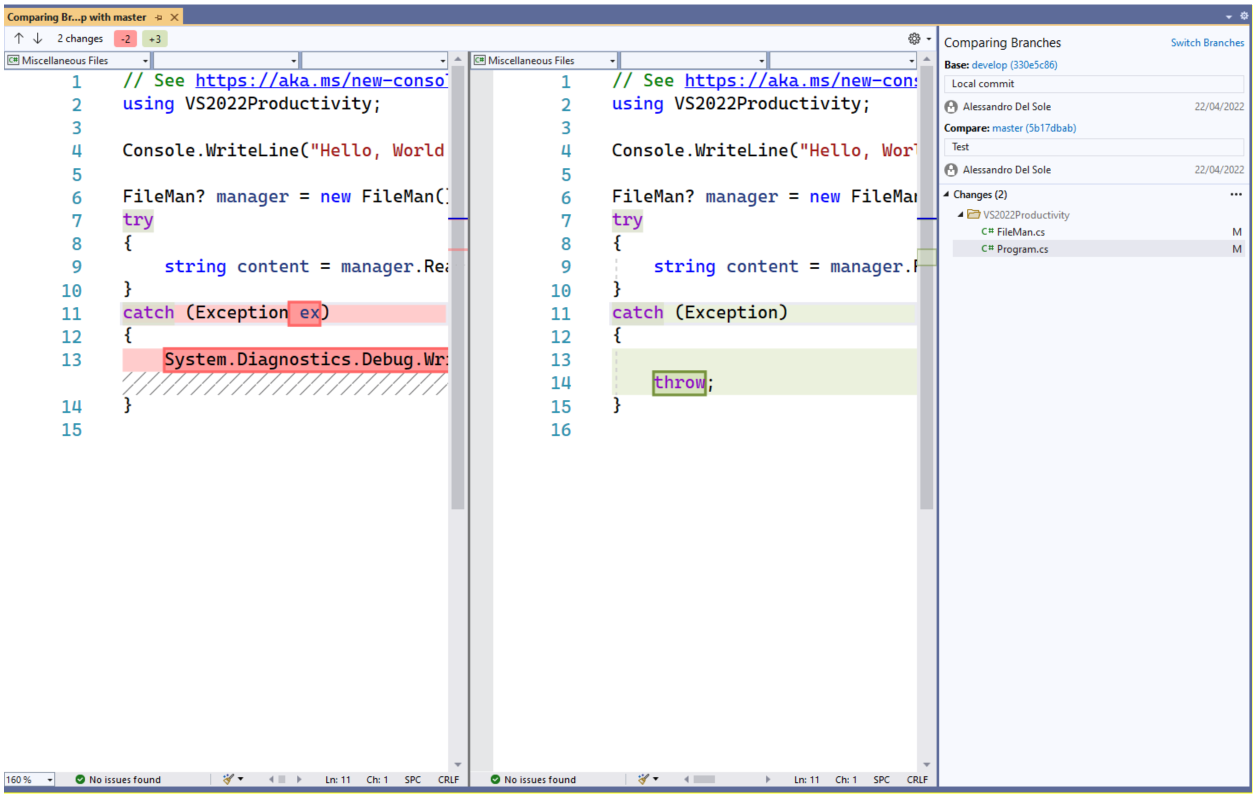


Figure 57: New Shortcut to Compare Branches

Tip: Branches can be compared only if they do not have any pending changes, so all changes must be committed before comparison.

When you click this command, Visual Studio will show files from the current branch on the left, the file difference with the branch selected for comparison at center, and the commit details with a list of modified files on the right. Figure 58 shows an example.



```
// See https://aka.ms/new-conso
using VS2022Productivity;

Console.WriteLine("Hello, World

FileMan? manager = new FileMan(
try
{
    string content = manager.Re
}
catch (Exception ex)
{
    System.Diagnostics.Debug.Wr

}
```

```
// See https://aka.ms/new-con
using VS2022Productivity;

Console.WriteLine("Hello, Wor

FileMan? manager = new FileMa
try
{
    string content = manager.
}
catch (Exception)
{

    throw;
}
```

Comparing Branches
Base: develop (330e5c86)
Local commit
Alessandro Del Sole 22/04/2022
Compare: master (5b17dbab)
Test
Alessandro Del Sole 22/04/2022
Changes (2)
VS2022Productivity
FileMan.cs M
Program.cs M

Figure 58: Branch Comparison in Visual Studio 2022

Simply click one of the files in the list to compare changes with the selected branch.

Working on specific commits

Sometimes you need to work on specific commits in a repository. For example, you might need to review a colleague's work without checking out a branch where you do not plan to contribute. In this case, you can follow these steps to check out a commit:

1. Select **View** > **Git Repository**.
2. When the Git Repository window appears, in the history of the code, right-click the commit you are interested in and then select **Checkout (--detach)** as shown in Figure 59.
3. You will need to accept Visual Studio's warning about the fact that you will work on a commit that is in a detached state before accessing the code.

Note: Put succinctly, when code is in a detached state, you point to a commit that works like a local folder, rather than to a branch. This means that you can still make changes to the code, but you will need to create and push a new branch to save and share those changes, otherwise they will be lost.

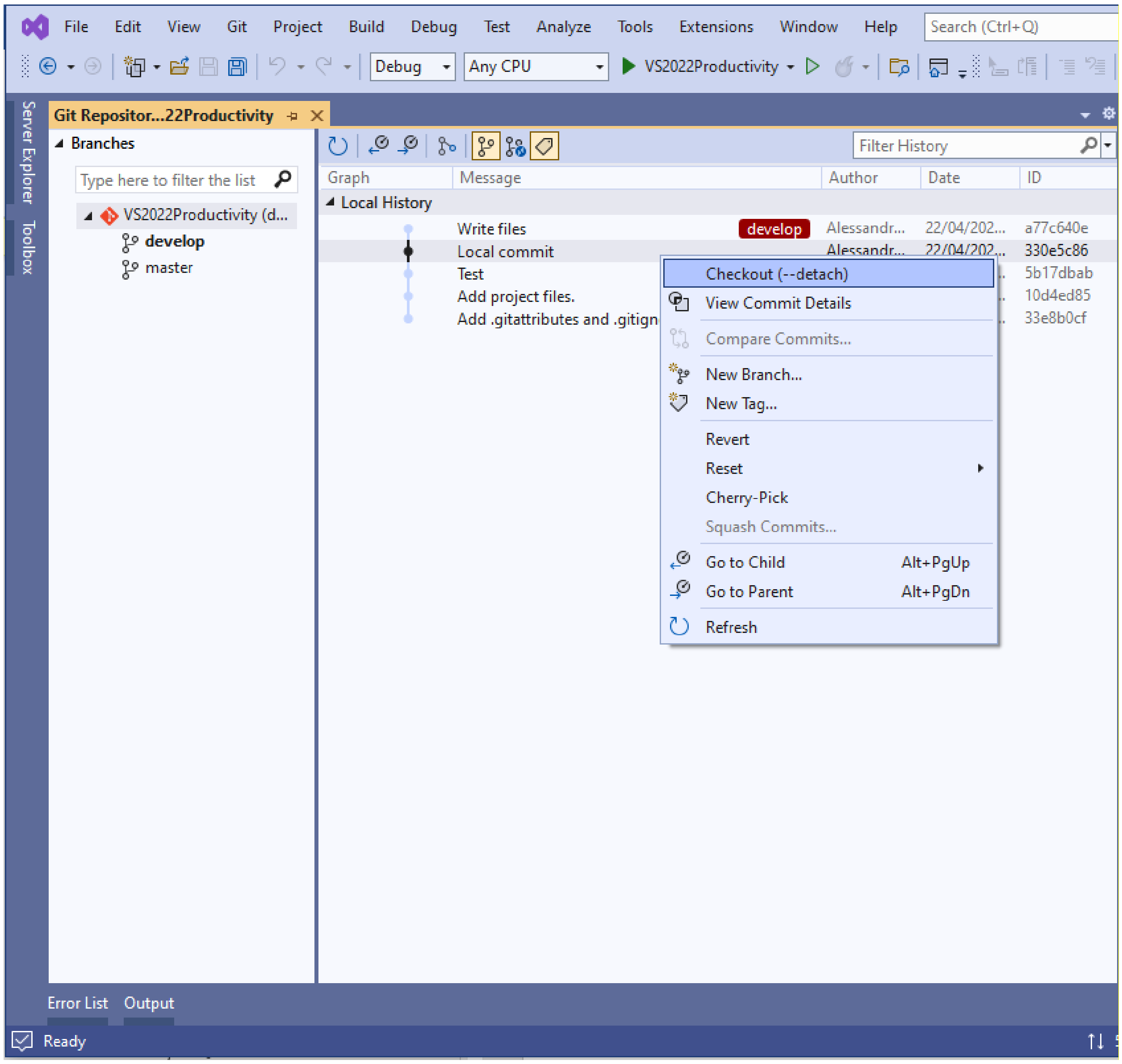
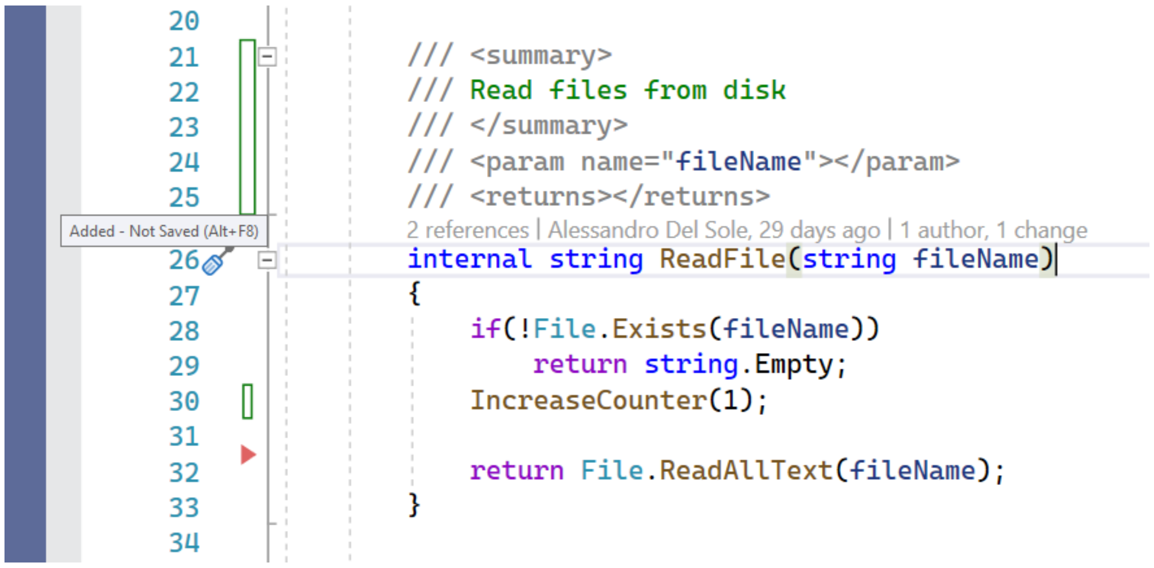


Figure 59: Checking Out Commits in a Detached State

Once code has been checked out, you will be able to run it and change it as usual (keeping in mind the comment in the previous note). This feature is very useful, especially for code reviews. Obviously, it needs some attention when you think the code needs changes or if you plan to contribute to the branch where the selected commit resides.

Colored editor margins

Visual Studio 2022 adds a new feature to the code editor with projects under source control, known as **colored editor margins**. In short, when you make changes to a file, the margin of the code editor shows a color that represents the change you made: green for new code, red for deleted code, and blue for modified code. Figure 60 shows an example where lines have been added to and removed from a method called `ReadFile`.



```
/// <summary>
/// Read files from disk
/// </summary>
/// <param name="fileName"></param>
/// <returns></returns>
2 references | Alessandro Del Sole, 29 days ago | 1 author, 1 change
internal string ReadFile(string fileName)
{
    if(!File.Exists(fileName))
        return string.Empty;
    IncreaseCounter(1);

    return File.ReadAllText(fileName);
}
```

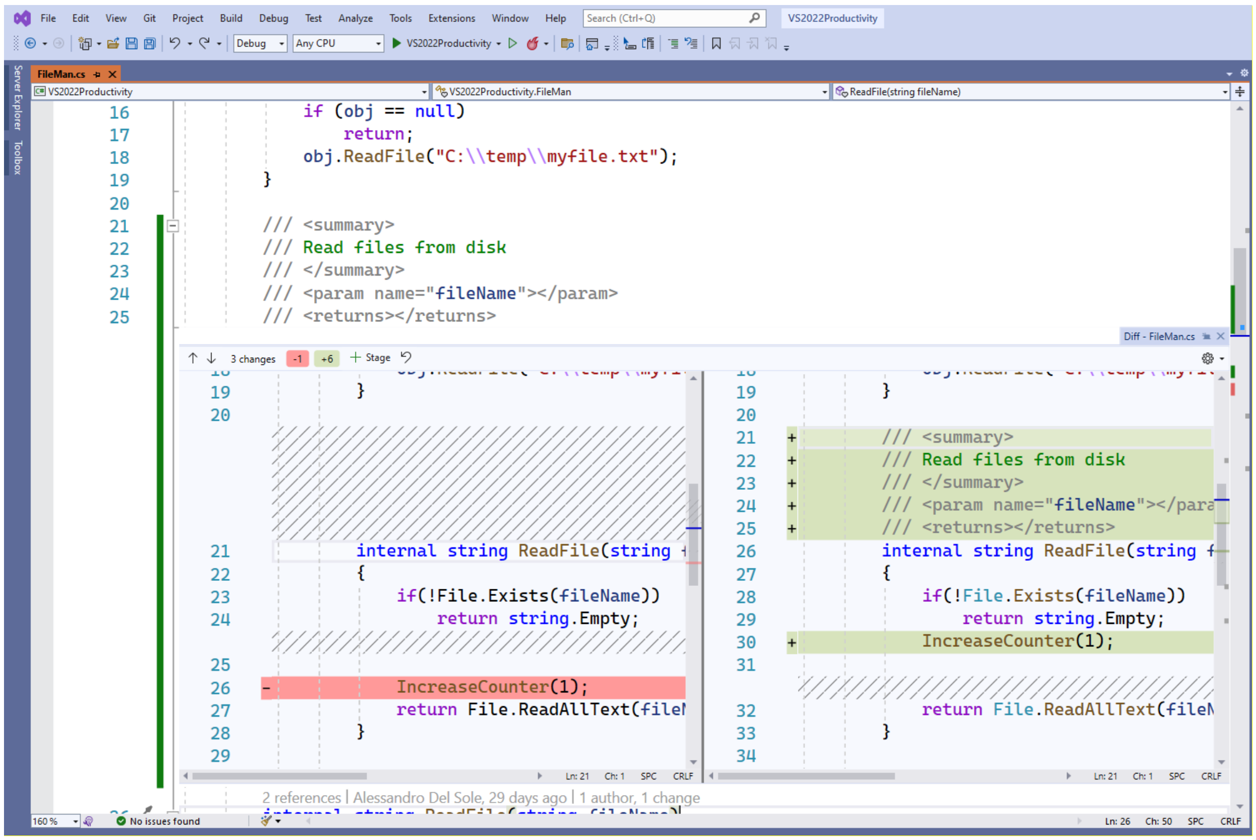
Figure 60: Colored Editor Margins

Note: Colored editor margins are still in preview at the time of this writing. You can enable this feature in the *Preview Features* ***of the*** *Options* ***dialog, selecting the*** *Enable line-staging support* ***option. This is also required for the inline staging feature discussed in the next section. You can look at Figure 47 to see where this option is.***

A red glyph indicates the point where code was removed. Green vertical lines indicate points where new code was added. If the code file has not been saved yet, such lines only show a border, whereas they are filled if the file has been saved. When you hover over a change indicator, you can see a tooltip that describes the current status—for example, **Added – Not Saved**, which you can see in Figure 60. The purpose of this feature is not just to give you a visual representation of changes in the code editor, but also to provide support for a feature called **inline staging**.

Inline staging

Inline staging is a new feature in Visual Studio 2022 that allows you to stage chunks of code files, instead of staging an entire file. This can be useful when you want to stage changes over different commits. Staging code blocks is accomplished by clicking one of the indicators in the colored editor margins. Figure 61 shows how Git changes are now shown inline.



```
        if (obj == null)
            return;
        obj.ReadFile("C:\\temp\\myfile.txt");
    }

    /// <summary>
    /// Read files from disk
    /// </summary>
    /// <param name="fileName"></param>
    /// <returns></returns>
```

```
    }

    internal string ReadFile(string f
    {
        if(!File.Exists(fileName))
            return string.Empty;

-       IncreaseCounter(1);
        return File.ReadAllText(fileN
    }
```

```
    }

+   /// <summary>
+   /// Read files from disk
+   /// </summary>
+   /// <param name="fileName"></para
+   /// <returns></returns>
    internal string ReadFile(string f
    {
        if(!File.Exists(fileName))
            return string.Empty;
+       IncreaseCounter(1);

        return File.ReadAllText(fileN
    }
```

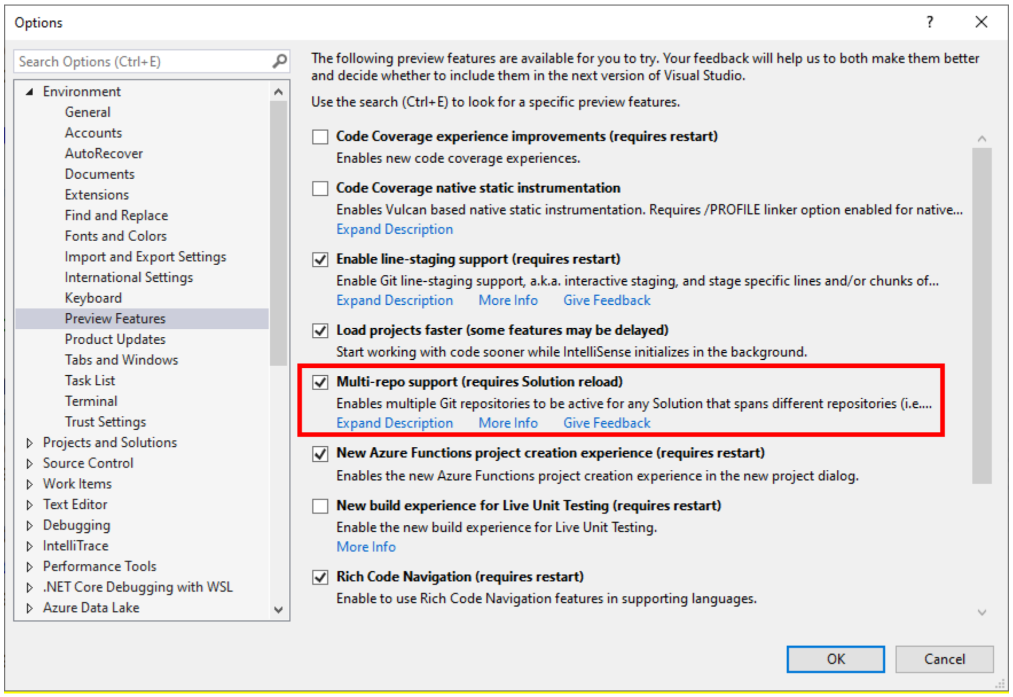
2 references | Alessandro Del Sole, 29 days ago | 1 author, 1 change

Figure 61: Inline Staging

The code editor highlights the differences before the previous and current versions of the code block. If you are satisfied with your changes, you can click the **Stage** button at the top. Once you stage changes, these will also be visible in the Git Changes tool window, under the **Staged Changes** node. From here, you will have the option to unstage edits if you change your mind.

Support for multiple Git repositories

Among others, support for multiple Git repositories in Visual Studio 2022 has been the most publicized new feature when it comes to team collaboration. With this feature, it is now possible to work on solutions made of projects that are hosted in different Git repositories on any provider (e.g., Azure DevOps, GitHub, Bitbucket, etc.) and Visual Studio will track changes accordingly. At this writing, this feature is still in preview, so it must be explicitly enabled. To do so, select **Tools** > **Options**, and in the Options window locate the **Preview Features** under the **General** node. Here you must turn on the feature called **Enable multi-repo support (requires Solution reload)**. Figure 62 shows how it appears in Visual Studio.



The following preview features are available for you to try. Your feedback will help us to both make them better and decide whether to include them in the next version of Visual Studio.

Use the search (Ctrl+E) to look for a specific preview features.

- [ ] **Code Coverage experience improvements (requires restart)**
  Enables new code coverage experiences.
- [ ] **Code Coverage native static instrumentation**
  Enables Vulcan based native static instrumentation. Requires /PROFILE linker option enabled for native...
  Expand Description
- [✓] **Enable line-staging support (requires restart)**
  Enable Git line-staging support, a.k.a. interactive staging, and stage specific lines and/or chunks of...
  Expand Description More Info Give Feedback
- [✓] **Load projects faster (some features may be delayed)**
  Start working with code sooner while IntelliSense initializes in the background.
- [✓] **Multi-repo support (requires Solution reload)**
  Enables multiple Git repositories to be active for any Solution that spans different repositories (i.e....
  Expand Description More Info Give Feedback
- [✓] **New Azure Functions project creation experience (requires restart)**
  Enables the new Azure Functions project creation experience in the new project dialog.
- [ ] **New build experience for Live Unit Testing (requires restart)**
  Enable the new build experience for Live Unit Testing.
  More Info
- [✓] **Rich Code Navigation (requires restart)**
  Enable to use Rich Code Navigation features in supporting languages.

Figure 62: Enabling Support for Multiple Git Repositories

In the next section, you will learn how to work against multiple repositories from one solution. Because it is not possible to predict which Git accounts you have, and for the sake of simplicity, you will get some hints about setting up an appropriate solution, and then the explanation will be based on a solution on my personal machine.

Note: It is not uncommon to have solutions with projects hosted in different repositories. For example, imagine you develop a component library that is hosted in one repository and app projects that reference the library while hosted in different repositories. Instead of referencing the library binaries, you can use the source code and update it without working with separate instances of Visual Studio and without the need to update the binary references.

Setting up a solution

There is no limit on the number of Git repositories that Visual Studio can handle from one solution, but two is enough for demonstration purposes. You can generally follow these steps to set up a solution:

1. Push an existing project, even if empty, to a Git repository of your choice (Azure DevOps, GitHub, Bitbucket, etc.).
2. Push another project to a Git repository. If you have multiple accounts, it is worth trying against a different provider.
3. In Visual Studio 2022, create a blank solution.
4. Add both projects that you have pushed previously to the solution.

This is enough for now. It is not necessary to push the solution file to Git, but it can be done if you wish. In the example you will see in the next paragraphs, there are two projects that have been added to a blank solution. One project is hosted in an Azure DevOps repository, and the other one is hosted in GitHub. Figure 63 shows how this solution looks in Solution Explorer. Both are personal projects of mine. Memorello is a Xamarin.Forms solution, and YouTubeChannelReader is a .NET Standard library hosted on GitHub. So, the two projects are not only in different Git repos, but also on different providers.

Tip: Should you be interested, YouTubeChannelReader is an open-source project that you can use in your .NET projects to retrieve information from a YouTube channel.

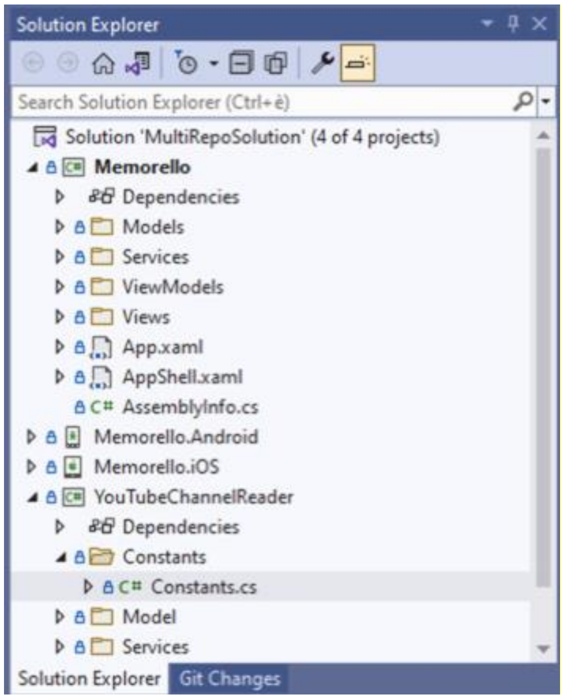


Figure 63: Solution Containing Projects Hosted in Different Repos

As you can see in Figure 63, Visual Studio shows the usual source control icons close to the file names. In the next sections, you will see how the Git tooling in VS 2022 can target multiple repositories.

Displaying and filtering repositories

You can display and filter the list of repositories involved in the current solution by clicking the number of repositories located at the bottom right of Visual Studio's status bar, as shown in Figure 64.

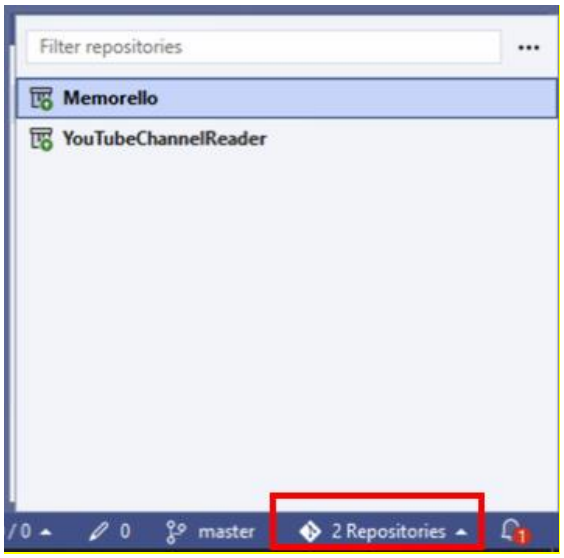


Figure 64: Displaying and Filtering the List of Repositories

If you hover over a repo's name, you will see its physical path on disk.

Handling Git changes

When you make changes to files that are under source control, such files are highlighted in the Git Changes tool window. This behavior remains exactly the same even when your solution is based on multiple repositories. Figure 65 shows an example, where you can see two files from different repositories that have been changed.

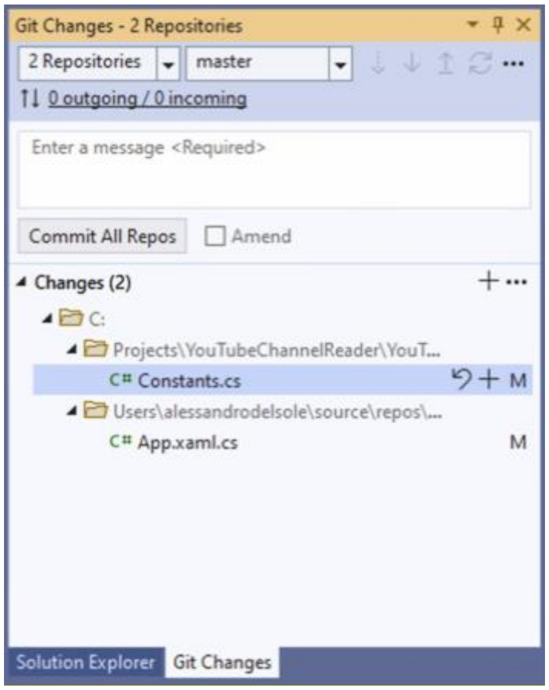

Git Changes - 2 Repositories
2 Repositories
master
0 outgoing / 0 incoming
Enter a message <Required>
Commit All Repos
Amend
Changes (2)
C:
Projects\YouTubeChannelReader\YouT...
Constants.cs
M
Users\alessandrodelsole\source\repos\...
App.xaml.cs
M
Solution Explorer
Git Changes

Figure 65: Highlighting Changes Across Repositories

At the top left corner of the window, you can see a dropdown that shows the list of involved repositories. You can filter the list of changes by selecting only one repository. At the right side of this dropdown, a second dropdown allows for showing and selecting branches (see Figure 66).

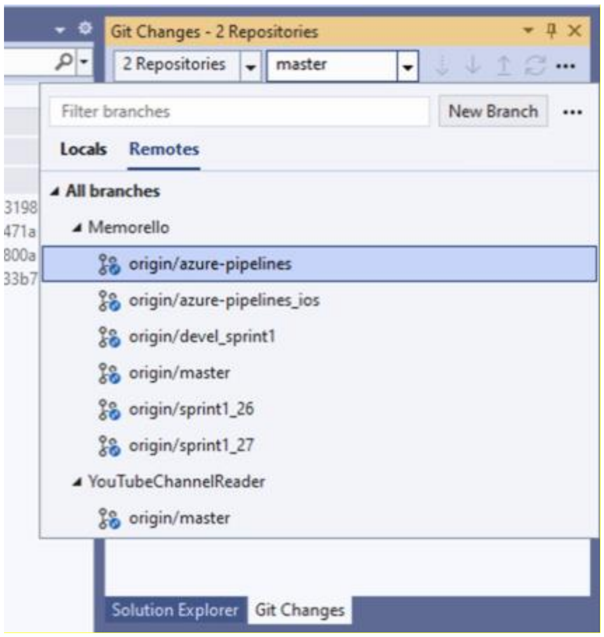


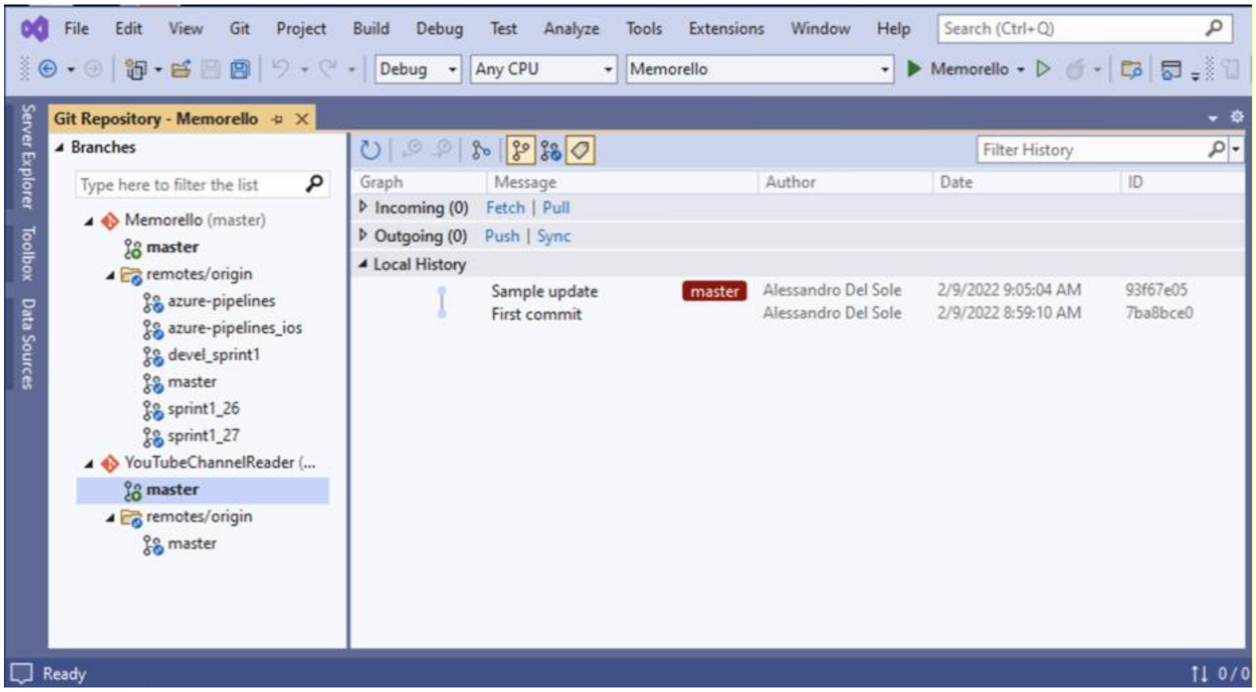
Figure 66: Branch Selection Inside Git Changes

You can select between local and remote branches individually for each project so that it is easier to decide where your code commits must go. You can also right-click a branch name to access shortcuts that would normally be available when you select a branch from the status bar.

When you are ready to submit your changes, add a comment and then click the **Commit All Repos** button. As you can see, nothing is different from what you would usually do against one repository.

Managing branches

You can select **Git** > **Manage Branches** to nicely manage your branches in all the involved repositories. Figure 67 shows an example.



Graph	Message	Author	Date	ID
	Sample update master	Alessandro Del Sole	2/9/2022 9:05:04 AM	93f67e05
	First commit	Alessandro Del Sole	2/9/2022 8:59:10 AM	7ba8bce0

Figure 67: Branch Management with Multiple Repositories

On the left side, a tree view shows the list of branches per repo, both local and remote. When you click a branch, you can see the history inside the main space of the window. Right-clicking a branch name provides shortcuts that are related to branches, such as checking out, code synchronization, deletion, rebasing, or resetting the branch. Right-clicking a commit in the window's main space provides shortcuts that are specific to individual commits, such as commit comparison and details view, but they are also for working on individual commits, as you learned in the first part of this chapter. Obviously, you can also create new branches from here. Right-click anywhere in the user interface and then select the New Branch command.

Final considerations about multirepo support

Supporting solutions based on multiple Git repositories is a tremendous addition to the Visual Studio toolbox, because many real-world solutions are based on projects that reside in different repositories, which does not necessarily mean different Git providers. In fact, a company might have several projects hosted by the same provider, such as Azure DevOps or private GitHub workspaces, that are interconnected with one another. This is the typical case of libraries that are referenced by application projects. The new team collaboration feature avoids the need of having multiple instances of Visual Studio open if you need to work on different projects at the same time, and it avoids the need of adding a binary reference, such as a NuGet package, of another project that must be continuously updated and maintained. Supporting multiple Git repositories means saving a lot of time and resources, improving team collaboration and productivity.

Chapter summary

Team collaboration is the core of software development, and Visual Studio has always offered first-class tools to support teams. Visual Studio 2022 introduces chats in the Live Share collaboration tool, and it rearranges Git tooling by moving Team Explorer into Git Changes and simplifying the tools required to work with Git repositories and source control. In terms of new features, you can now quickly compare branches, work against individual code commits rather than against branches, and you can work with solutions based on projects that are hosted on different Git repositories, regardless of the provider. These are all important additions, especially now that remote team collaboration has been widely adopted by many companies as a result of the recent pandemic.

Chapter 7 Web and Cloud Support

There are important changes in the way Visual Studio supports web and cloud projects. This is due to the natural evolution of the Azure platform on the cloud side and the development tools on the web side. These changes could be impactful to you as a developer, so this chapter summarizes what's new in web and cloud development with Visual Studio 2022, highlighting what scenarios are no longer supported and the workaround you could use.

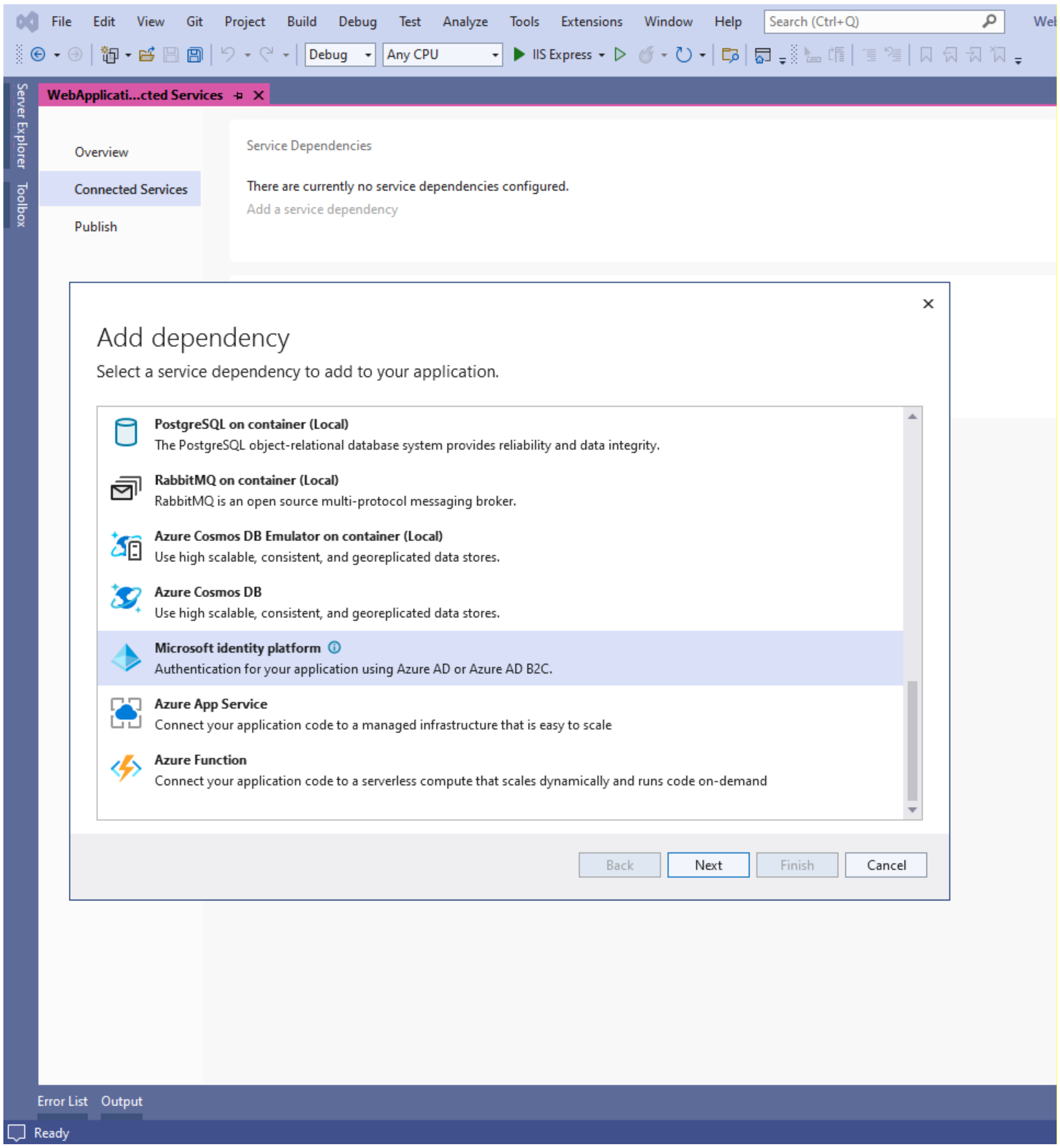
Note: The focus of this chapter is on tooling, not development technologies, so you will not find explanations about techniques and implementation. Appropriate links will be provided to the official documentation where required.

What's new with web development tools

Web development tools in Visual Studio allow you to build ASP.NET applications and publish them to both web and cloud hosts, so the concepts described here also apply to Azure. The following sections describe the most relevant new features in the development tools offered in VS 2022.

Adding identity support from Connected Services

Connected Services is a window that appears at the end of the generation of a new ASP.NET Core project. You can also enable it at any time by clicking **Connected Services** in Solution Explorer. In Visual Studio 2022, it is possible to add authentication based on the Microsoft identity platform directly from Connected Services. When in this window, click the **Add a service dependency** shortcut, and you will see the **Add dependency** dialog (see Figure 68).



Service Dependencies

There are currently no service dependencies configured.

Add a service dependency

Add dependency

Select a service dependency to add to your application.

PostgreSQL on container (Local)
The PostgreSQL object-relational database system provides reliability and data integrity.

RabbitMQ on container (Local)
RabbitMQ is an open source multi-protocol messaging broker.

Azure Cosmos DB Emulator on container (Local)
Use high scalable, consistent, and georeplicated data stores.

Azure Cosmos DB
Use high scalable, consistent, and georeplicated data stores.

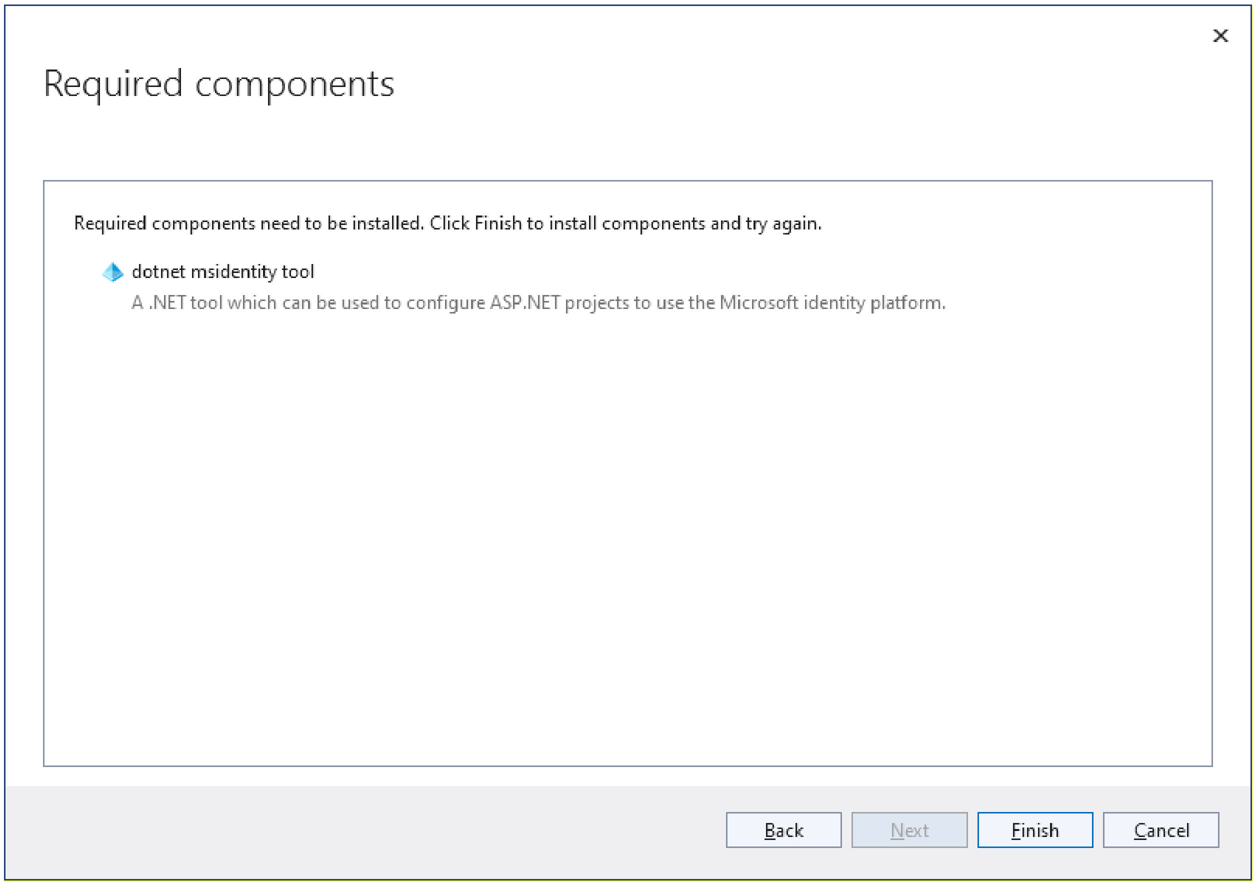
Microsoft identity platform
Authentication for your application using Azure AD or Azure AD B2C.

Azure App Service
Connect your application code to a managed infrastructure that is easy to scale

Azure Function
Connect your application code to a serverless compute that scales dynamically and runs code on-demand

Figure 68: Adding the Microsoft Identity Platform as a Connected Service

Locate the Microsoft identity platform service and then click **Next**. Depending on your setup configuration, you might get the warning shown in Figure 69, where you can see Visual Studio saying that some additional components are required, more specifically **dotnet msidentity**. Click **Finish** when ready.



Required components need to be installed. Click Finish to install components and try again.

dotnet msidentity tool

A .NET tool which can be used to configure ASP.NET projects to use the Microsoft identity platform.

Figure 69: Installing Microsoft Identity Platform Components

Tip: At the time of this writing, Visual Studio closes the dialog after installing the components without doing any work. This means you will need to again select the Microsoft identity platform in the Add dependency dialog and start the process again.

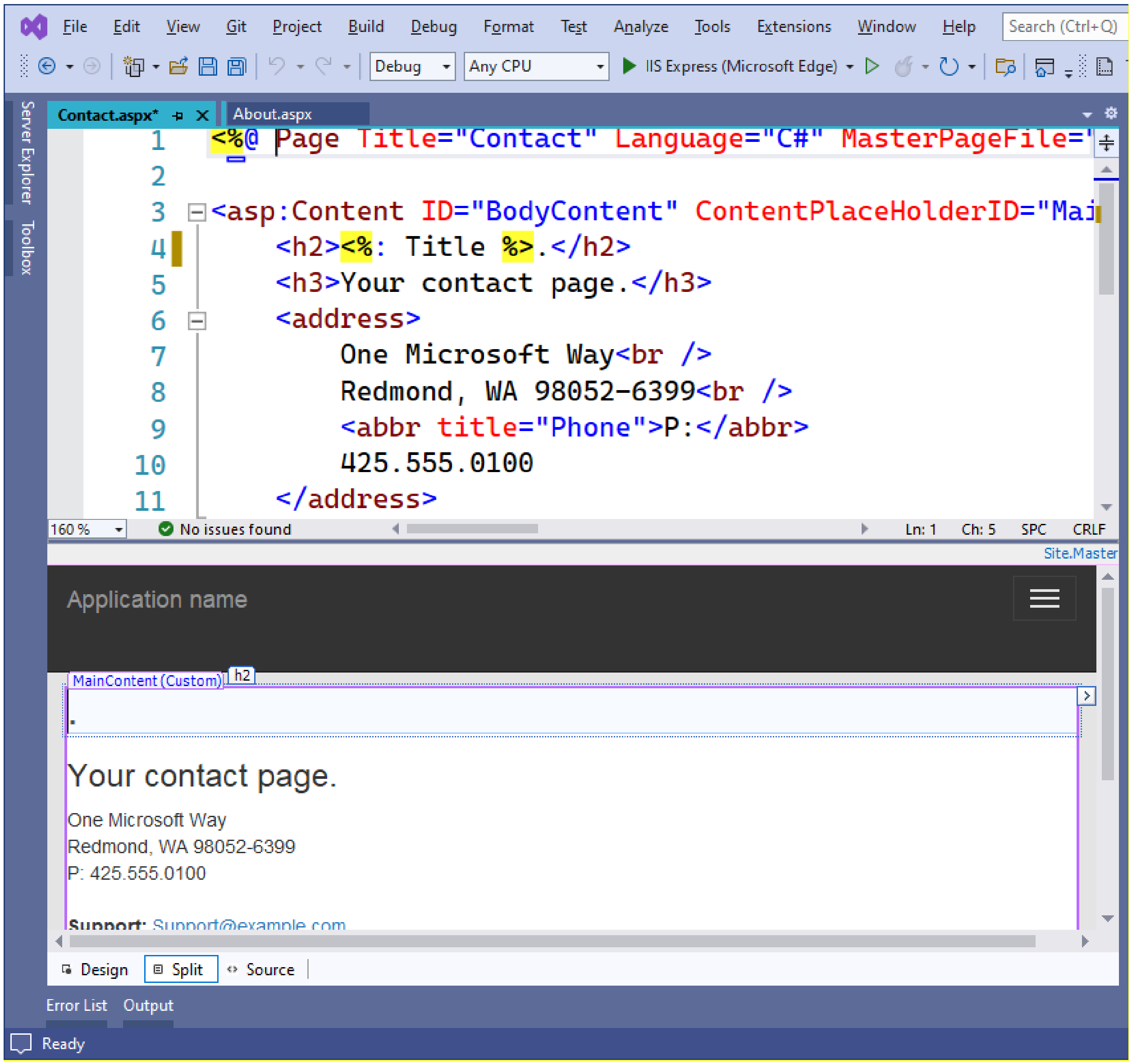
At this point you will be able to select your Azure subscription and the application you want to target. You can complete the registration of the application by following the Microsoft official guide.

Empty projects and minimal APIs

Another thing you might want to know when developing with ASP.NET Core in VS 2022 is that the ASP.NET Core Empty project template is now automatically based on a minimal API if the target framework is .NET 6. The purpose of a minimal API is to create web API projects with minimal dependencies and with the minimum set of files, features, and dependencies of ASP.NET Core.

Updated designer for Web Forms

Despite the evolution of development platforms for web applications, there are many sites built with ASP.NET Framework on the market that need continuous maintenance and updates. In order to better support such applications, Visual Studio 2022 introduces an updated designer for Web Forms with several interesting features. When you split the designer in two parts, the code editor and a designer, you enable a feature called Web Live Preview, discussed in the next sentences. Figure 70 shows an example.



```
<%@ Page Title="Contact" Language="C#" MasterPageFile="
<asp:Content ID="BodyContent" ContentPlaceHolderID="Mai
    <h2><%: Title %>.</h2>
    <h3>Your contact page.</h3>
    <address>
        One Microsoft Way<br />
        Redmond, WA 98052-6399<br />
        <abbr title="Phone">P:</abbr>
        425.555.0100
    </address>
```

Application name

Your contact page.

One Microsoft Way
Redmond, WA 98052-6399
P: 425.555.0100

Support: Support@example.com

Figure 70: Web Live Preview in Action

You can click the **Split** button at the bottom of the designer so that both the source markup and the live preview are visible. When you click a view in the code editor, it will also be highlighted in the preview and vice versa. The updated designer allows you to work with well-known techniques, for example the ability to drag and drop controls from the Toolbox onto the design surface, or double-clicking a control to add a default event handler. However, it also adds new capabilities. For example, the preview is powered by the Microsoft Edge engine. This makes it possible to work with the support of the latest web technologies.

Additionally, changes you make to markup code, such as to .aspx and .css files, are visible in the Web Live Preview as you type. You do not need to save changes every time. Last but not least, the designer now supports live data instead of placeholder data. For a better illustration, if you had a control that was data-bound in earlier versions of Visual Studio, the Web Forms designer displayed auto-generated placeholder data. Now the designer is able to instantiate the data-bound object and show its result in the user interface at design time.

What's new for cloud development

In Visual Studio 2022, the Azure Cloud Service project template has been deprecated. The new **Azure Cloud Service (extended support)** template should be used for new projects (see Figure 71).

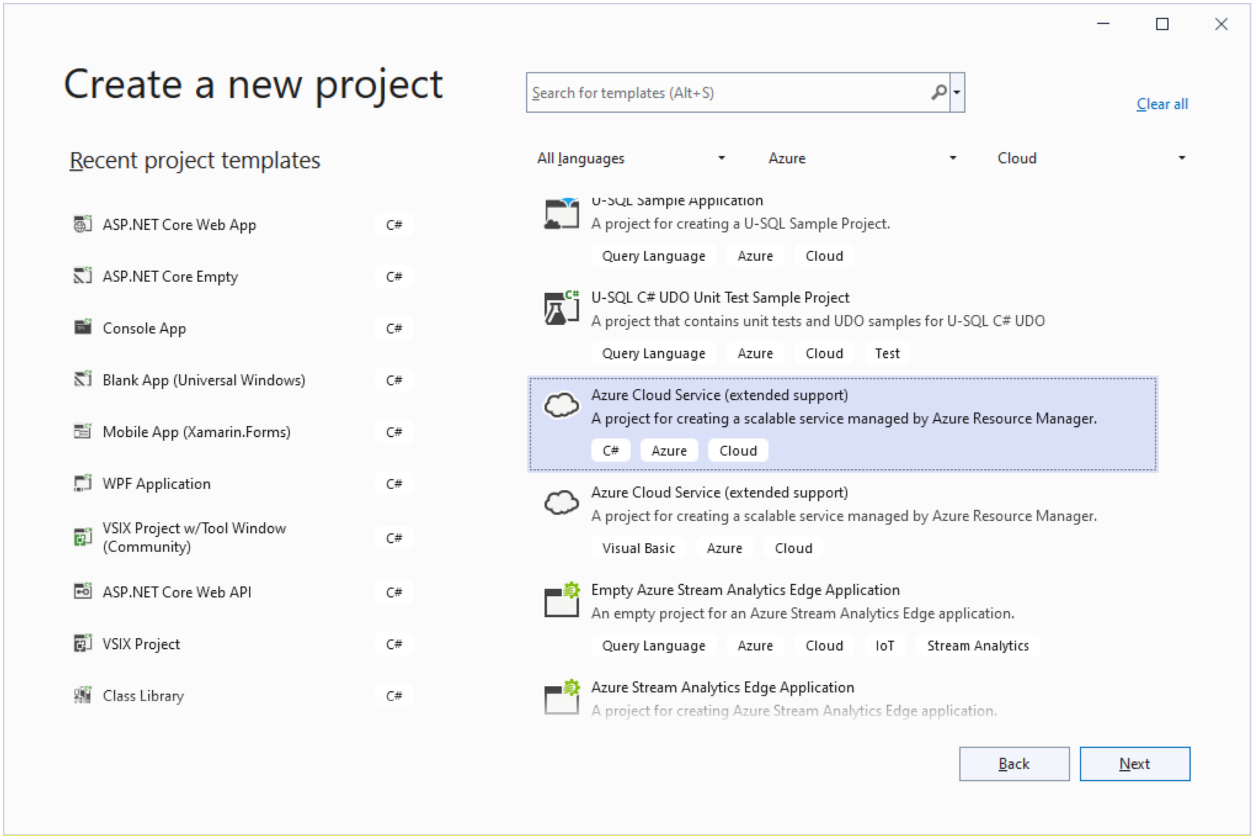

Create a new project
Search for templates (Alt+S)
Clear all
Recent project templates
ASP.NET Core Web App C#
ASP.NET Core Empty C#
Console App C#
Blank App (Universal Windows) C#
Mobile App (Xamarin.Forms) C#
WPF Application C#
VSIX Project w/Tool Window (Community) C#
ASP.NET Core Web API C#
VSIX Project C#
Class Library C#
All languages
Azure
Cloud
U-SQL Sample Application
A project for creating a U-SQL Sample Project.
Query Language Azure Cloud
U-SQL C# UDO Unit Test Sample Project
A project that contains unit tests and UDO samples for U-SQL C# UDO
Query Language Azure Cloud Test
Azure Cloud Service (extended support)
A project for creating a scalable service managed by Azure Resource Manager.
C# Azure Cloud
Azure Cloud Service (extended support)
A project for creating a scalable service managed by Azure Resource Manager.
Visual Basic Azure Cloud
Empty Azure Stream Analytics Edge Application
An empty project for an Azure Stream Analytics Edge application.
Query Language Azure Cloud IoT Stream Analytics
Azure Stream Analytics Edge Application
A project for creating Azure Stream Analytics Edge application.
Back
Next

Figure 71: New Azure Cloud Service Project Template

The reason for the deprecation of the old template can be found in many infrastructural changes that Microsoft made to the Azure platform for new customers, so a new template was needed. In practice, when you create a new Azure Cloud Service project in VS 2022, you will still be able to add roles like in the past (see Figure 72).

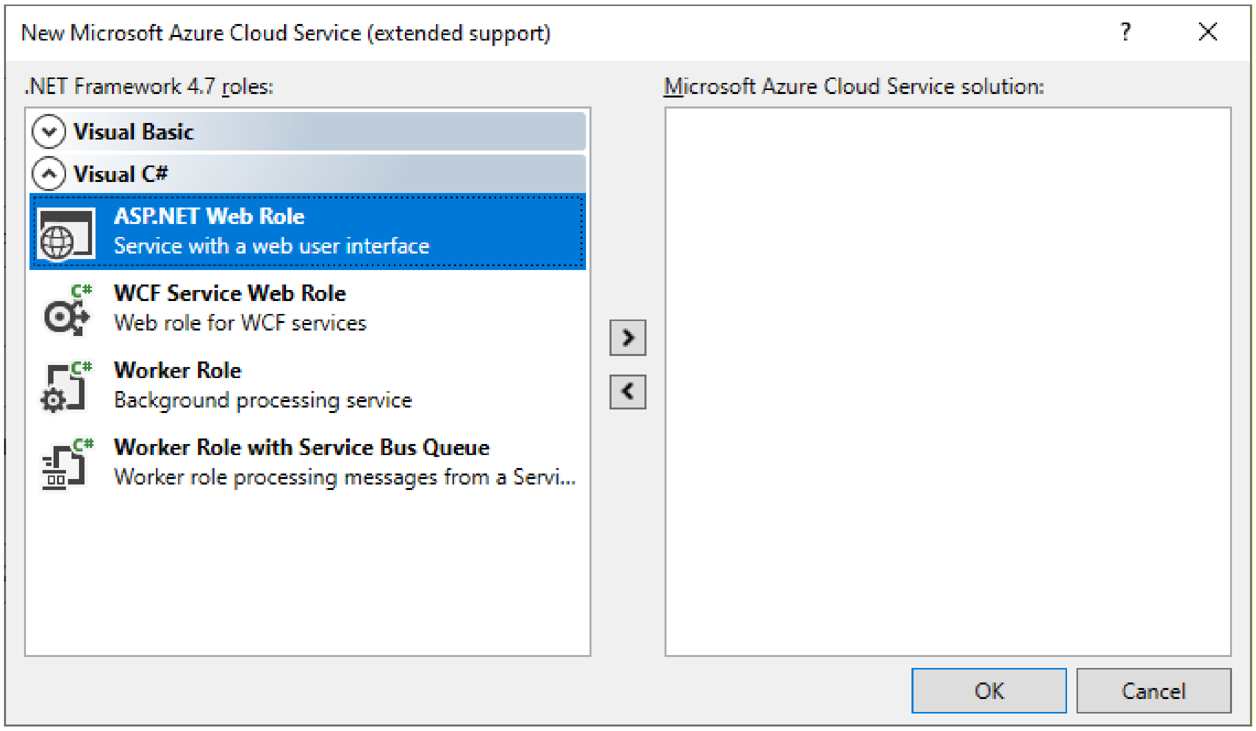

New Microsoft Azure Cloud Service (extended support)
?
.NET Framework 4.7 roles:
Visual Basic
Visual C#
ASP.NET Web Role
Service with a web user interface
WCF Service Web Role
Web role for WCF services
Worker Role
Background processing service
Worker Role with Service Bus Queue
Worker role processing messages from a Servi...
Microsoft Azure Cloud Service solution:
OK
Cancel

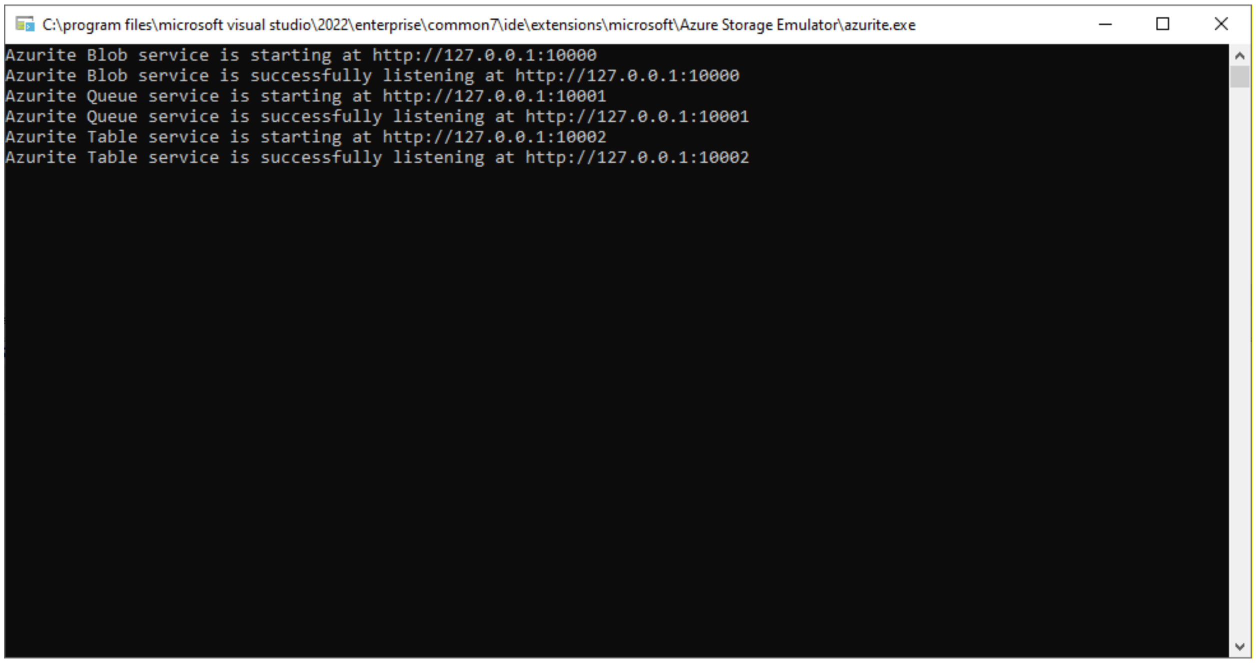
Figure 72: Adding Roles to a New Azure Cloud Service Project

Azure projects in Visual Studio 2022 no longer use the Azure Storage emulator, which has also been deprecated in favor of a tool called **Azurite**, introduced in the next paragraphs. This is one of the main reasons a new project template was needed. Existing Azure customers will still be able to open cloud service projects created with the previous versions.

Local storage emulation with Azurite

As you might know, one of the most important and popular services offered by Azure is storage, where files, tables, and message queues can be hosted. Applications you develop for Azure can access the storage and, for debugging purposes, the Microsoft SDKs have always offered a local emulator of the storage that you could use instead of debugging against remote storage.

In Visual Studio 2022, the local storage emulation is no longer provided by the Azure Storage Emulator tool. Instead, it is now provided by an open-source tool called Azurite. Azurite is based on Node.js and TypeScript code. It supports the majority of the Azure Storage API, and it allows local storage with a cross-platform architecture. Azurite runs as a console application (see Figure 73) that keeps listening to Storage API calls.


C:\program files\microsoft visual studio\2022\enterprise\common7\ide\extensions\microsoft\Azure Storage Emulator\azurite.exe

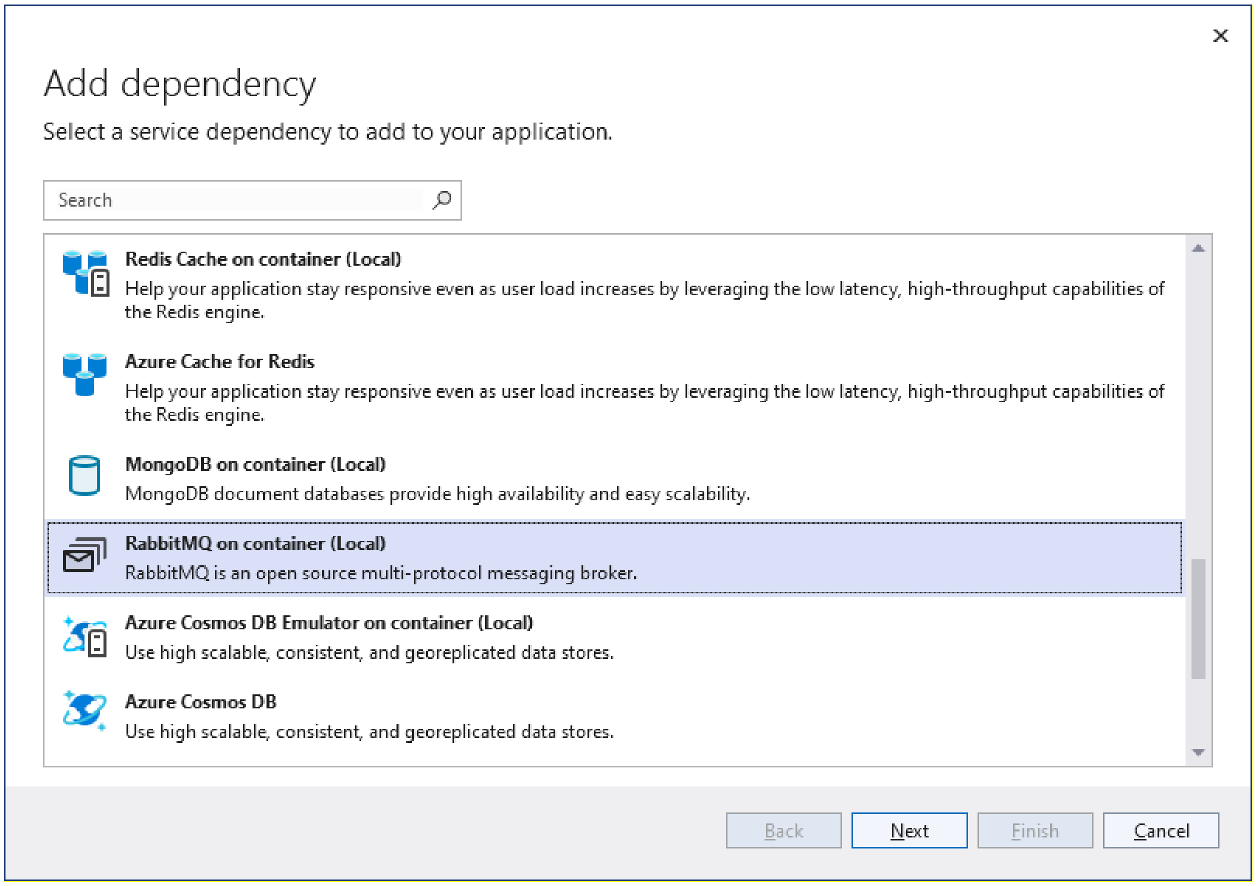
```
Azurite Blob service is starting at http://127.0.0.1:10000
Azurite Blob service is successfully listening at http://127.0.0.1:10000
Azurite Queue service is starting at http://127.0.0.1:10001
Azurite Queue service is successfully listening at http://127.0.0.1:10001
Azurite Table service is starting at http://127.0.0.1:10002
Azurite Table service is successfully listening at http://127.0.0.1:10002
```

Figure 73: Azurite Listening to Storage API Calls

With Visual Studio 2022, like the old storage emulator, you do not need to directly interact with Azurite. The IDE will take care of sending all the necessary Storage API requests to Azurite.

New Connected Services

The Connected Services tool has been extended to include additional services when creating Azure projects. More specifically, Connected Services now allows you to use container images to provide a local debugging experience for areas like Redis Cache, MongoDB, RabbitMQ, Cosmos DB, Azure Storage, and SQL. Even if these dependencies are not listed one after the other in the **Add dependency** dialog, they can be easily recognized because their names end with the following literal: **on container (Local)**. Figure 74 shows the dialog with some of the aforementioned options.



Add dependency

Select a service dependency to add to your application.

Redis Cache on container (Local)
Help your application stay responsive even as user load increases by leveraging the low latency, high-throughput capabilities of the Redis engine.

Azure Cache for Redis
Help your application stay responsive even as user load increases by leveraging the low latency, high-throughput capabilities of the Redis engine.

MongoDB on container (Local)
MongoDB document databases provide high availability and easy scalability.

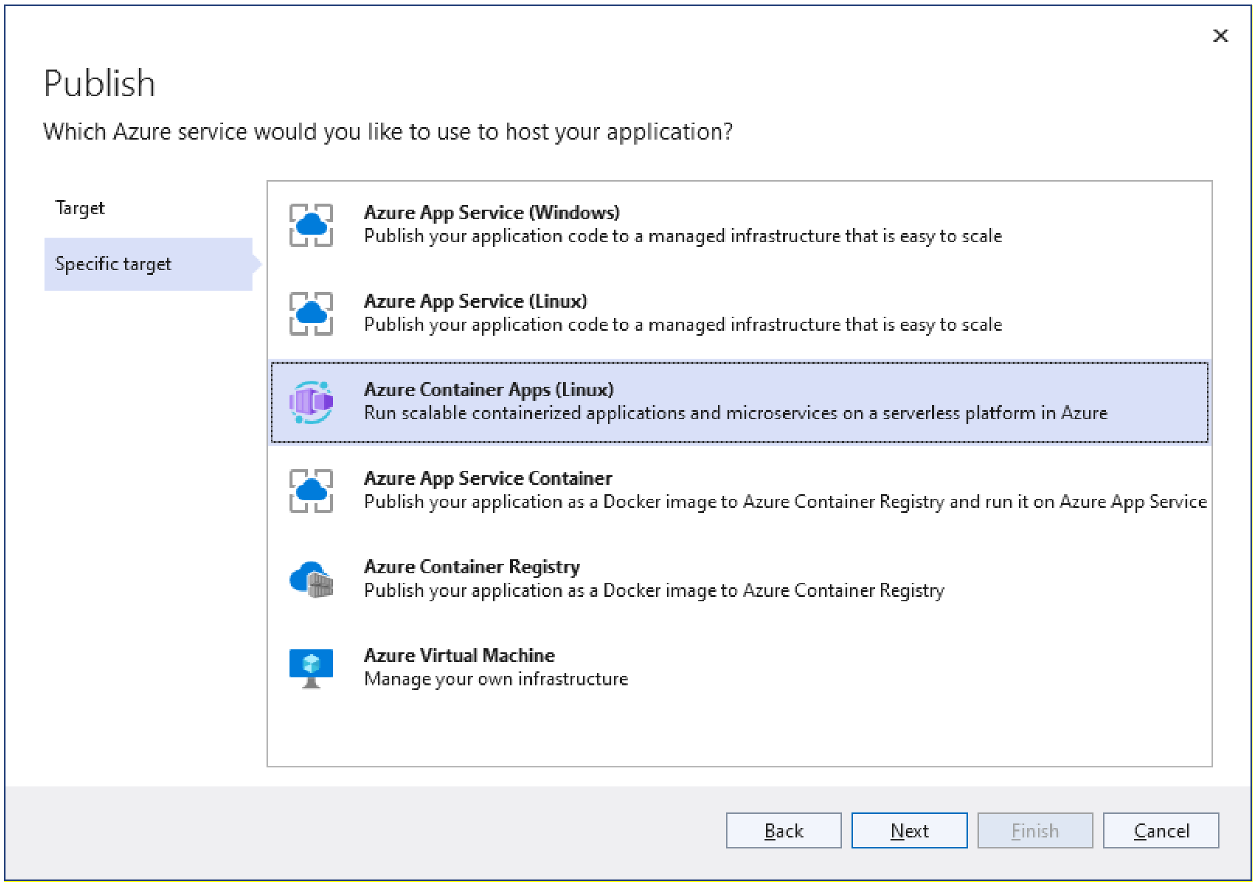
RabbitMQ on container (Local)
RabbitMQ is an open source multi-protocol messaging broker.

Azure Cosmos DB Emulator on container (Local)
Use high scalable, consistent, and georeplicated data stores.

Azure Cosmos DB
Use high scalable, consistent, and georeplicated data stores.

Figure 74: Container Images for Local Data Services are Now Supported

Some new options are also available for publishing applications to Azure. Visual Studio 2022 allows you to publish apps to a new container called Azure Container Apps, which is based on Linux. When you start the **Publish** wizard in Visual Studio 2022, you will have an option to choose the target, as shown in Figure 75.



Publish

Which Azure service would you like to use to host your application?

Azure App Service (Windows)
Publish your application code to a managed infrastructure that is easy to scale

Azure App Service (Linux)
Publish your application code to a managed infrastructure that is easy to scale

Azure Container Apps (Linux)
Run scalable containerized applications and microservices on a serverless platform in Azure

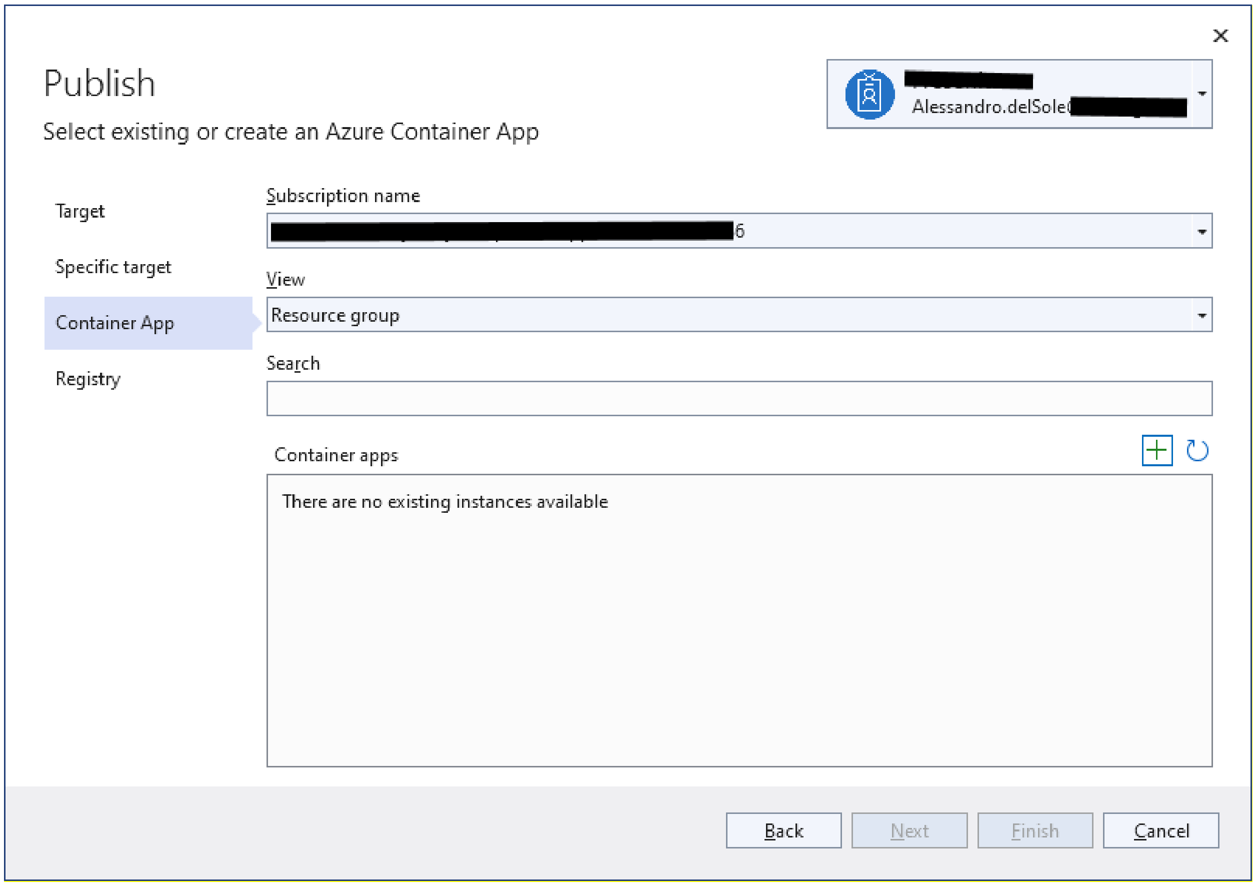
Azure App Service Container
Publish your application as a Docker image to Azure Container Registry and run it on Azure App Service

Azure Container Registry
Publish your application as a Docker image to Azure Container Registry

Azure Virtual Machine
Manage your own infrastructure

Figure 75: Choosing a Publish Target

The **Azure Container Apps (Linux)** option allows you to publish your apps to this service. When you select this option, you will be prompted to select an Azure subscription and a container app based on the Azure Container Apps service, as shown in Figure 76.



Publish

Select existing or create an Azure Container App

[redacted]
Alessandro.delSole[redacted]

Subscription name: [redacted]6

View: Resource group

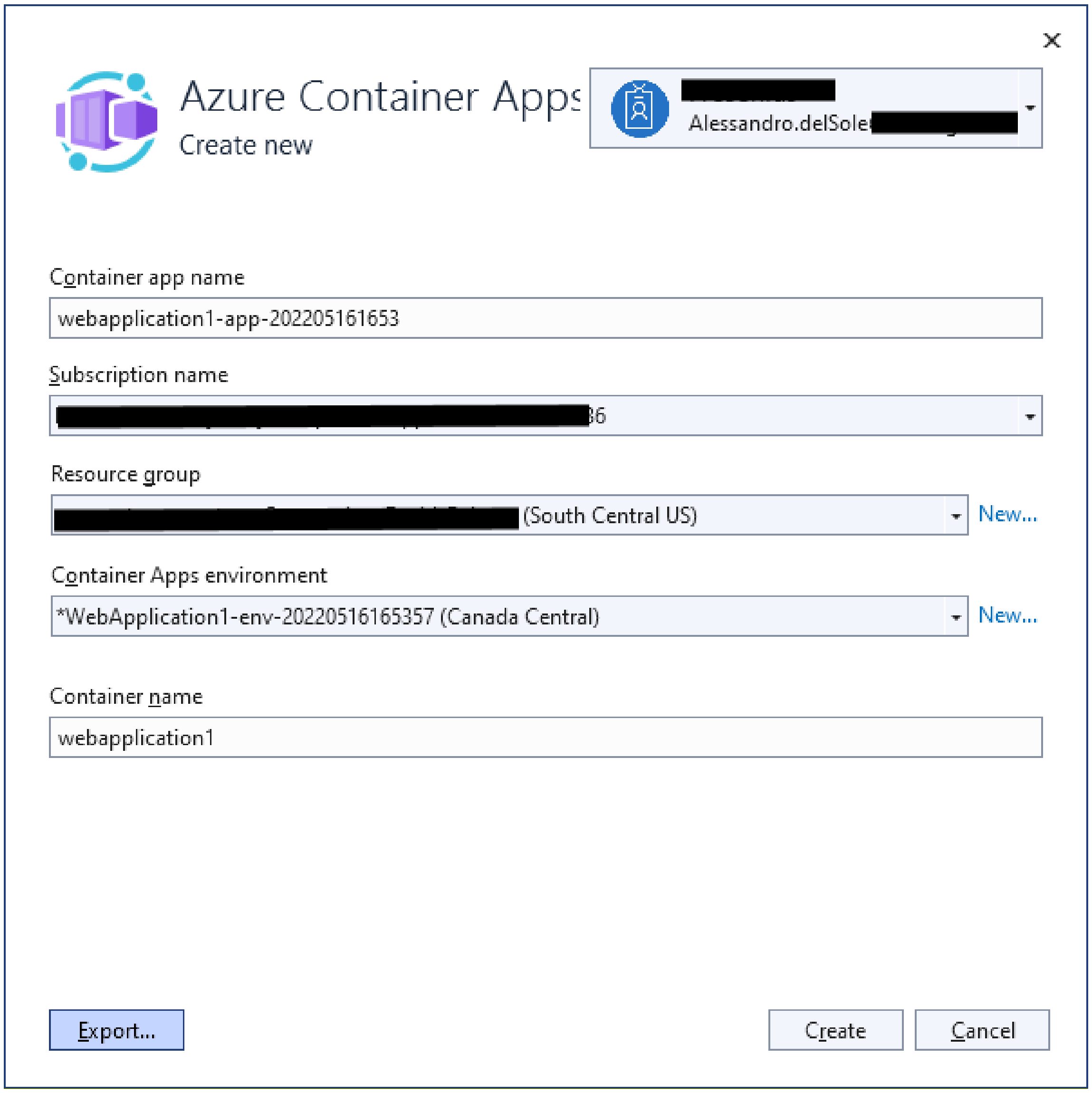
Search:

Container apps

There are no existing instances available

Figure 76: Selecting an Azure Subscription for the Container App

You can connect to an existing container app or click the **+** button at the right of the header of the **Container apps** group to create a new one. If you create a new container app from this dialog, you will need to enter all the service details shown in Figure 77.


Azure Container Apps

Create new

[redacted] Alessandro.delSole[redacted]

Container app name: webapplication1-app-202205161653

Subscription name: [redacted]6

Resource group: [redacted] (South Central US)

Container Apps environment: *WebApplication1-env-20220516165357 (Canada Central)

Container name: webapplication1

Figure 77: Details for Creating a Container App

You will need to specify the name, the subscription, the Azure resource group, the host web application with its data center, and finally the container name. If you click Create, Visual Studio will generate all the services you need. Obviously, it is out of the scope of this book to go further on this topic, but you can always look at the official documentation about Azure Container Apps if you are interested in learning more about this topic.

Razor editor improvements

Razor is the markup language of choice with ASP.NET Core web projects, and Visual Studio has always offered the usual first-class code editor for it. In Visual Studio 2022, additional interesting improvements were made to the Razor editor. The first addition is Hot Reload, which you saw in Chapter 4, and that works in the same way. If you want to quickly try it yourself, you can follow these steps:

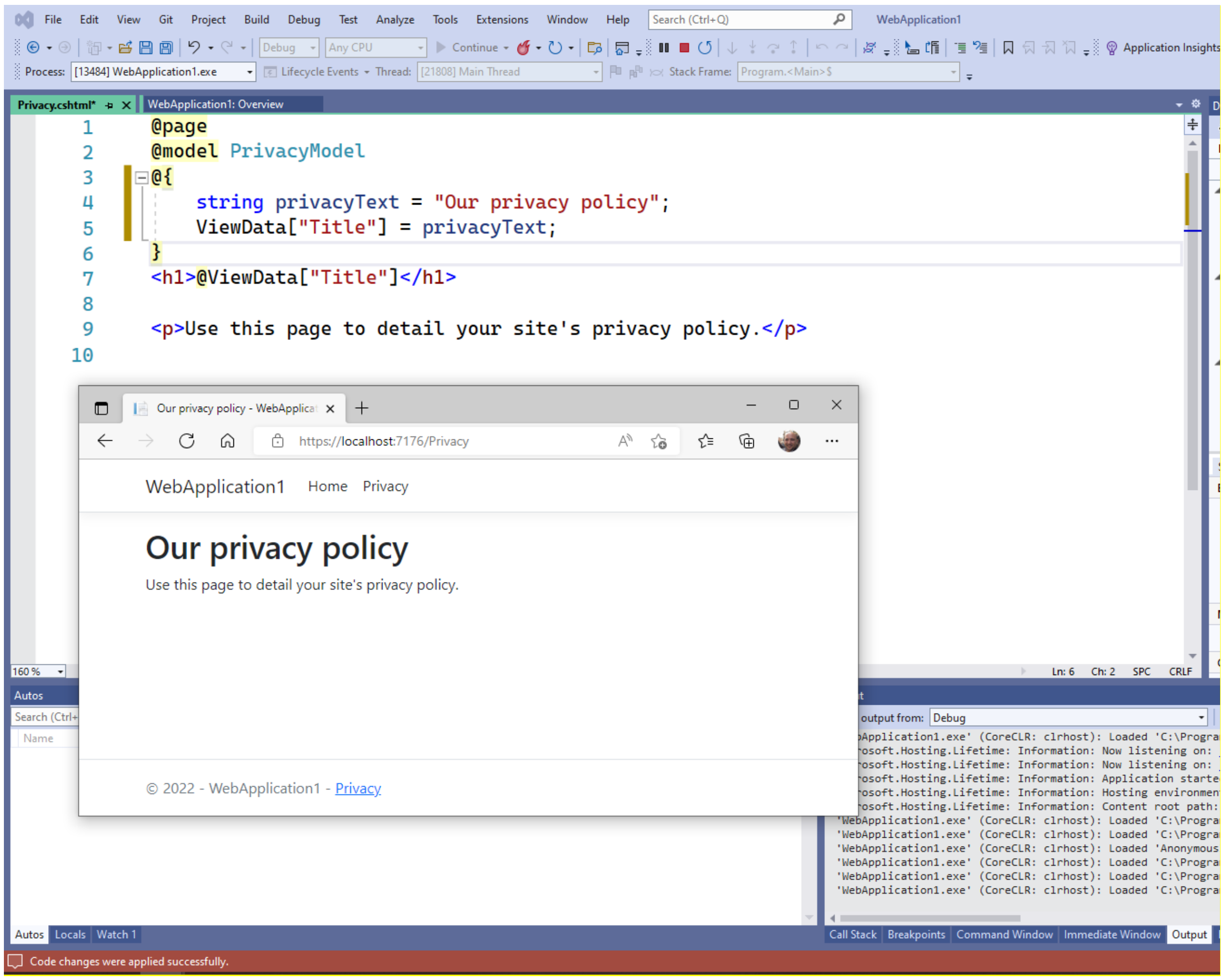
1. Create a new ASP.NET Core web project.
2. Run the project by pressing **F5** so that the debugger is enabled.
3. Open the **Privacy** page of the auto-generated application (see Figure 78).
4. Change the following line of code:

```
ViewData["Title"] = "Privacy policy";
```

As follows:

```
string privacyText = "Our privacy policy";
ViewData["Title"] = privacyText;
```

This way, you have not only changed the markup and the text, but also added managed references.



```
@page
@model PrivacyModel
@{
    string privacyText = "Our privacy policy";
    ViewData["Title"] = privacyText;
}
<h1>@ViewData["Title"]</h1>

<p>Use this page to detail your site's privacy policy.</p>
```

Our privacy policy

Use this page to detail your site's privacy policy.

© 2022 - WebApplication1 - Privacy

```
pApplication1.exe' (CoreCLR: clrhost): Loaded 'C:\Progra
rosoft.Hosting.Lifetime: Information: Now listening on:
rosoft.Hosting.Lifetime: Information: Now listening on:
rosoft.Hosting.Lifetime: Information: Application starte
rosoft.Hosting.Lifetime: Information: Hosting environmen
rosoft.Hosting.Lifetime: Information: Content root path:
'WebApplication1.exe' (CoreCLR: clrhost): Loaded 'C:\Progra
'WebApplication1.exe' (CoreCLR: clrhost): Loaded 'C:\Progra
'WebApplication1.exe' (CoreCLR: clrhost): Loaded 'Anonymous
'WebApplication1.exe' (CoreCLR: clrhost): Loaded 'C:\Progra
'WebApplication1.exe' (CoreCLR: clrhost): Loaded 'C:\Progra
'WebApplication1.exe' (CoreCLR: clrhost): Loaded 'C:\Progra
```

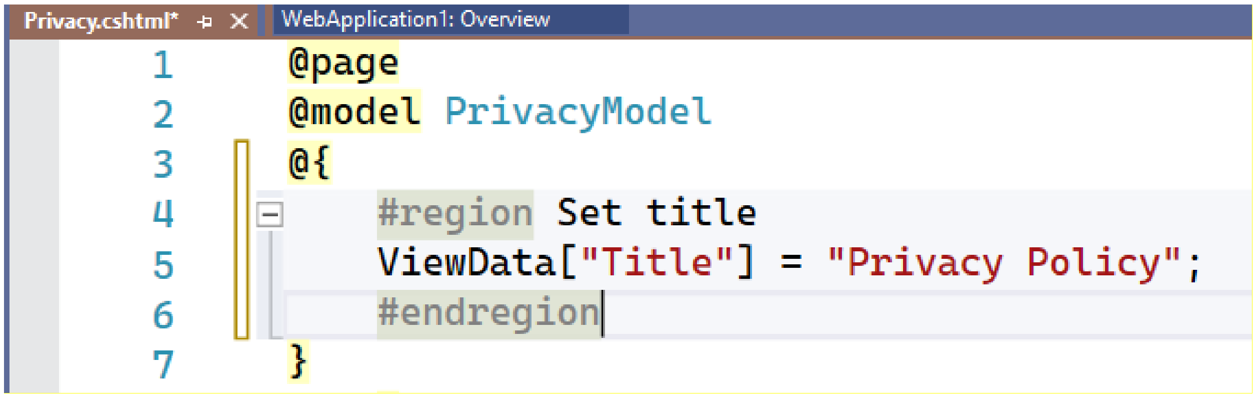
Code changes were applied successfully.

Figure 78: Hot Reload with Razor Files

Click the **Hot Reload** button on the toolbar (the flame icon) and you will see how changes will be reflected in the webpage without needing to restart the application.

General editor improvements

There are several additional improvements in the Razor code editor that are worth mentioning. First, performance: The code editor is now much faster. Second, the editor now fixes indentation and formatting much better than before. Another new feature of the Razor code editor is the ability to collapse regions, as you can see in Figure 79.

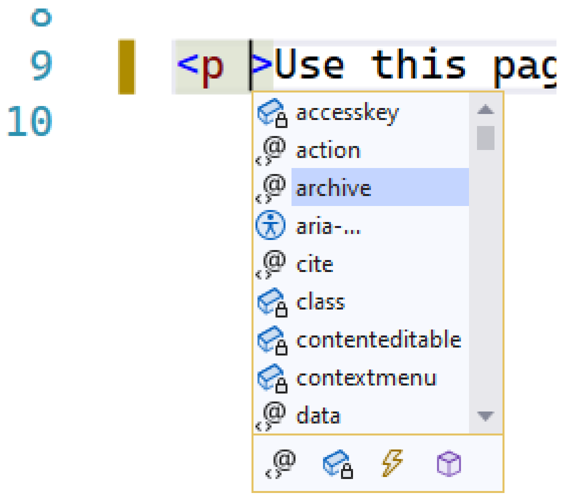


Privacy.cshtml* WebApplication1: Overview

```
@page
@model PrivacyModel
@{
    #region Set title
    ViewData["Title"] = "Privacy Policy";
    #endregion
}
```

Figure 79: An Option to Collapse Regions Inside Razor Files Has Been Added

This is certainly a nice addition to improve code readability and organization. Visual Studio 2022 also adds new colors to the Razor editing experience that you can customize via **Tools** > **Options** > **Environment** > **Fonts & Colors**. Moreover, when you add comments, they now have autocompletion, and they automatically include commenting continuation and smart indentation. Finally, something new has been added to tag helpers. Look at Figure 80.



```
<p >Use this pag
```

Figure 80: Tag Helper Improvements

Tag helpers are now classified in the IntelliSense pop-up together with other members, such as fields, events, and methods. This allows you to quickly filter the list by tag helpers.

Chapter summary

As the Azure platform and web technologies evolve, so does Visual Studio. In the new version, specific tools have been added to simplify integration with the Microsoft identity platform and minimal APIs. For cloud development, VS 2022 provides a new project template for Azure Cloud Services, which is required to support many infrastructural changes in the platform, including the usage of Azurite as the new storage simulator for local development and debugging. The Razor editor has also been updated to support Hot Reload, and has received several improvements like smarter indentation and better performance.