NUGET IN-HOUSE

SUCCINCTLY

BY **JOSÉ ROBERTO OLIVAS MENDOZA**

SUCCINCTLY EBOOK SERIES

Syncfusion®

NuGet In-House Succinctly

By
José Roberto Olivas Mendoza

Foreword by Daniel Jebaraj

Technical Reviewer: James McCaffrey
Copy Editor: Courtney Wright
Acquisitions Coordinator: Tres Watkins, content development manager, Syncfusion, Inc.
Proofreader: Graham High, senior content producer, Syncfusion, Inc.

Table of Contents

The Story Behind the *Succinctly* Series of Books

Daniel Jebaraj, Vice President
Syncfusion, Inc.

Staying on the cutting edge

As many of you may know, Syncfusion is a provider of software components for the Microsoft platform. This puts us in the exciting but challenging position of always being on the cutting edge.

Whenever platforms or tools are shipping out of Microsoft, which seems to be about every other week these days, we have to educate ourselves, quickly.

Information is plentiful but harder to digest

In reality, this translates into a lot of book orders, blog searches, and Twitter scans.

While more information is becoming available on the internet and more and more books are being published, even on topics that are relatively new, one aspect that continues to inhibit us is the inability to find concise technology overview books.

We are usually faced with two options: read several 500+ page books or scour the web for relevant blog posts and other articles. Just as everyone else who has a job to do and customers to serve, we find this quite frustrating.

The *Succinctly* series

This frustration translated into a deep desire to produce a series of concise technical books that would be targeted at developers working on the Microsoft platform.

We firmly believe, given the background knowledge such developers have, that most topics can be translated into books that are between 50 and 100 pages.

This is exactly what we resolved to accomplish with the *Succinctly* series. Isn't everything wonderful born out of a deep desire to change things for the better?

The best authors, the best content

Each author was carefully chosen from a pool of talented experts who shared our vision. The book you now hold in your hands, and the others available in this series, are a result of the authors' tireless work. 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.

Free forever

Syncfusion will be working to produce books on several topics. The books will always be free. Any updates we publish will also be free.

Free? What is the catch?

There is no catch here. Syncfusion has a vested interest in this effort.

As a component vendor, our unique claim has always been that we offer deeper and broader frameworks than anyone else on the market. Developer education greatly helps us market and sell against competing vendors who promise to "enable AJAX support with one click," or "turn the moon to cheese!"

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 succinctly-series@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 Twitter and "Like" us on Facebook to help us spread the word about the *Succinctly* series!

About the Author

As I mentioned in my previous ebooks published in the Syncfusion *Succinctly* series, I am an IT business entrepreneur, a software developer, and a huge technology fan. Since my company went to market in 1990, I have been in charge of product development and the innovation department. We were born as a custom software development firm, basing our projects on COBOL as our main programming language. Of course, there have been a lot changes and innovations in the IT industry since 1990. Therefore, we have been evolving along these years, from COBOL to .NET and other innovative technologies. An important part of my daily schedule is dedicated to researching cutting-edge technologies, and searching for new tools that can help us automate some or all of our processes regarding our products' development life cycles.

Electronic invoicing was a huge change that came from the Mexican government's Tax Authority (SAT) in 2010, and it became a requirement for all business transactions on Mexican soil. Therefore, the Mexican IT industry had to embrace this change. The result was a huge challenge in terms of product updating and integration. In addition, due the new regulations, a whole set of new products had to be developed.

As a Product Development Manager, I was leading all these changes and new product designs. I divided my crew into small teams, where each received a particular project to develop. In some cases, the result of a project was a .NET assembly that would be a part of another project. As a result, the team or teams that used these assemblies had to manually reference each one of them in their projects. At first, there were no issues with this process. However, as the number of projects increased, there were several problems. The recurrent issue was assembly versions: in many cases, teams forgot to reference the last updated assembly, and bugs appeared. This caused a huge waste of resources, and prevented some projects from being delivered on time. All these events led me to a challenge: assembly-referencing automation.

I had heard about NuGet, as I used it for referencing some public domain assemblies, such as Entity Framework, MySQL.NET connectors, and some file-zipping libraries. However, at that time, I did not know much about it. Therefore, my initial activity was digging around to learn how to use NuGet, and creating a NuGet account at www.nuget.org. I started browsing all NuGet website content in order to use its power for assembly-referencing automation. I was so excited thinking about the end of our problems that I did not foresee a huge issue: in order to use NuGet, we needed to convert all of our copyrighted code to public-domain code.

This book is about the challenges I faced to get the power of NuGet, without compromising our copyrighted code. The adventure gave me several sleepless nights, but the result was worth all the effort. I hope you, the reader of this book, can take advantage of NuGet capabilities to improve your software development process.

Who Is This Book For?

This book is for.NET developers who need to distribute their reusable code as NuGet packages, and do not want to make those packages public in the NuGet Gallery. In order to take advantage of the contents of this book, the reader should have some C# programming language skills and basic knowledge about IIS and Microsoft Visual Studio.

The book starts with an overview of NuGet, the NuGet Gallery, and the architecture of NuGet packages. The second chapter covers the process of setting up a NuGet server on our local computer. Then, the next chapter focuses on creating a Visual Studio project and setting up a development environment that allows us to create NuGet packages. The last two chapters will explain how to create and publish packages on the local computer, and how to use the published packages as dependencies for another project.

We will use Windows 10 as our computer operating system, and IIS (Internet Information Services) as our web server. Microsoft Edge will be our web browser for examples that use the NuGet Gallery. In addition, we will employ C# to write all code examples. Microsoft Visual Studio 2017 will be the IDE to work with the projects explained in this book. All examples described along this book are available to download here.

Chapter 1 Introduction

What is NuGet?

NuGet is an open-source package manager developed by the Microsoft .NET Foundation for the Microsoft development platform. The first release of NuGet, called NuPack, appeared in October of 2010. Since then, it has evolved into a great ecosystem of services and tools. NuGet is free and licensed under the Apache License 2.0, and comes preinstalled with Visual Studio 2012 and later.

NuGet is primarily a tool for sharing .NET code in the form of packages. It defines a way to create, host, and consume these packages. It also provides the tools required for each one of the roles involved in code sharing: creators, hosts, and consumers.

Note: A NuGet package is a unit of code that can be shared at the developer‘s convenience.

The following figure displays how packages flow between creators, hosts, and consumers.

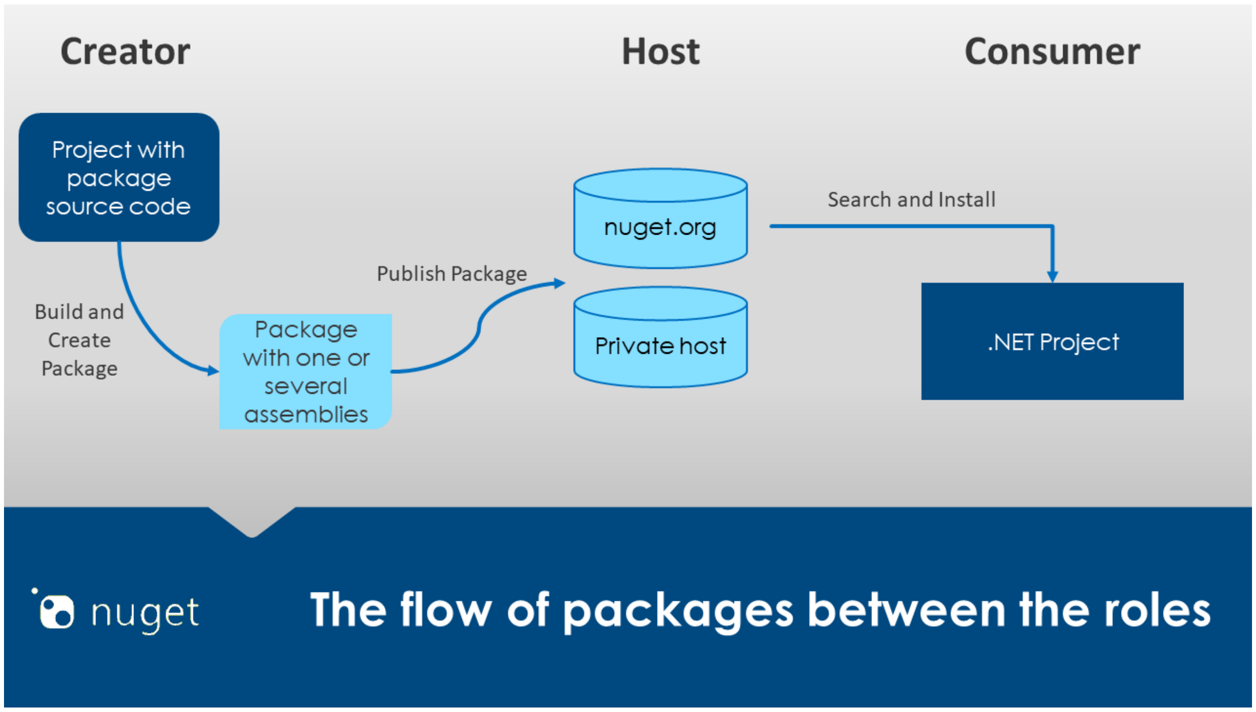


Figure 1: NuGet flow

The NuGet package

A NuGet package consists of .NET code compiled as one or several assemblies, delivered in a single compressed file with a .nupkg extension. Essentially, a .nupkg file is a .zip file with a different extension. Therefore, if we want to browse the content of a NuGet package, we can use any of the .zip file browser tools that exist on the market. One of the topics of this book is creating NuGet packages for sharing our code. This will be explained later.

The NuGet Gallery

The NuGet Gallery is the public repository for sharing NuGet packages. To access the NuGet Gallery, navigate to https://www.nuget.org. The following figure shows the NuGet Gallery home page.

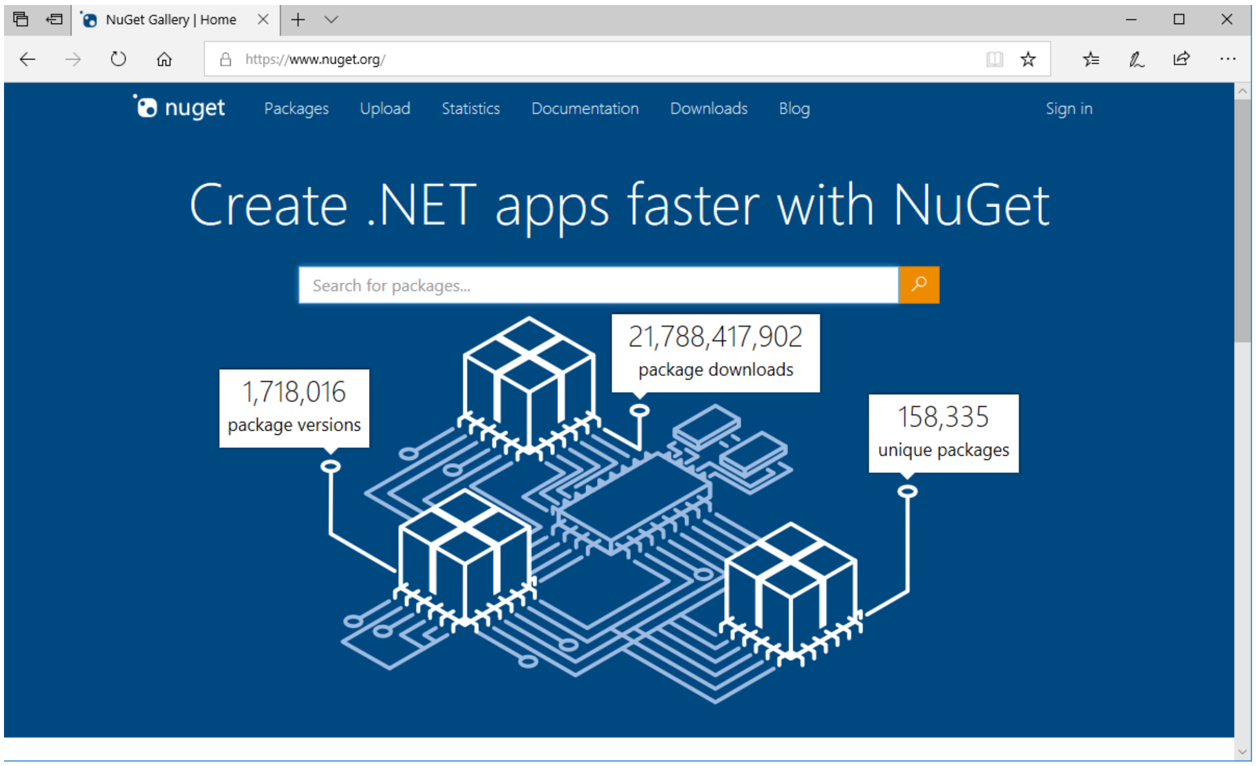


Figure 2: NuGet Gallery home page

In Figure 2, we can see three metrics that are automatically calculated by the NuGet Gallery. The first one points to the number of downloads made by all .NET developers who browse the NuGet Gallery. The second tells us how many packages are hosted in the gallery, regardless of the number of versions for each one. The last metric points to the number of package versions hosted (a unique package can have multiple versions). If we want to update these metrics, we can click our browser's **Refresh** button, usually located to the left of the address bar.

Signing in to the NuGet Gallery

We can browse the NuGet Gallery without an account. However, if we want to upload a package manually, we need to create an account and sign in to the gallery.

Creating an account for the NuGet Gallery

At the time I am writing this book, the NuGet Gallery has deprecated nuget.org accounts and supports only Microsoft accounts. Therefore, the first step is to create a Microsoft account if we do not already have one.

Creating a Microsoft account

We need an email address in order to create a Microsoft account. Once we make sure we have a valid email address, we should navigate to https://account.microsoft.com. The following figure displays the Microsoft account home page.

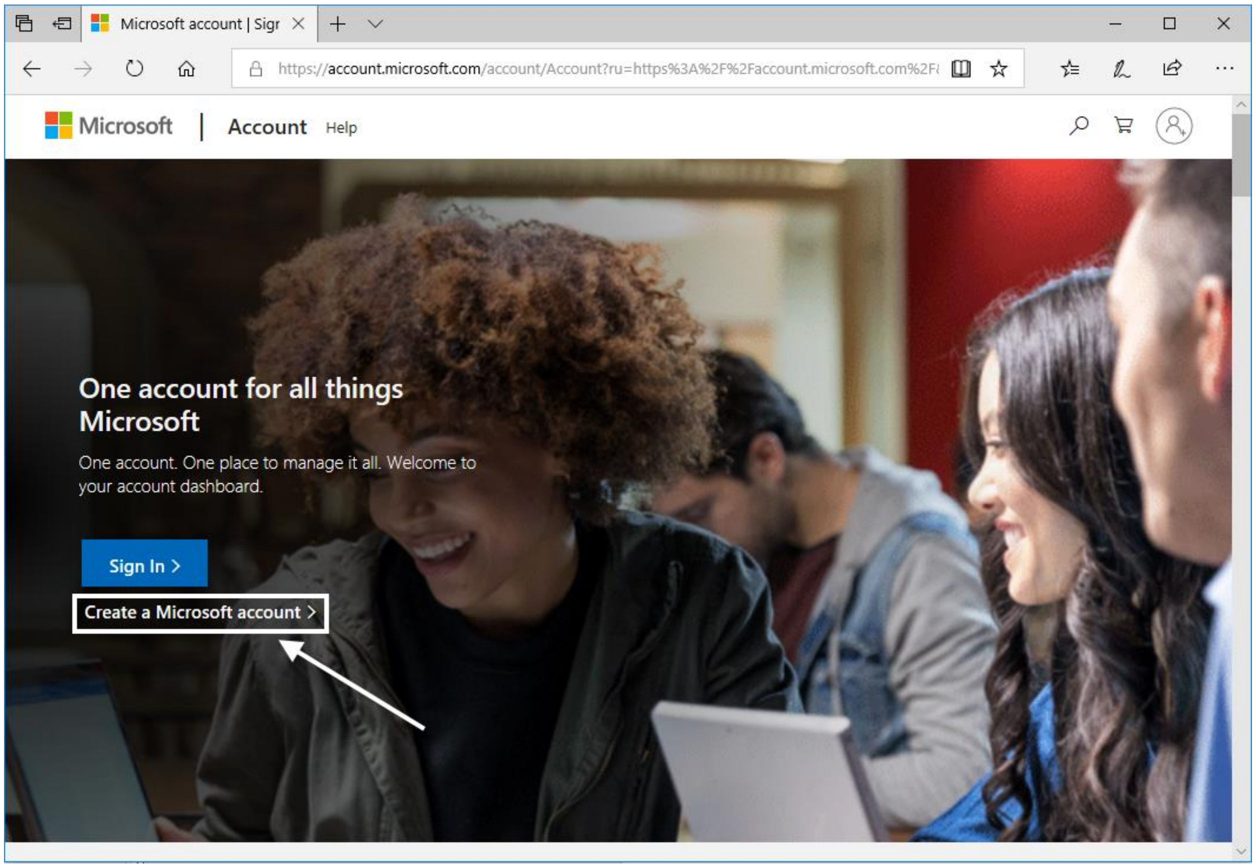

Microsoft account | Sigr
https://account.microsoft.com/account/Account?ru=https%3A%2F%2Faccount.microsoft.com%2F
Microsoft
Account
Help
One account for all things Microsoft
One account. One place to manage it all. Welcome to your account dashboard.
Sign In >
Create a Microsoft account >

Figure 3: Microsoft account home page

Next, click **Create a Microsoft account**, displayed in Figure 3, outlined by a rectangle. The account creation form will appear, as shown in the following figure.

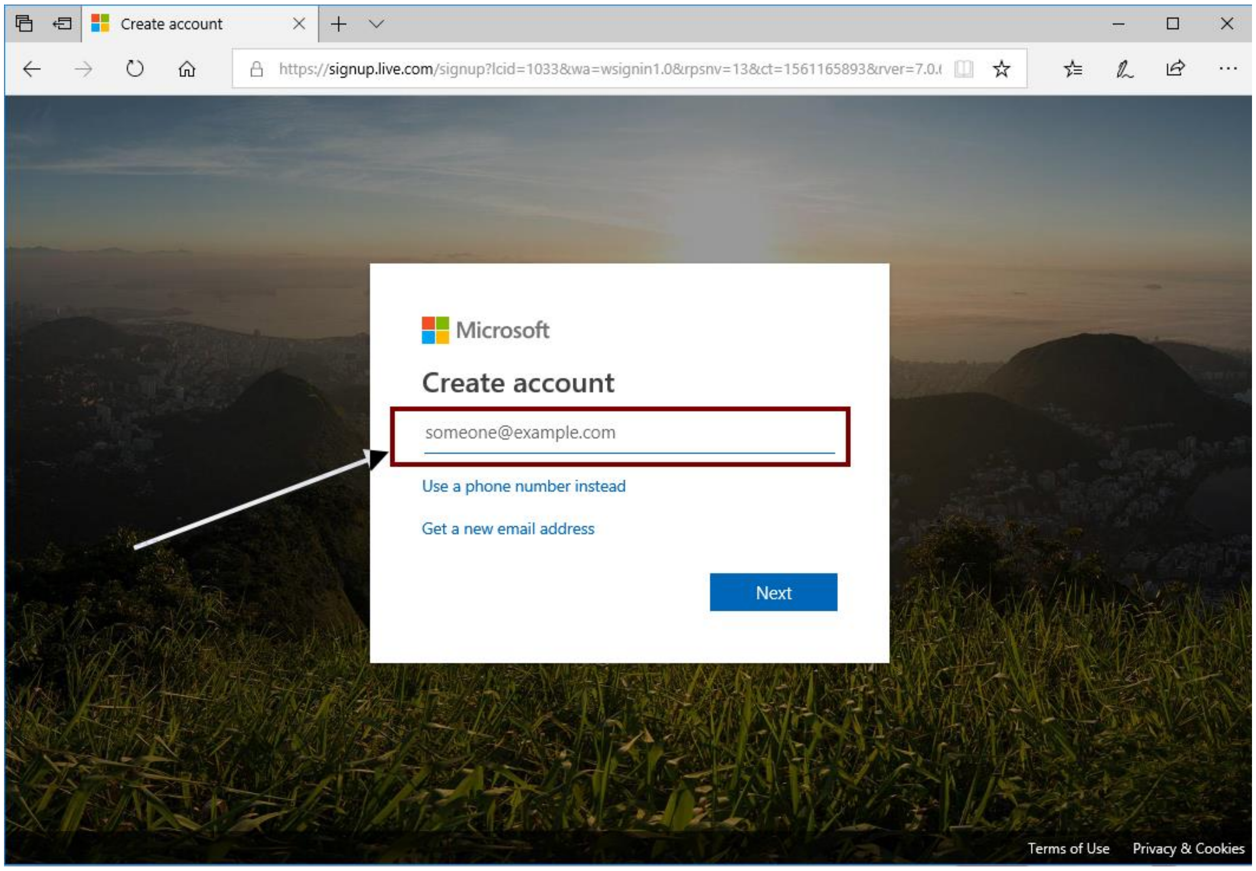

Create account
https://signup.live.com/signup?lcid=1033&wa=wsignin1.0&rpsnv=13&ct=1561165893&rver=7.0.
Microsoft
Create account
someone@example.com
Use a phone number instead
Get a new email address
Next
Terms of Use
Privacy & Cookies

Figure 4: Microsoft account creation form

As suggested by the form displayed in the previous image, we need to type our email address. Then, click **Next** to continue. Now, the web browser displays a form asking for a password. We need to create a password for our new account, and then click **Next**. The following image displays the password form.

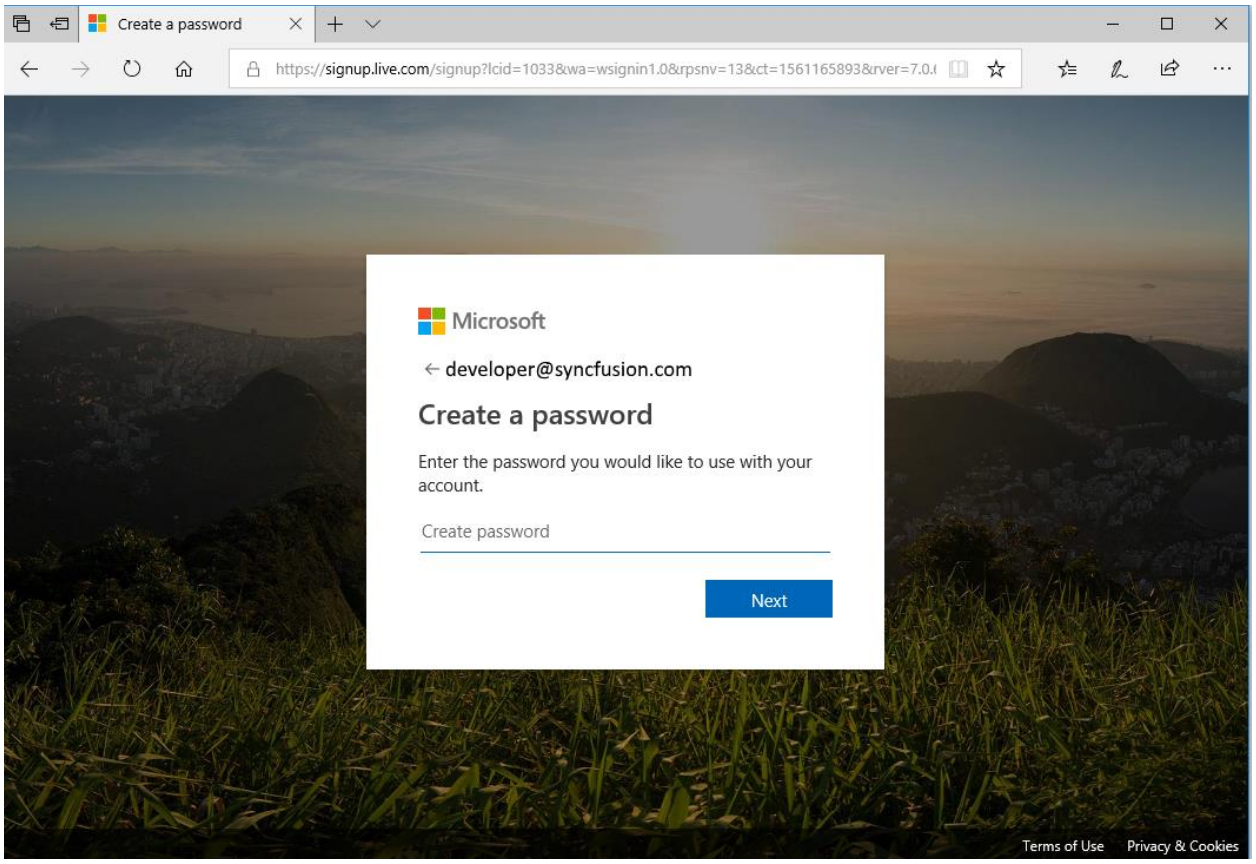
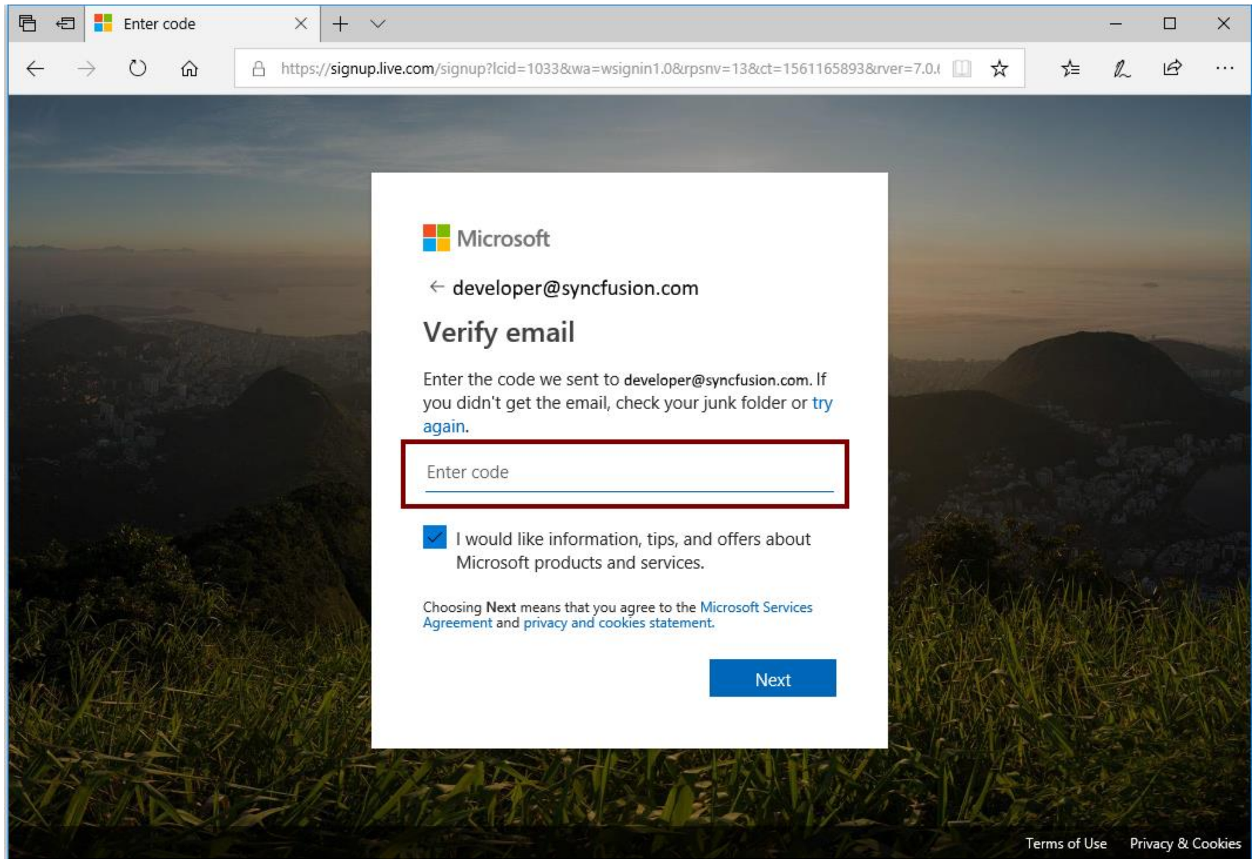

Create a password
https://signup.live.com/signup?lcid=1033&wa=wsignin1.0&rpsnv=13&ct=1561165893&rver=7.0.(
Microsoft
developer@syncfusion.com
Create a password
Enter the password you would like to use with your account.
Create password
Next
Terms of Use
Privacy & Cookies

Figure 5: Create password form

The Microsoft account system asks for a verification code to ensure that we use an email address that belongs to us. We need to check our email inbox and locate the message with the subject line “Verify your email address” from the Microsoft account team. After that, we should find the security code and type it into the form displayed in the following image.


Enter code

https://signup.live.com/signup?lcid=1033&wa=wsignin1.0&rpsnv=13&ct=1561165893&rver=7.0.

Microsoft

developer@syncfusion.com

Verify email

Enter the code we sent to developer@syncfusion.com. If you didn't get the email, check your junk folder or try again.

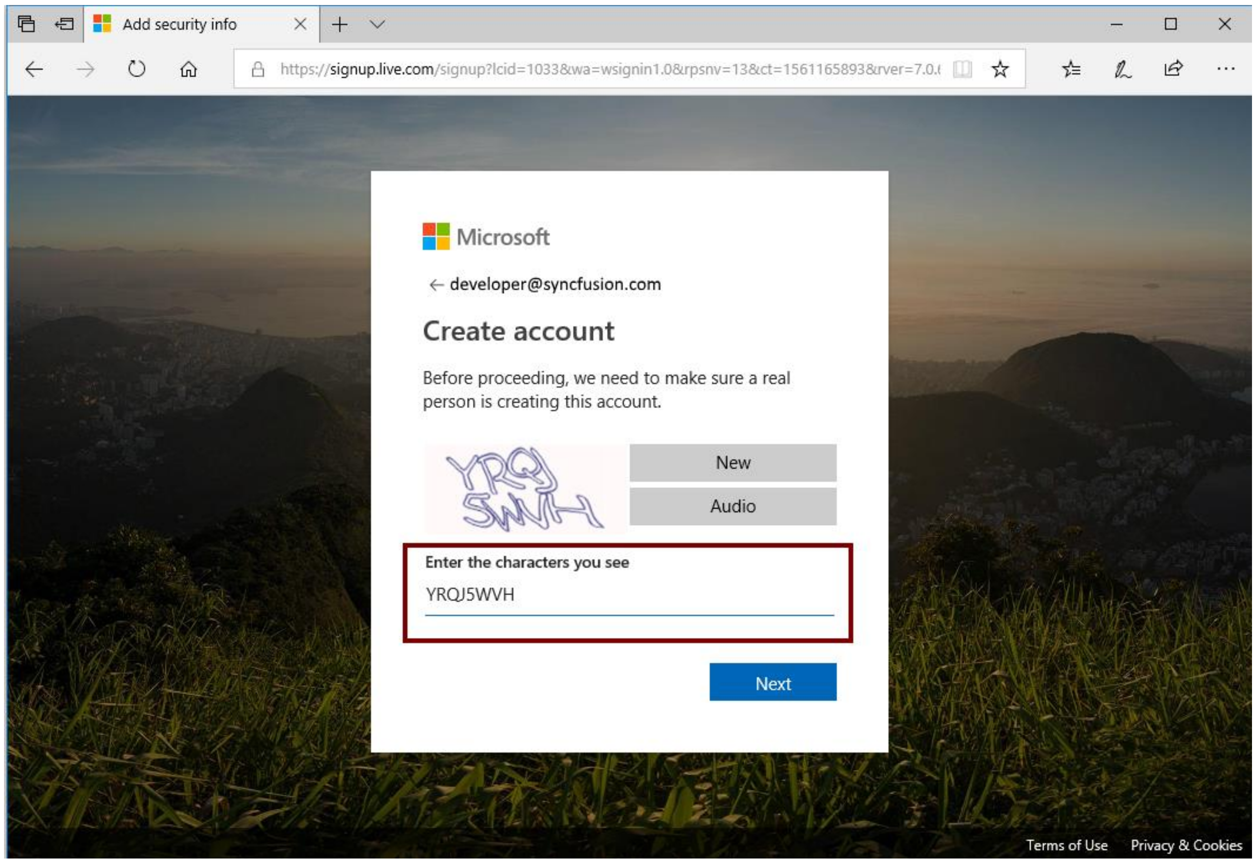
Enter code

I would like information, tips, and offers about Microsoft products and services.

Choosing **Next** means that you agree to the Microsoft Services Agreement and privacy and cookies statement.

Figure 6: Verify email form

Click **Next**, and a form asking for a CAPTCHA will appear. This form helps to ensure an actual human being is creating the account. Type the characters displayed by the CAPTCHA, and click **Next**.



Add security info
https://signup.live.com/signup?lcid=1033&wa=wsignin1.0&rpsnv=13&ct=1561165893&rver=7.0.
Microsoft
New
Audio
Next
Terms of Use
Privacy & Cookies

← developer@syncfusion.com

Create account

Before proceeding, we need to make sure a real person is creating this account.

Enter the characters you see

YRQJ5WVH

Figure 7: The CAPTCHA form

The system creates the account, and the Microsoft account manager home page appears in the web browser.

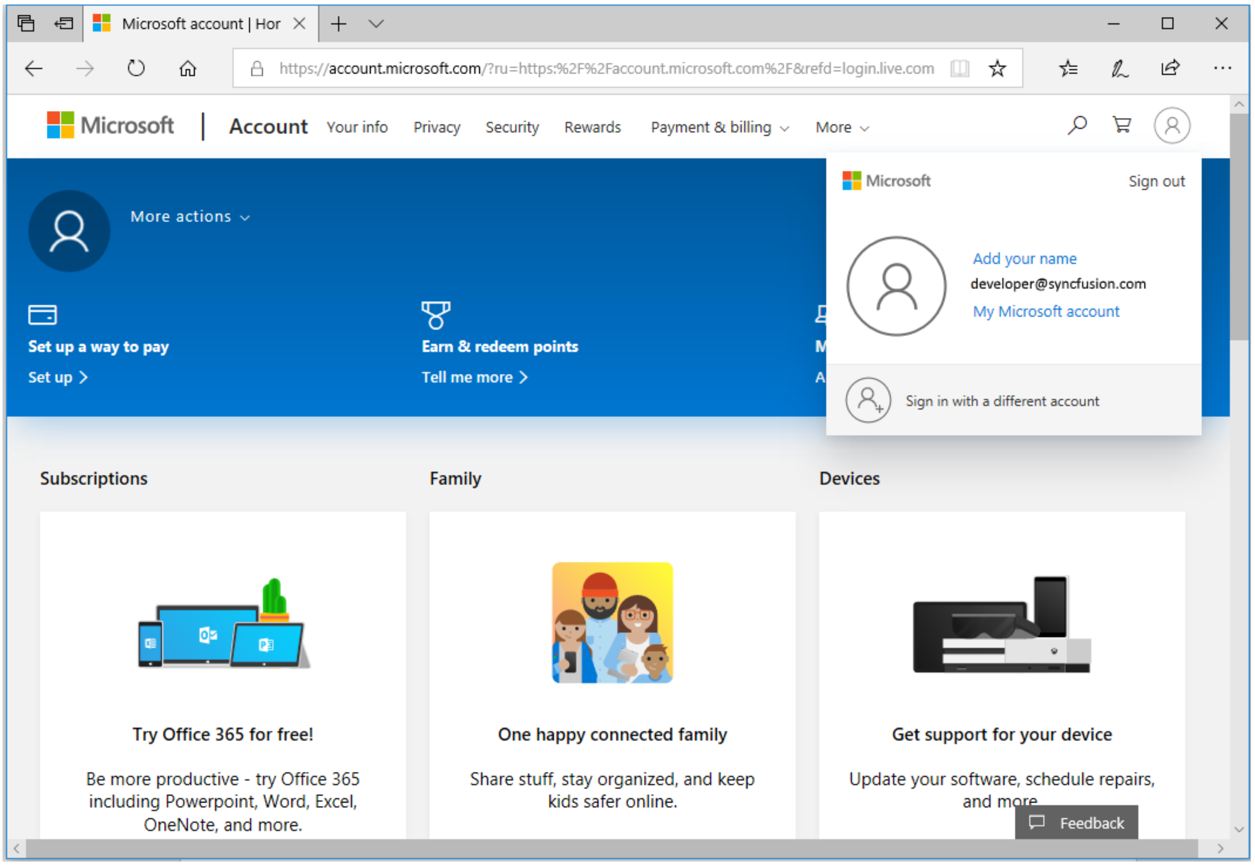

Microsoft account | Hor
https://account.microsoft.com/?ru=https%2F%2Faccount.microsoft.com%2F&refd=login.live.com
Microsoft
Account
Your info
Privacy
Security
Rewards
Payment & billing
More
More actions
Set up a way to pay
Set up
Earn & redeem points
Tell me more
Microsoft
Sign out
Add your name
developer@syncfusion.com
My Microsoft account
Sign in with a different account
Subscriptions
Family
Devices
Try Office 365 for free!
Be more productive - try Office 365 including Powerpoint, Word, Excel, OneNote, and more.
One happy connected family
Share stuff, stay organized, and keep kids safer online.
Get support for your device
Update your software, schedule repairs, and more
Feedback

Figure 8: Microsoft account manager home page

Now we are ready to sign in to the NuGet Gallery.

Using the Microsoft account to access the NuGet Gallery

Once we are on the NuGet Gallery home page, we should click the **Sign in** hyperlink displayed in the following image.

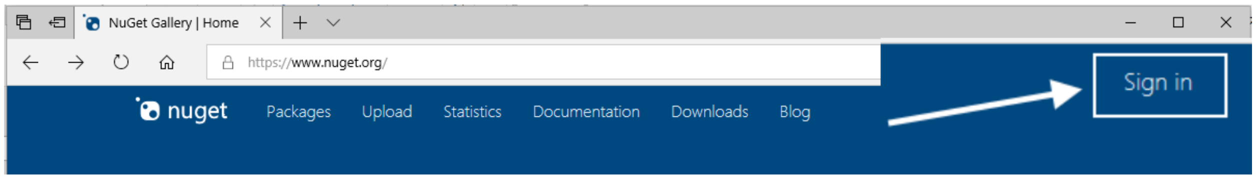


Figure 9: Sign in

The browser will display a form asking for the Microsoft account to sign in to the gallery, as we can see in the following figure.

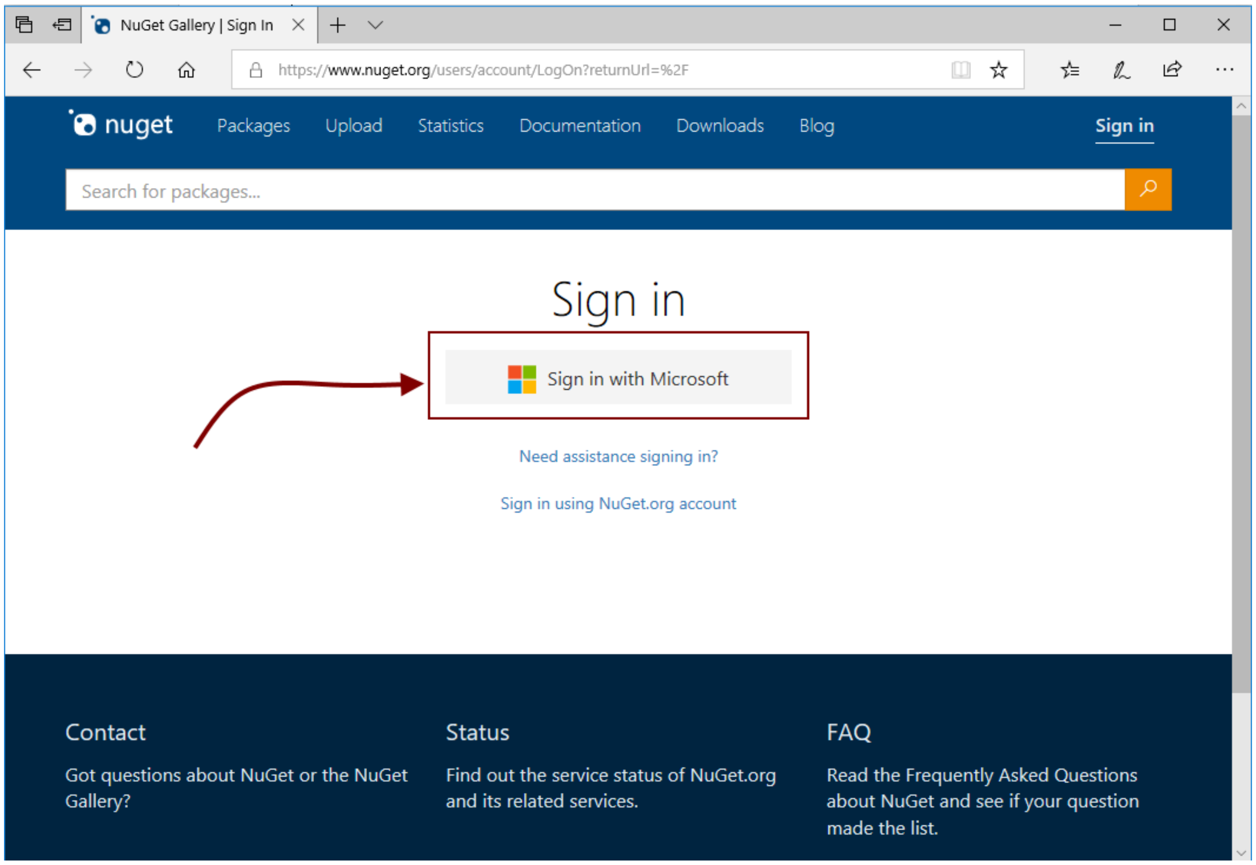

NuGet Gallery | Sign In
https://www.nuget.org/users/account/LogOn?returnUrl=%2F
nuget
Packages
Upload
Statistics
Documentation
Downloads
Blog
Sign in
Search for packages...
Sign in
Sign in with Microsoft
Need assistance signing in?
Sign in using NuGet.org account
Contact
Got questions about NuGet or the NuGet Gallery?
Status
Find out the service status of NuGet.org and its related services.
FAQ
Read the Frequently Asked Questions about NuGet and see if your question made the list.

Figure 10: Signing in to NuGet Gallery, step 1

As suggested by the previous image, we should click **Sign in with Microsoft** to continue the sign-in process. Next, the web browser will show the form displayed in the following figure.

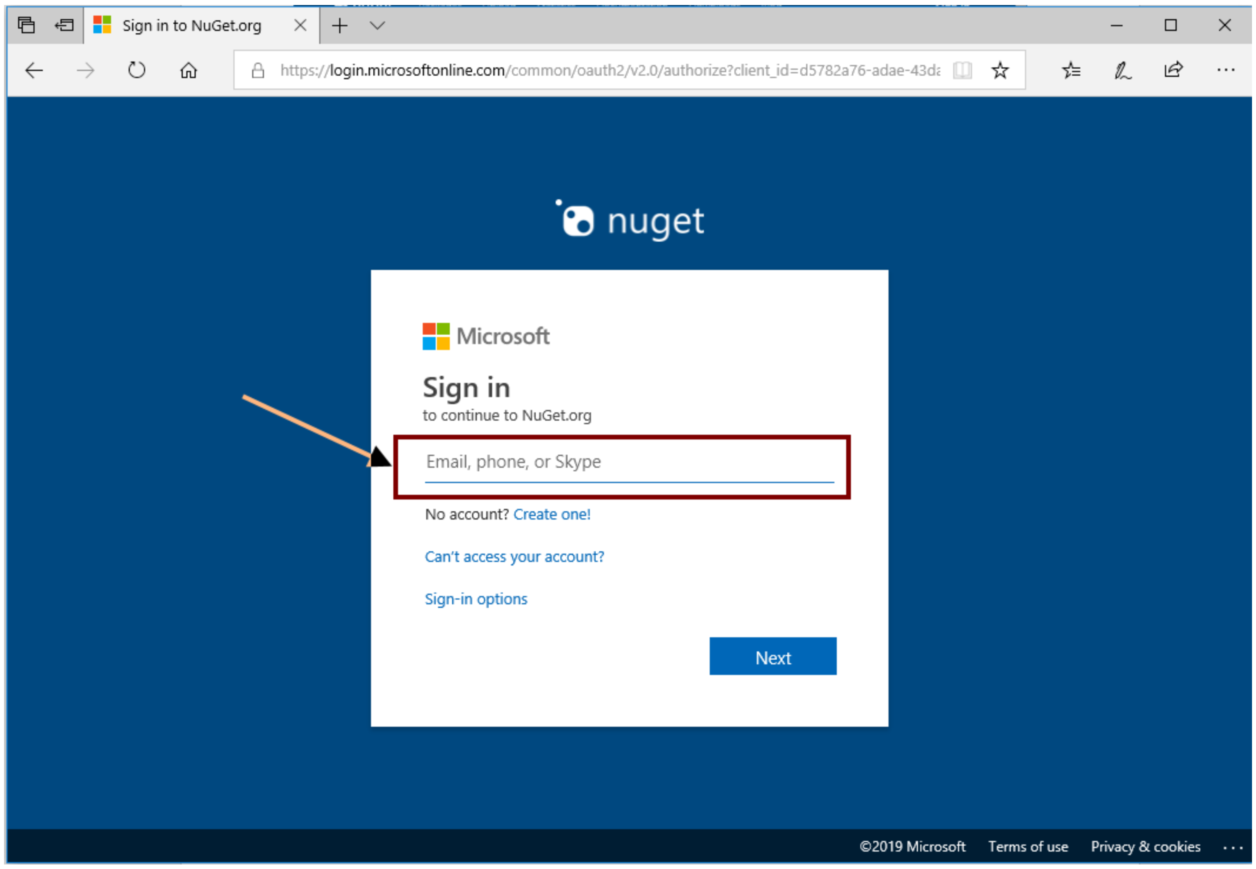
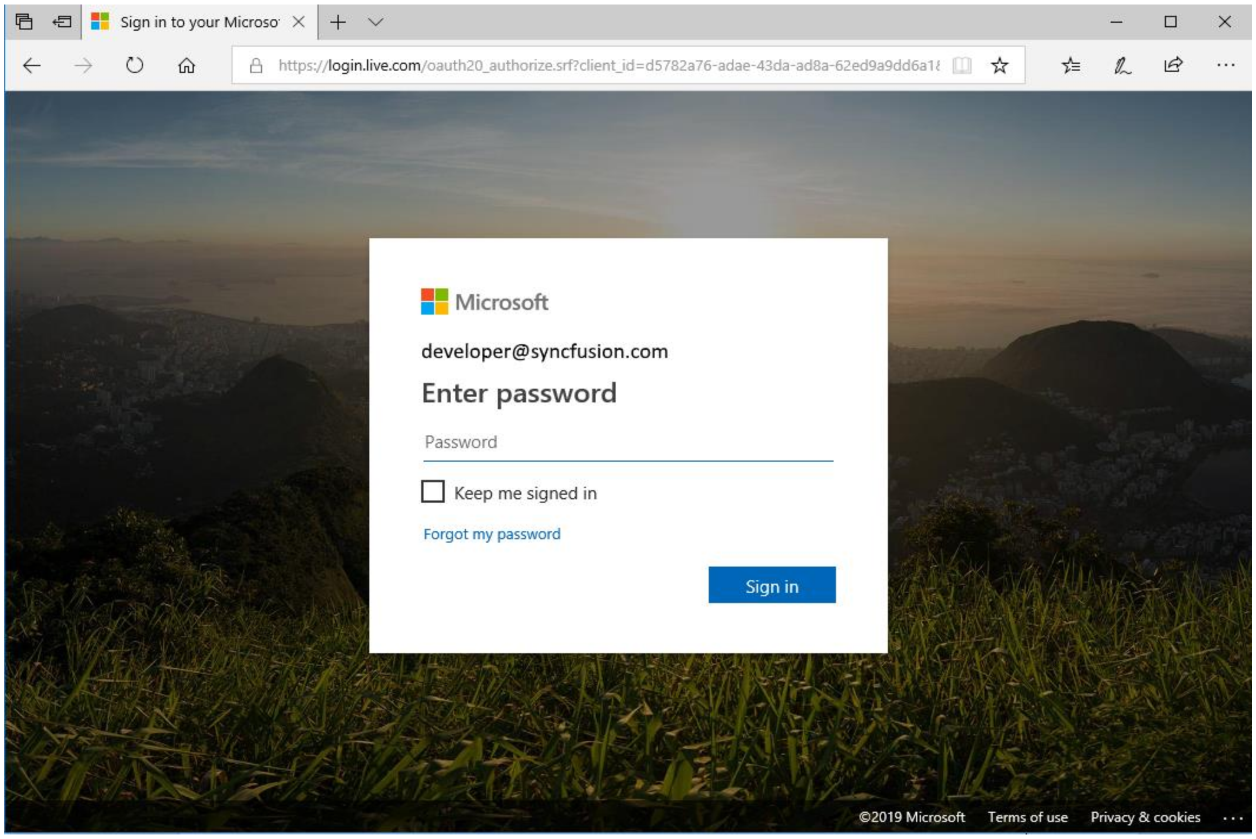


Figure 11: Signing in to NuGet Gallery, step 2

We should type the email address linked to our Microsoft account and click **Next**. A form appears in the web browser, asking for the password we assigned to our Microsoft account. Again, we should type the password and click **Next**. If everything is OK, the NuGet Gallery home page appears with our username located at the top-right corner.


Sign in to your Microso

https://login.live.com/oauth20_authorize.srf?client_id=d5782a76-adae-43da-ad8a-62ed9a9dd6a1

Microsoft

developer@syncfusion.com

Enter password

Password

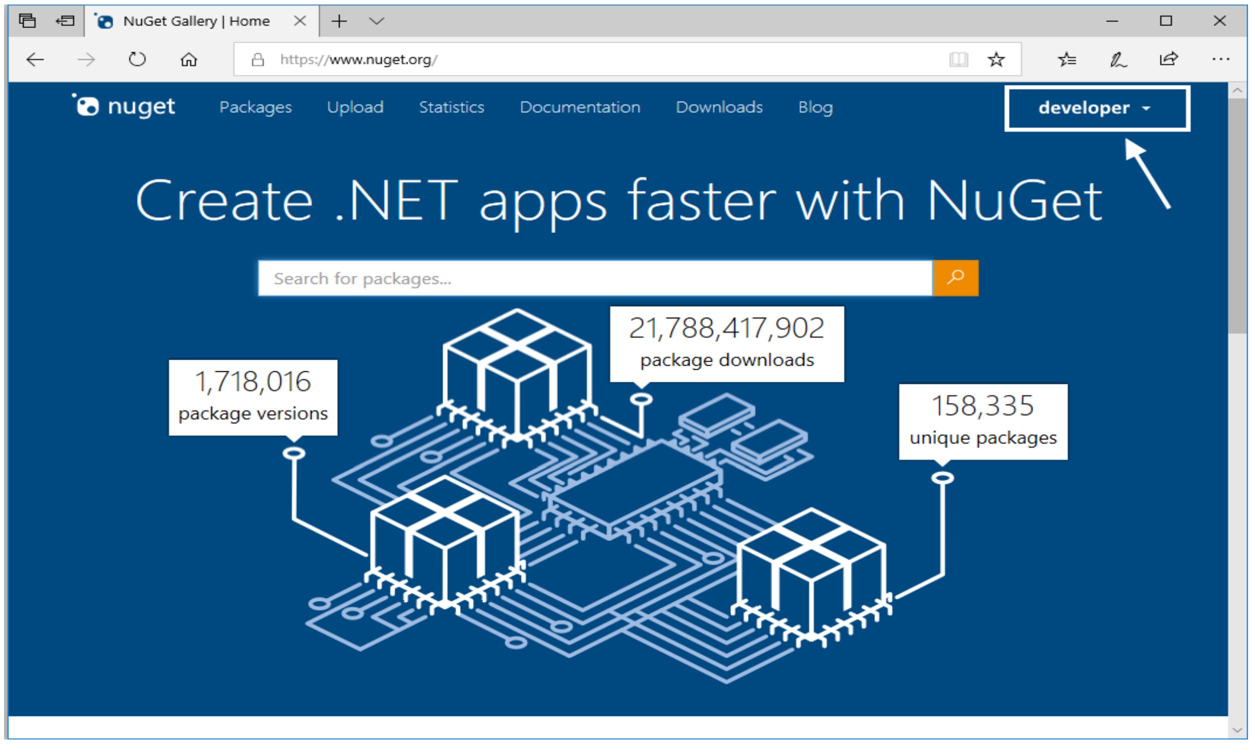
Keep me signed in

Forgot my password

Sign in

©2019 Microsoft Terms of use Privacy & cookies

Figure 12: Signing in to NuGet Gallery, step 3


NuGet Gallery | Home

https://www.nuget.org/

nuget Packages Upload Statistics Documentation Downloads Blog developer

Create .NET apps faster with NuGet

Search for packages...

1,718,016 package versions

21,788,417,902 package downloads

158,335 unique packages

Figure 13: NuGet Gallery home page after signing in

Uploading a package to the NuGet Gallery

If we have a code package to share, we can upload this code to the NuGet Gallery by clicking the **Upload** hyperlink displayed in the following figure.

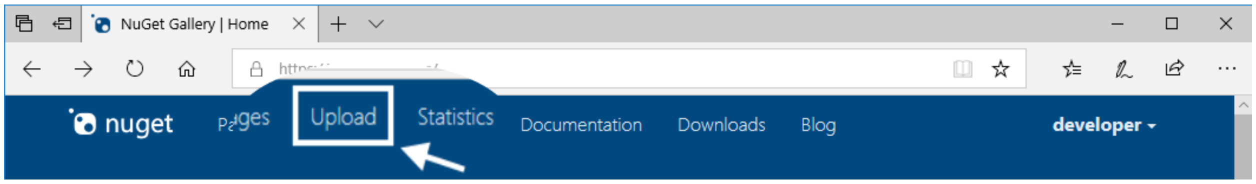
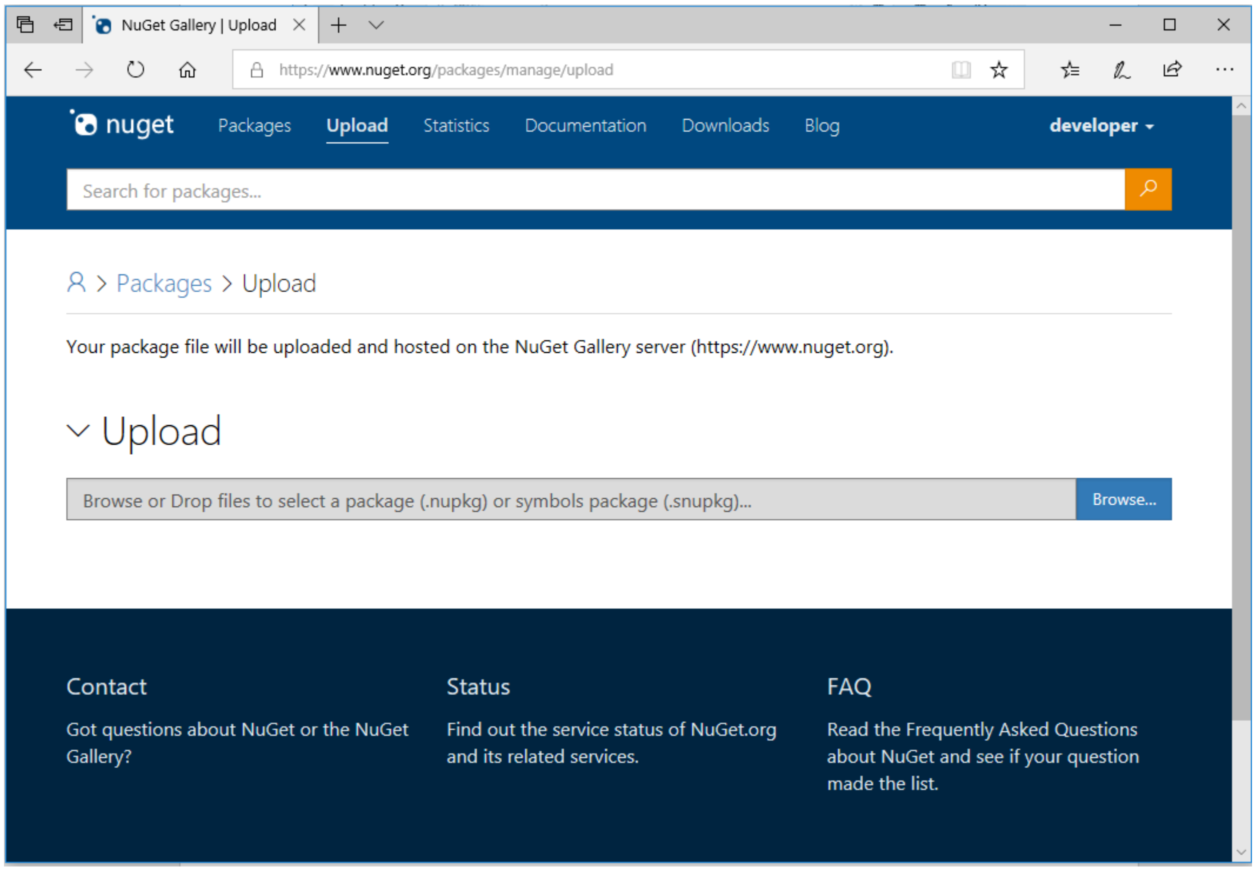


Figure 14: Upload hyperlink in NuGet Gallery home page

The Upload form will appear and prompt us for the file name of the package we want to upload.



Your package file will be uploaded and hosted on the NuGet Gallery server (https://www.nuget.org).

Upload

Browse or Drop files to select a package (.nupkg) or symbols package (.snupkg)... Browse...

Contact

Got questions about NuGet or the NuGet Gallery?

Status

Find out the service status of NuGet.org and its related services.

FAQ

Read the Frequently Asked Questions about NuGet and see if your question made the list.

Figure 15: Upload form in NuGet Gallery

Click **Browse**, and the Open dialog box will appear on the screen. For the purposes of this book, I placed a file named **myfirstpackage.nupkg** in the Downloads special folder. I should select this file and click **Open**.

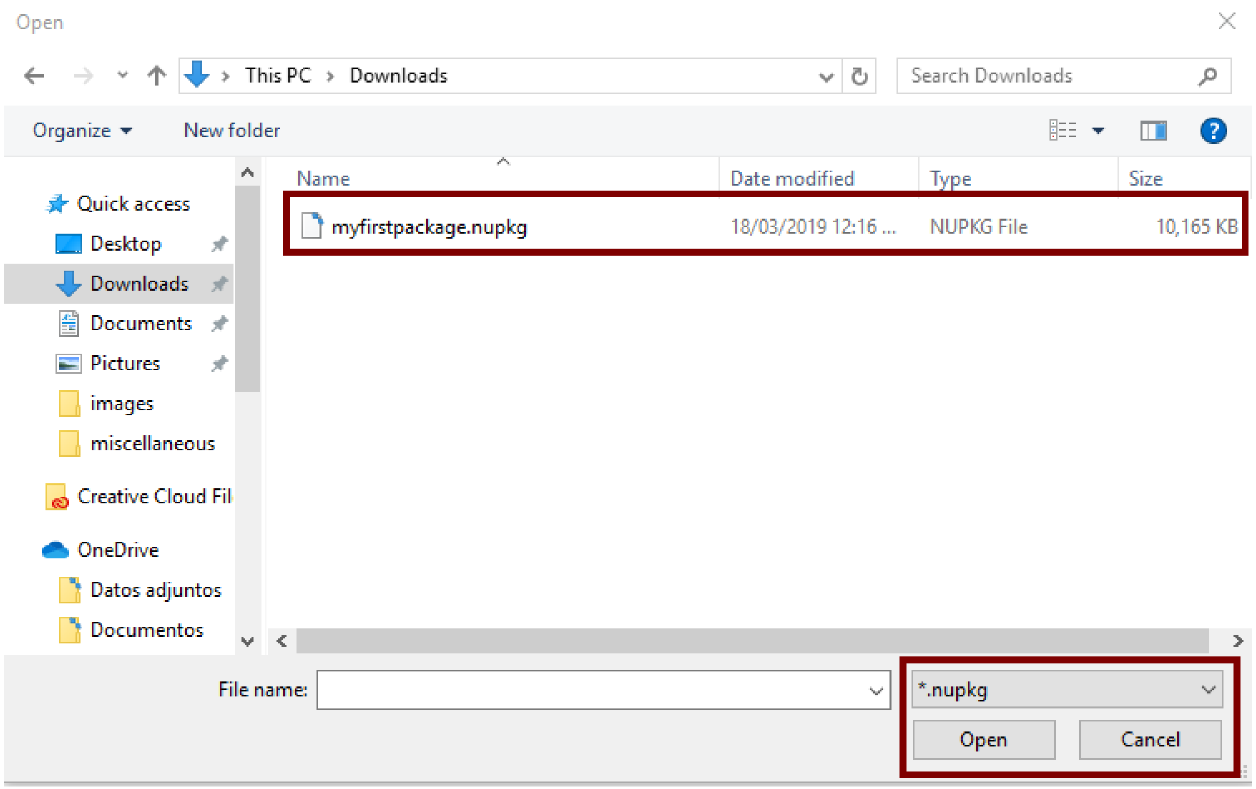
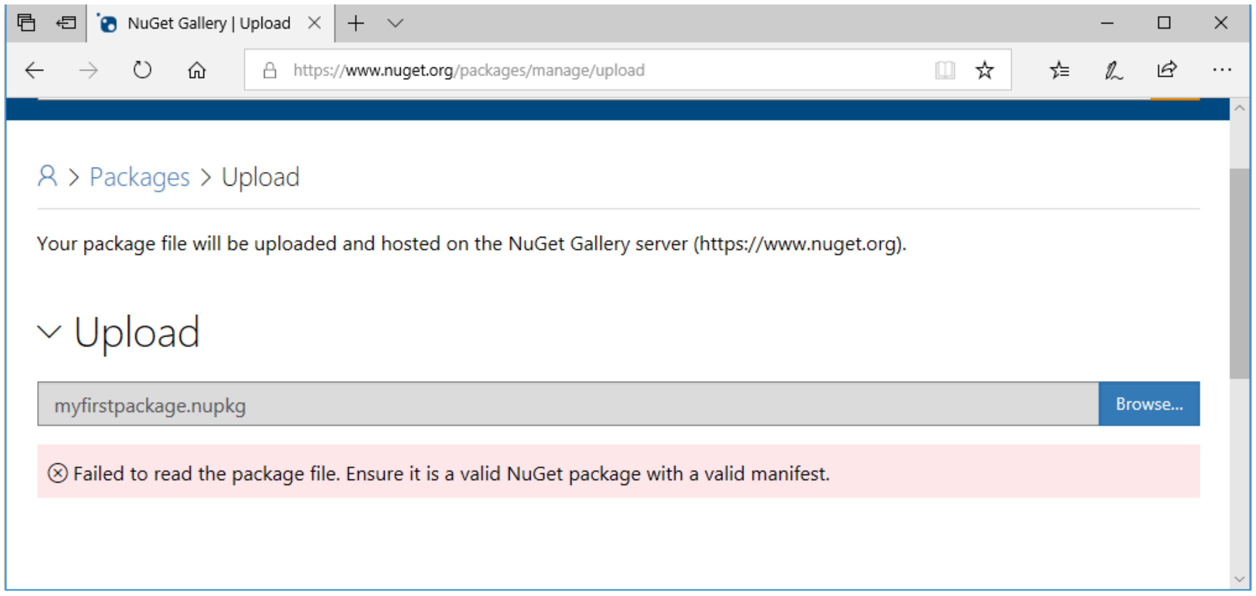


Figure 16: The Open dialog box when selecting a package to upload

The NuGet Gallery starts to process the file. The file I used for this example is a dummy package, so NuGet Gallery will not upload the package, and will display the message shown in the following figure.



Your package file will be uploaded and hosted on the NuGet Gallery server (https://www.nuget.org).

Failed to read the package file. Ensure it is a valid NuGet package with a valid manifest.

Figure 17: NuGet package upload failed

Note: The dummy NuGet package is not included in the examples available for downloading; you can try to upload a real package to the NuGet Gallery for practice purposes.

Chapter summary

NuGet is an open-source package manager developed by the Microsoft .NET Foundation, and designed for the Microsoft development platform. The first release of NuGet, called NuPack, appeared in 2010. NuGet is free and licensed under the Apache License 2.0, and comes preinstalled with Visual Studio 2012 and later.

NuGet is primarily a tool for sharing .NET code in the form of packages. The roles involved in code-sharing using NuGet are creators, hosts, and consumers.

A NuGet package consists of .NET code compiled as one or several assemblies, delivered in a single compressed file with a .nupkg extension, and the NuGet Gallery is the public repository for sharing NuGet packages. To access the NuGet Gallery all we need to do is navigate to https://www.nuget.org.

The NuGet Gallery displays three metrics: the number of downloads made by all .NET developers, the number of packages the gallery hosts, and the total number of package versions hosted.

We can browse the NuGet Gallery without an account. However, if we want to upload a package manually, we need to create an account to sign in to the gallery.

The NuGet Gallery has deprecated nuget.org accounts and supports only Microsoft accounts at this time. Therefore, we need to create a Microsoft account if we do not already have one. We also need a valid email address before creating the account. Once we create the account, we click the **Sign in** hyperlink in the NuGet Gallery home page to access the gallery.

If we have a code package to share, we can upload this code to the NuGet Gallery using the **Upload** hyperlink in the NuGet Gallery home page. Then, an upload form will appear asking for the file name of the package we want to upload. We should click **Browse** to bring up the Open dialog box, and then select the file to upload. For the purposes of this book, I placed a file named **myfirstpackage.nupkg** in the Downloads special folder, selected this file from the Open dialog, and clicked **Open**. Because the file is a dummy package, the NuGet Gallery will show an error message instead of uploading the package.

Chapter 2 The NuGet Server

What is a NuGet server?

A NuGet server is an application employed to host NuGet packages. The NuGet Gallery is one, for example, albeit a public one as mentioned in Chapter 1. The purpose of this chapter is to show you how to create a local NuGet server in order to keep your code private.

For the purposes of this book, we first need to set up IIS on our local computer before creating our own NuGet server.

Setting up IIS to host a NuGet server

The first thing we should do is go to the **Control Panel** and click the **Programs and Features** icon.

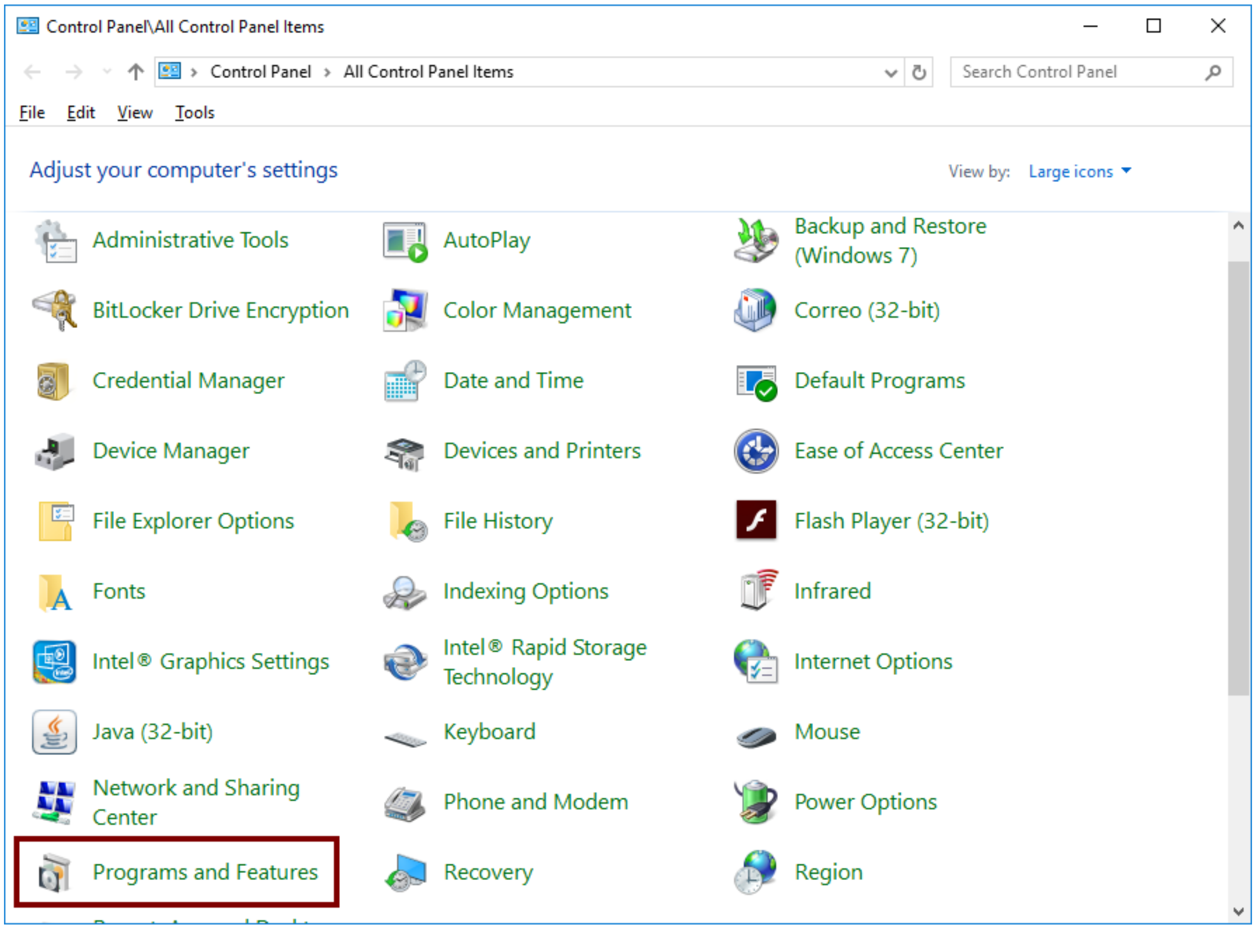
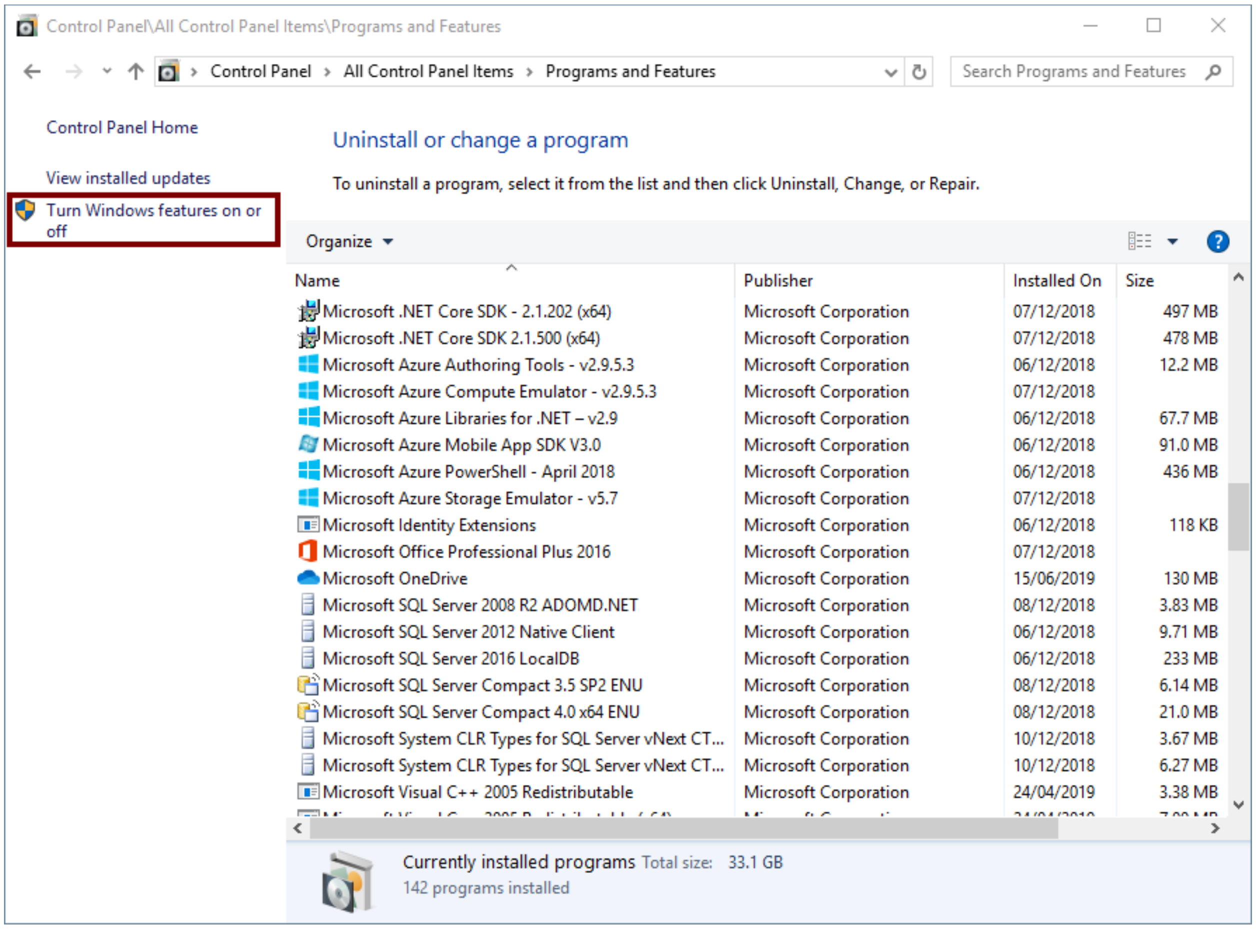


Figure 18: "Programs and Features" in Control Panel

Now, in the **Programs and Features** dialog, we should click the **Turn Windows features on or off** hyperlink located at the left side.


Control Panel\All Control Panel Items\Programs and Features
Control Panel > All Control Panel Items > Programs and Features
Search Programs and Features
Control Panel Home
View installed updates
Turn Windows features on or off
Organize

Uninstall or change a program

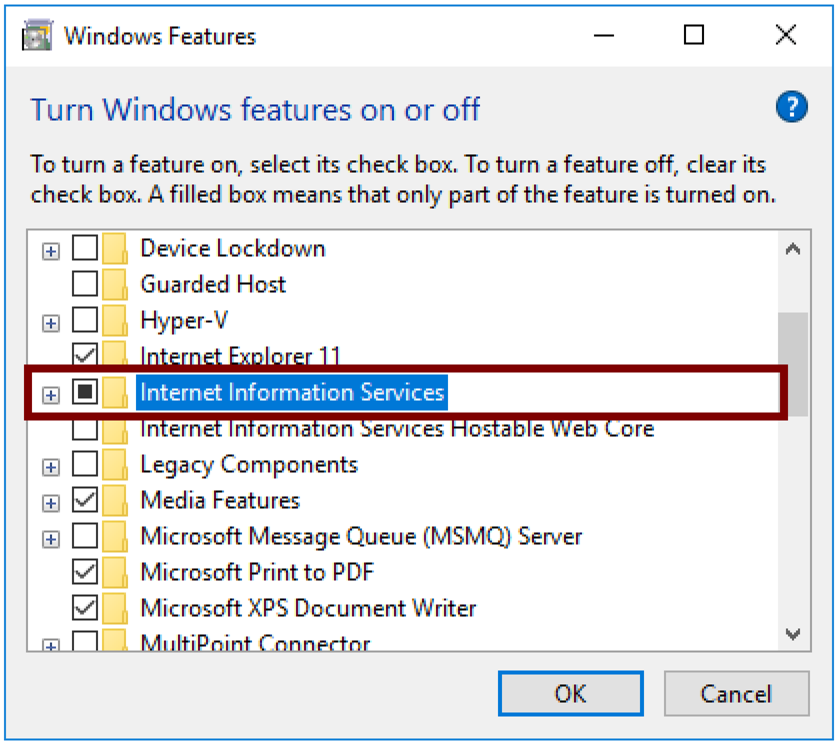
To uninstall a program, select it from the list and then click Uninstall, Change, or Repair.

Name	Publisher	Installed On	Size
Microsoft .NET Core SDK - 2.1.202 (x64)	Microsoft Corporation	07/12/2018	497 MB
Microsoft .NET Core SDK 2.1.500 (x64)	Microsoft Corporation	07/12/2018	478 MB
Microsoft Azure Authoring Tools - v2.9.5.3	Microsoft Corporation	06/12/2018	12.2 MB
Microsoft Azure Compute Emulator - v2.9.5.3	Microsoft Corporation	07/12/2018	
Microsoft Azure Libraries for .NET – v2.9	Microsoft Corporation	06/12/2018	67.7 MB
Microsoft Azure Mobile App SDK V3.0	Microsoft Corporation	06/12/2018	91.0 MB
Microsoft Azure PowerShell - April 2018	Microsoft Corporation	06/12/2018	436 MB
Microsoft Azure Storage Emulator - v5.7	Microsoft Corporation	07/12/2018	
Microsoft Identity Extensions	Microsoft Corporation	06/12/2018	118 KB
Microsoft Office Professional Plus 2016	Microsoft Corporation	07/12/2018	
Microsoft OneDrive	Microsoft Corporation	15/06/2019	130 MB
Microsoft SQL Server 2008 R2 ADOMD.NET	Microsoft Corporation	08/12/2018	3.83 MB
Microsoft SQL Server 2012 Native Client	Microsoft Corporation	06/12/2018	9.71 MB
Microsoft SQL Server 2016 LocalDB	Microsoft Corporation	06/12/2018	233 MB
Microsoft SQL Server Compact 3.5 SP2 ENU	Microsoft Corporation	08/12/2018	6.14 MB
Microsoft SQL Server Compact 4.0 x64 ENU	Microsoft Corporation	08/12/2018	21.0 MB
Microsoft System CLR Types for SQL Server vNext CT...	Microsoft Corporation	10/12/2018	3.67 MB
Microsoft System CLR Types for SQL Server vNext CT...	Microsoft Corporation	10/12/2018	6.27 MB
Microsoft Visual C++ 2005 Redistributable	Microsoft Corporation	24/04/2019	3.38 MB

Currently installed programs Total size: 33.1 GB
142 programs installed

Figure 19: "Turn Windows features on or off" hyperlink

As soon as the **Windows Features** dialog box appears, we should locate the **Internet Information Services** node and select it. The filled check box displayed after clicking indicates that Windows will install the default components of IIS only.


Windows Features

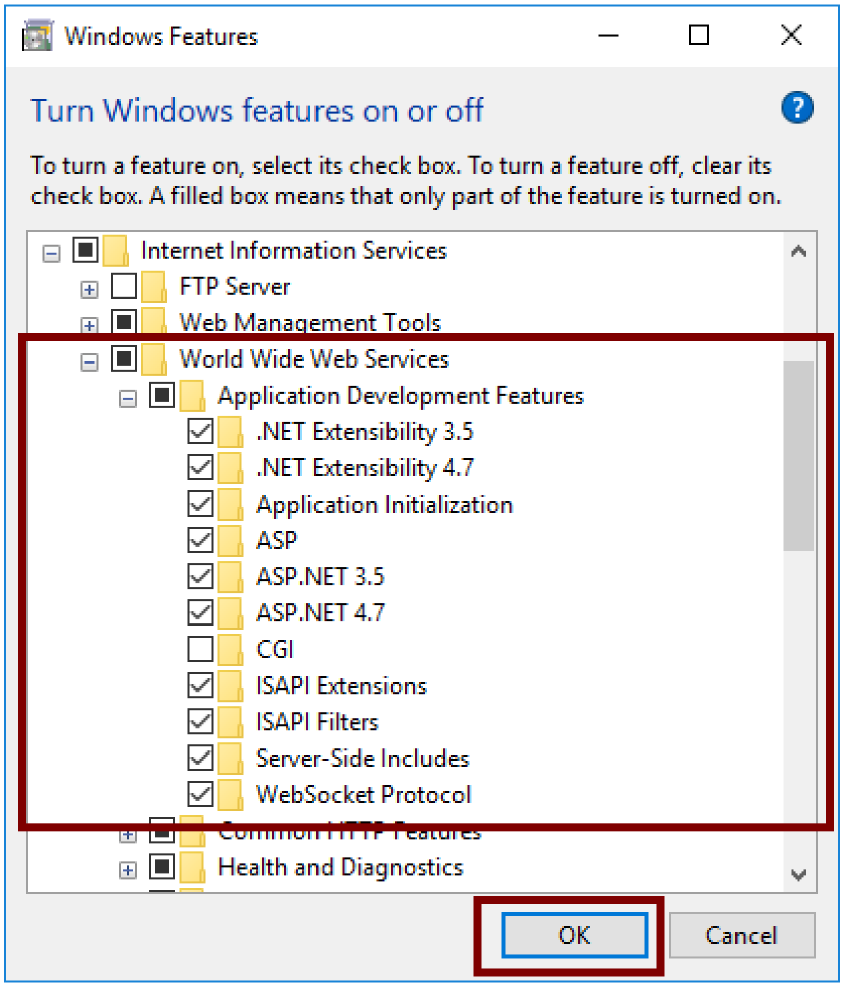
Turn Windows features on or off

To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- [ ] Device Lockdown
- [ ] Guarded Host
- [ ] Hyper-V
- [x] Internet Explorer 11
- [■] Internet Information Services
- [ ] Internet Information Services Hostable Web Core
- [ ] Legacy Components
- [x] Media Features
- [ ] Microsoft Message Queue (MSMQ) Server
- [x] Microsoft Print to PDF
- [x] Microsoft XPS Document Writer
- [ ] MultiPoint Connector

Figure 20: Selecting the "Internet Information Services" feature

For the purposes of hosting our NuGet server in IIS, we need to open the **Internet Information Services** node, and then the **Application Development Features** node, which is a child node of **World Wide Web Services**. Now, we are going to select all the features displayed as child nodes, except **CGI**, as shown in the following image.


Windows Features

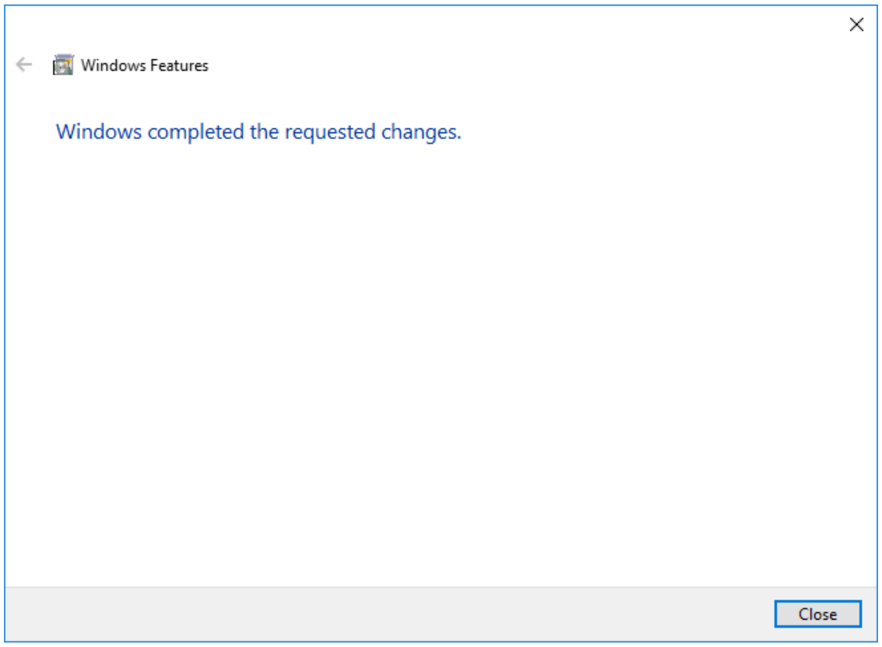
Turn Windows features on or off

To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- ■ Internet Information Services
  - ☐ FTP Server
  - ■ Web Management Tools
  - ■ World Wide Web Services
    - ■ Application Development Features
      - ✓ .NET Extensibility 3.5
      - ✓ .NET Extensibility 4.7
      - ✓ Application Initialization
      - ✓ ASP
      - ✓ ASP.NET 3.5
      - ✓ ASP.NET 4.7
      - ☐ CGI
      - ✓ ISAPI Extensions
      - ✓ ISAPI Filters
      - ✓ Server-Side Includes
      - ✓ WebSocket Protocol
    - ■ Common HTTP Features
    - ■ Health and Diagnostics

Figure 21: Selecting "Application Development Features" components

Now, click **OK** to start IIS installation. A progress dialog will appear, indicating how installation is proceeding. When the installation ends, the dialog box displayed in the following figure will appear.


Windows Features

Windows completed the requested changes.

Figure 22: Dialog box indicating that installation ended

Click **Close** to finish the process. Next, we need to test the IIS installation. Open your web browser and type http://localhost into the address bar. If everything is OK, the home page for our local web server will appear, as shown in the following figure.

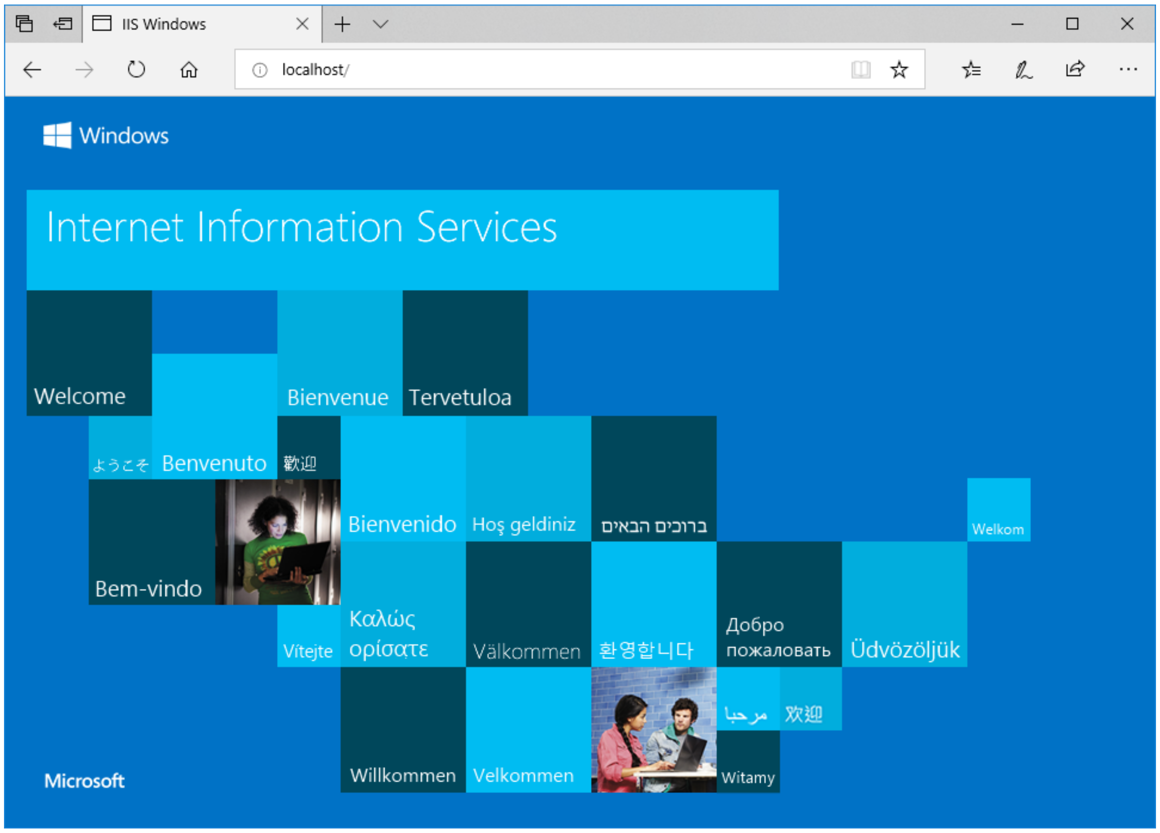


Figure 23: Local web server home page

Creating the NuGet server with Visual Studio

To make our own NuGet server, we will start by creating a new ASP.NET Web application project using Visual Studio. For the purposes of this book, we should save the project into the **Documents\Visual Studio 2017\Projects** folder. In addition, we will name this project **MyNuGetServer**.

Creating NuGet server project in Visual Studio

First, we will launch Visual Studio using the **Run as administrator** option. Then, we will click **File** > **New** > **Project** in the Visual Studio menu bar.

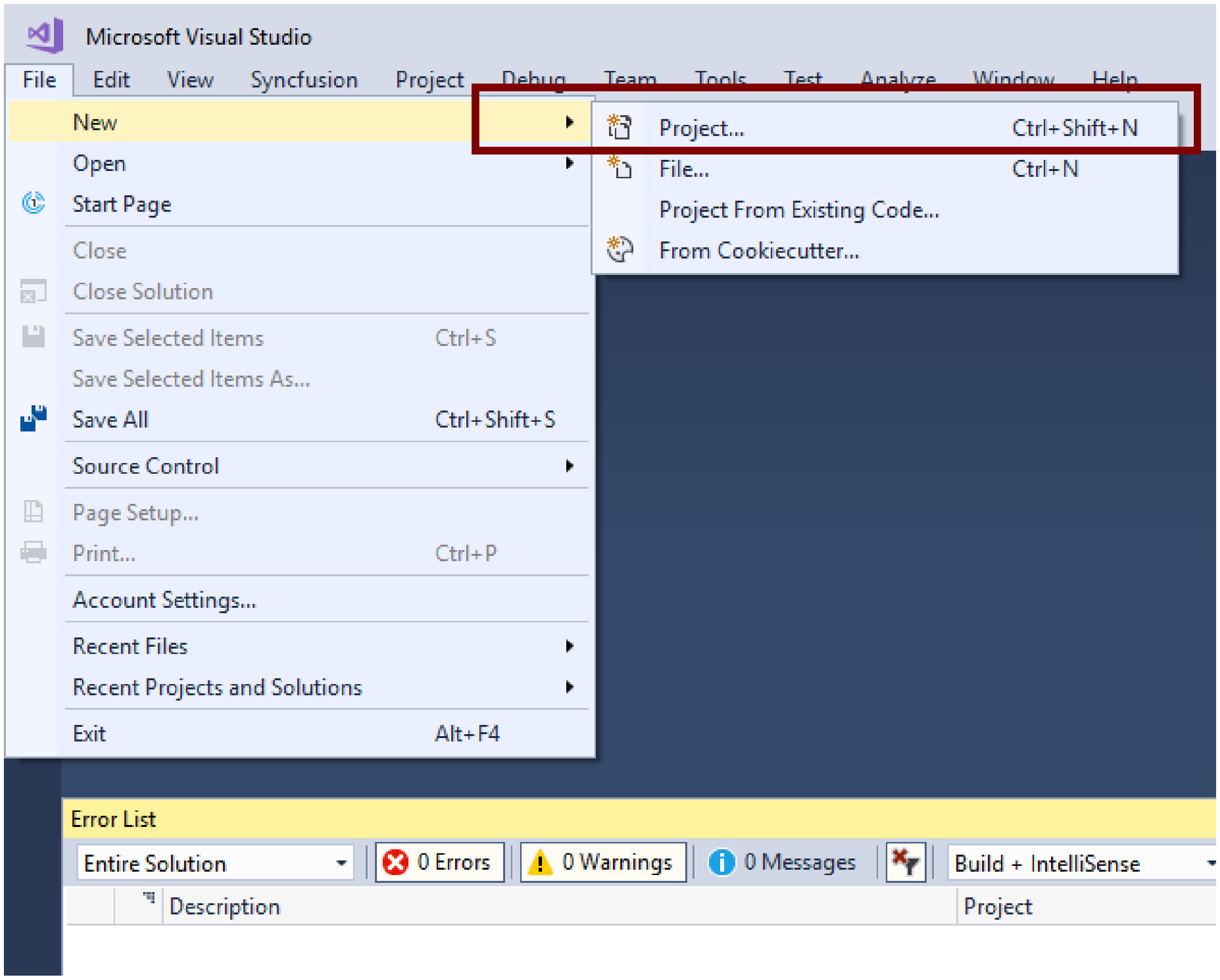
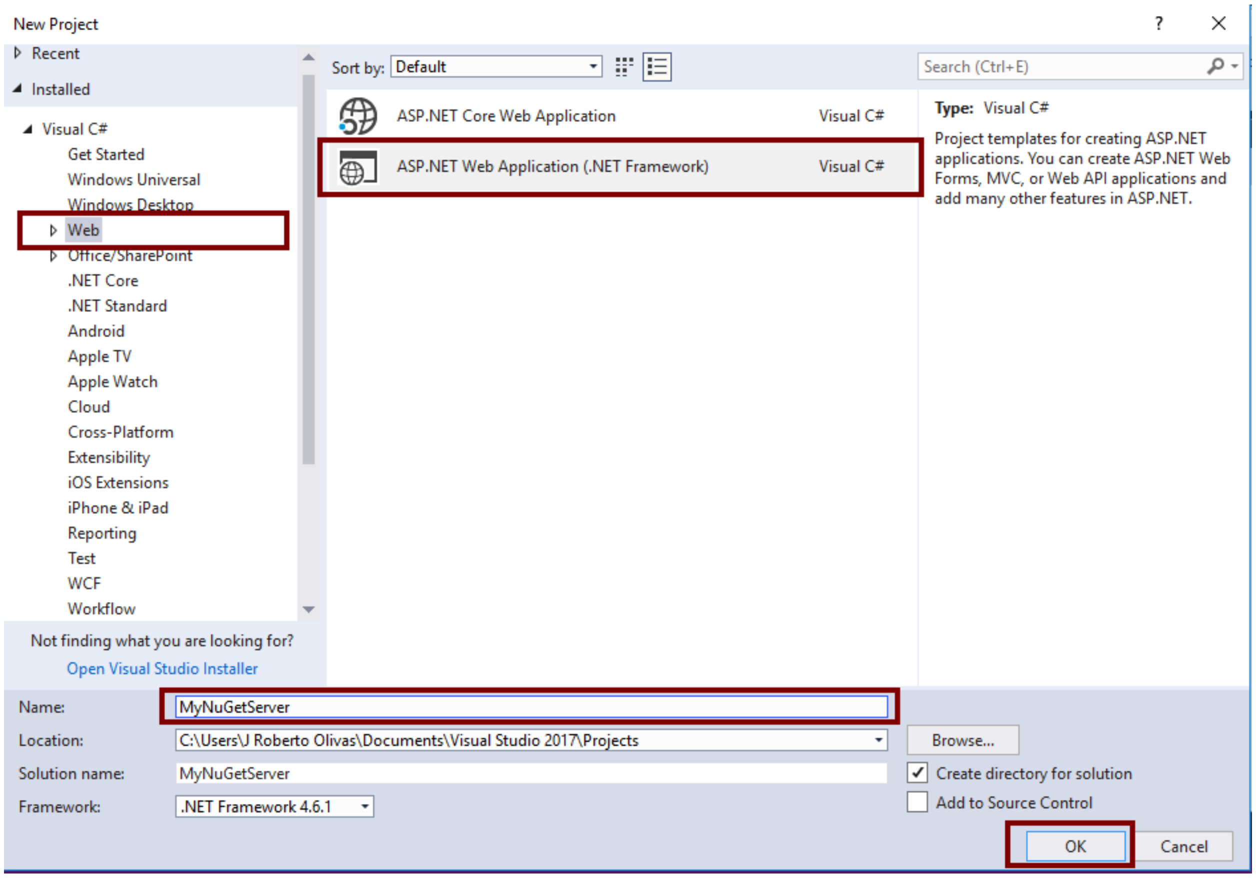


Figure 24: File > New > Project item in Visual Studio

Select **ASP.NET Web Application** from the **Web** section and type **MyNuGetServer** in the **Name** text box. We are using the classic ASP.NET Web application template rather than the newer ASP.NET Core Web application because we do not need a cross-platform system. Click **OK** to complete the process.



Type: Visual C#

Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Name: MyNuGetServer

Location: C:\Users\J Roberto Olivas\Documents\Visual Studio 2017\Projects

Solution name: MyNuGetServer

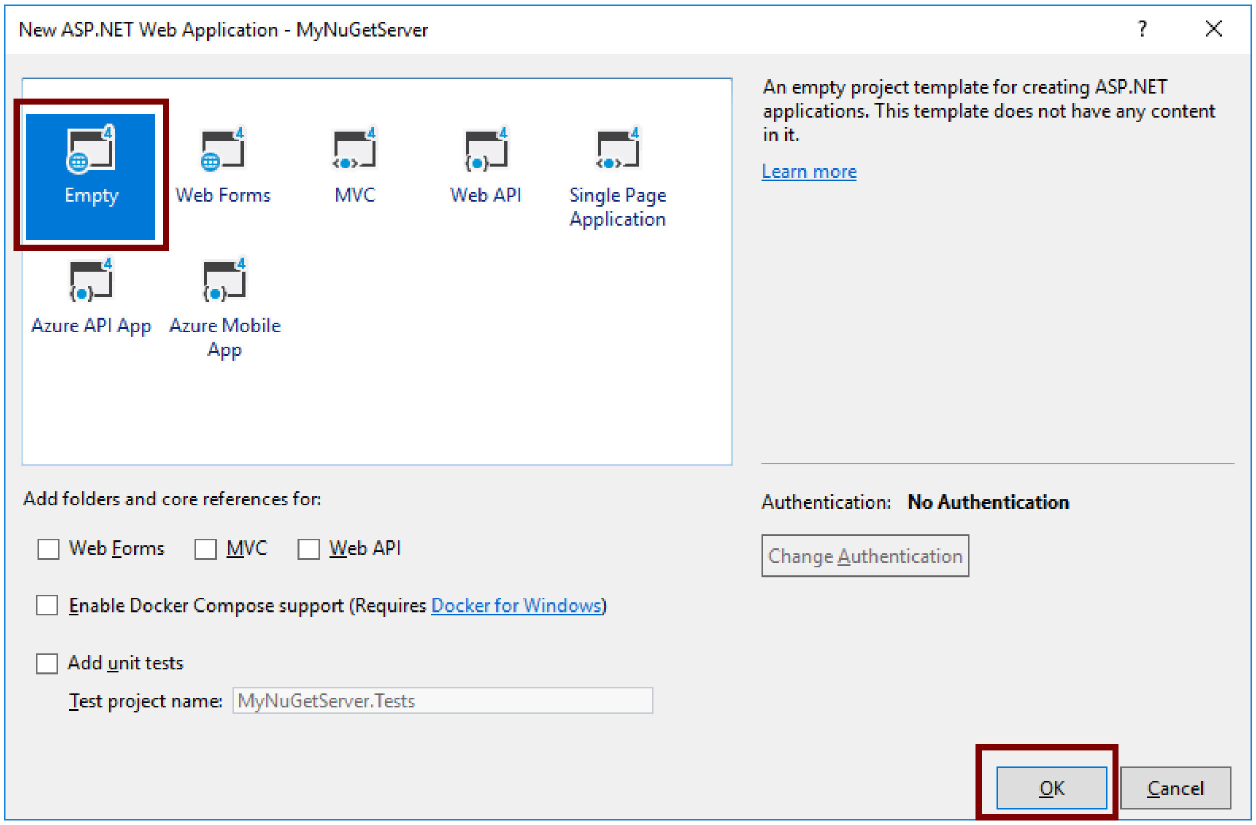
Framework: .NET Framework 4.6.1

[✓] Create directory for solution

[ ] Add to Source Control

Figure 25: Creating the NuGet server project in Visual Studio

Now, Visual Studio asks for the kind of template we will use for the project. Click the **Empty** template and click **OK**. After that, Visual Studio will create the project.


New ASP.NET Web Application - MyNuGetServer

An empty project template for creating ASP.NET applications. This template does not have any content in it.

Learn more

Add folders and core references for:

- [ ] Web Forms
- [ ] MVC
- [ ] Web API
- [ ] Enable Docker Compose support (Requires Docker for Windows)
- [ ] Add unit tests

Test project name: MyNuGetServer.Tests

Authentication: **No Authentication**

Figure 26: Choosing a Web Project Template

Now, the **Solution Explorer** in Visual Studio should look like the following figure.

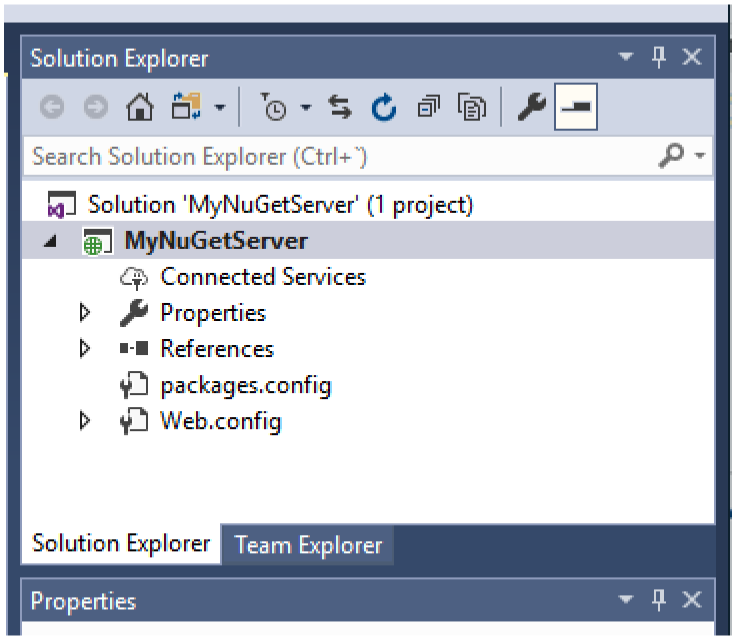


Figure 27: MyNuGetServer project in the Solution Explorer

Adding the NuGet.Server package to the project

Our NuGet server project will need a package named NuGet.Server in order to host its own NuGet packages on our local web server. This package is public, and we can download it from the NuGet Gallery by right-clicking the **MyNuGetServer** node in the **Solution Explorer**, and then clicking the **Manage NuGet Packages** item in the context menu.

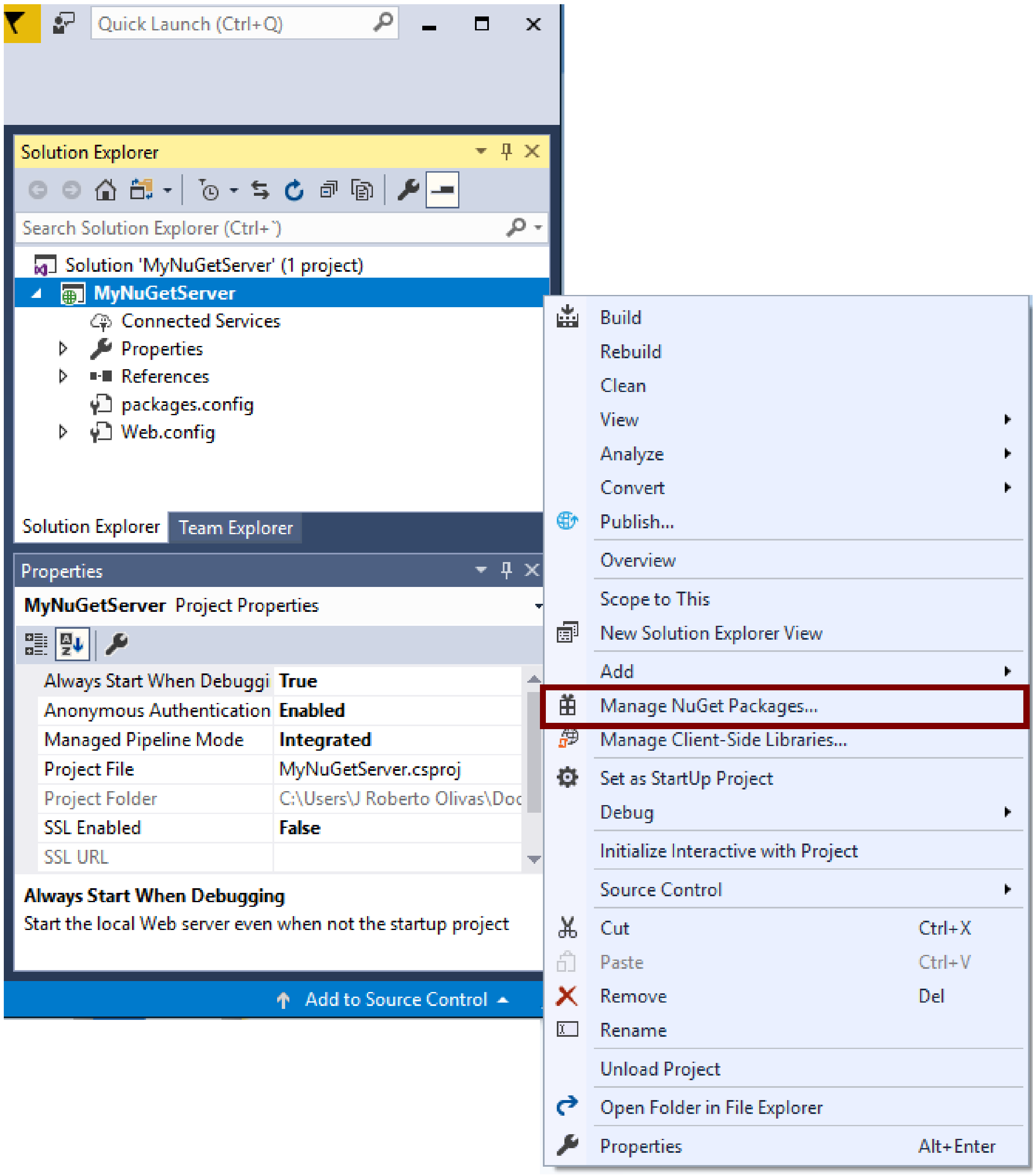
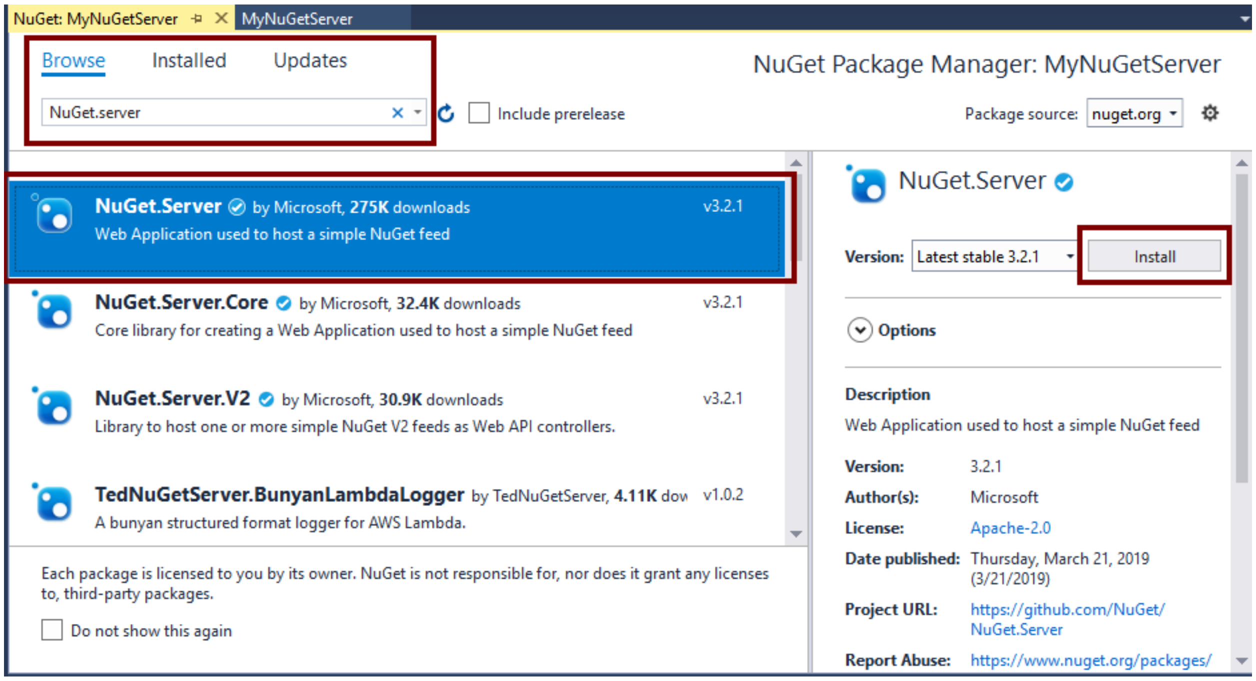

Quick Launch (Ctrl+Q)
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'MyNuGetServer' (1 project)
MyNuGetServer
Connected Services
Properties
References
packages.config
Web.config
Solution Explorer
Team Explorer
Properties
MyNuGetServer Project Properties
Always Start When Debuggi True
Anonymous Authentication Enabled
Managed Pipeline Mode Integrated
Project File MyNuGetServer.csproj
Project Folder C:\Users\J Roberto Olivas\Doc
SSL Enabled False
SSL URL
Always Start When Debugging
Start the local Web server even when not the startup project
Add to Source Control
Build
Rebuild
Clean
View
Analyze
Convert
Publish...
Overview
Scope to This
New Solution Explorer View
Add
Manage NuGet Packages...
Manage Client-Side Libraries...
Set as StartUp Project
Debug
Initialize Interactive with Project
Source Control
Cut Ctrl+X
Paste Ctrl+V
Remove Del
Rename
Unload Project
Open Folder in File Explorer
Properties Alt+Enter

Figure 28: Managing NuGet packages in the NuGet server project

The NuGet Package Manager will appear, and we can search for the **NuGet.Server** package.



NuGet.Server by Microsoft, 275K downloads v3.2.1
Web Application used to host a simple NuGet feed

NuGet.Server.Core by Microsoft, 32.4K downloads v3.2.1
Core library for creating a Web Application used to host a simple NuGet feed

NuGet.Server.V2 by Microsoft, 30.9K downloads v3.2.1
Library to host one or more simple NuGet V2 feeds as Web API controllers.

TedNuGetServer.BunyanLambdaLogger by TedNuGetServer, 4.11K dow v1.0.2
A bunyan structured format logger for AWS Lambda.

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.

Do not show this again

NuGet.Server

Version: Latest stable 3.2.1

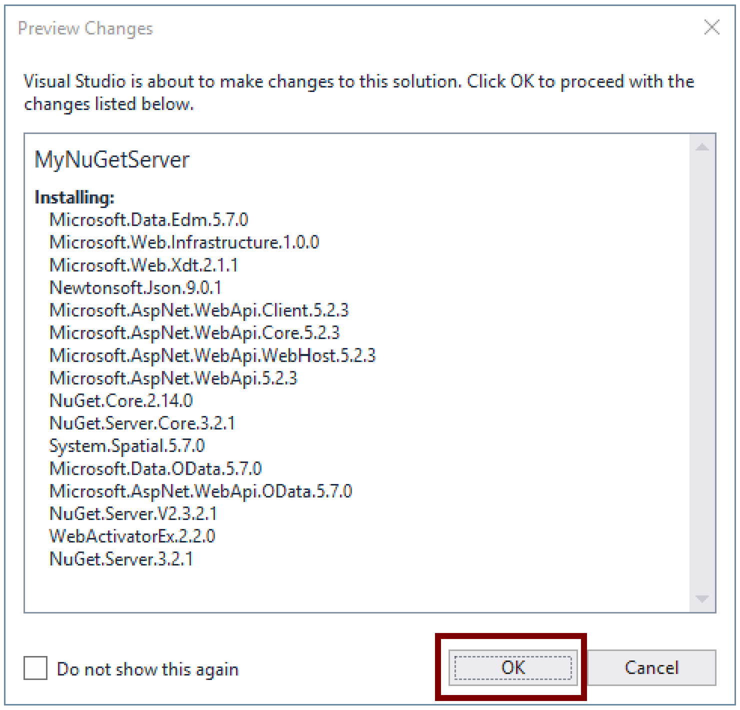
Description
Web Application used to host a simple NuGet feed

Version:	3.2.1
Author(s):	Microsoft
License:	Apache-2.0
Date published:	Thursday, March 21, 2019 (3/21/2019)
Project URL:	https://github.com/NuGet/NuGet.Server
Report Abuse:	https://www.nuget.org/packages/

Figure 29: NuGet Package manager

As suggested by the previous figure, we should type **NuGet.server** in the search text box of the **Browse** section (underscored in blue). The Package Manager searches the NuGet Gallery and shows us the NuGet.Server package at the top. Click **Install** to begin package installation.

Now, the NuGet Package Manager displays a dialog box indicating the changes that it will make to the project after installing the package. We should click **OK** if we agree with these changes.


Preview Changes

Visual Studio is about to make changes to this solution. Click OK to proceed with the changes listed below.

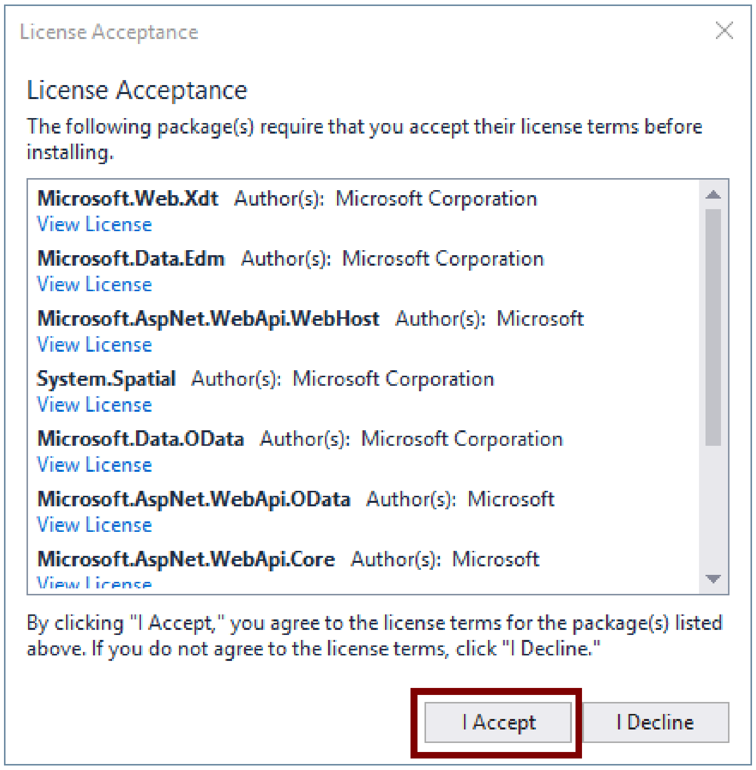
MyNuGetServer

Installing:

```
Microsoft.Data.Edm.5.7.0
Microsoft.Web.Infrastructure.1.0.0
Microsoft.Web.Xdt.2.1.1
Newtonsoft.Json.9.0.1
Microsoft.AspNet.WebApi.Client.5.2.3
Microsoft.AspNet.WebApi.Core.5.2.3
Microsoft.AspNet.WebApi.WebHost.5.2.3
Microsoft.AspNet.WebApi.5.2.3
NuGet.Core.2.14.0
NuGet.Server.Core.3.2.1
System.Spatial.5.7.0
Microsoft.Data.OData.5.7.0
Microsoft.AspNet.WebApi.OData.5.7.0
NuGet.Server.V2.3.2.1
WebActivatorEx.2.2.0
NuGet.Server.3.2.1
```

Figure 30: Preview Changes dialog box

In this case, some packages require the acceptance of a license for using them. Therefore, the License Acceptance dialog box appears.


License Acceptance

License Acceptance

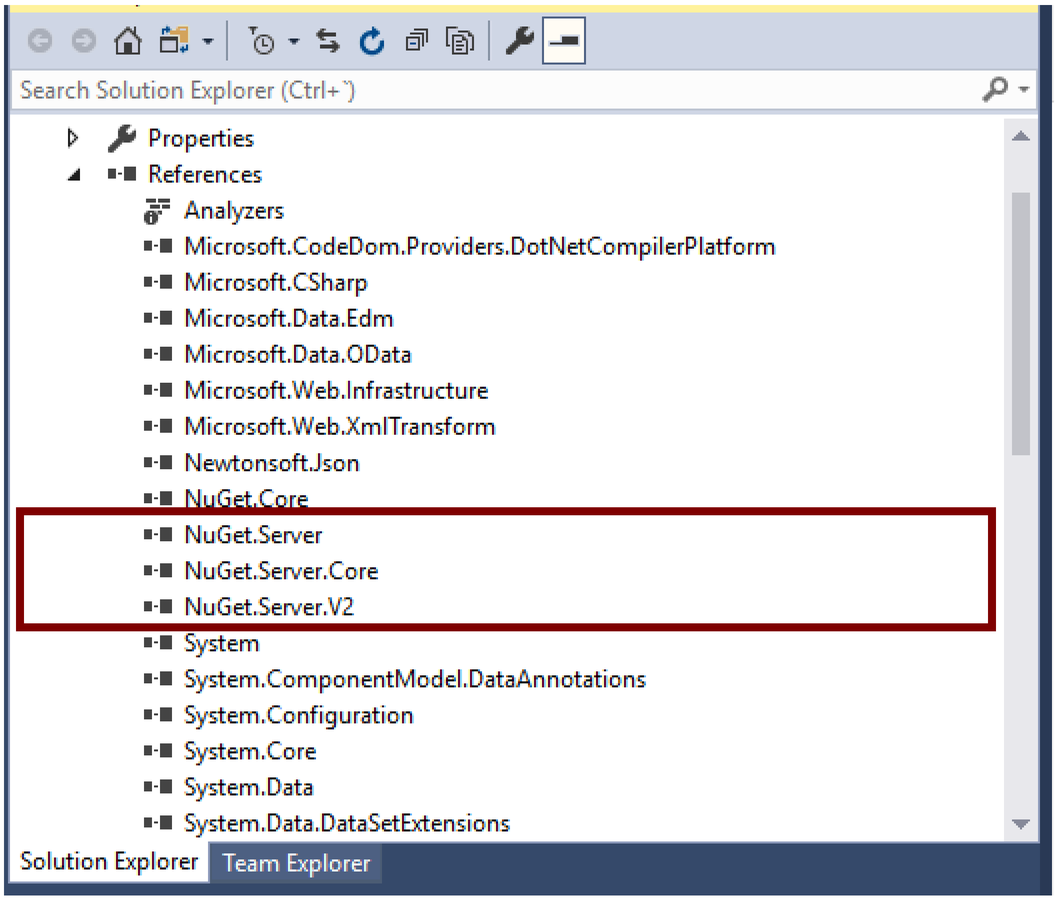
The following package(s) require that you accept their license terms before installing.

Microsoft.Web.Xdt Author(s): Microsoft Corporation
View License
Microsoft.Data.Edm Author(s): Microsoft Corporation
View License
Microsoft.AspNet.WebApi.WebHost Author(s): Microsoft
View License
System.Spatial Author(s): Microsoft Corporation
View License
Microsoft.Data.OData Author(s): Microsoft Corporation
View License
Microsoft.AspNet.WebApi.OData Author(s): Microsoft
View License
Microsoft.AspNet.WebApi.Core Author(s): Microsoft
View License

By clicking "I Accept," you agree to the license terms for the package(s) listed above. If you do not agree to the license terms, click "I Decline."

Figure 31: License Acceptance dialog box

We should click **I Accept** if we agree with the license terms for these packages. For the purposes of this book, we will assume that everything is OK. The Package Manager installs all the packages and adds the proper references to the project. Now, the **Solution Explorer** should look like the following image.



Search Solution Explorer (Ctrl+`)

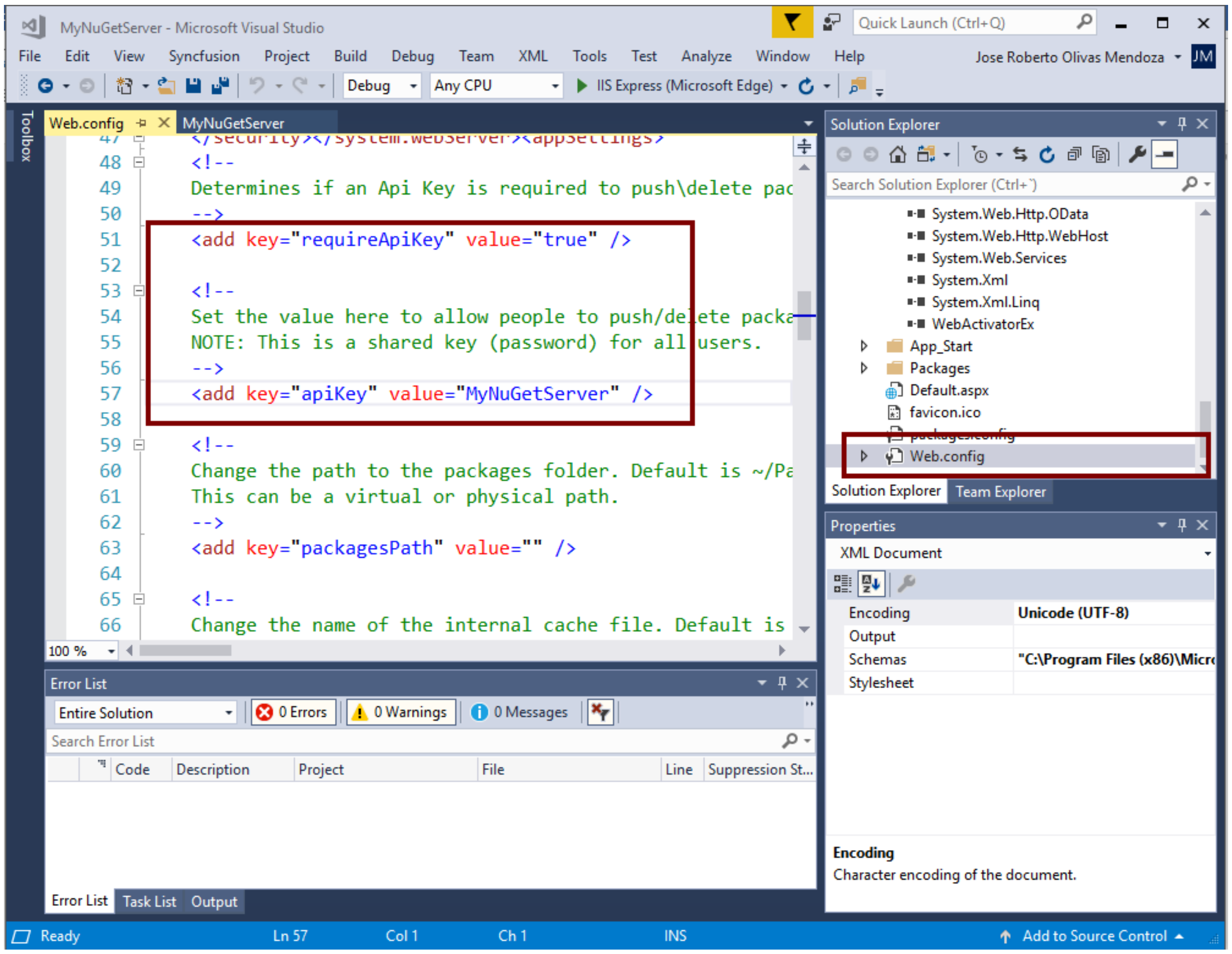
- Properties
- References
  - Analyzers
  - Microsoft.CodeDom.Providers.DotNetCompilerPlatform
  - Microsoft.CSharp
  - Microsoft.Data.Edm
  - Microsoft.Data.OData
  - Microsoft.Web.Infrastructure
  - Microsoft.Web.XmlTransform
  - Newtonsoft.Json
  - NuGet.Core
  - NuGet.Server
  - NuGet.Server.Core
  - NuGet.Server.V2
  - System
  - System.ComponentModel.DataAnnotations
  - System.Configuration
  - System.Core
  - System.Data
  - System.Data.DataSetExtensions

Figure 32: The Solution Explorer after installing the NuGet.Server package

Now we can see the references to NuGet.Server in our project.

Setting the API key for the project

If we want to upload our NuGet packages into our local NuGet server, we need to define an API key before building the project. We are going to define the API key for the NuGet server in the **Web.config** file, using **MyNuGetServer** as a key for the purposes of this book.


```
</security></system.webServer><appSettings>
<!--
Determines if an Api Key is required to push\delete pac
-->
<add key="requireApiKey" value="true" />

<!--
Set the value here to allow people to push/delete packa
NOTE: This is a shared key (password) for all users.
-->
<add key="apiKey" value="MyNuGetServer" />

<!--
Change the path to the packages folder. Default is ~/Pa
This can be a virtual or physical path.
-->
<add key="packagesPath" value="" />

<!--
Change the name of the internal cache file. Default is
```

Encoding	Unicode (UTF-8)
Output	
Schemas	"C:\Program Files (x86)\Micr
Stylesheet	

Encoding
Character encoding of the document.

Figure 33: Specifying the API key in the Web.config file

Building and testing the project

Once we define the API key, we can test the project by clicking the **Start Debugging** button located on the Visual Studio toolbar.

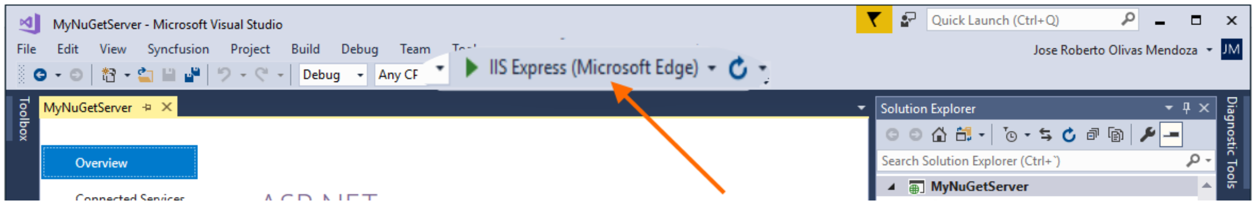
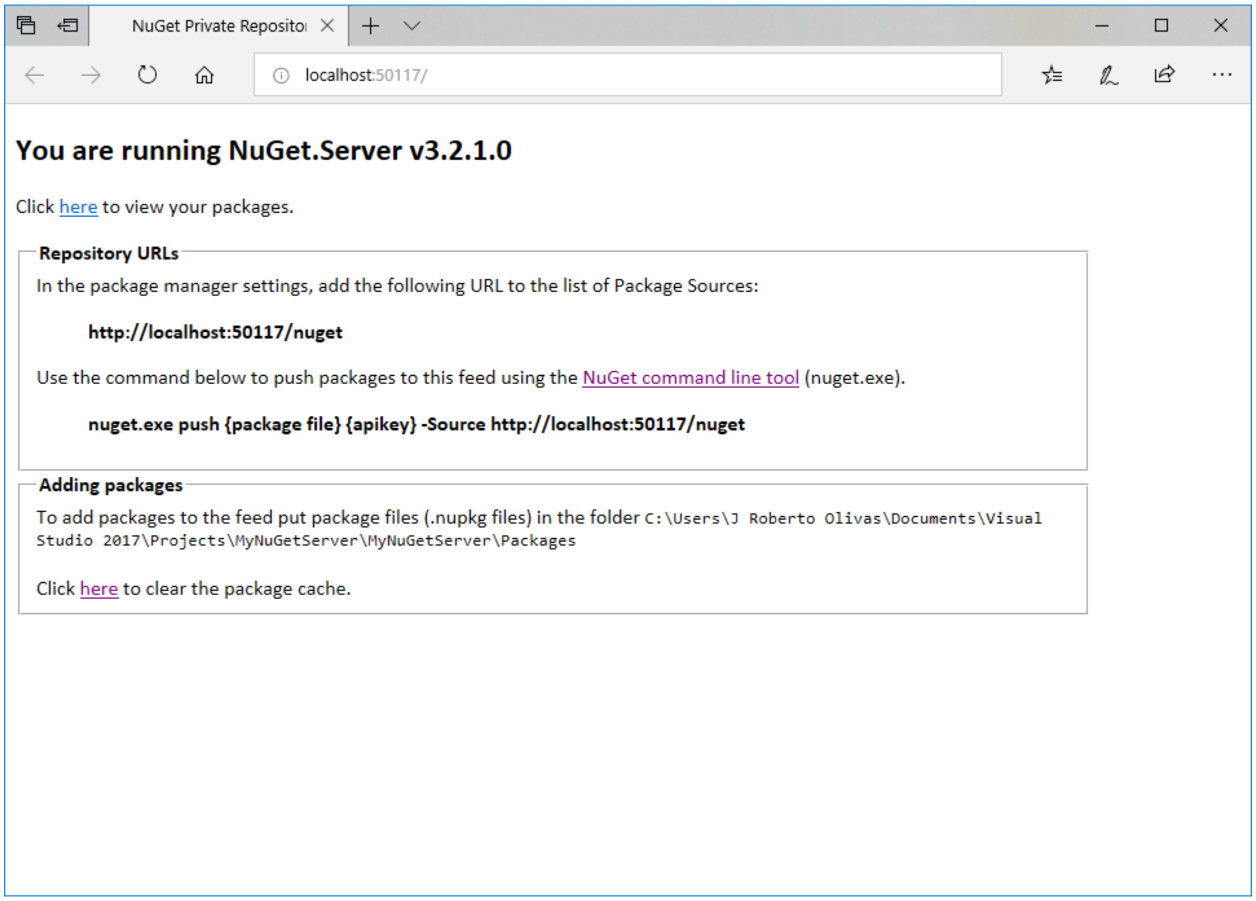

MyNuGetServer - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
File Edit View Syncfusion Project Build Debug Team
Jose Roberto Olivas Mendoza
Debug
Any CPU
IIS Express (Microsoft Edge)
MyNuGetServer
Overview
Connected Services
ASP.NET
Solution Explorer
Search Solution Explorer (Ctrl+;)
MyNuGetServer

Figure 34: Start Debugging for MyNuGetServer project

Visual Studio builds the project and launches the Microsoft Edge browser. The following image displays our NuGet server in action.

Note: If you get an error message that states something like "HTTP Error 500.19 - Internal Server Error The requested page cannot be accessed because the related configuration data for the page is invalid," carefully review the Web.config file and look for a duplicate entry, such as the <compilation> tag, and remove any duplicates.



You are running NuGet.Server v3.2.1.0

Click here to view your packages.

Repository URLs

In the package manager settings, add the following URL to the list of Package Sources:

http://localhost:50117/nuget

Use the command below to push packages to this feed using the NuGet command line tool (nuget.exe).

nuget.exe push {package file} {apikey} -Source http://localhost:50117/nuget

Adding packages

To add packages to the feed put package files (.nupkg files) in the folder `C:\Users\J Roberto Olivas\Documents\Visual Studio 2017\Projects\MyNuGetServer\MyNuGetServer\Packages`

Click here to clear the package cache.

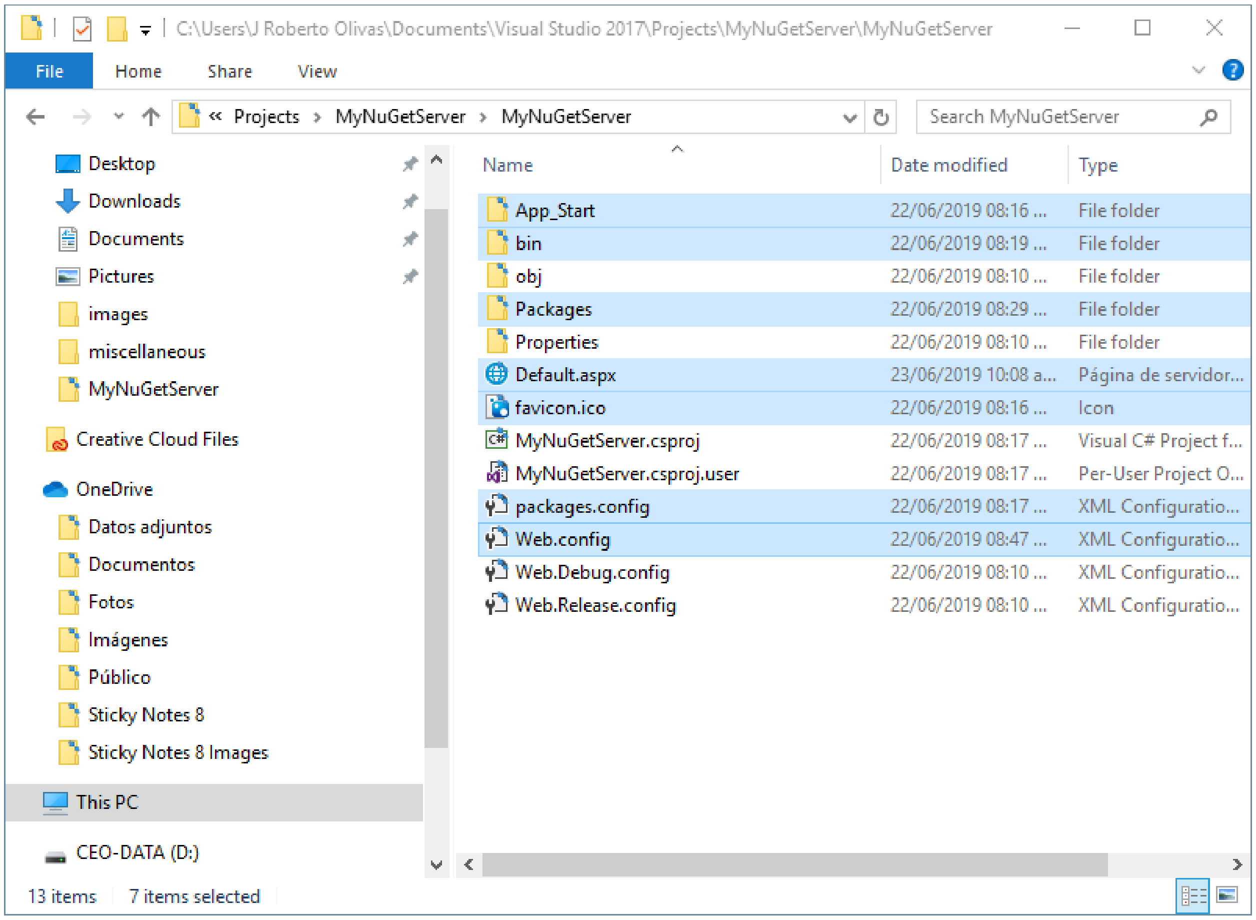
Figure 35: NuGet Server webpage in Microsoft Edge

Hosting the NuGet server in IIS

Copying our project compile code to IIS default website

Now, it is time to make our NuGet server available for use with Visual Studio. To accomplish this task, we need to go to the website root folder of our local computer. This is usually **C:\Inetpub\wwwroot**. Then, we are going to create a new folder named **mynugetserver**. In this folder, we will save the compiled code of our NuGet server project, created in the previous section.

Now, we need to go to the folder where our compiled code is, using Windows File Explorer. For the purposes of this book, this folder is **Documents\Visual Studio 2017\Projects\MyNuGetServer\MyNuGetServer**. Once the File Explorer window appears, we need to select the files and folders highlighted in the following figure.

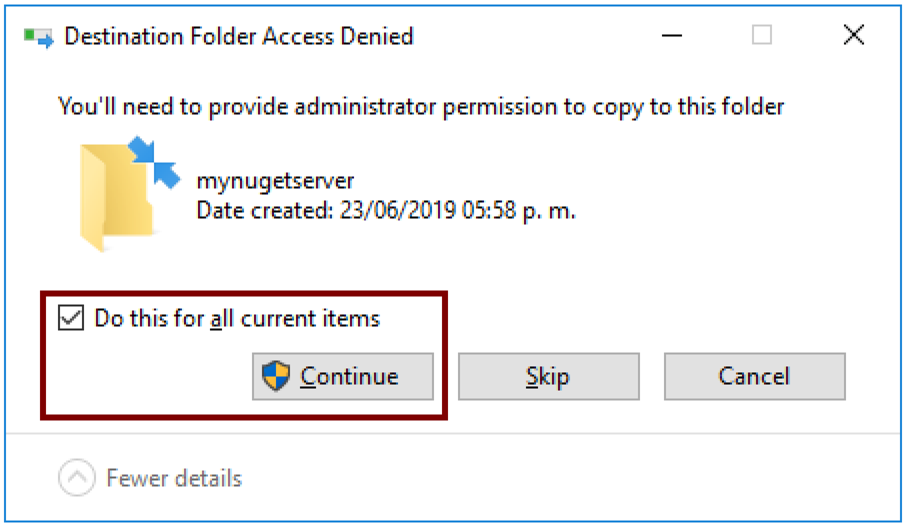


Name	Date modified	Type
App_Start	22/06/2019 08:16 ...	File folder
bin	22/06/2019 08:19 ...	File folder
obj	22/06/2019 08:10 ...	File folder
Packages	22/06/2019 08:29 ...	File folder
Properties	22/06/2019 08:10 ...	File folder
Default.aspx	23/06/2019 10:08 a...	Página de servidor...
favicon.ico	22/06/2019 08:16 ...	Icon
MyNuGetServer.csproj	22/06/2019 08:17 ...	Visual C# Project f...
MyNuGetServer.csproj.user	22/06/2019 08:17 ...	Per-User Project O...
packages.config	22/06/2019 08:17 ...	XML Configuratio...
Web.config	22/06/2019 08:47 ...	XML Configuratio...
Web.Debug.config	22/06/2019 08:10 ...	XML Configuratio...
Web.Release.config	22/06/2019 08:10 ...	XML Configuratio...

13 items 7 items selected

Figure 36: The list of files and folders to copy into the mynugetserver folder

Copy these files and folders to the **mynugetserver** folder created in **C:\Inetpub\wwwroot**. Since Windows considers the folder created for administrative purposes, a dialog box appears, asking for administrator permissions.


Destination Folder Access Denied

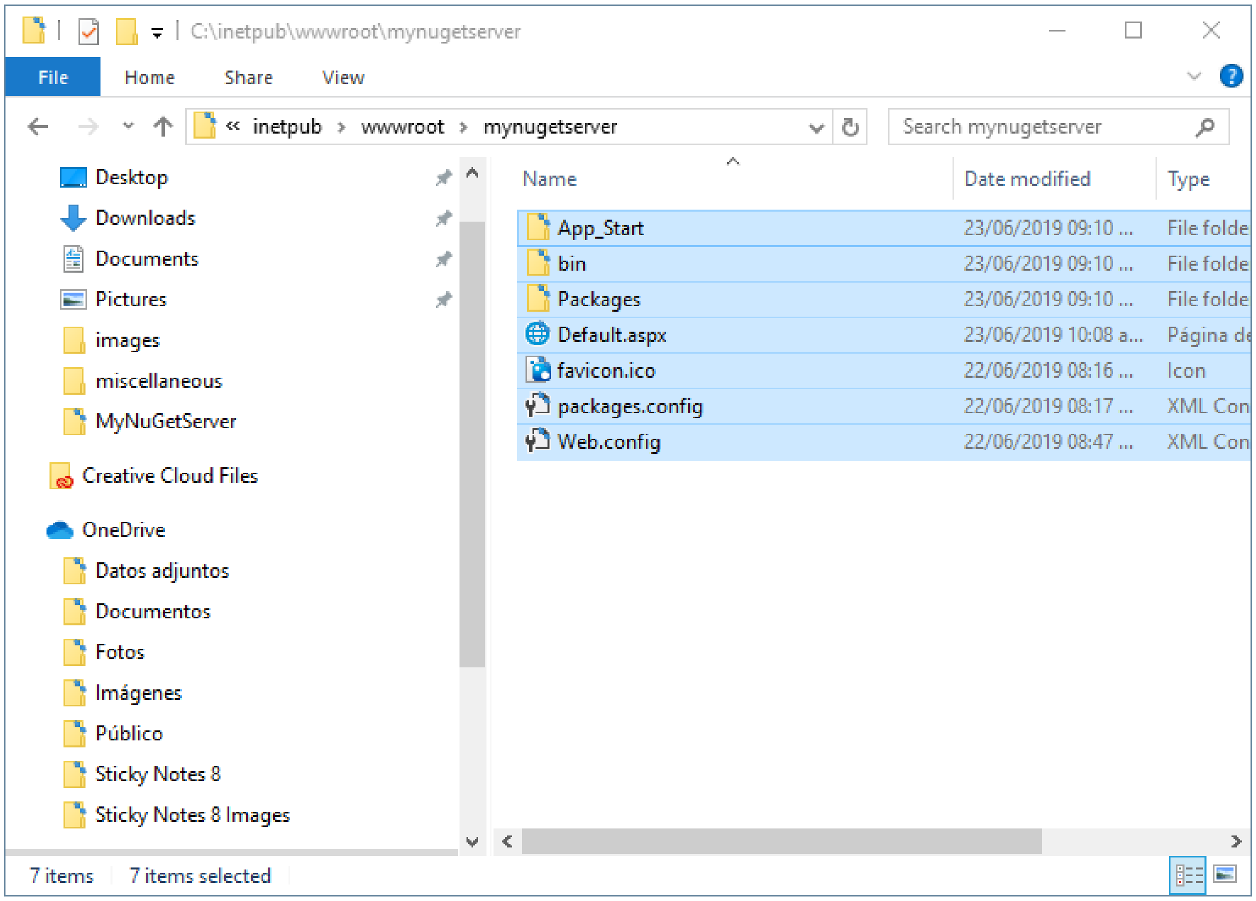
You'll need to provide administrator permission to copy to this folder

mynugetserver
Date created: 23/06/2019 05:58 p. m.

☑ Do this for all current items

Figure 37: Dialog box asking for administrative permissions

Select the **Do this for all current items** check box to prevent the dialog box from appearing each time Windows attempts to copy a file or folder. Then, click **Continue**. The following image displays the **mynugetserver** folder with all copied files and folders.


C:\inetpub\wwwroot\mynugetserver

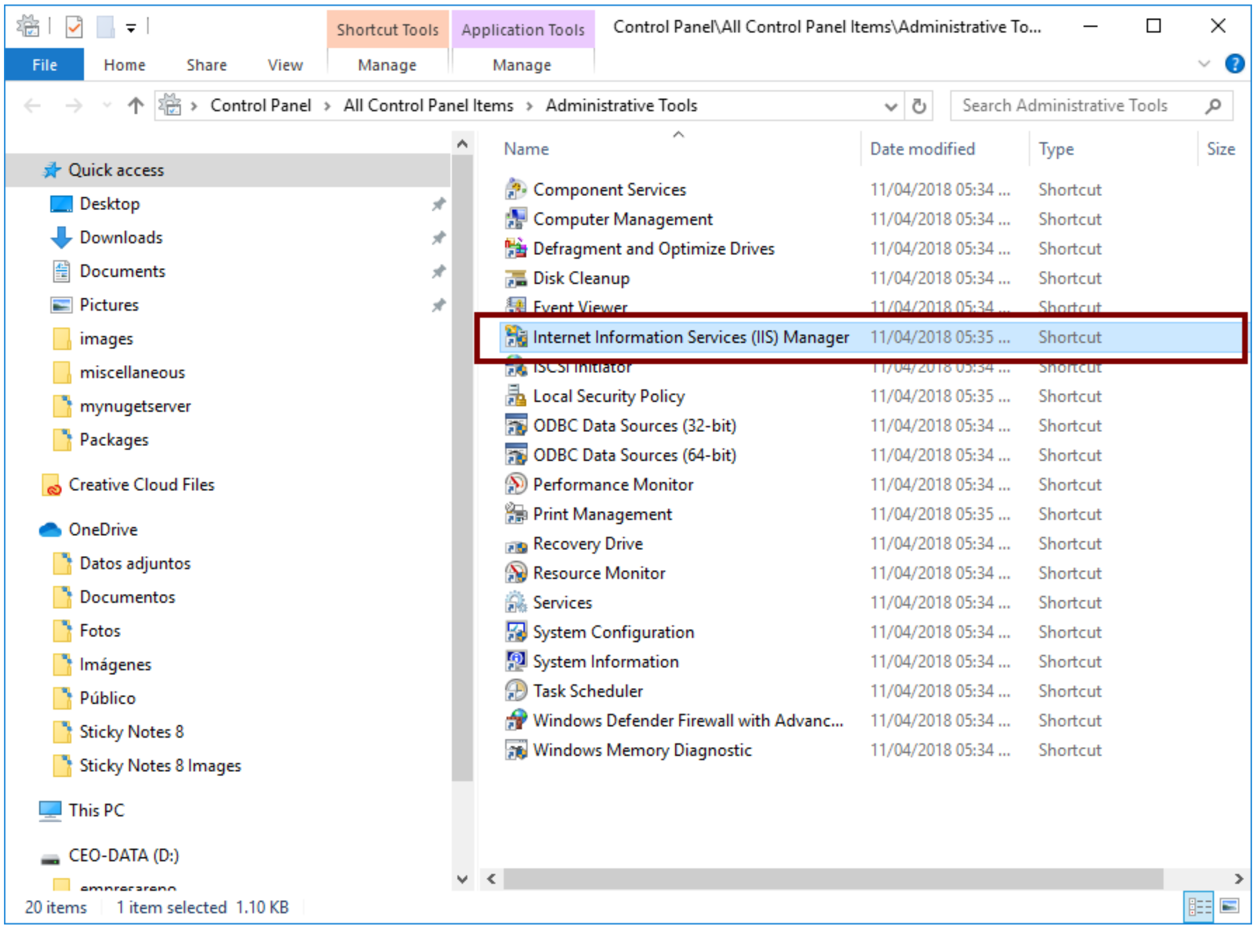
Name	Date modified	Type
App_Start	23/06/2019 09:10 ...	File folde
bin	23/06/2019 09:10 ...	File folde
Packages	23/06/2019 09:10 ...	File folde
Default.aspx	23/06/2019 10:08 a...	Página de
favicon.ico	22/06/2019 08:16 ...	Icon
packages.config	22/06/2019 08:17 ...	XML Con
Web.config	22/06/2019 08:47 ...	XML Con

7 items   7 items selected

Figure 38: MyNuGetServer files and folders

Setting mynugetserver as an IIS application

The next step is to configure the **mynugetserver** folder as an application using the Internet Information Services Manager. We need to go to **Control Panel** and click the **Administrative Tools** icon. Then, we should click **Internet Information Services (IIS) Manager**, as shown in the following image.



Name	Date modified	Type	Size
Component Services	11/04/2018 05:34 ...	Shortcut	
Computer Management	11/04/2018 05:34 ...	Shortcut	
Defragment and Optimize Drives	11/04/2018 05:34 ...	Shortcut	
Disk Cleanup	11/04/2018 05:34 ...	Shortcut	
Event Viewer	11/04/2018 05:34 ...	Shortcut	
Internet Information Services (IIS) Manager	11/04/2018 05:35 ...	Shortcut	
iSCSI Initiator	11/04/2018 05:34 ...	Shortcut	
Local Security Policy	11/04/2018 05:35 ...	Shortcut	
ODBC Data Sources (32-bit)	11/04/2018 05:34 ...	Shortcut	
ODBC Data Sources (64-bit)	11/04/2018 05:34 ...	Shortcut	
Performance Monitor	11/04/2018 05:34 ...	Shortcut	
Print Management	11/04/2018 05:35 ...	Shortcut	
Recovery Drive	11/04/2018 05:34 ...	Shortcut	
Resource Monitor	11/04/2018 05:34 ...	Shortcut	
Services	11/04/2018 05:34 ...	Shortcut	
System Configuration	11/04/2018 05:34 ...	Shortcut	
System Information	11/04/2018 05:34 ...	Shortcut	
Task Scheduler	11/04/2018 05:34 ...	Shortcut	
Windows Defender Firewall with Advanc...	11/04/2018 05:34 ...	Shortcut	
Windows Memory Diagnostic	11/04/2018 05:34 ...	Shortcut	

Figure 39: Internet Information Services Manager icon in Administrative Tools

Now, in the **Internet Information Services Manager** dialog box, open the **Sites** node and right-click the **Default Web Site** child node. Then, click **Add Application** in the context menu.

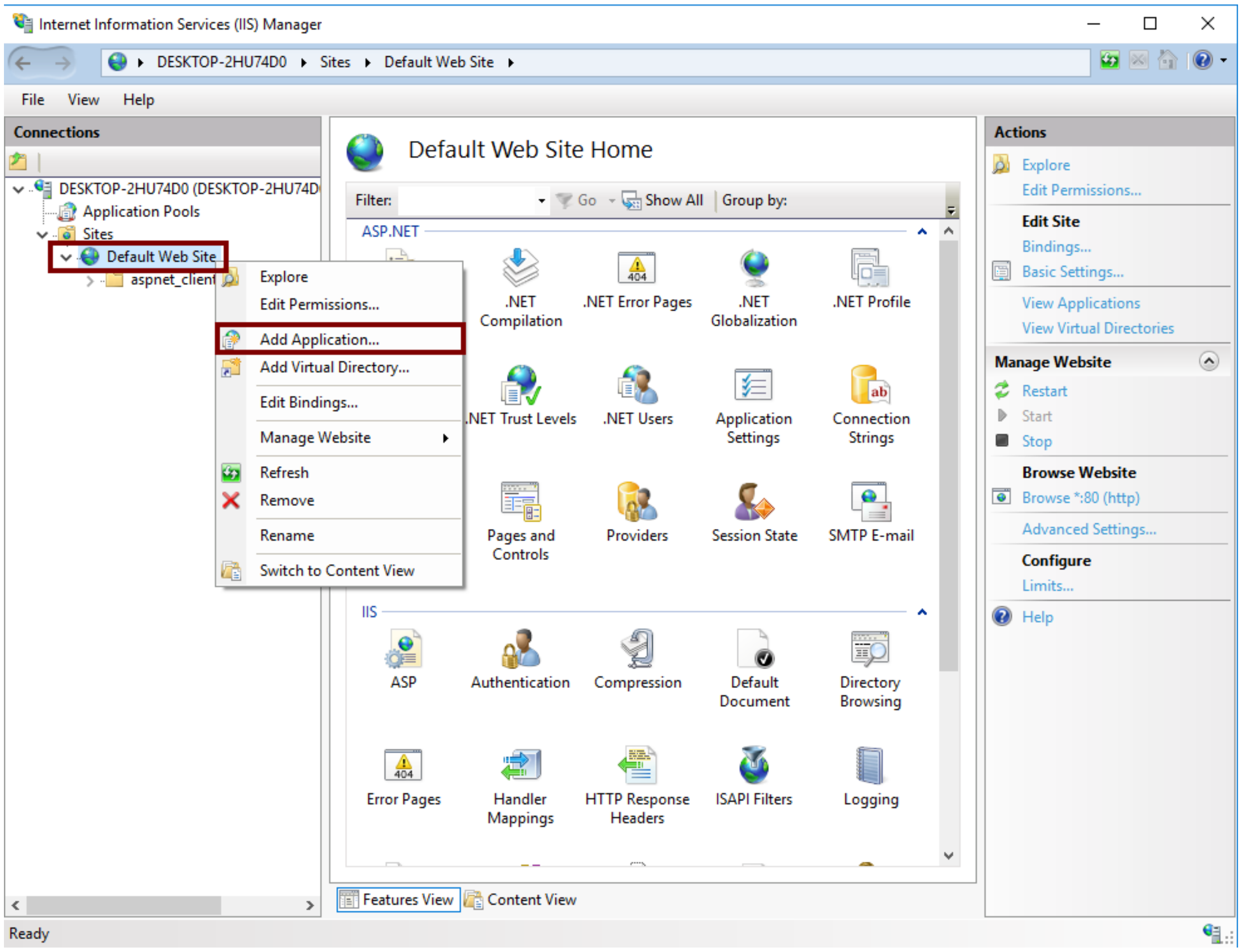
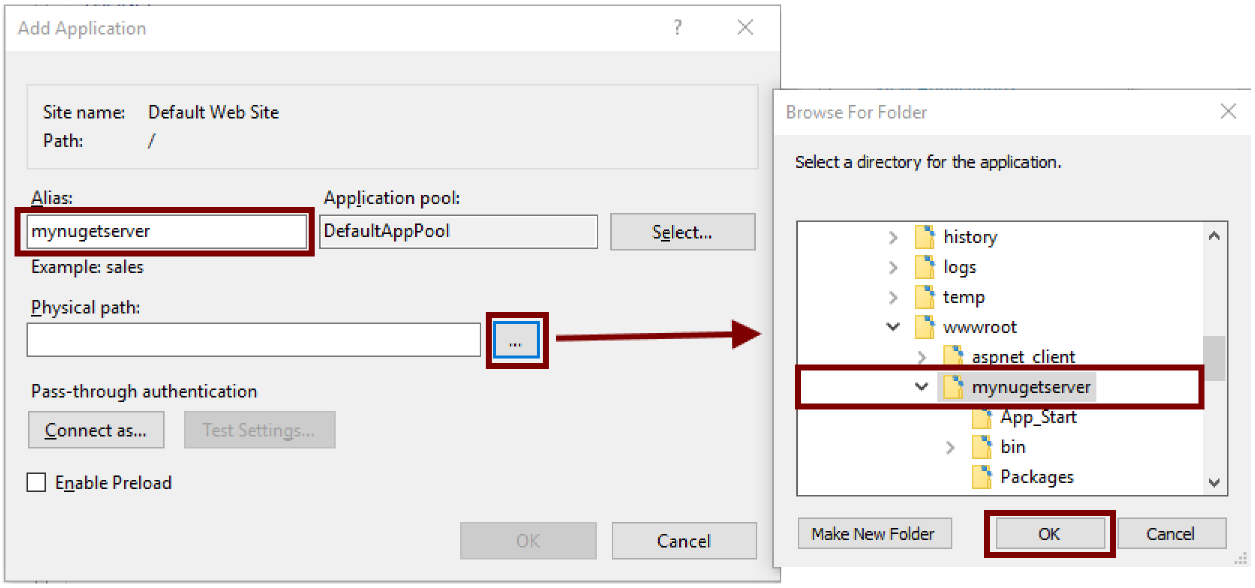


Figure 40: Adding an application to IIS

The **Add Application** dialog box appears on the screen. For the purposes of this book, we should type **mynugetserver** in the **Alias** text box. Then, we should click the **ellipsis (…)** button located to the right of the **Physical Path** text box. Now, in the **Browse For Folder** dialog box, locate and point to the **mynugetserver** folder created in **C:\Inetpub\wwwroot**. Click **OK** to select this folder.



Add Application

Site name: Default Web Site
Path: /

Alias: mynugetserver
Application pool: DefaultAppPool
Example: sales
Physical path:

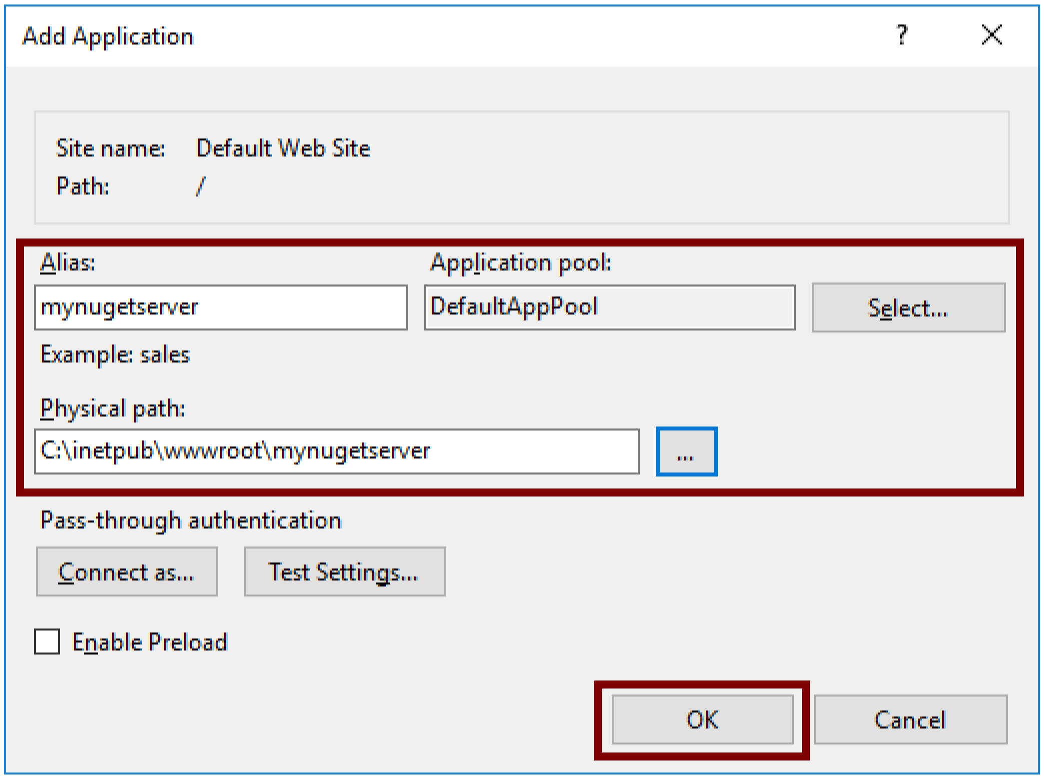
Browse For Folder

Select a directory for the application.

- history
- logs
- temp
- wwwroot
  - aspnet_client
  - mynugetserver
    - App_Start
    - bin
    - Packages

Figure 41: Setting application parameters

Now, the **Add Application** dialog box should look like the following image.



Add Application

Site name: Default Web Site
Path: /

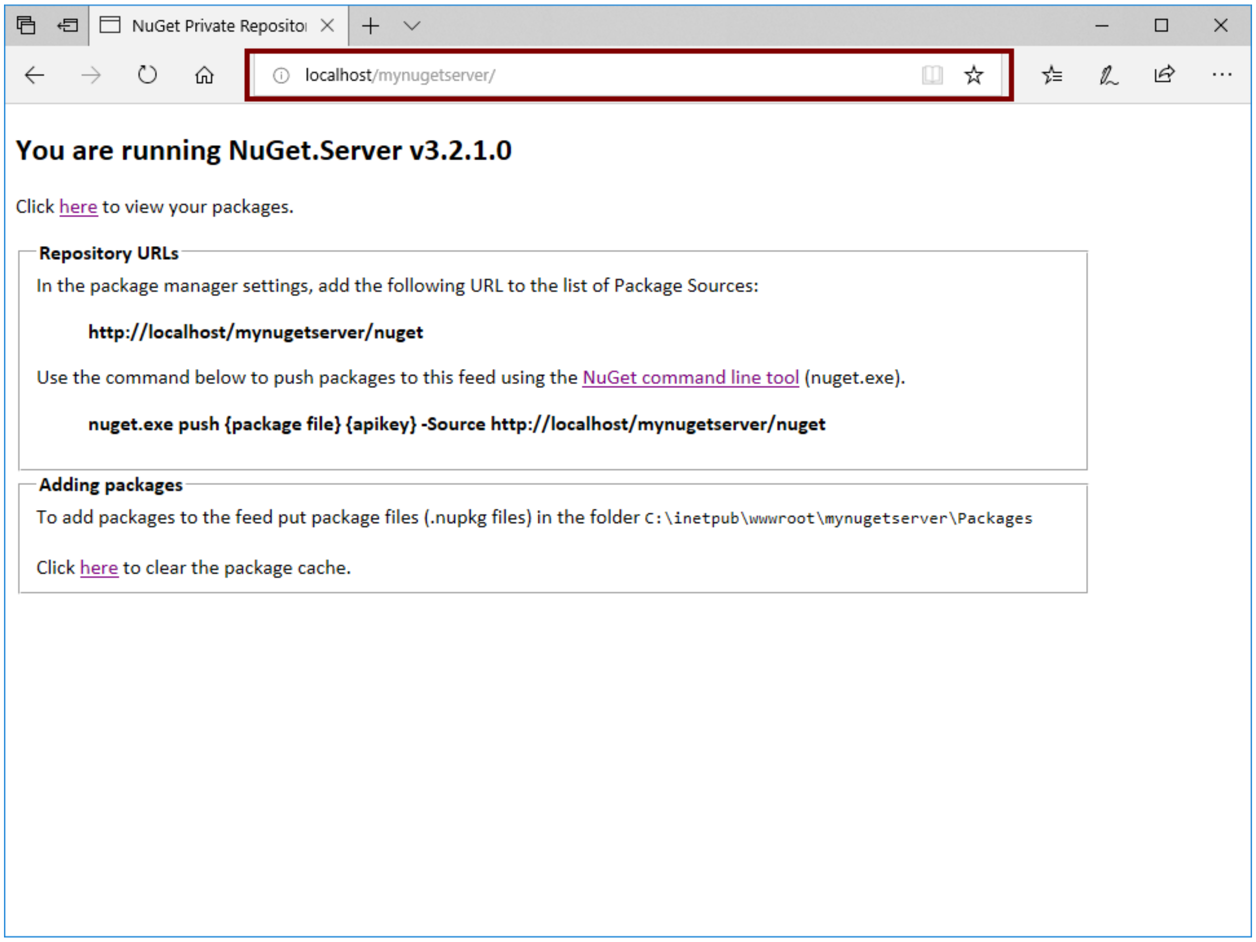
Alias: mynugetserver
Application pool: DefaultAppPool
Example: sales
Physical path: C:\inetpub\wwwroot\mynugetserver

Figure 42: Application parameters

Click **OK** to finish the process.

Testing the NuGet server in IIS

We need to ensure that our NuGet server is working properly. To accomplish that, we will open Microsoft Edge and type **http://localhost/mynugetserver** in the address bar. If everything is right, the internet browser should look like the following figure.



You are running NuGet.Server v3.2.1.0

Click here to view your packages.

Repository URLs

In the package manager settings, add the following URL to the list of Package Sources:

http://localhost/mynugetserver/nuget

Use the command below to push packages to this feed using the NuGet command line tool (nuget.exe).

```
nuget.exe push {package file} {apikey} -Source http://localhost/mynugetserver/nuget
```

Adding packages

To add packages to the feed put package files (.nupkg files) in the folder `C:\inetpub\wwwroot\mynugetserver\Packages`

Click here to clear the package cache.

Figure 43: Our NuGet server running on IIS

Chapter summary

A NuGet server is an application employed to host NuGet packages. The NuGet Gallery is the most popular host for NuGet packages, but every package hosted there is public. The purpose of this chapter was to show you how to create a local NuGet server to keep your code private.

To host a local NuGet server on our computer, we needed to set up IIS first. To accomplish this, we went to Control Panel and clicked the **Programs and Features** icon. Then, when the **Programs and Features** dialog appeared on the screen, we clicked **Turn Windows features on or off**. After that, we selected the **Internet Information Services** node. Since this action told Windows to install the basic IIS features only, we opened this node and selected all components located in the **Application Development Features** node, except for **CGI**. We typed http://localhost in the address bar of our web browser to verify the installation, and the web browser displayed the IIS home page for our local computer.

We got our own NuGet server by creating an ASP.NET Web application project with Visual Studio. We named this project **MyNuGetServer** and saved it into the **Documents\Visual Studio 2017\Projects** folder. We employed the **Empty** template to create the project, and added a public package named **NuGet.Server** from the NuGet Gallery using the **Manage NuGet Packages** item from the context menu displayed by right-clicking the **Solution Explorer**.

After the NuGet Package Manager added NuGet.Server references to our project, we modified the **Web.config** file to set an API key necessary to upload our NuGet packages into our local NuGet server. For the purposes of this book, we used **MyNuGetServer** as an API key. Finally, we clicked **Start Debugging** to build our project and test it on the Microsoft Edge web browser.

To make our NuGet server available for Visual Studio, we created a folder named **mynugetserver** in **C:\Inetpub\wwwroot**, which is the website root folder of our local computer. Then, we copied the compiled code of the MyNuGetServer project, located in **Documents\Visual Studio 2017\Projects\MyNuGetServer\MyNuGetServer**, to the **mynugetserver** folder. After that, we configured **mynugetserver** folder as an IIS application using Internet Information Services Manager. Then, we tested the IIS application by typing **http://localhost/mynugetserver** in the address bar of the Microsoft Edge web browser.

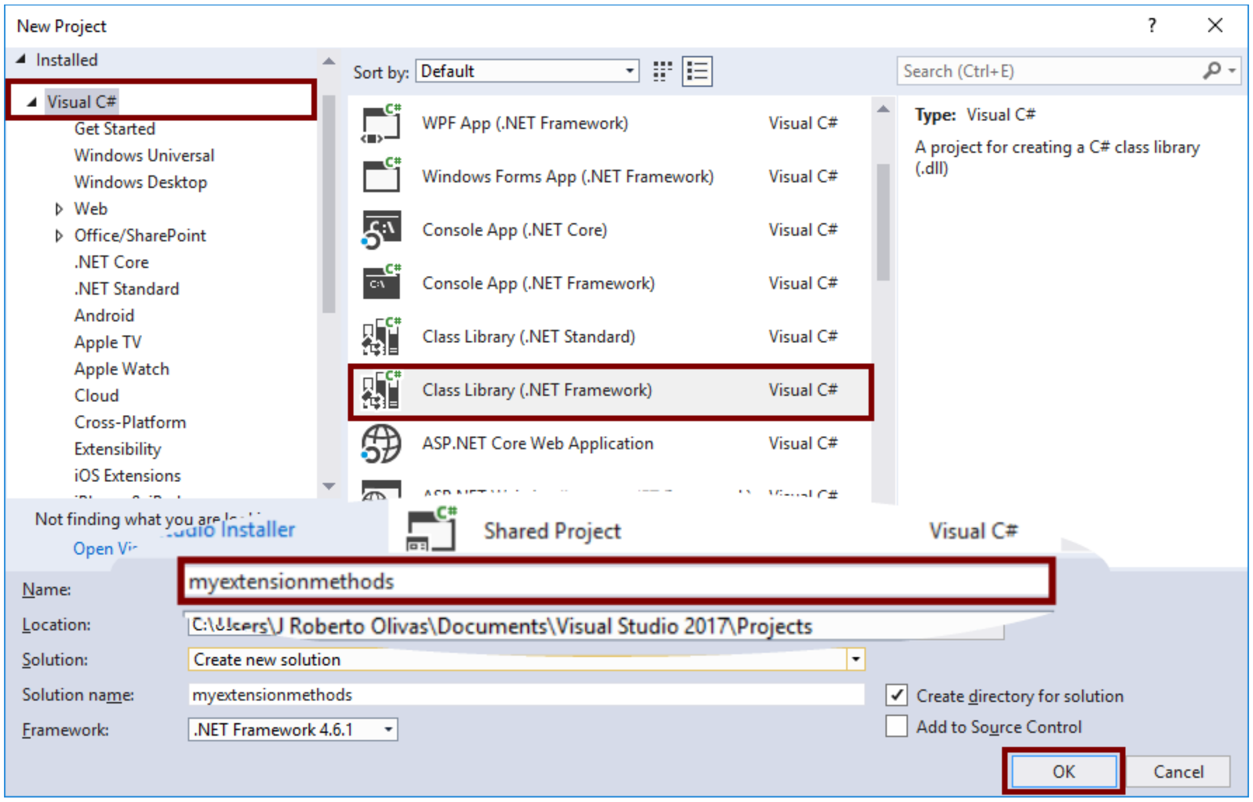
Chapter 3 Creating the Project for a Distributable Package

Defining the project

For the purposes of this book, we will create a .NET Framework class library that will contain a set of extension methods. We will name this project **myextensionmethods** and save it in the **Documents\Visual Studio 2017\Projects** folder. We will use this library in a Windows Forms application detailed in Chapter 5.

Creating the project

First, open Visual Studio and click **File** > **New** > **Project** in the menu bar. Now, select the **Visual C#** node in the template categories tree of the **New Project** dialog box, and select the **Class Library (.NET Framework)** template. After that, type **myextensionmethods** in the **Name** text box and click **OK**.


New Project
Installed
Visual C#
Get Started
Windows Universal
Windows Desktop
Web
Office/SharePoint
.NET Core
.NET Standard
Android
Apple TV
Apple Watch
Cloud
Cross-Platform
Extensibility
iOS Extensions
Not finding what you are looking for?
Open Visual Studio Installer
Sort by: Default
Search (Ctrl+E)
WPF App (.NET Framework) Visual C#
Windows Forms App (.NET Framework) Visual C#
Console App (.NET Core) Visual C#
Console App (.NET Framework) Visual C#
Class Library (.NET Standard) Visual C#
Class Library (.NET Framework) Visual C#
ASP.NET Core Web Application Visual C#
Shared Project Visual C#
OK
Cancel

Type: Visual C#

A project for creating a C# class library (.dll)

Name: myextensionmethods

Location: C:\Users\J Roberto Olivas\Documents\Visual Studio 2017\Projects

Solution: Create new solution

Solution name: myextensionmethods

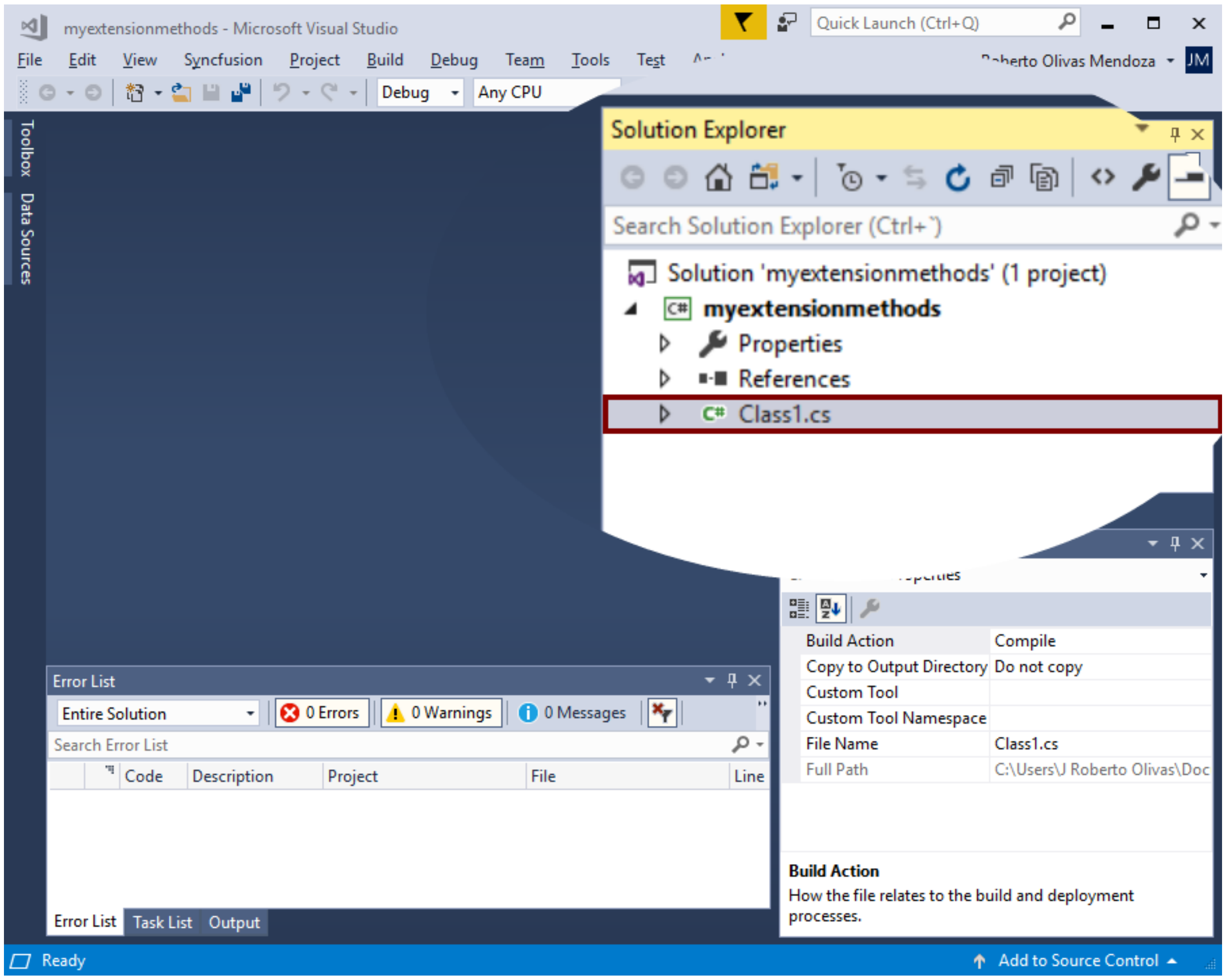
Framework: .NET Framework 4.6.1

[✓] Create directory for solution

[ ] Add to Source Control

Figure 44: Creating the myextensionmethods project

Visual Studio will create the project, and the **Solution Explorer** should now look like the following figure.



Build Action	Compile
Copy to Output Directory	Do not copy
Custom Tool	
Custom Tool Namespace	
File Name	Class1.cs
Full Path	C:\Users\J Roberto Olivas\Doc

Build Action
How the file relates to the build and deployment processes.

Figure 45: Visual Studio after creating the project

We can see in the previous image that Visual Studio automatically added a file named **Class1.cs** to the project. For the purposes of this book, we should rename this file to **MyExtensionMethods.cs**. To do this, right-click the name of the file and click **Rename**. When Visual Studio highlights the file name, type **MyExtensionMethods** and press **Enter**.

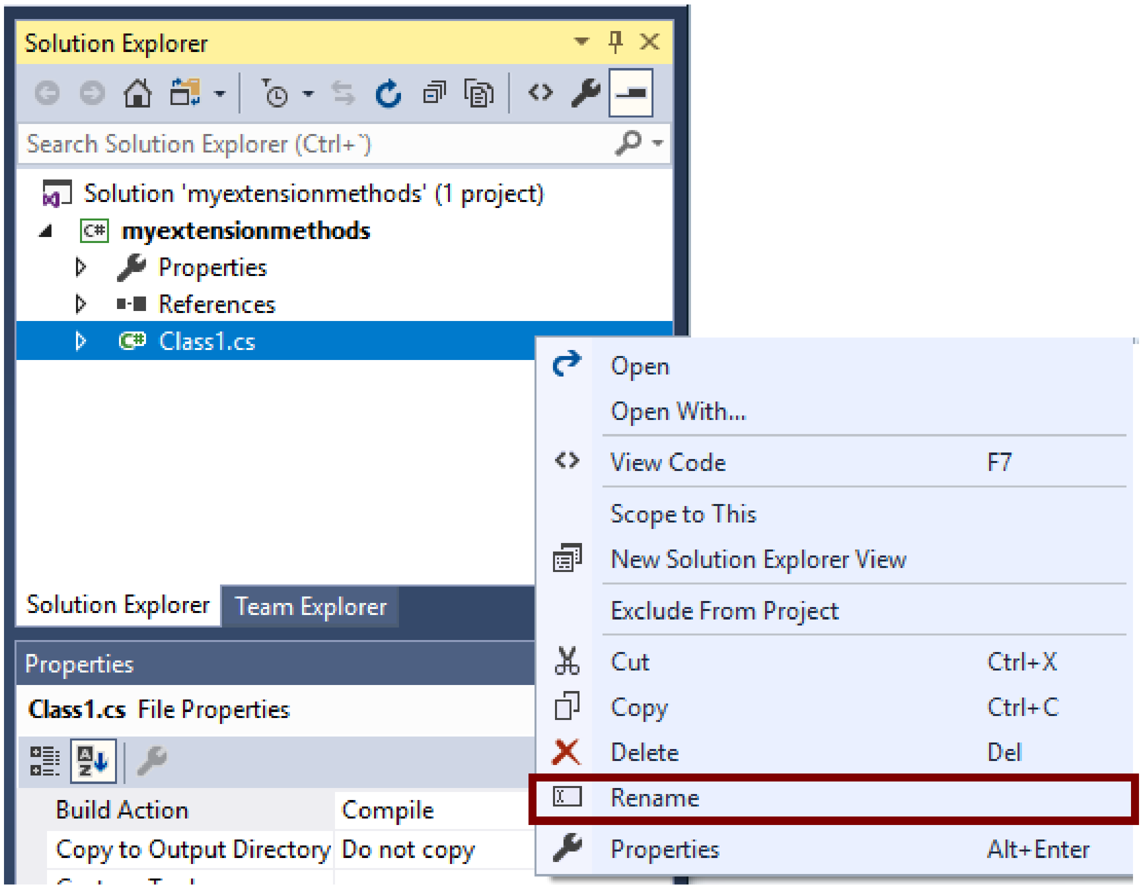
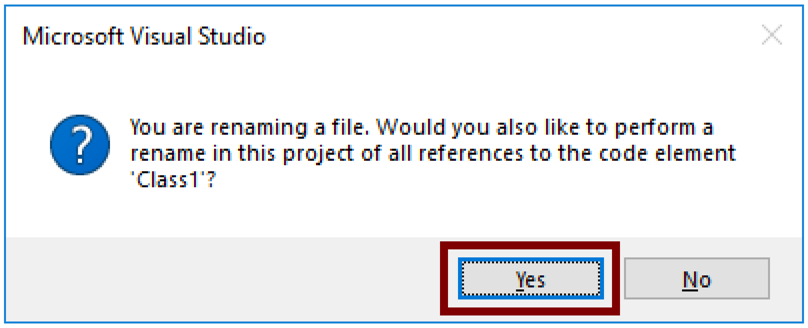


Figure 46: Renaming the Class1.cs file

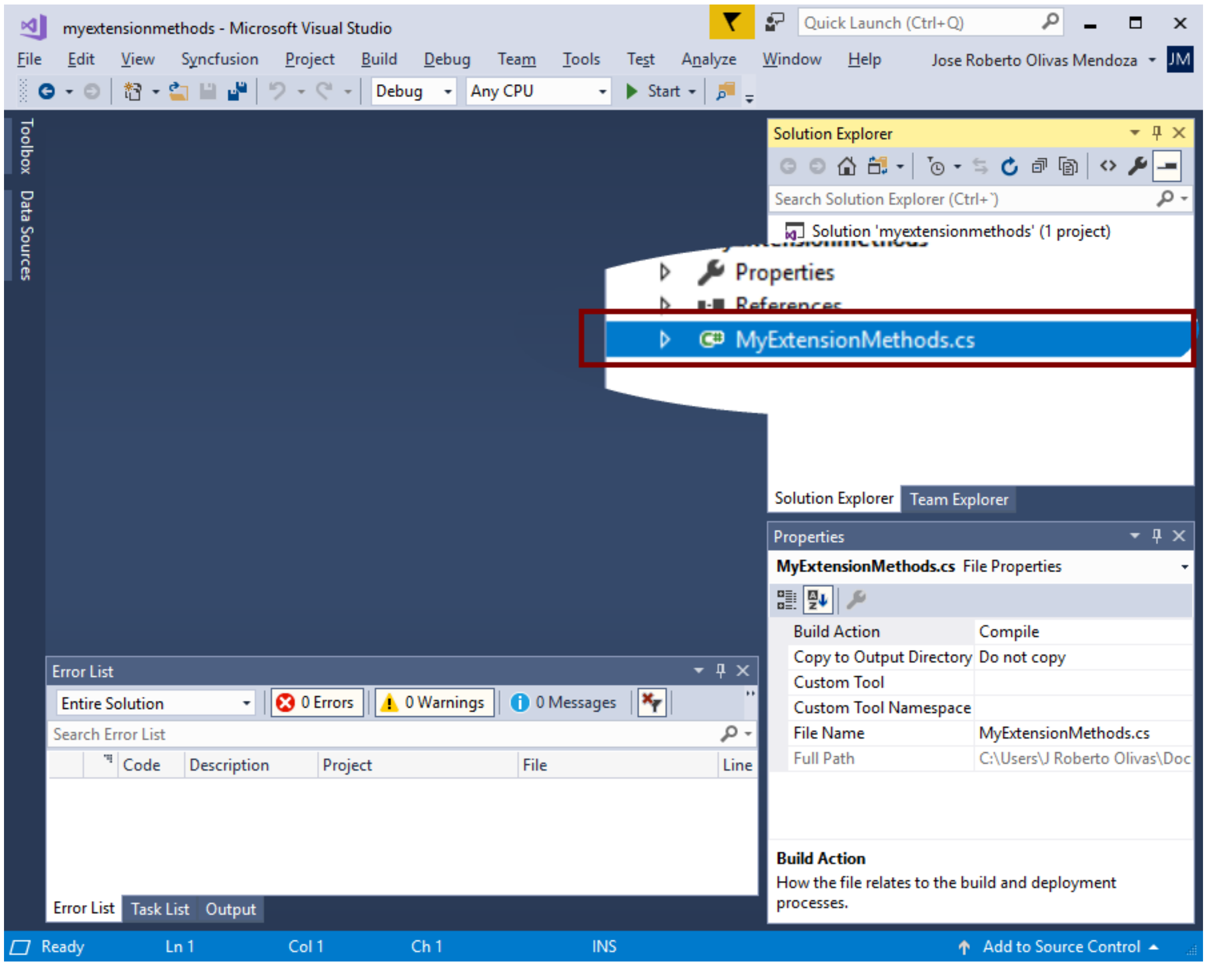
Now, Visual Studio asks to rename the dependencies of **Class1.cs** using the dialog box displayed in the following image.


Microsoft Visual Studio

You are renaming a file. Would you also like to perform a rename in this project of all references to the code element 'Class1'?

Figure 47: Visual Studio asking for renaming dependencies

Click **Yes**. Visual Studio renames the file and all its references in the project. The **Solution Explorer** will look like the following figure.


myextensionmethods - Microsoft Visual Studio
File Edit View Syncfusion Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
Start
Solution Explorer
Solution 'myextensionmethods' (1 project)
Properties
References
MyExtensionMethods.cs
Properties
MyExtensionMethods.cs File Properties
Error List
Entire Solution
0 Errors
0 Warnings
0 Messages
Ready

Build Action	Compile
Copy to Output Directory	Do not copy
Custom Tool	
Custom Tool Namespace	
File Name	MyExtensionMethods.cs
Full Path	C:\Users\J Roberto Olivas\Doc

Build Action
How the file relates to the build and deployment processes.

Figure 48: Our project after renaming the Class1.cs file

Writing code and building the assembly

Now, we will write the code for the extension methods to expose in the class library. At the end, the code should look like the following example.

Code Listing 1: MyExtensionMethods class code

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Windows.Forms;

namespace myextensionmethods
{
    public static class MyExtensionMethods
    {
        /// <summary>
        /// Message posted when the user clicks the left mouse button while
        /// the cursor is within the nonclient area of a window. The window
        /// that has the cursor receives the message.
        /// </summary>
        private const int WM_NCLBUTTONDOWN = 0x00A1;

        /// <summary>
        /// Indicates that the screen coordinates belong to the
        /// title bar.
        /// </summary>
        private const int HTCAPTION = 2;

        /// <summary>
        /// Creates a WM_NCLBUTTONDOWN message for the specified form. This
        /// method provides a mechanism to drag the form from anywhere,
including the
        /// title bar, if there is one.
        /// </summary>
        /// <param name="targetForm">The form instance that will receive
the message</param>
        /// <returns></returns>
        public static Message CreateWmNclButtonDown(this Form targetForm)
=> Message.Create(targetForm.Handle, WM_NCLBUTTONDOWN, new
IntPtr(HTCAPTION), IntPtr.Zero);

        /// <summary>
        /// Draw the hands for an analog clock face in a client area
(usually a form's client area).
        /// </summary>
        /// <param name="graphicsInstance">Graphics object associated with
the client area.</param>
        /// <param name="clientSize">Width and height of the client
area.</param>
        /// <param name="handsColor">Color used to draw the hour and minute
hands.</param>
        /// <param name="secondHandColor">Color used to draw the second
hand.</param>
        public static void DrawClockHands(this Graphics graphicsInstance,
Size clientSize,Color handsColor, Color secondHandColor)
        {
            using (var handsPen = new Pen(handsColor, 4))
            {
                // Get the hour and the minute including any fraction
elapsed.
                DateTime currentDateTime = DateTime.Now;
                var currentHour = currentDateTime.Hour +
currentDateTime.Minute / 60f + currentDateTime.Second / 3600f;
```

```
                var currentMinute = currentDateTime.Minute +
currentDateTime.Second / 60f;

                // Gets the center point for the clock face.
                var clockFaceCenter = new PointF(0, 0);

                //Set the scale factor for drawing the hour hand.
                var hourHandXFactor = 0.2f * clientSize.Width;
                var hourHandYFactor = 0.2f * clientSize.Height;

                //Gets the angle for the current hour.
                var hourHandAngle = -Math.PI / 2 + 2 * Math.PI *
currentHour / 12.0;

                //Gets the hour hand endpoint.
                var hourHandEndPoint = new PointF((float)(hourHandXFactor *
Math.Cos(hourHandAngle)),(float)(hourHandYFactor *
Math.Sin(hourHandAngle)));

                //Assigns the hands color to the drawing pen.
                handsPen.Color = handsColor;

                //Draws the hour hand.
                graphicsInstance.DrawLine(handsPen, hourHandEndPoint,
clockFaceCenter);

                //Set the scale factor for the minute hand.
                float minuteHandXFactor = 0.3f * clientSize.Width;
                float minuteHandYFactor = 0.3f * clientSize.Height;

                //Gets the angle for the current minute.
                double minuteHandAngle = -Math.PI / 2 + 2 * Math.PI *
currentMinute / 60.0;

                //Gets the endpoint for the minute hand.
                var minuteHandEndPoint = new
PointF((float)(minuteHandXFactor * Math.Cos(minuteHandAngle)),
(float)(minuteHandYFactor * Math.Sin(minuteHandAngle)));

                //Sets the pen width to 2 pixels for the minute hand.
                handsPen.Width = 2;

                //Draws the minute hand.
                graphicsInstance.DrawLine(handsPen, minuteHandEndPoint,
clockFaceCenter);

                //Sets the scale factor for the second hand.
                var secondHandXFactor = 0.4f * clientSize.Width;
                var secondHadYFactor = 0.4f * clientSize.Height;
                //Gets the angle for the current second.
                var second_angle = -Math.PI / 2 + 2 * Math.PI *
currentDateTime.Second / 60.0;

                //Gets the endpoint for the second hand
                var secondEndPoint = new PointF((float)(secondHandXFactor *
Math.Cos(second_angle)),(float)(secondHadYFactor *
Math.Sin(second_angle)));

                //Sets the color for the second hand
                handsPen.Color = secondHandColor;
                handsPen.Width = 1;

                //Draws the second hand.
                graphicsInstance.DrawLine(Pens.Red, secondEndPoint,
clockFaceCenter);
            }
        }

        /// <summary>
        /// Draws an analog clock face in a client area (usually a form's
client area).
        /// </summary>
        /// <param name="graphicsInstance">Graphics object associated with
the client area.</param>
        /// <param name="clientSize">Width and height of the client
area.</param>
        /// <param name="penColor">Color used to draw the analog clock
face.</param>
        /// <param name="penWidth">Width of the pen used to draw the analog
clock face.</param>
        public static void DrawClockFace(this Graphics graphicsInstance,
Size clientSize, Color penColor, int penWidth)
        {
            using (var clockPen = new Pen(penColor, penWidth))
            {
                //Draws the clock face’s outline.
                graphicsInstance.DrawCircle(clientSize, penColor,
penWidth);

                //Draws the tick marks around the clock's face outline.

                //Defines a round cap for beginning and ending of every
line.
                clockPen.StartCap = LineCap.Round;
                clockPen.EndCap = LineCap.Round;

                //Defines the scale factors used to draw
                //the tick marks around the clock's face.

                //Scale factors for the mark's outer point (x,y).
                var outXFactor = 0.45f * clientSize.Width;
                var outYFactor = 0.45f * clientSize.Height;

                //Scale factors for the mark's inner point (x,y).
                var innXFactor = 0.425f * clientSize.Width;
                var innYFactor = 0.425f * clientSize.Height;

                //Scale factors for the hour mark's inner point (x,y).
                var hourXFactor = 0.4f * clientSize.Width;
                var hourYFactor = 0.4f * clientSize.Height;

                //Loop for iterating the hour's minutes.
                for (int minute = 1; minute <= 60; minute++)
                {
                    //Calculates the angle for every minute.
                    var minuteAngle = Math.PI * minute / 30.0;
                    var cosineAngle = (float)Math.Cos(minuteAngle);
                    var sineAngle = (float)Math.Sin(minuteAngle);

                    //Calculates inner and outer points
                    //for the current tick mark.
                    var innerPoint = (minute % 5 == 0) ? new
PointF(hourXFactor * cosineAngle,hourYFactor * sineAngle) : new
PointF(innXFactor * cosineAngle, innYFactor * sineAngle);
                    var outerPoint = new PointF(outXFactor *
cosineAngle,outYFactor * sineAngle);

                    graphicsInstance.DrawLine((minute % 5 == 0) ? clockPen
: Pens.Black,innerPoint,outerPoint);
                }
            }
        }

        /// <summary>
        /// Draws a circle within a client area (usually a form's client
area).
        /// </summary>
        /// <param name="graphicsInstance">Graphics object associated with
the client area.</param>
        /// <param name="clientSize">Width and height of the client
area.</param>
        /// <param name="penColor">Color used to draw the circle.</param>
        /// <param name="penWidth">Width of the pen used to draw the
circle.</param>
        public static void DrawCircle(this Graphics graphicsInstance,Size
clientSize,Color penColor, int penWidth)
```

```
        {
            using (var circlePen = new Pen(penColor, penWidth))
            {
                graphicsInstance.DrawEllipse(circlePen, -clientSize.Width / 
2, -clientSize.Height / 2, clientSize.Width, clientSize.Height);
            }
        }

        /// <summary>
        /// Builds a context menu based on the elements of a Dictionary.
        /// </summary>
        /// <param name="contextMenu">Instance of the context menu to 
build.</param>
        /// <param name="menuItems">Dictionary with all menu items. The key 
is used in the Tag property to identify the item.</param>
        /// <param name="clickEventHandler">Handler for the Click event of 
every item.</param>
        public static void BuildContextMenu(this ContextMenuStrip 
contextMenu, Dictionary<string,string> menuItems, EventHandler 
clickEventHandler)
        {
            menuItems.ToList().ForEach(item =>
            {
                var menuItem = new ToolStripMenuItem
                {
                    Text = item.Value,
                    Tag = item.Key
                };
                menuItem.Click += clickEventHandler;

                contextMenu.Items.Add(menuItem);
            });
        }

    }
}
```

Next, we need to add references to the **`System.Drawing`** and **`System.Windows.Forms`** assemblies. Click **Build** > **Build Solution** in the Visual Studio menu bar to create the assembly.

Setting up a development environment for building the NuGet package

In this section, we will review all tools we need to set up our development environment in order to build and publish a NuGet package.

The NuGet Command Line Interface

The NuGet Command Line Interface (CLI) nuget.exe is an application that provides the full power of NuGet functionality to install, create, publish, and manage packages without making changes to our project files.

Installing nuget.exe on our computer

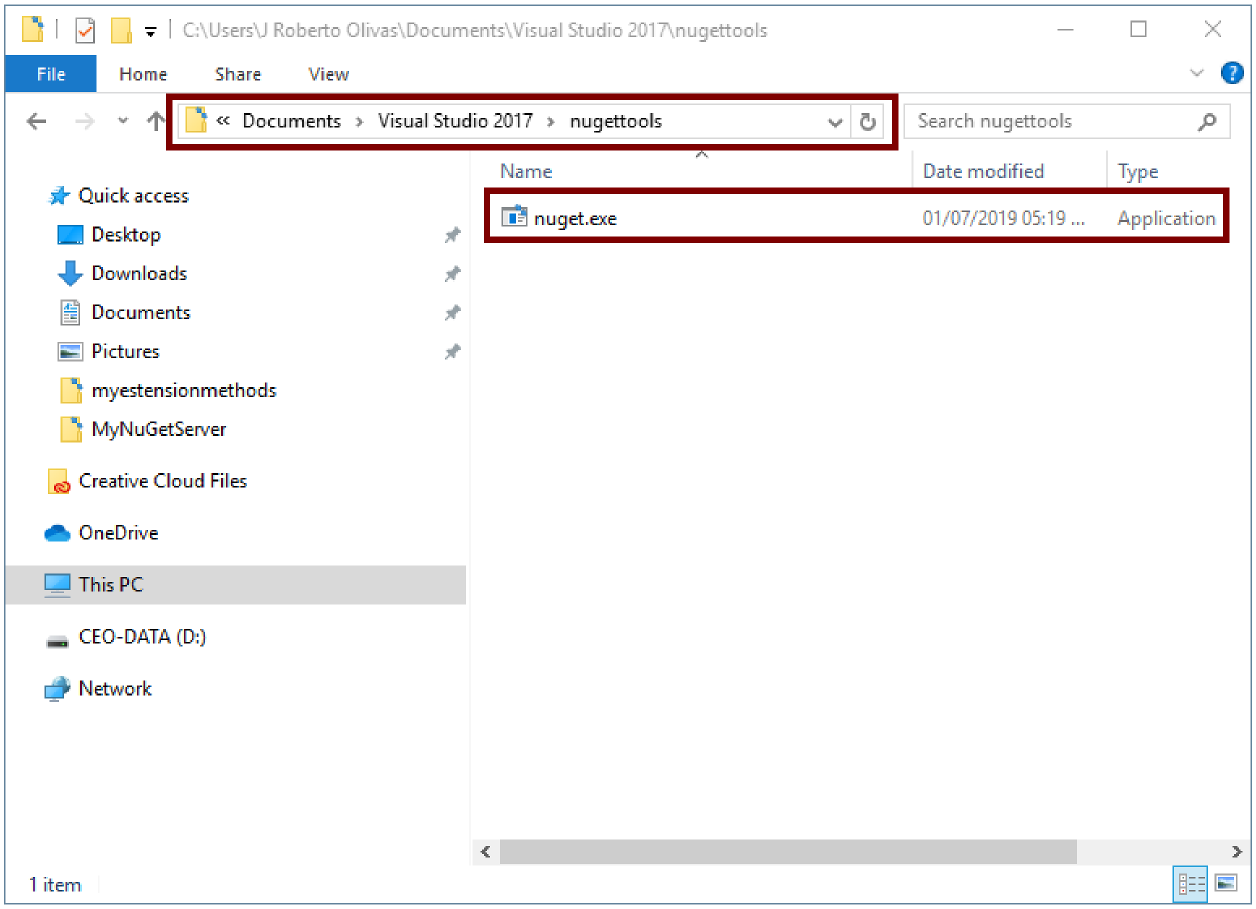
We can install nuget.exe by performing the following steps.

Create a folder named **nugettools** in **Documents\Visual Studio 2017**. Visit this website and select the latest version of nuget.exe. At the time of writing this book, I am using version 5.0.2.

Next, we will instruct our browser to save the file in the folder created in the previous step. Add the **Documents\Visual Studio 2017\nugettools** folder to the PATH system environment variable to make nuget.exe available from anywhere.

Note: The nuget.exe file downloaded is the CLI application, not an installer.

The following figure shows the NuGet Command Line Interface application in the folder after it is downloaded.



Name	Date modified	Type
nuget.exe	01/07/2019 05:19 ...	Application

Figure 49: nuget.exe application after it is downloaded

Nuget.exe available commands

The following table summarizes the available nuget.exe commands.

Table 1: Nuget.exe commands summary

Command	Role	NuGet Version	Description
`pack`	Creation	2.7+	Creates a NuGet package from a .nuspec or project file.
`push`	Publishing	All	Publishes a package to a package source.
`config`	All	All	Gets or sets NuGet configuration values.
`help` or `?`	All	All	Displays help information or help for a command.
`locals`	Consumption	3.3+	Lists locations of the **global-packages**, **http-cache**, and **temp** folders and clears the contents of those folders.
`restore`	Consumption	2.7+	Restores all packages referenced by the package management format in use.
`setapikey`	Publishing, Consumption	All	Saves an API key for a given package source when that package source requires a key for access.
`spec`	Creation	All	Generates a .nuspec file, using tokens if generating the file from a Visual Studio project.
`add`	Publishing	3.3+	Adds a package to a non-HTTP package source using hierarchical layout. (For HTTP sources, use `push`.)
`delete`	Publishing	All	Removes or un-lists a package from a package source.

Command	Role	NuGet Version	Description
`init`	Creation	3.3+	Adds packages from a folder to a package source using hierarchical layout.
`install`	Consumption	All	Installs a package into the current project, but does not modify projects or reference files.
`list`	Consumption, perhaps Publishing	All	Displays packages from a given source.
`mirror`	Publishing	Deprecated in 3.2+	Mirrors a package and its dependencies from a source to a target repository.
`sources`	Consumption, Publishing	All	Manages package sources in configuration files.
`update`	Consumption	All	Updates a project's packages to the latest available versions.

Note: A complete explanation of each of the commands displayed in Table 1 is outside the scope of this book.

The NuGet Package Manager

The NuGet Package Manager is a tool for simply installing, uninstalling, and updating NuGet packages in projects and solutions. Visual Studio 2017 includes this tool with two options: the Package Manager Console, and a graphical user interface. We can execute both options from the **Tools** > **NuGet Package Manager** submenu located in the Visual Studio menu bar. We will explain the use of the Package Manager Console later in this book.

Note: The NuGet Package Manager graphical user interface was explained in the "[Adding the NuGet.Server package to the project](#)" section of Chapter 2.

The NuGet Package Explorer

The NuGet Package Explorer is an application that allows for easily creating and exploring NuGet packages. We can load .nupkg files from disk or directly from a host, such as http://nuget.org.

Installing NuGet Package Explorer

We can install NuGet Package Explorer from the Microsoft Store. First, launch the Microsoft Store application in Windows 10, and then type **NuGet Package Explorer** in the **Search** text box and press **Enter**. Once the search results appear in the dialog box, click the **NuGet Package** icon displayed in the following figure.

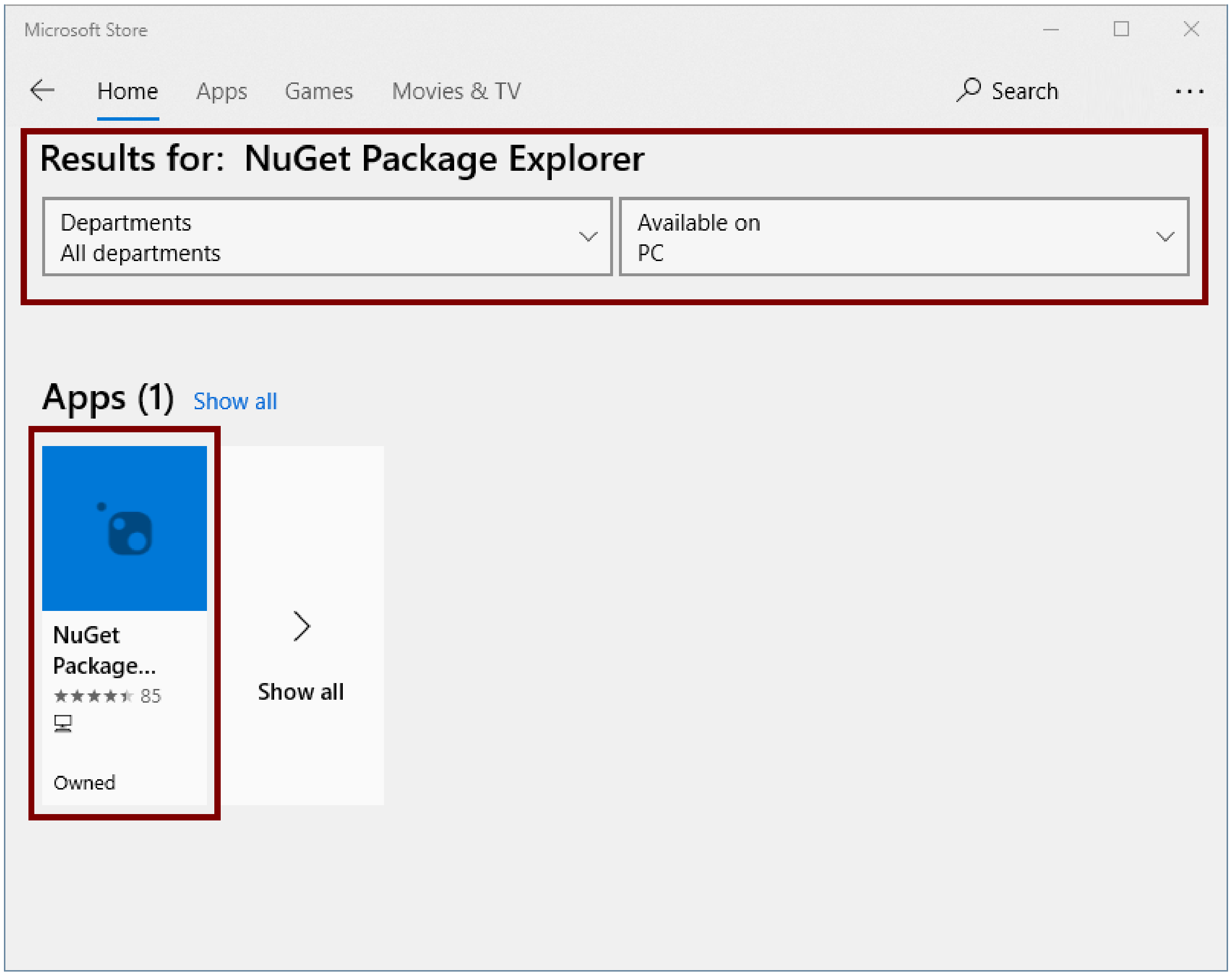
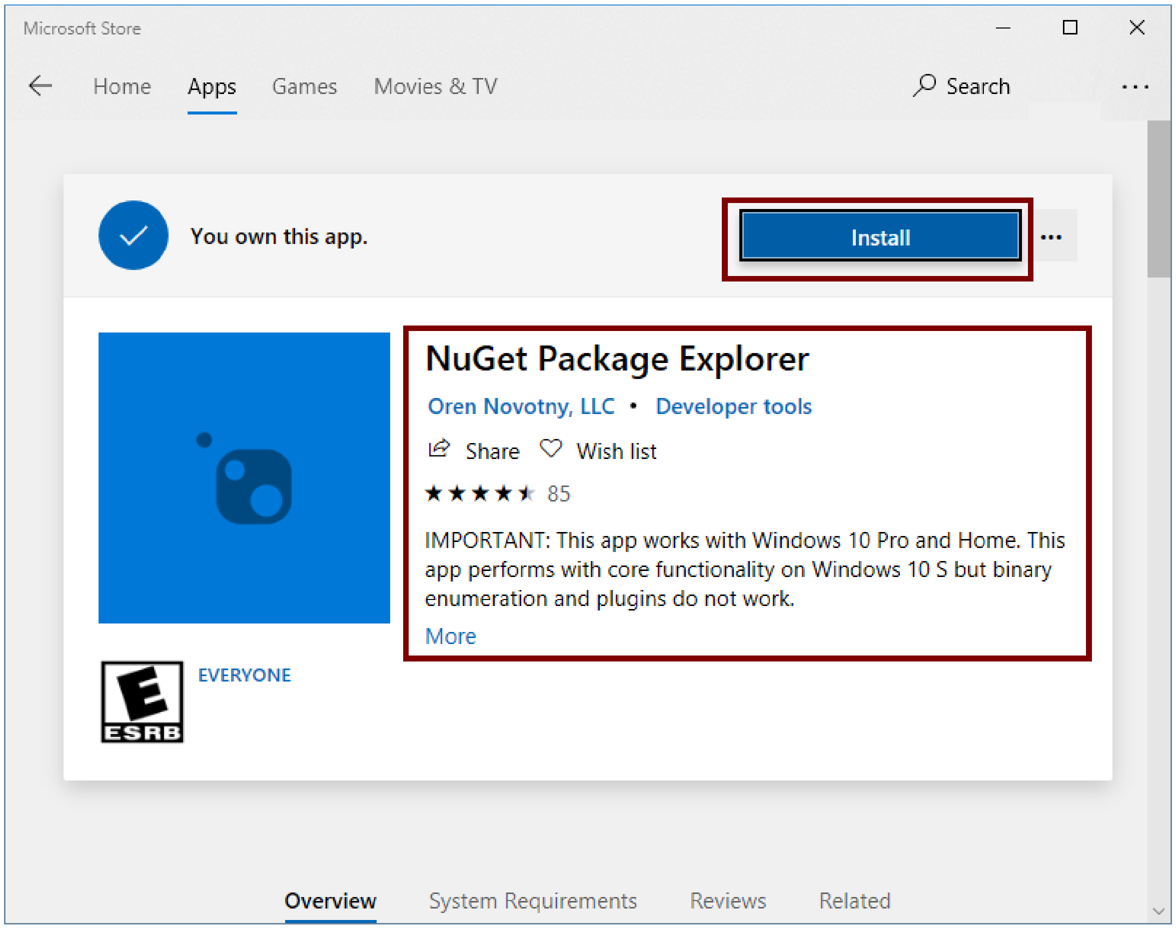


Figure 50: Displaying search results

The application info will appear on the screen. Click the **Install** button located at the top of the application info area. The Microsoft Store application will download the NuGet Package Explorer and install it to our local computer. The following image shows the application info in the Microsoft Store application.



You own this app.

NuGet Package Explorer

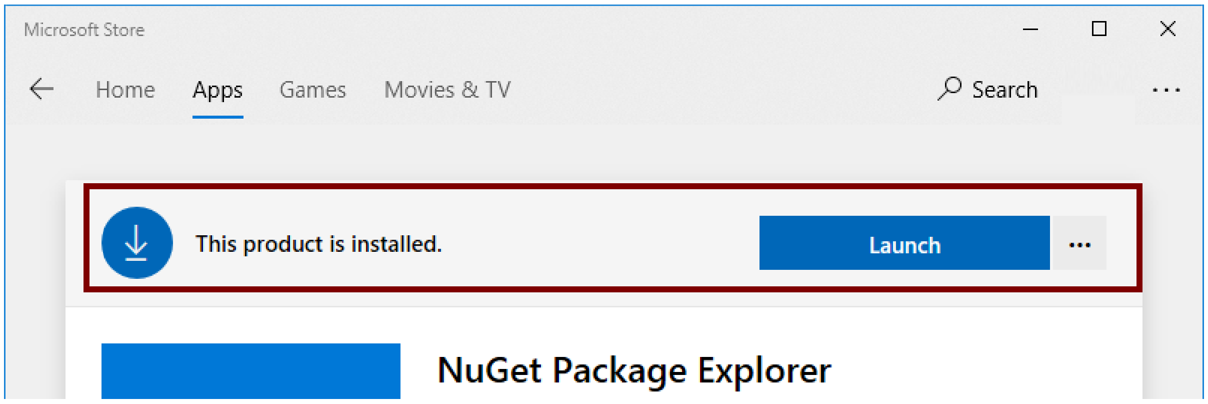
Oren Novotny, LLC • Developer tools

★★★★☆ 85

IMPORTANT: This app works with Windows 10 Pro and Home. This app performs with core functionality on Windows 10 S but binary enumeration and plugins do not work.
More

Figure 51: NuGet Package Explorer application info

When installation ends, the Microsoft Store application will replace the Install button with the Launch button, as displayed in the following figure.



This product is installed.

NuGet Package Explorer

Figure 52: Microsoft Store application after installation process

Click **Launch** to make sure that the NuGet Package Explorer installation was successful. If everything is fine, we should see the dialog box displayed in the following image.

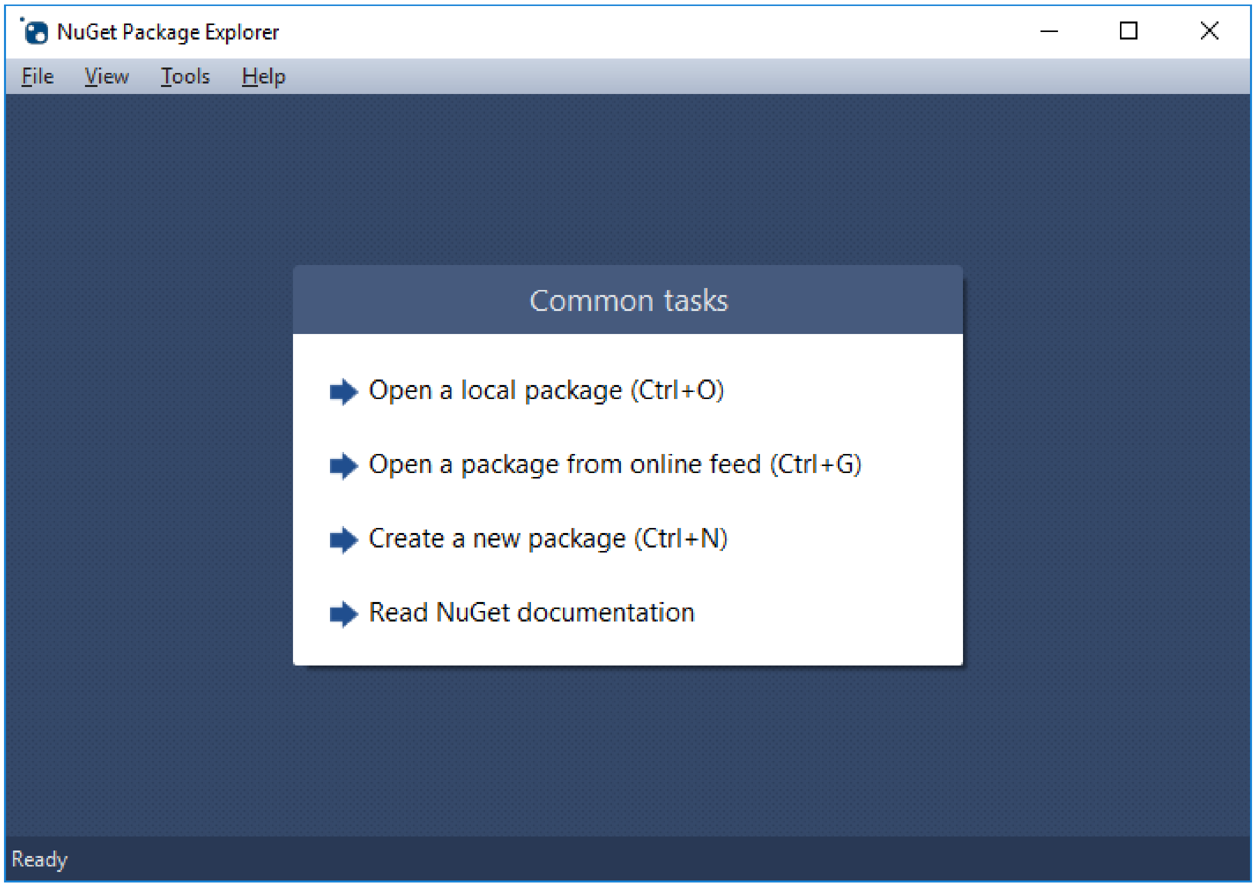

NuGet Package Explorer
File
View
Tools
Help
Common tasks
Open a local package (Ctrl+O)
Open a package from online feed (Ctrl+G)
Create a new package (Ctrl+N)
Read NuGet documentation
Ready

Figure 53: NuGet Package Explorer main window

Note: I will explain the use of the NuGet Package Explorer in the next chapter of this book.

Chapter summary

In this chapter, we defined a project to publish as a NuGet package. We created a .NET Framework class library with a set of extension methods named **myextensionmethods**, and we saved it in the **Documents\Visual Studio 2017\Projects** folder. We will use the compiled class library in a Windows Forms application, which will be explained in Chapter 5.

After creating and building the class library project, we reviewed all the necessary tools for creating, managing, and publishing NuGet packages, such as the NuGet Command Line Interface (CLI). The NuGet CLI is an application that provides the functionality to work with NuGet Packages without making changes to our project files. In addition, we reviewed the NuGet Package Manager, which is a tool included in Visual Studio 2017 that can install, uninstall, and update NuGet packages in projects and solutions. Finally, we reviewed the NuGet Package Explorer, which is an application for easily creating and exploring NuGet packages.

Chapter 4 Creating and Publishing the Package in a Local NuGet Server

Creating a folder structure to build the package

Let us go back to the **myextensionmethods** project created in Chapter 3. We will share the compiled code for this project as a NuGet package. In order to do this, we should make a few changes to our project. First, we are going to add a couple of folders in order to save the necessary files for building and deploying the package.

To create these folders, right-click the **myextensionmethods** node in Solution Explorer, and then click **Add** > **New Folder** in the context menu. For the purposes of this book, we will create two folders: **packagesource** and **packagedeploy**.

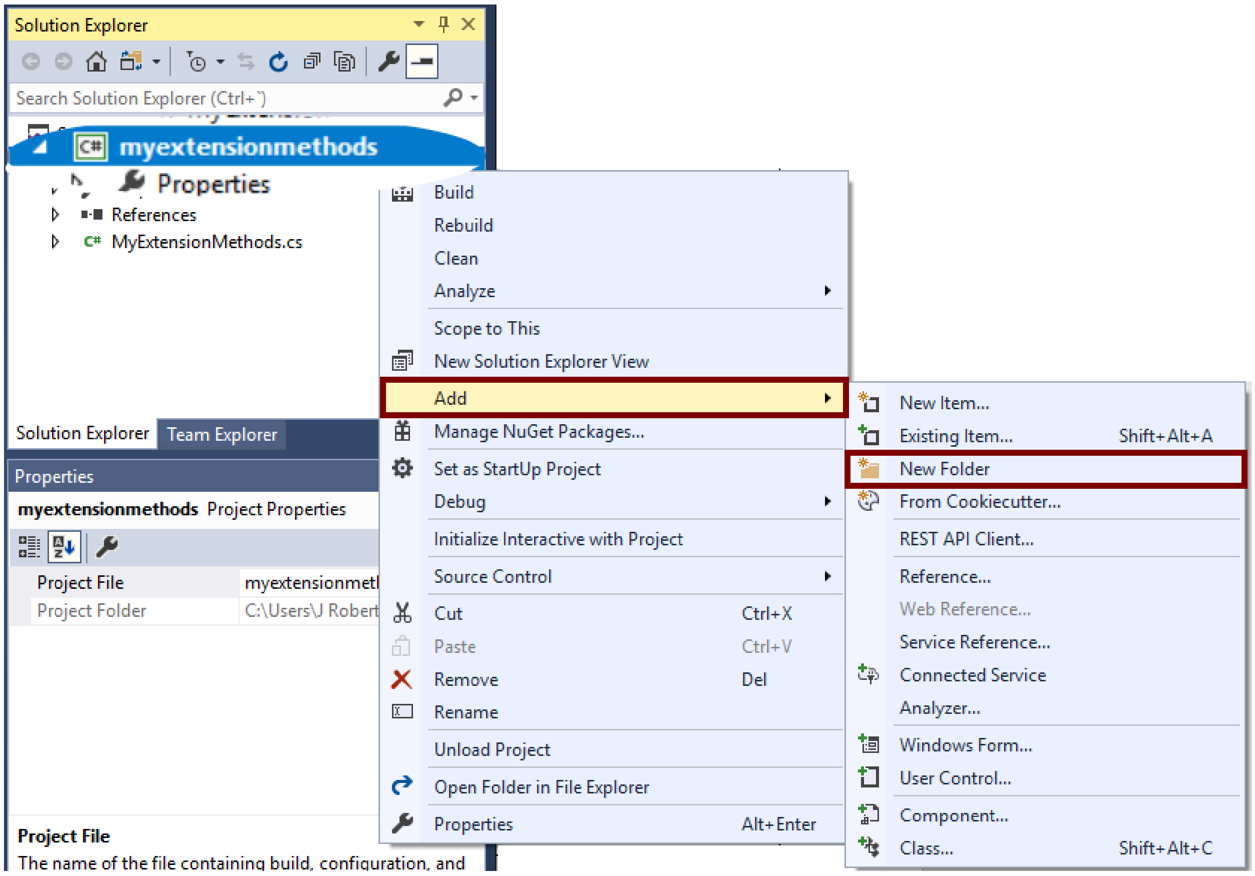


Figure 54: Creating the folders for our project

After creating the folders, the Solution Explorer will look like the following figure.

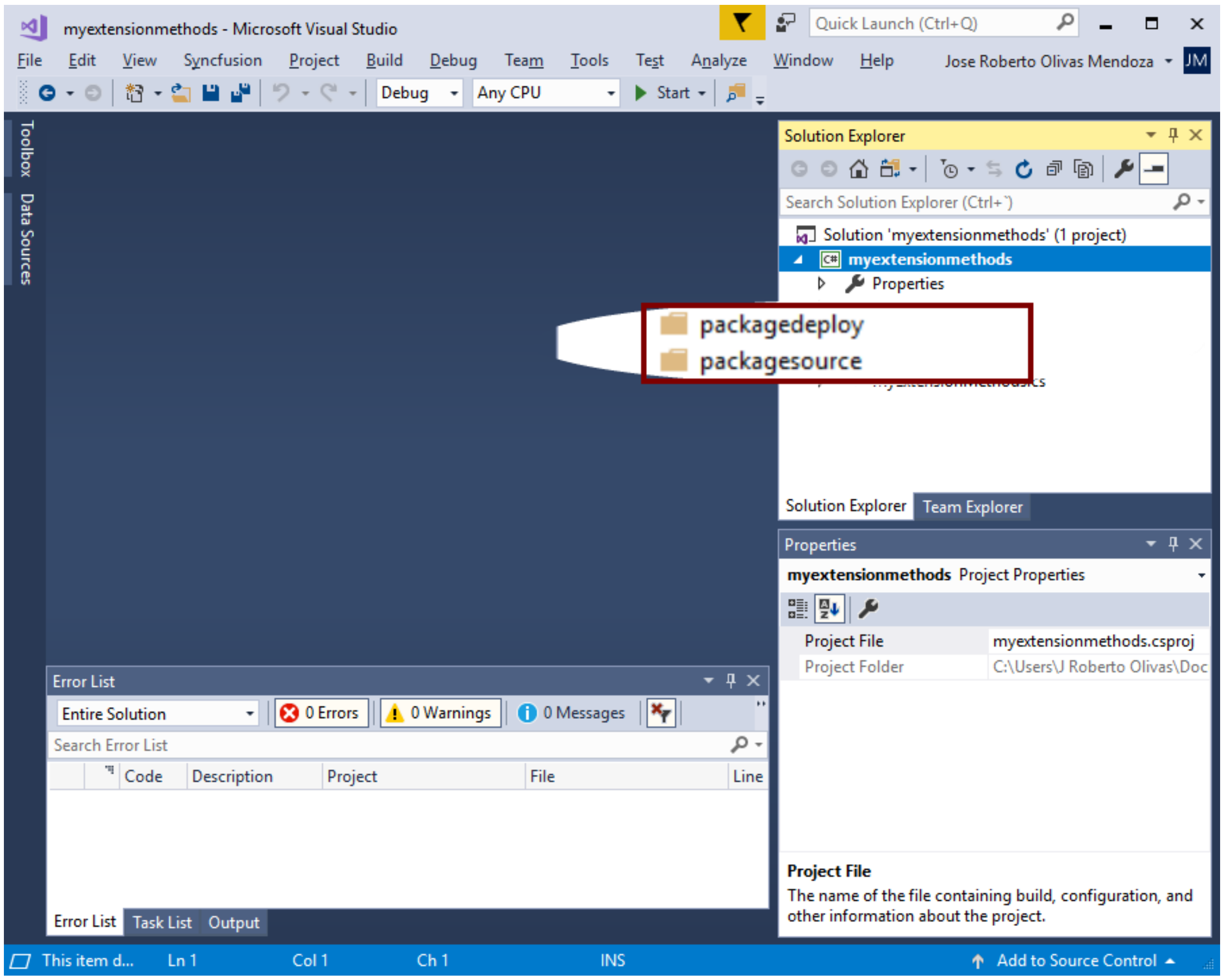


Figure 55: The Solution Explorer after creating the folders

At this point, we are ready to start the process for building our NuGet package.

The .nuspec file

A .nuspec file is a text file in XML format. This file contains the metadata used to build the NuGet package and to offer information to consumers. The package will always contain the .nuspec file.

The .nuspec file bases its syntax and structure in the nuspec.xsd schema file. We can download this file from GitHub. According to this schema, a general form for a .nuspec file should look like the following example.

Code Listing 2: A generic .nuspec file

```
<?xml version="1.0" encoding="utf-8"?>
<package xmlns="http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd">
```

```
    <metadata>
        <!-- Required elements-->
        <id></id>
        <version></version>
        <description></description>
        <authors></authors>

        <!-- Optional elements -->
        <!-- ... -->
    </metadata>
    <!-- Optional 'files' node -->
</package>
```

We can see in Code Listing 2 that some elements appear as required, and other elements appear as optional. The following table summarizes the most relevant required and optional elements.

Table 2: .nuspec file metadata elements

Element	Description	Required
`id`	The package identifier. This identifier must be unique across the gallery the package resides in (nuget.org or a private host). IDs should not contain spaces or characters that are invalid for a URL. They usually follow .NET namespace rules.	Yes
`version`	The version of the package. It should follow the **`major.minor.patch`** pattern. We may include one of the following prerelease suffixes. • **`-alpha`**: Typically used for works in progress and research. • **`-beta`**: Typically, one that is complete according to the next planned release. This version may contain known bugs. • **`-rc`**: Typically, a release that's potentially stable, unless significant bugs emerge.	Yes
`description`	A long description of the package, for UI display purposes.	Yes
`authors`	A comma-separated list of packages authors. When the package is hosted at nuget.org, these names should match the profile names on the NuGet Gallery because the gallery displays them and also uses them to cross-reference packages by the same authors.	Yes
`title`	A human-friendly title of the package. This is typically for UI display purposes. Also, it appears at nuget.org if we host the package there, and at the Package Manager in Visual Studio. The package ID replaces this element if this is not specified.	No

Element	Description	Required
`owners`	A comma-separated list of the package creators. This is similar to the list in **`authors`**, and is ignored when we host the package at nuget.org.	No
`projectUrl`	The URL for the package's home page, for UI display purposes. This URL is also shown when we host the package at nuget.org.	No
`summary`	This is a short description of the package for UI display purposes. A truncated version of **`description`** is used if this element is omitted.	No
`copyright`	Copyright credits for the package.	No
`files`	This is a node contained in the package node, and specifies which assembly files are included in the package and its target location at consuming time.	No

Note: An in-depth explanation of all .nuspec file elements is outside the scope of this book.

Creating the .nuspec file for the project

Using nuget.exe

We need to build a .nuspec file before creating a NuGet package. We will use the NuGet Command Line Interface to accomplish this task. Assuming that the **myextensionmethods** assembly is in the **Documents\Visual Studio 2017\Projects\myextensionmethods\myextensionmethods\bin\Debug** folder, we will launch the **Command Prompt** application and use the **`cd`** command to navigate to the folder, so that it is the working directory. After that, we are going to use **nuget.exe** to create the .nuspec file by using the command shown in the following example.

Code Listing 3: Command for creating the .nuspec file

```
C:\Users\Your user name\Documents\Visual Studio
2017\Projects\myextensionmethods\myextensionmethods\bin\Debug> nuget spec
myextensionmethods
Created 'myextensionmethods.nuspec' successfully.
```

We can notice in the previous example that **nuget.exe** tells us when the **myextensionmethods.nuspec** file is created. Now, if we open the file using the **Notepad** application, the contents should look like the following example.

Code Listing 4: The contents of the myextensionmethods.nuspec file

```
<?xml version="1.0"?>
<package >
  <metadata>
    <id>myextensionmethods</id>
    <version>1.0.0</version>
    <authors>Your user name</authors>
    <owners>Your user name</owners>
    <licenseUrl>http://LICENSE_URL_HERE_OR_DELETE_THIS_LINE</licenseUrl>
    <projectUrl>http://PROJECT_URL_HERE_OR_DELETE_THIS_LINE</projectUrl>
    <iconUrl>http://ICON_URL_HERE_OR_DELETE_THIS_LINE</iconUrl>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>Package description</description>
    <releaseNotes>Summary of changes made in this release of the
package.</releaseNotes>
    <copyright>Copyright 2019</copyright>
    <tags>Tag1 Tag2</tags>
    <dependencies>
      <dependency id="SampleDependency" version="1.0" />
    </dependencies>
  </metadata>
</package>
```

Tuning up the .nuspec file

The previous example displays the default **myextensionmethods.nuspec** file created by **nuget.exe**. It is our responsibility to make the necessary changes to this file before creating the package. Therefore, we need to delete all unnecessary elements and add those not included by **nuget.exe**. After that, our **myextensionmethods.nuspec** file should look like the following example.

Code Listing 5: The myextensionmethods.nuspec file after tune-up

```
<?xml version="1.0"?>
<package >
  <metadata>
    <id>myextensionmethods</id>
    <version>1.0.0</version>
    <authors>Your username</authors>
    <owners>Your username</owners>
    <description>Extension Methods for NuGet In-House
Succinctly</description>
    <copyright>Copyright 2019 NuGet In-House Succinctly</copyright>
  </metadata>
  <files>
   <file src="..\bin\Debug\myextensionmethods.dll" target="lib\net46" />
  </files>
</package>
```

In this example, the **`<files>`** node allows us to declare all the assemblies and files to include in the package. Every file declared corresponds to a **`<file>`** child node, which employs two attributes for file declaration. The first attribute is **`src`**, which points to the path of the included file or files (we can use wild cards). The second attribute, **`target`**, points to a relative path to the folder within the package, where the NuGet Command Line Interface (or NuGet Package Explorer) places the file. According to NuGet conventions, this path should start with **`lib`**. We will explain this attribute later in this book.

After saving the modified file, we should copy it into the **Documents\Visual Studio 2017\Projects\myextensionmethods\myextensionmethods\packagesource** folder, created in the first section of this chapter.

Replacement tokens

As noted in Code Listing 5, the values for the **`<metadata>`** child nodes are string constants, meaning we hard-coded these values. Sometimes, we need to supply variable values to these nodes. To make this happen, NuGet offers a set of replacement tokens. We should place these tokens instead of values, and we can send values to them from the NuGet Command Line Interface (nuget.exe) at the time of creating the package. The following table summarizes the available replacement tokens.

Table 3: NuGet replacement tokens

Token	Value source	Value
`$id$`	Project File	Assembly name (title) from the project file.
`$version$`	Assembly Info	AssemblyInformationalVersion if present, otherwise AssemblyVersion.
`$description$`	Assembly Info	Assembly Description
`$author$`	Assembly Info	Assembly Company
`$title$`	Assembly Info	Assembly Title
`$copyright$`	Assembly Info	Assembly Copyright
`$configuration$`	Assembly DLL	Configuration used to build the assembly (Debug is the default).

Note: For the `$configuration$` token, when using the Release configuration, we should always employ `-properties Configuration=Release` on the NuGet Command Line Interface. We will explain this later in this book.

Adding support for multiple .NET Framework versions

We can add support for multiple framework versions in a single package. We can accomplish this by using the **`target`** attribute of the **`<file>`** node. As we explained in the previous section, the value of this attribute points to a relative path to the folder within the package, where NuGet Command Line Interface (or NuGet Package Explorer) places the file.

NuGet conventions for relative paths

There are some NuGet conventions about how to specify a relative path in the **`<file>`** node. Let's look at a couple of them in the following table.

Table 4: NuGet folder naming conventions

Folder	Description	Action upon package install
(root)	Location for the **readme.txt** file.	Visual Studio displays a **readme.txt** file in the package root when the package is installed.
lib\{tfm}	Location for the assemblies (.dll), documentation (.xml), and symbol (.pdb) files for the specified target framework moniker (TFM).	Visual Studio adds the assemblies as references for compile time as well as runtime. Also, it copies .xml and .pdb files into project folders. The target framework moniker corresponds to a subfolder for a specific framework version.
ref\{tfm}	Location for the assemblies (.dll) and symbol (.pdb) files for the specified target framework moniker (TFM).	Visual Studio adds the assemblies as references only for compile time; therefore, nothing will be copied into the project **bin** folder.
content	Arbitrary files, according to the project's architecture.	Visual Studio copies these files to the project root of the target application that consumes the package. Therefore, if we need to add a file named **transform.xsd** in the target application's **xmlstylesheets** folder, we should place this file in the package's **content\xmlstylesheets** folder.

Target framework monikers

A target framework moniker (TFM) is a standardized token format for specifying the target framework of a .NET application or library. The following table summarizes the available target framework monikers.

Table 5: NuGet folder naming conventions

Target Framework	Target Framework Monikers
.NET Standard	netstandard1.0 netstandard1.1 netstandard1.2 netstandard1.3 netstandard1.4 netstandard1.5 netstandard1.6 netstandard2.0
.NET Core	netcoreapp1.0 netcoreapp1.1 netcoreapp2.0 netcoreapp2.1 netcoreapp2.2
.NET Framework	net11 net20 net35 net40 net403 net45 net451 net452 net46 net461 net462 net47 net471 net472 net48
Windows Store	netcore [netcore45] netcore45 [win] [win8] netcore451 [win81]

Target Framework	Target Framework Monikers
.NET Micro Framework	netmf
Universal Windows Platform	uap [uap10.0] uap10.0 [win10] [netcore50]

Now, we will modify the .nuspec file displayed in Code Listing 5 in order to support both .NET Framework 4.7 and .NET Framework 4.6. After making changes, we should save the file. The contents should look like the following example.

Code Listing 6: .nuspec file with multiple frameworks support

```
<?xml version="1.0"?>
<package >
  <metadata>
    <id>myextensionmethods</id>
    <version>1.0.0</version>
    <authors>Your username</authors>
    <owners>Your username</owners>
    <description>Extension Methods for NuGet In-House
Succinctly</description>
    <copyright>Copyright 2019 NuGet In-House Succinctly</copyright>
  </metadata>
  <files>
   <file src="..\bin\Debug\myextensionmethods.dll" target="lib\net47" />
   <file src="..\bin\Debug\myextensionmethods.dll" target="lib\net46" />
  </files>
</package>
```

Using NuGet Package Explorer to view the .nuspec file

We can use the NuGet Package Explorer to view our .nuspec file. The first step is to launch the application and click **Open a local package**.

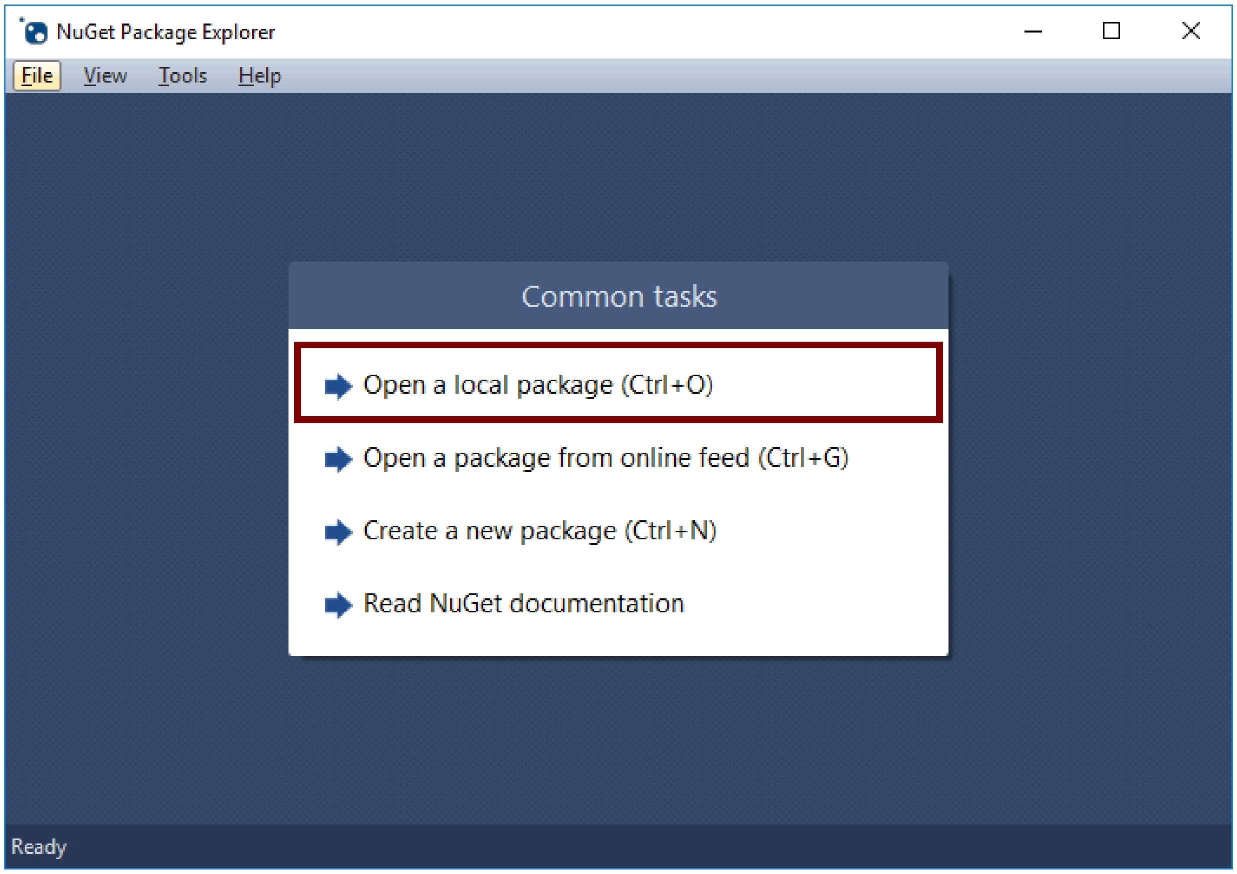
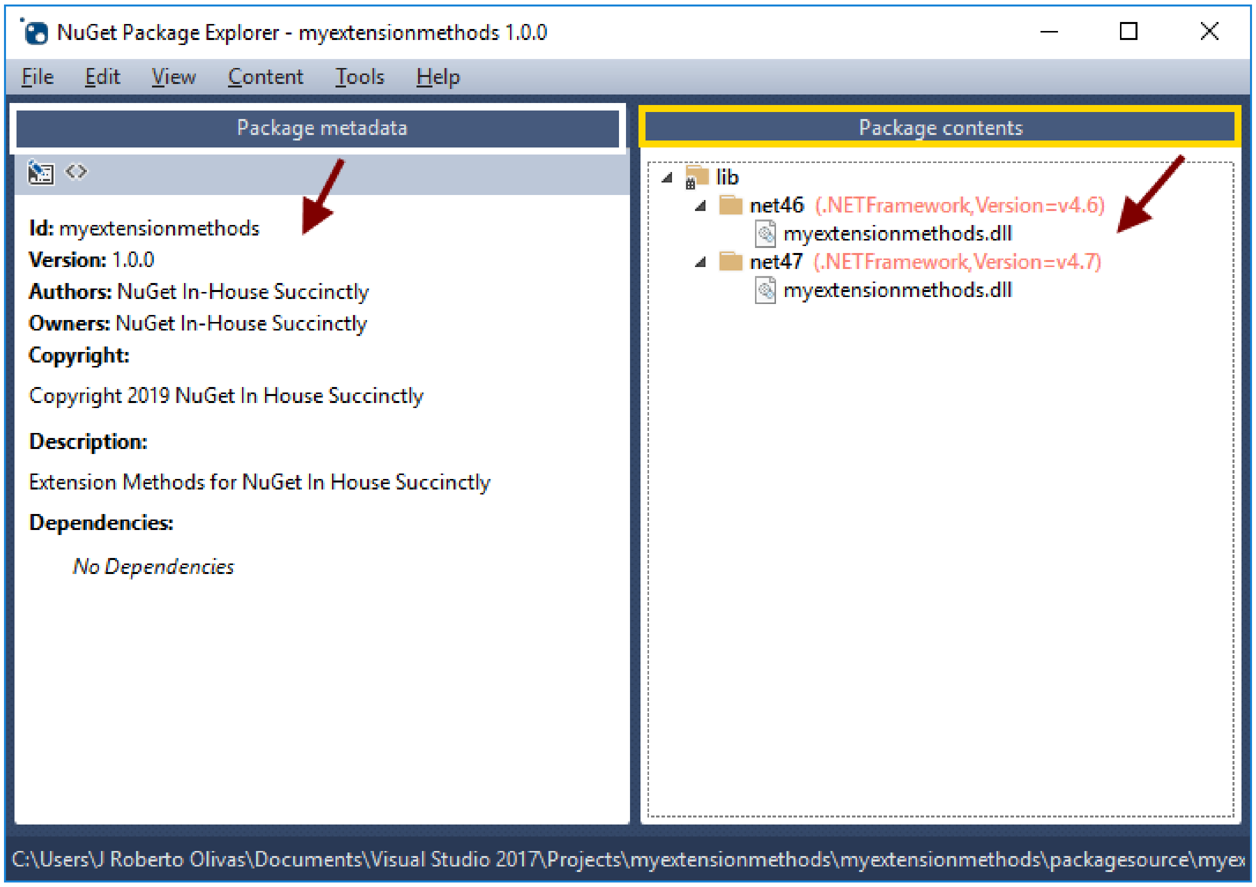

NuGet Package Explorer
File
View
Tools
Help
Common tasks
Open a local package (Ctrl+O)
Open a package from online feed (Ctrl+G)
Create a new package (Ctrl+N)
Read NuGet documentation
Ready

Figure 56: Using the NuGet Package Explorer to view a .nuspec file

Now, when the Select File dialog box appears, we should go to the **Documents\Visual Studio 2017\Projects\myextensionmethods\myextensionmethods\packagesource** folder and double-click the **myextensionmethods.nuspec** file name. The package explorer should look like the following figure.


NuGet Package Explorer - myextensionmethods 1.0.0

Id: myextensionmethods
Version: 1.0.0
Authors: NuGet In-House Succinctly
Owners: NuGet In-House Succinctly
Copyright:
Copyright 2019 NuGet In House Succinctly
Description:
Extension Methods for NuGet In House Succinctly
Dependencies:
No Dependencies

```
lib
  net46 (.NETFramework,Version=v4.6)
    myextensionmethods.dll
  net47 (.NETFramework,Version=v4.7)
    myextensionmethods.dll
```

C:\Users\J Roberto Olivas\Documents\Visual Studio 2017\Projects\myextensionmethods\myextensionmethods\packagesource\myex

Figure 57: NuGet Package Explorer after opening .nuspec file

Creating the package

Using the NuGet Command Line Interface

With our .nuspec file ready for use, it is time to create the NuGet package for publishing on our local server.

Using replacement tokens

We will make a few changes to our .nuspec file before creating the package: we will place replacement tokens (discussed previously) in some child nodes of the **`<metadata>`** and **`<file>`** nodes in order to send values to them at package creation time. The following example displays our .nuspec file after making these changes.

Code Listing 7: .nuspec file with replacement tokens

```
<?xml version="1.0"?>
<package >
  <metadata>
```

```
    <id>myextensionmethods</id>
    <version>$version$</version>
    <authors>$author$</authors>
    <owners>$author$</owners>
    <description>$description$</description>
    <copyright>$copyright$</copyright>
  </metadata>
  <files>
    <file src="..\bin\$configuration$\myextensionmethods.dll"
target="lib\net47" />
    <file src="..\bin\$configuration$\myextensionmethods.dll"
target="lib\net46" />
  </files>
</package>
```

In this example, we can identify the use of replacement tokens for the **`<version>`**, **`<copyright>`**, and **`<file>`** nodes. The **`$configuration$`** token in the **`<file>`** node tells NuGet the name of the configuration used to create the assembly, either **`Debug`** or **`Release`**. Therefore, depending on the value sent to this token, the path for the assembly file could be **..\bin\Debug\myextensionmethods.dll** or **..\bin\Release\myextensionmethods.dll**. We also need to send values for the rest of these tokens when executing the NuGet Command Line Interface. Table 3, displayed previously, shows the values for each one of the tokens that we could use in the .nuspec file.

Note: A .nuspec file with replacement tokens should not be used with NuGet Package Browser.

Executing NuGet Command Line Interface

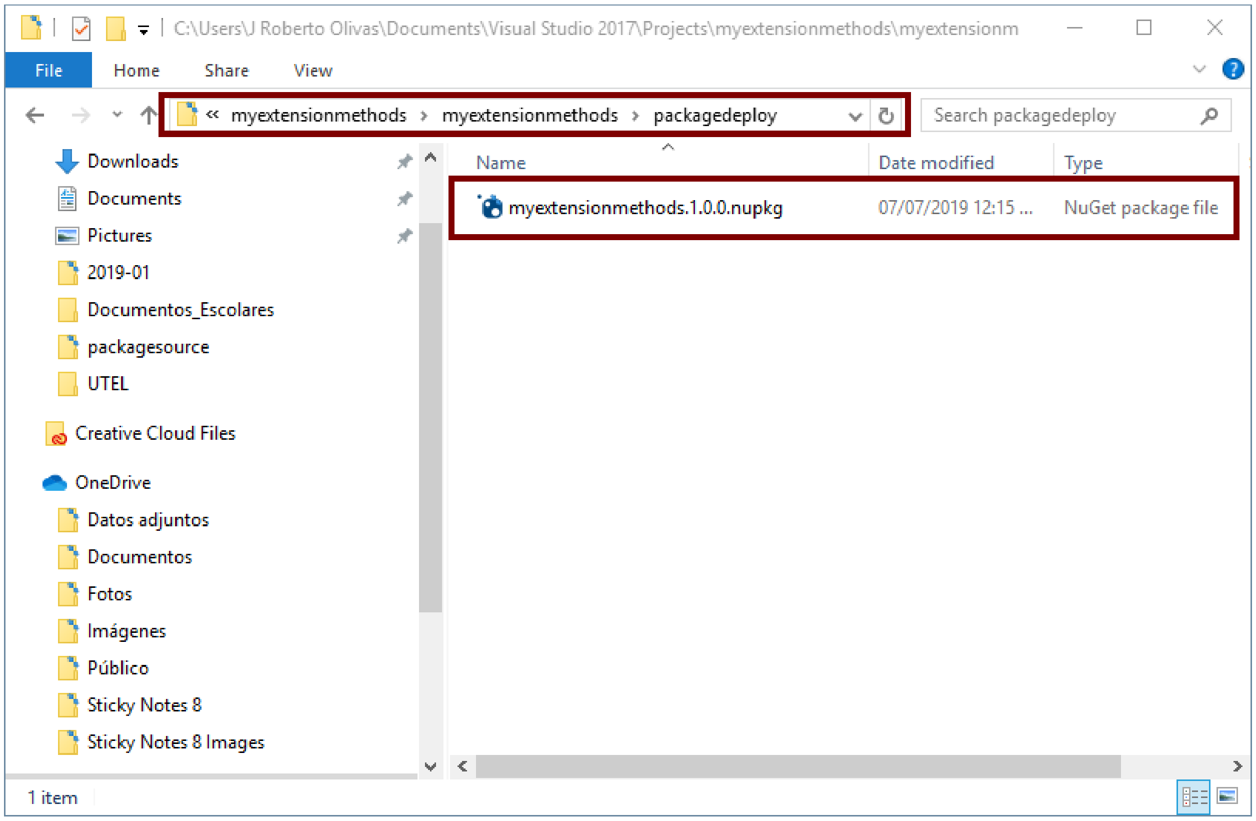
We should open a command prompt instance by pressing the **Windows logo key**+**R**, typing **cmd** in the text box of the **Run** dialog box that appears, and then pressing **Enter**. In the Command Prompt window, we should go to the **Documents\Visual Studio 2017\Projects\myextensionmethods\myextensionmethods\packagesource** folder. After that, we should type the command displayed in the following example, and then press **Enter**.

Code Listing 8: Creating our NuGet package from the command line

```
nuget pack myextensionmethods.nuspec -OutputDirectory ..\packagedeploy -
Properties Configuration=Debug;Version=1.0.0.0;Description="Extension
Methods for NuGet in House Succinctly";Author="NuGet In House Succinctly"
```

Let us dig a little bit into the previous example. The **`pack`** command of the NuGet Command Line Interface creates our NuGet package. This command takes a list of parameters; the first one is the name of our .nuspec file. Then, the **`-OutputDirectory`** parameter tells the interface where to place the package. For the purposes of this book, we will save the package in the **packagedeploy** folder created previously. Now, the **`-Properties`** parameter sends the values for the replacement tokens that appear in the .nuspec file. Every replacement token has its equivalent property in the **`-Properties`** command. We specify the value for each token to the right of every property, led by an equal sign. In addition, we need to separate properties with a semicolon.

Now, if we go to the **packagedeploy** folder, we can see our NuGet package as displayed in the following figure.



Name	Date modified	Type
myextensionmethods.1.0.0.nupkg	07/07/2019 12:15 ...	NuGet package file

Figure 58: Our NuGet package in the output folder

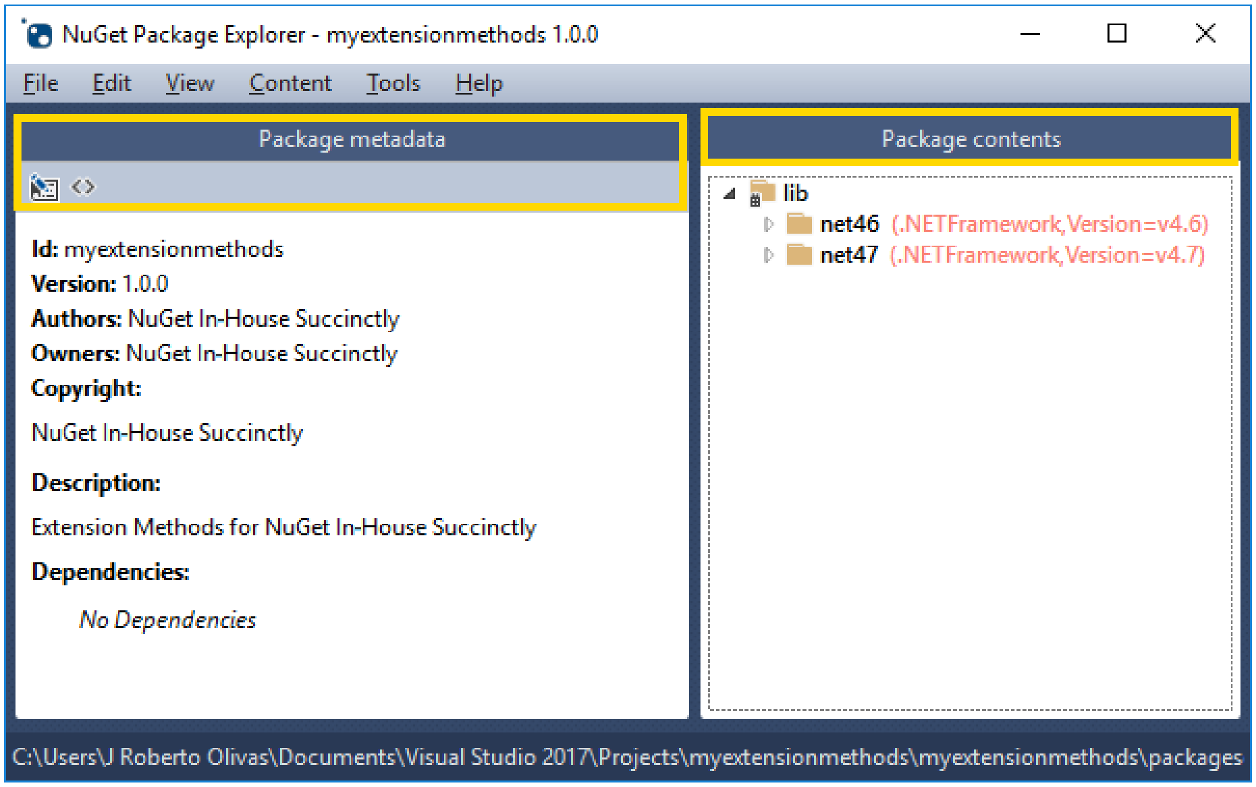
Using NuGet Package Explorer

We also can use the NuGet Package Explorer to create our NuGet package. The difference here is the replacement tokens—the NuGet Package Explorer cannot use them. Therefore, we need to change our .nuspec file and place constant values directly, instead of using replacement tokens in our file. After these changes, the .nuspec file should look like the following example.

Code Listing 9: The .nuspec file for use with NuGet Package Explorer

```
<?xml version="1.0"?>
<package >
  <metadata>
    <id>myextensionmethods</id>
    <version>1.0.0.0</version>
    <authors>NuGet In-House Succinctly</authors>
    <owners>NuGet In-House Succinctly</owners>
    <description>Extension Methods for NuGet In-House
Succinctly</description>
    <copyright> NuGet In-House Succinctly</copyright>
  </metadata>
  <files>
    <file src="..\bin\Debug\myextensionmethods.dll" target="lib\net47" />
    <file src="..\bin\Debug\myextensionmethods.dll" target="lib\net46" />
  </files>
</package>
```

Now, launch the NuGet Package Explorer and click **Open a local package**. Then, using the **Select File** dialog box, select our .nuspec file and click **Open**. The NuGet Package Explorer should look like the following figure.



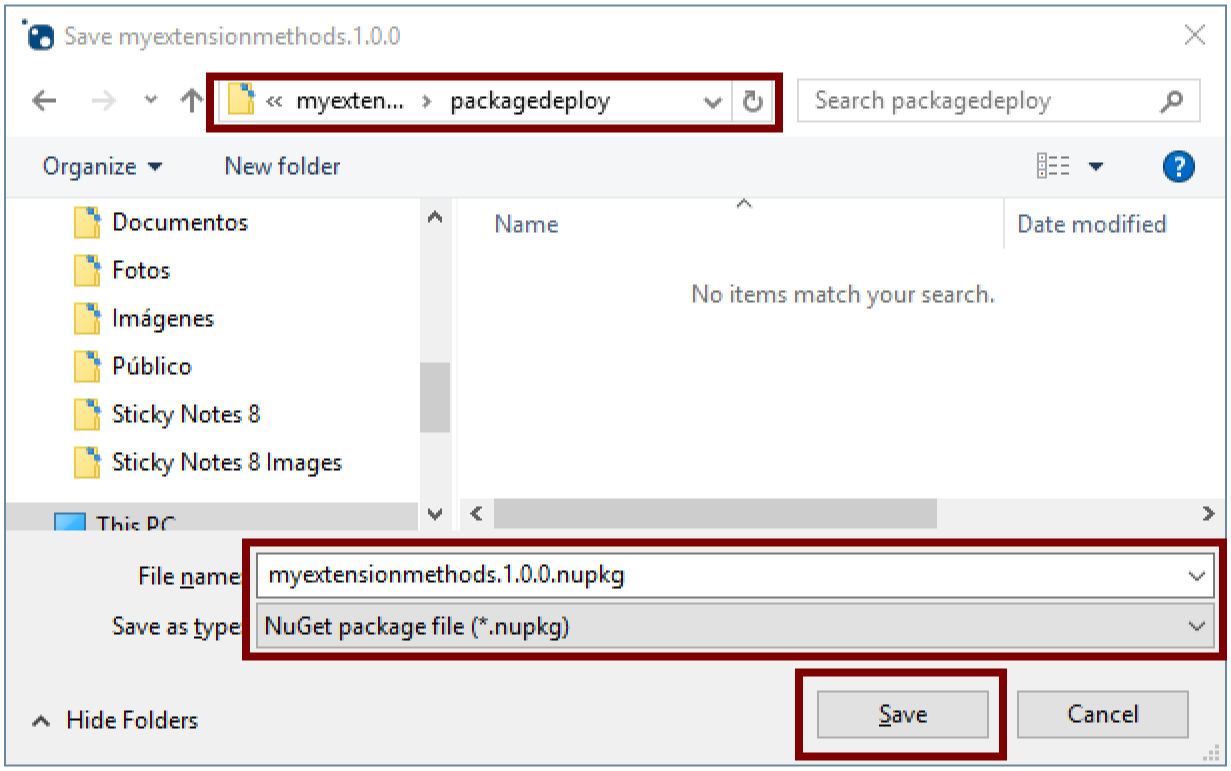
Id: myextensionmethods
Version: 1.0.0
Authors: NuGet In-House Succinctly
Owners: NuGet In-House Succinctly
Copyright:
NuGet In-House Succinctly
Description:
Extension Methods for NuGet In-House Succinctly
Dependencies:
No Dependencies

lib
net46 (.NETFramework,Version=v4.6)
net47 (.NETFramework,Version=v4.7)

C:\Users\J Roberto Olivas\Documents\Visual Studio 2017\Projects\myextensionmethods\myextensionmethods\packages

Figure 59: Our .nuspec file in NuGet Package Explorer

We can create our NuGet package by clicking **File** > **Save** in the NuGet Package Explorer's menu bar. NuGet Package Explorer first saves every change we made to our .nuspec file. Then, it asks for the NuGet Package output folder. For the purposes if this book, we should go to the **packagedeploy** folder created previously, and click **Save**. Now, the NuGet Package Explorer will create the NuGet package.



File name: myextensionmethods.1.0.0.nupkg

Save as type: NuGet package file (*.nupkg)

Figure 60: Saving the NuGet package

Publishing the package in the local server

With our package created, we are ready to publish it in our local server. As in package building, we can use the NuGet Command Line Interface or the NuGet Package Explorer to perform this action.

Using the NuGet Command Line Interface

In the NuGet Command Line Interface, the **`push`** command is responsible for publishing NuGet packages. The following example displays the syntax for this command.

Code Listing 10: push command syntax

```
nuget push {package file} {apikey} -Source {NuGet Host URL}
```

The **`{package file}`** parameter corresponds to the name of our NuGet package. The **`{apikey}`** parameter matches the API key used to identify the NuGet host. According to Chapter 2 of this book, the value for this parameter is **`MyNuGetServer`**. Finally, the **`-Source`** parameter tells the NuGet Command Line Interface the location of the NuGet host. In this case, for our local NuGet server, the location is http://localhost/mynugetserver/nuget. Therefore, for the purposes of this book, we should go to the **packagedeploy** folder and employ the following command to publish our NuGet package.

Code Listing 11: Publishing our NuGet package to the local server

```
nuget push myextensionmethods.1.0.0.nupkg MyNuGetServer -Source
http://localhost/mynugetserver/nuget
```

The following example displays the response from the command.

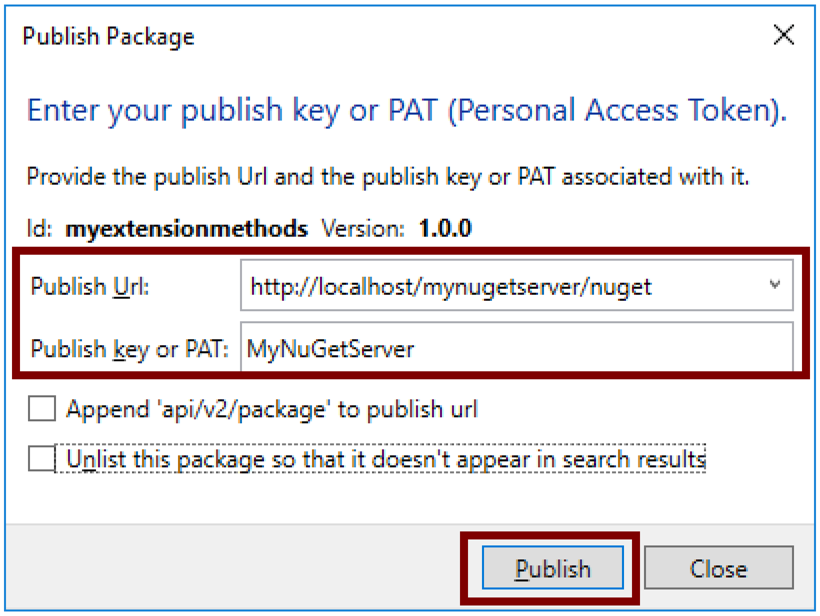
Code Listing 12: Response from NuGet push command

```
nuget push myextensionmethods.1.0.0.nupkg MyNuGetServer -Source
http://localhost/mynugetserver/nuget
Pushing myextensionmethods.1.0.0.nupkg to 'http:
//localhost/mynugetserver/nuget'...
  PUT http://localhost/mynugetserver/nuget/
  Created http://localhost/mynugetserver/nuget/ 811ms
Your package was pushed.
```

Note: We need to make sure that the *Packages* ***folder in our local NuGet server has write permissions for IIS_IUSRS.***

Using NuGet Package Explorer

We can also use NuGet Package Explorer to publish our package. First, we need to launch NuGet Package Explorer and open our NuGet package in the same way we opened our .nuspec file previously. Click **File** > **Publish** in the menu bar to display the **Publish Package** dialog box.


Publish Package

Enter your publish key or PAT (Personal Access Token).

Provide the publish Url and the publish key or PAT associated with it.

Id: **myextensionmethods** Version: **1.0.0**

Publish Url: http://localhost/mynugetserver/nuget

Publish key or PAT: MyNuGetServer

☐ Append 'api/v2/package' to publish url

☐ Unlist this package so that it doesn't appear in search results

Figure 61: NuGet Package Explorer Publish Package dialog box

In Figure 61, we can see the location for the NuGet host URL and the server API key. In addition, we can see a couple of check boxes located above the Publish button. These check boxes should be cleared for the purposes of a local NuGet server. At the end, we should click **Publish** to publish our package.

Note: Since the package was previously published with NuGet Command Line Interface, the process with NuGet Package Explorer will fail.

Chapter summary

This chapter explained how to create our NuGet package and publish it to our local NuGet server. First, we modified the **myextensionmethods** project, and added two folders to its structure: **packagesource** and **packagedeploy**. Next, we explained the concept of a .nuspec file: a text file in XML format that contains the metadata required for creating NuGet packages. This file bases its syntax and structure on the nuspec.xsd schema file available on GitHub. We created the .nuspec file using the NuGet Command Line Interface, and for the purposes of this book, we saved the file to **Documents\Visual Studio 2017\Projects\myextensionmethods\myextensionmethods\bin\Debug**.

After that, we used the Notepad application to review and modify the file before creating our package. We changed the string constants of the file for replacement tokens. A replacement token is a special identifier that allows us to send variable values to the .nuspec file when using the NuGet Command Line Interface for creating a NuGet package. In addition, we added support for multiple target .NET Framework versions using the **`target`** attribute of the **`<file>`** node, and following the NuGet naming conventions for relative paths, along with the use of target framework monikers.

A target framework moniker (TFM) is a standardized token format for specifying the target framework of a .NET application or library. For the purposes of this book, we supported .NET Framework 4.7 and .NET Framework 4.6. We also used the NuGet Package Explorer to view a modified version of our .nuspec file with no replacement tokens, because NuGet Package Explorer does not support these tokens. Then, we created the NuGet package using the **`pack`** command of the NuGet Command Line Interface. Once we created the package, we published it to our local server using the **`push`** command in the command line. In addition, we discussed how to create and publish packages using the NuGet Package Explorer.

Chapter 5 Using Packages Published in a Local NuGet Server

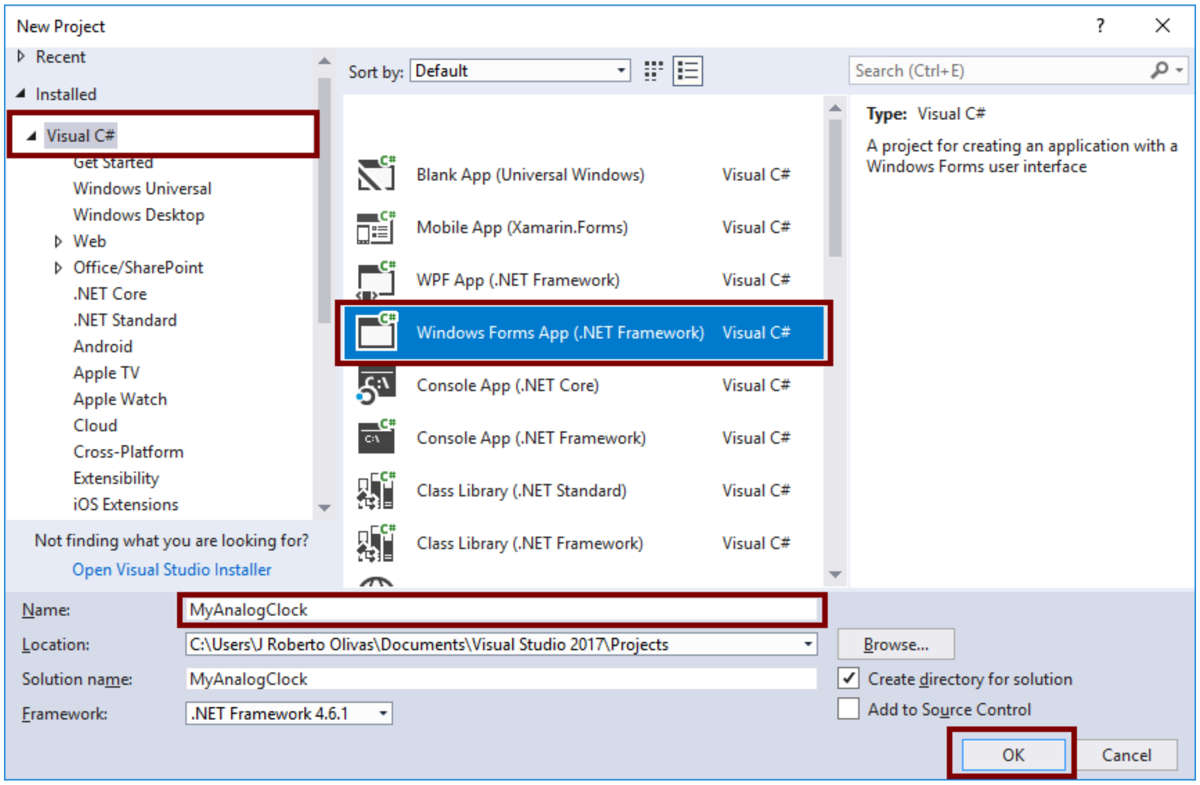
Defining and creating a project for using published packages

Defining the project

Now it is time to use the published package from our private NuGet server. For the purposes of this book, we are going to create a Windows Forms application called **MyAnalogClock**. The application name explains its purpose: displaying an analog clock on the screen.

Creating the project

Click **File** > **New** > **Project** in Visual Studio's menu bar, and then click the **Visual C#** templates category. Next, click the **Windows Forms App (.NET Framework)** template and type **MyAnalogClock** in the **Name** text box. Finally, click **OK** to create the project.


New Project
?
Recent
Installed
Visual C#
Get Started
Windows Universal
Windows Desktop
Web
Office/SharePoint
.NET Core
.NET Standard
Android
Apple TV
Apple Watch
Cloud
Cross-Platform
Extensibility
iOS Extensions
Not finding what you are looking for?
Open Visual Studio Installer
Sort by: Default
Search (Ctrl+E)
Blank App (Universal Windows) Visual C#
Mobile App (Xamarin.Forms) Visual C#
WPF App (.NET Framework) Visual C#
Windows Forms App (.NET Framework) Visual C#
Console App (.NET Core) Visual C#
Console App (.NET Framework) Visual C#
Class Library (.NET Standard) Visual C#
Class Library (.NET Framework) Visual C#
Browse...
OK
Cancel

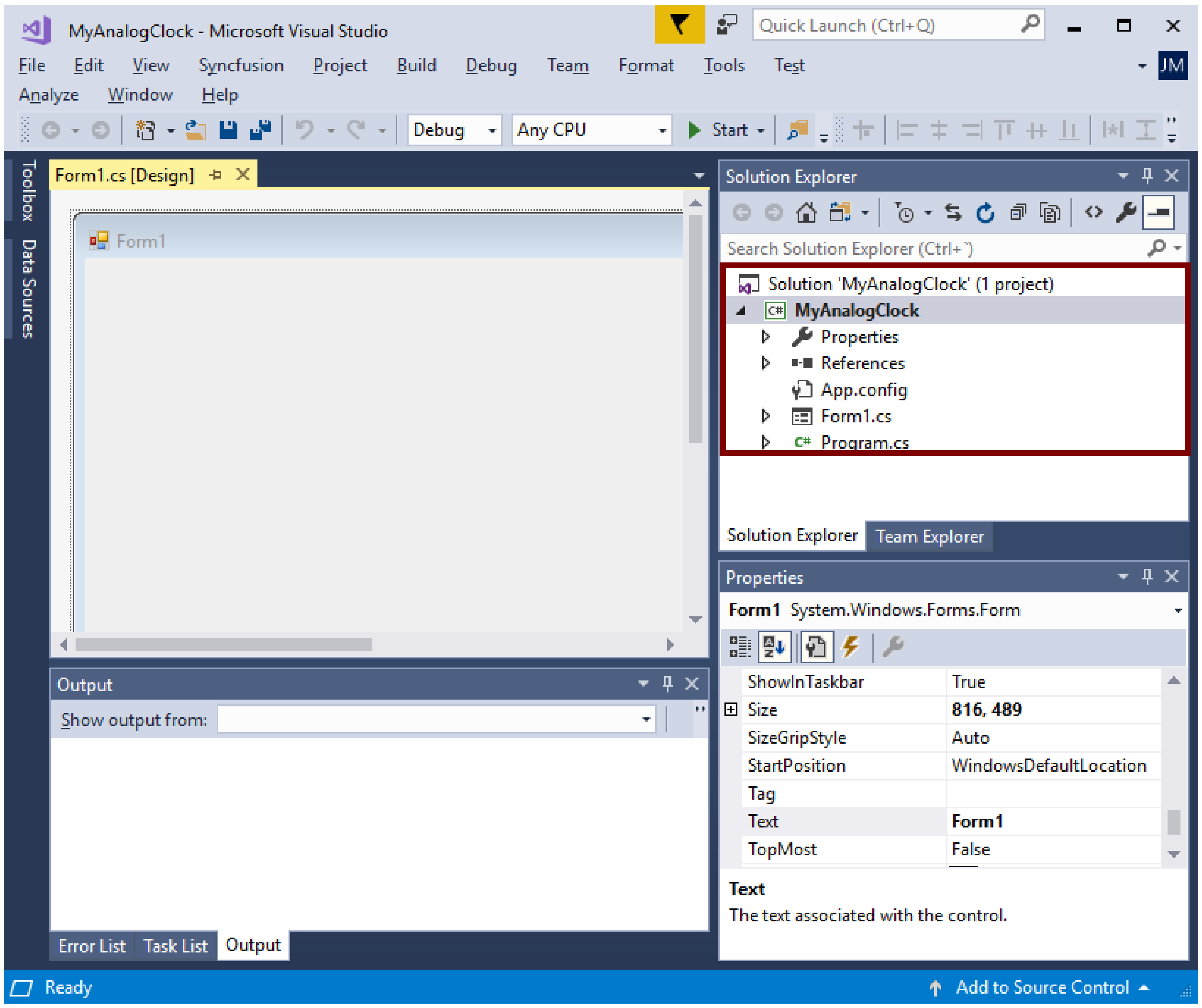
Type: Visual C#

A project for creating an application with a Windows Forms user interface

Name: MyAnalogClock
Location: C:\Users\J Roberto Olivas\Documents\Visual Studio 2017\Projects
Solution name: MyAnalogClock
Framework: .NET Framework 4.6.1
✓ Create directory for solution
Add to Source Control

Figure 62: Creating the MyAnalogClock project

After creating the project, our Visual Studio interface looks like the following image.



Solution 'MyAnalogClock' (1 project)
- MyAnalogClock
  - Properties
  - References
  - App.config
  - Form1.cs
  - Program.cs

Form1 System.Windows.Forms.Form

ShowInTaskbar	True
Size	**816, 489**
SizeGripStyle	Auto
StartPosition	WindowsDefaultLocation
Tag	
Text	**Form1**
TopMost	False

Text
The text associated with the control.

Figure 63: Solution Explorer for the MyAnalogClock project

For the purposes of this book, we are going to rename the **Form1.cs** and **Program.cs** files to **MyAnalogClockForm.cs** and **MyAnalogClock.cs**, respectively.

Specifying package sources in Visual Studio

The first step to using our NuGet package is telling Visual Studio where the sources for downloading and installing packages are. To accomplish that, click **Tools** > **NuGet Package Manager** > **Package Manager Settings** in the Visual Studio menu bar.

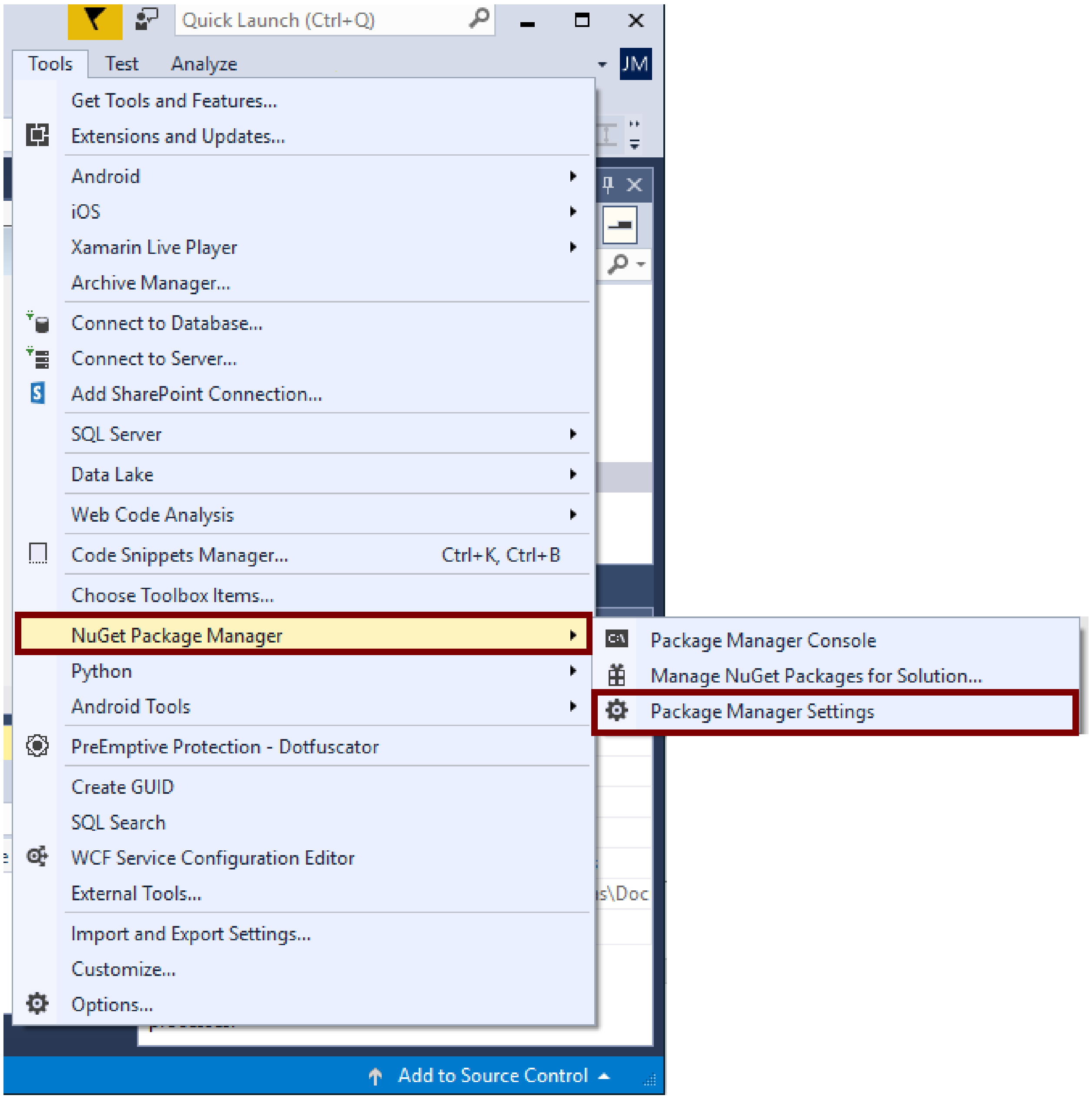
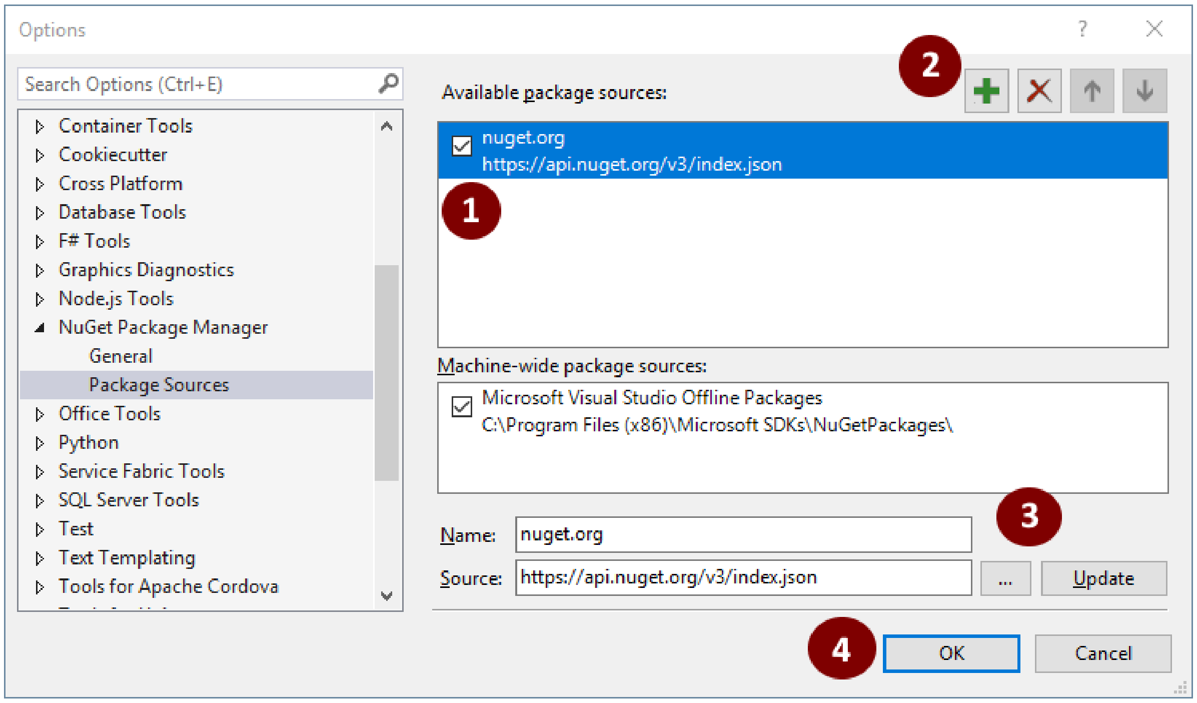


Figure 64: NuGet Package Manager Settings

When the **Options** dialog box appears on the screen, we should click the **Package Sources** section located at the left of the dialog box. Now, we can see the package sources section for the NuGet Package Manager, as displayed in the following figure.



Available package sources:

nuget.org
https://api.nuget.org/v3/index.json

Machine-wide package sources:

Microsoft Visual Studio Offline Packages
C:\Program Files (x86)\Microsoft SDKs\NuGetPackages\

Name: nuget.org

Source: https://api.nuget.org/v3/index.json

Figure 65: NuGet package sources

The following table summarizes the usage of each one of the numbered elements that appear in Figure 65.

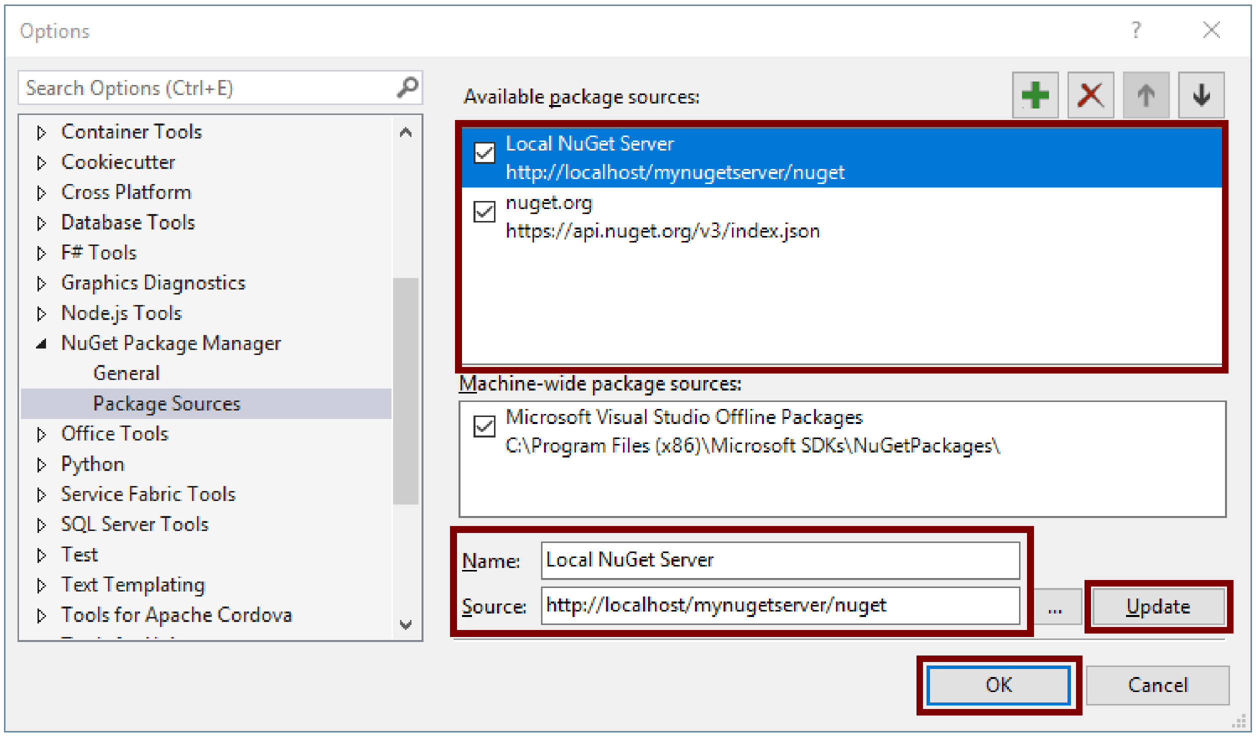
Table 6: NuGet Package Sources interface elements

Element	Purpose
1	The list box displays all available package sources for Visual Studio.
2	The command buttons allow us to add, delete, and reorder the package sources. Reordering package sources tells Visual Studio the order in which sources should be queried when we search for a package.
3	The package source settings include a descriptive name for the source and the URL for the source's location.
4	The OK and Cancel buttons allow us to accept or reject the changes we've made.

Click the plus-sign icon to add our local NuGet server as a package source, as explained in Table 6. The second source automatically appears in the list box. Now, we'll set the following values for the source's settings:

- Name: Local NuGet server
- Source: http://localhost/mynugetserver/nuget

Click the **Update** button located to the right of the source's settings and click the up-arrow button located at the top-right corner. The dialog box should look like the following figure.



Local NuGet Server
http://localhost/mynugetserver/nuget

nuget.org
https://api.nuget.org/v3/index.json

Microsoft Visual Studio Offline Packages
C:\Program Files (x86)\Microsoft SDKs\NuGetPackages\

Name: Local NuGet Server
Source: http://localhost/mynugetserver/nuget

Figure 66: The Options dialog box after setting the local package source

Finally, click **OK** to accept changes and make our local NuGet server available for Visual Studio.

Managing packages from the NuGet local server

Since our project depends on the package published in our local NuGet server, we need to add this package before starting to write any lines of code. We are going to use the NuGet Package Manager for installing our package. First, we should right-click the **MyAnalogClock** node in the **Solution Explorer**, and then click **Manage NuGet Packages** in the context menu.

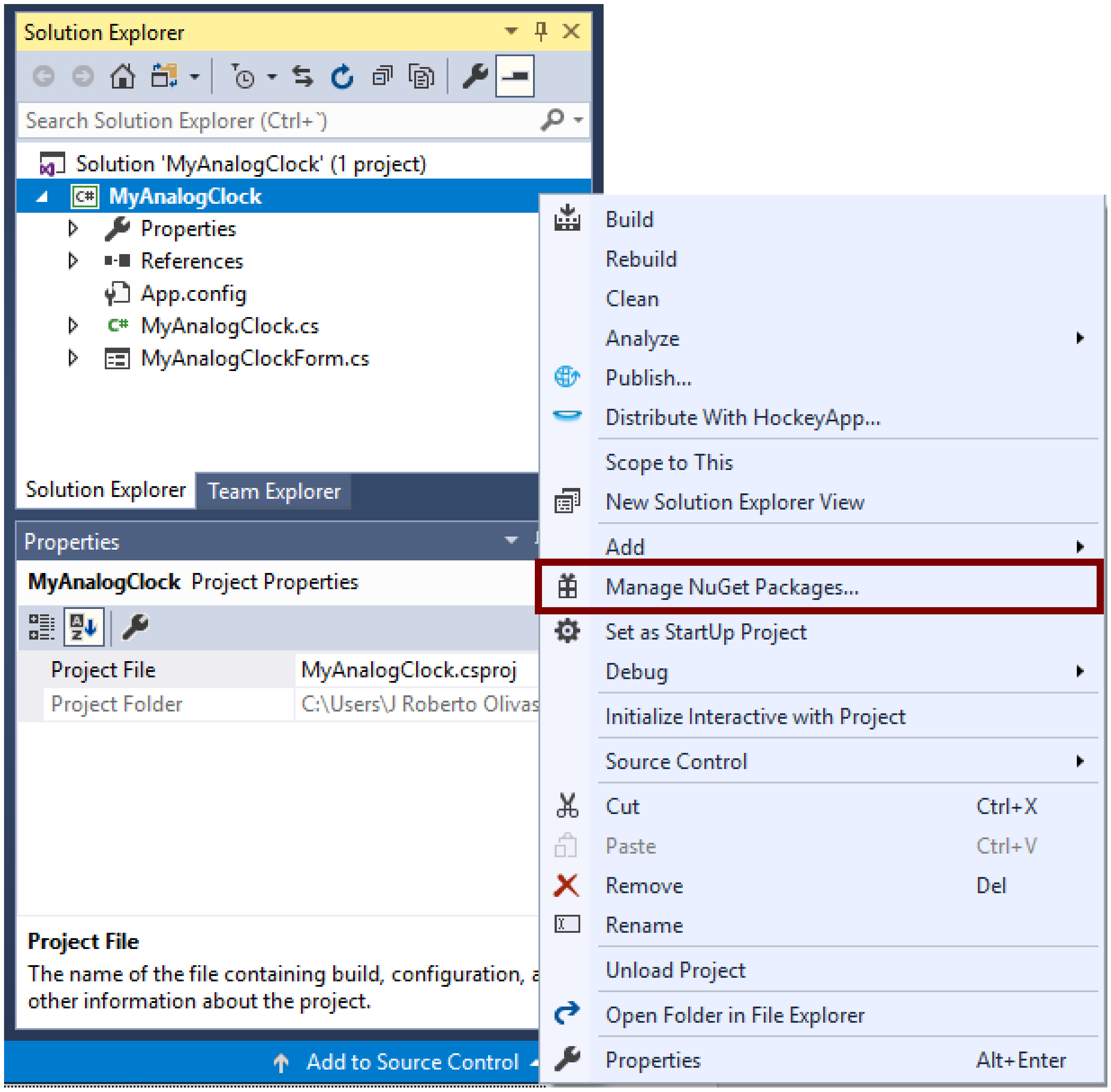
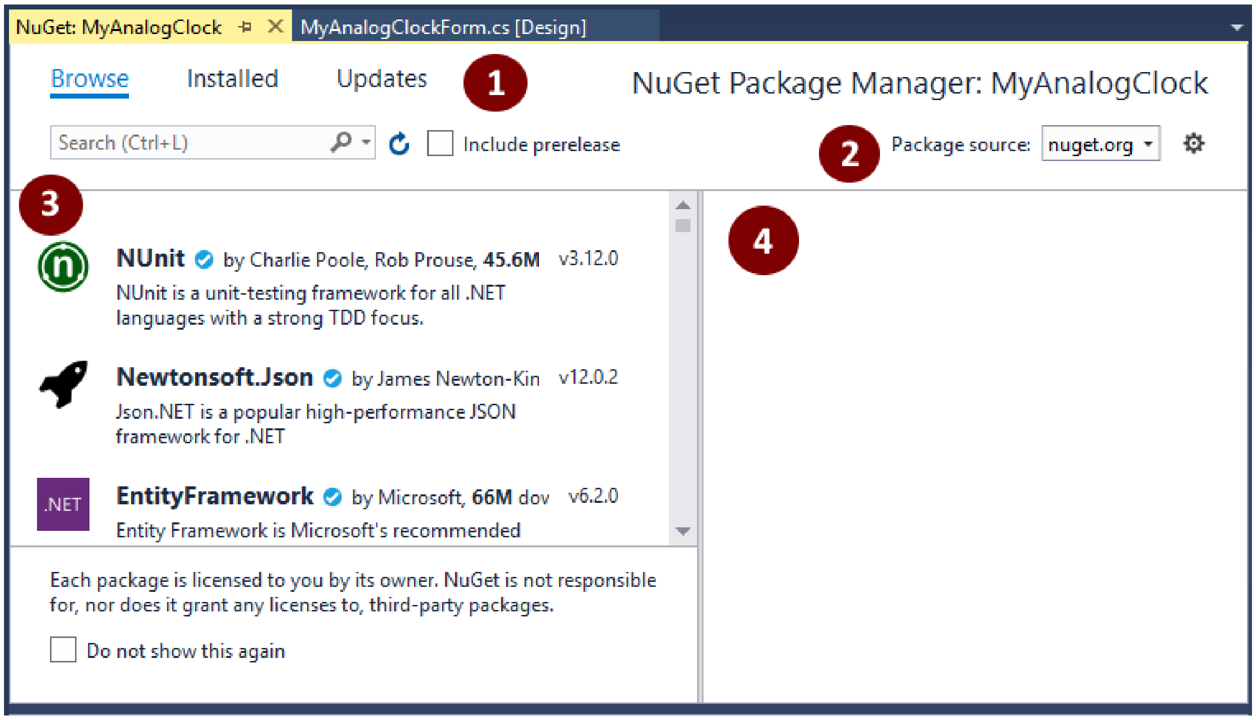


Figure 67: Managing NuGet packages for our project

Now, the NuGet Package Manager appears in a tab inside our project. We can see its interface in the following image.


NuGet: MyAnalogClock   MyAnalogClockForm.cs [Design]

Browse   Installed   Updates

NuGet Package Manager: MyAnalogClock

Search (Ctrl+L)   Include prerelease   Package source: nuget.org

NUnit by Charlie Poole, Rob Prouse, 45.6M v3.12.0
NUnit is a unit-testing framework for all .NET languages with a strong TDD focus.

Newtonsoft.Json by James Newton-Kin v12.0.2
Json.NET is a popular high-performance JSON framework for .NET

EntityFramework by Microsoft, 66M dov v6.2.0
Entity Framework is Microsoft's recommended

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.

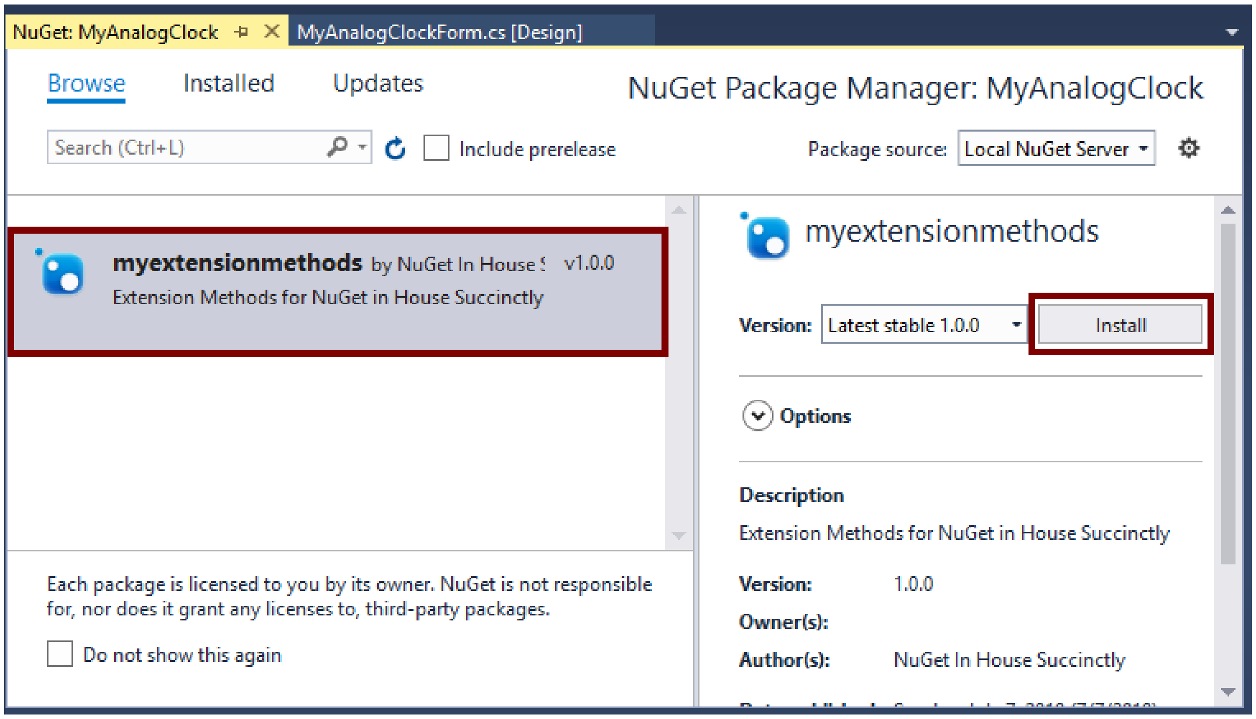
Do not show this again

Figure 68: NuGet Package Manager inside our project

The following list describes the purpose of each highlighted item in Figure 68:

- **Management categories (1):** We have three categories available: **Browse**, which allows us to explore the current package source; **Installed**, which displays the installed packages in our project; and **Updates**, which shows us if there are updates for any of the installed packages in our project.
- **Current package source (2):** Displays the current source for installing packages. We can select a different source by clicking the drop-down button in the combo box.
- **List of available packages (3):** Displays the available packages for the current package source, and the search expression entered in the search bar at the top of the list.
- **Selected package info (4):** Displays information about the selected package from the list. In addition, it allows us to install the package.

We can also see in Figure 68 that the current package source is nuget.org. For the purposes of our project, we need to choose our local NuGet server as a source. Click the **Package source** drop-down and then select **Local NuGet Server**. Next, click the **myextensionmethods** package in the list. The interface should look like the following image.



NuGet: MyAnalogClock | MyAnalogClockForm.cs [Design]

NuGet Package Manager: MyAnalogClock

Browse | Installed | Updates

Search (Ctrl+L) | Include prerelease | Package source: Local NuGet Server

myextensionmethods by NuGet In House S v1.0.0
Extension Methods for NuGet in House Succinctly

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.

Do not show this again

myextensionmethods

Version: Latest stable 1.0.0 | Install

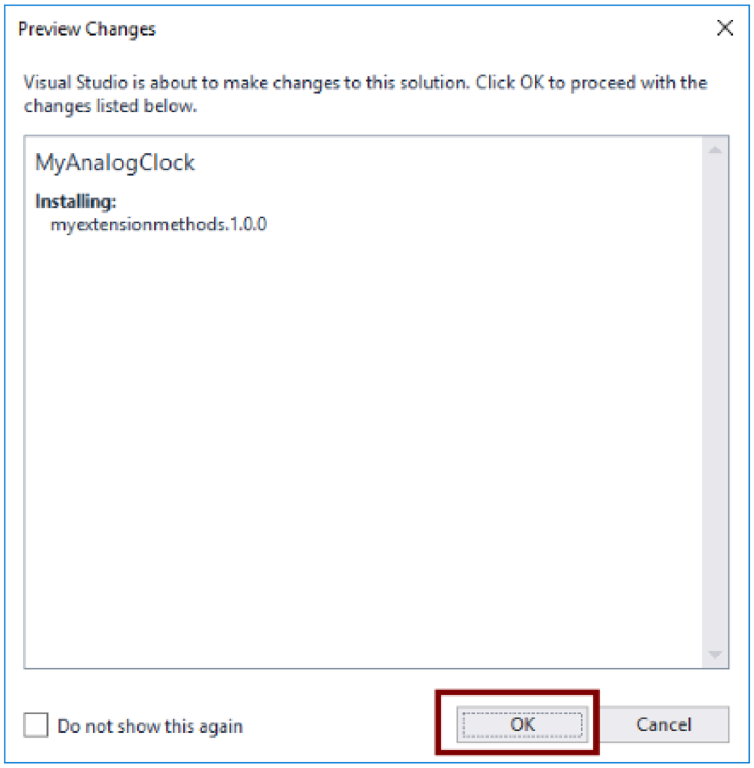
Options

Description
Extension Methods for NuGet in House Succinctly

Version: 1.0.0
Owner(s):
Author(s): NuGet In House Succinctly

Figure 69: Selecting the myextensionmethods package from our local NuGet server

Click **Install** to install the package. After that, the following dialog box is displayed.



Preview Changes

Visual Studio is about to make changes to this solution. Click OK to proceed with the changes listed below.

MyAnalogClock
Installing:
myextensionmethods.1.0.0

Do not show this again | OK | Cancel

Figure 70: Preview Changes dialog box

Click **OK** to finish the installation, and then close the NuGet Package Manager tab to continue our work.

Putting it all together: building our project

Because Visual Studio added a reference to our package, we are ready to start building our project. The first step is writing the code for the analog clock.

The code for our project

The following example displays the code for making our analog clock function properly. The comments added explain the code’s functionality.

Code Listing 13: MyAnalogClock's code

```
using myextensionmethods;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Drawing.Text;
using System.Windows.Forms;

namespace MyAnalogClock
{
    /// <summary>
    /// Displays an analog clock on the screen
    /// </summary>
    public partial class MyAnalogClockForm : Form
    {
        /// <summary>
        /// Instance of the context menu which allows us
        /// to exit the application.
        /// </summary>
        private ContextMenuStrip _contextMenu;

        /// <summary>
        /// Dictionary with the form's context menu items.
        /// The key is assigned to the Tag property of the ToolStripMenuItem
        /// </summary>
        private readonly Dictionary<string, string> _contextMenuItems = new
Dictionary<string, string>
        {
            {"EXIT","Exit"}
        };

        /// <summary>
        /// Instance of the timer in charge of updating the clock.
        /// </summary>
        private Timer _clockTimer;

        /// <summary>
        /// Creates the instance of the form.
        /// </summary>
        public MyAnalogClockForm()
        {
            InitializeComponent();

            //Subscribes the Paint event of the form
            Paint += MyAnalogClockForm_Paint;

            //Subscribes the Load event of the form
            Load += MyAnalogClockForm_Load;

            //Subscribes the MouseDown event of the form
            //in order to allow dragging it
            MouseDown += MyAnalogClockForm_MouseDown;

        }

        /// <summary>
        /// Handles the Load event of the form.
        /// </summary>
        /// <param name="sender">Instance of the form that raises the
event.</param>
        /// <param name="e">Instance of <see cref="EventArgs"/> which holds
the data for managing the event.</param>
        private void MyAnalogClockForm_Load(object sender, EventArgs e)
        {
            //Removes title bar and border
            //from the form.
            Text = string.Empty;
            FormBorderStyle = FormBorderStyle.None;
            ControlBox = false;

            //Creates the instance of the context menu
            _contextMenu = new ContextMenuStrip();

            //Builds the context menu using the items dictionary.
            //Subscribes the Click event for every item to the MenuItem_Click
method.
            _contextMenu.BuildContextMenu(_contextMenuItems, MenuItem_Click);

            // Attach the context menu to the form.
            ContextMenuStrip = _contextMenu;

            // Set the size to a square of 300x300.
            SetSize(300, 300);
            //Use double buffering to avoid flickering.
            DoubleBuffered = true;

            //The form will not be shown in the task bar.
            ShowInTaskbar = false;

            // Set focus to the form. You should do this
            // in order to make Alt+F4 close the clock properly.
            Focus();

            //Creates an instance of a timer
            //and adjusts the time for firing the
            //Tick event (1000 milliseconds == 1 second)
            _clockTimer = new Timer
            {
                Interval = 1000
            };

            //Subscribes the Tick event for the timer instance
            _clockTimer.Tick += _clockTimer_Tick;
            _clockTimer.Enabled = true;
        }

        /// <summary>
        /// Handles the MouseDown event of the form in order to
        /// allow dragging from anywhere on the clock's surface.
        /// </summary>
        /// <param name="sender">Instance of the form that raises the
event.</param>
        /// <param name="e">Instance of <see cref="EventArgs"/> which holds
the data for managing the event.</param>
        private void MyAnalogClockForm_MouseDown(object sender,
MouseEventArgs e)
        {
            //If we hold the mouse left button, we can drag the clock.
            if (e.Button == MouseButtons.Left)
            {
                // Release the mouse capture started by the mouse down.
                Capture = false;

                //Create the windows message
                var wMessage = this.CreateWmNclButtonDown();

                //Sends the message to the form's instance
                DefWndProc(ref wMessage);
            }
        }

        // Redraws the clock to show the current hand position.
        private void _clockTimer_Tick(object sender, EventArgs e) =>
Refresh();

        /// <summary>
        /// Handled the Paint event of the form and draws the clock.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void MyAnalogClockForm_Paint(object sender, PaintEventArgs e)
        {
            //Clear the drawing surface of the form
            //and fills it with the form's BackColor
            e.Graphics.Clear(BackColor);

            //Sets the quality for graphics rendering
            e.Graphics.SmoothingMode = SmoothingMode.AntiAlias;

            //Sets the quality for text rendering
            e.Graphics.TextRenderingHint =
TextRenderingHint.AntiAliasGridFit;

            //Translates the graphics coordinates to
            //center the drawing.
            e.Graphics.TranslateTransform(ClientSize.Width / 2,
ClientSize.Height / 2);

            // Draws the clock face including tick marks.
            e.Graphics.DrawClockFace(ClientSize, Color.Blue, 4);

            // Draws the center of the clock.
            e.Graphics.FillEllipse(Brushes.Blue, -5, -5, 10, 10);

            // Draws the clock hands.
            e.Graphics.DrawClockHands(ClientSize, Color.Red,
Color.OrangeRed);
        }

        /// <summary>
        /// Handles the click event for every item in the context menu.
        /// Since there is only one item (Exit), executes the Close method
        /// to close the form and exit the application.
        /// </summary>
        /// <param name="sender">Instance of the menu item which raises the
event.</param>
        /// <param name="e">Instance of <see cref="EventArgs"/> which holds
the data for managing the event.</param>
        private void MenuItem_Click(object sender, EventArgs e) => Close();

        /// <summary>
        /// Set the form's size.
        /// </summary>
        /// <param name="clientWidth">Form's width in pixels.</param>
        /// <param name="clientHeight">Form's height in pixels.</param>
        private void SetSize(int clientWidth, int clientHeight)
        {
            //Set the form's size.
            ClientSize = new Size(clientWidth, clientHeight);

            // Set the form's region to a circle.
            GraphicsPath path = new GraphicsPath();
            path.AddEllipse(ClientRectangle);
            Region = new Region(path);

            // Redraw the form.
            Refresh();
        }
    }

}
```

Building and testing our project

To build and test our project, click **Start** in the Visual Studio toolbar.

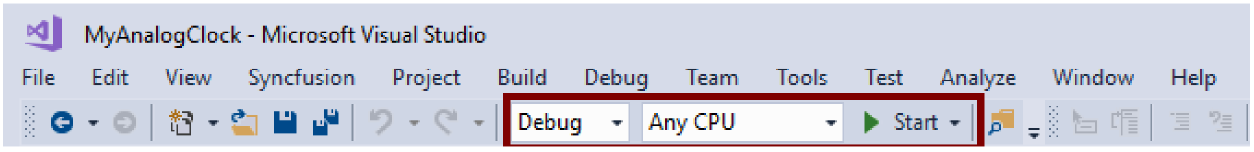

MyAnalogClock - Microsoft Visual Studio
File Edit View Syncfusion Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
Start

Figure 71: Building and testing our project with the Start button

If everything is OK, we will see our analog clock on the screen.

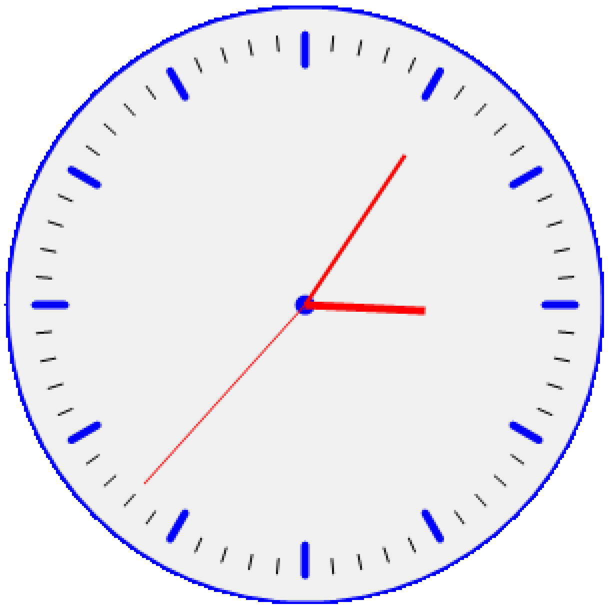


Figure 72: The analog clock

Publishing an update for our NuGet package

One benefit of NuGet is the ability to easily update our reusable code. We can improve our assemblies, publish our package's upgrades, and let NuGet deal with communicating upgrades to the consumer of that package. In this case, we are going to create a new version of the myextensionmethods package and publish this upgrade into our local NuGet server. First, let us see this part of the consumer's code for understanding the reason of the upgrade.

Code Listing 14: Code for drawing the clock

```
            // Draws the clock face including tick marks.
            e.Graphics.DrawClockFace(ClientSize, Color.Blue, 4);

            // Draws the center of the clock.
            e.Graphics.FillEllipse(Brushes.Blue, -5, -5, 10, 10);

            // Draws the clock hands.
            e.Graphics.DrawClockHands(ClientSize, Color.Red,
Color.OrangeRed);
```

We can see in Code Listing 14 that a separate line of code draws the center of the clock, instead of doing it using the **`DrawClockFace`** method. This works, but a better way to perform this task might be to draw the center in the **`DrawClockFace`** method. This will isolate the entire clock's face drawing from the rest of the code.

The following example displays the modified code for the **`DrawClockFace`** method.

Code Listing 15: Upgraded code for DrawClockFace method

```
        /// <summary>
        /// Draws an analog clock face in a client area (usually a form's
client area).
        /// </summary>
        /// <param name="graphicsInstance">Graphics object associated to the
client area.</param>
        /// <param name="clientSize">Width and Height of the client
area.</param>
        /// <param name="penColor">Color used to draw the analog clock
face.</param>
        /// <param name="penWidth">Width of the pen used to draw the analog
clock face.</param>
        public static void DrawClockFace(this Graphics graphicsInstance, Size
clientSize, Color penColor, int penWidth)
        {
            using (var clockPen = new Pen(penColor, penWidth))
            {
                //Draws the clock face's outline
                graphicsInstance.DrawCircle(clientSize, penColor, penWidth);

                //Draws the tick marks around the clock's face outline

                //Defines a round cap for beginning and ending of every line
                clockPen.StartCap = LineCap.Round;
                clockPen.EndCap = LineCap.Round;

                //Defines the scale factors used to draw
                //the tick marks around the clock's face

                //Scale factors for the mark's outer point (x,y)
                var outXFactor = 0.45f * clientSize.Width;
                var outYFactor = 0.45f * clientSize.Height;

                //Scale factors for the mark's inner point (x,y)
                var innXFactor = 0.425f * clientSize.Width;
                var innYFactor = 0.425f * clientSize.Height;

                //Scale factors for the hour mark's inner point (x,y)
                var hourXFactor = 0.4f * clientSize.Width;
                var hourYFactor = 0.4f * clientSize.Height;

                //Loop for iterating the hour's minutes
                for (int minute = 1; minute <= 60; minute++)
                {
                    //Calculates the angle for every minute
                    var minuteAngle = Math.PI * minute / 30.0;
                    var cosineAngle = (float)Math.Cos(minuteAngle);
                    var sineAngle = (float)Math.Sin(minuteAngle);
                    //Calculates inner and outer points
                    //for the current tick mark
                    var innerPoint = (minute % 5 == 0) ? new
PointF(hourXFactor * cosineAngle,hourYFactor * sineAngle) : new
PointF(innXFactor * cosineAngle, innYFactor * sineAngle);
                    var outerPoint = new PointF(outXFactor *
cosineAngle,outYFactor * sineAngle);

                    graphicsInstance.DrawLine((minute % 5 == 0) ? clockPen :
Pens.Black,innerPoint,outerPoint);
                }

                // Draws the center of the clock.
                // Now, we need to upgrade our package to version 1.1.0.0
                using (var centerBrush = new SolidBrush(penColor))
graphicsInstance.FillEllipse(centerBrush, -5, -5, 10, 10);

            }
        }
```

After building our project, we should go to the **packagesource** folder in the **myextensionmethods** project, and then create the new version of our package using the NuGet Command Line Interface, as displayed in the following example.

Code Listing 16: Creating the package upgrade for myextensionmethods

```
nuget pack myextensionmethods.nuspec -OutputDirectory ..\packagedeploy -
Properties Configuration=Debug;Version=1.1.0.0;Description="Extension
Methods for NuGet In-House Succinctly";Author="NuGet In-House Succinctly"
```

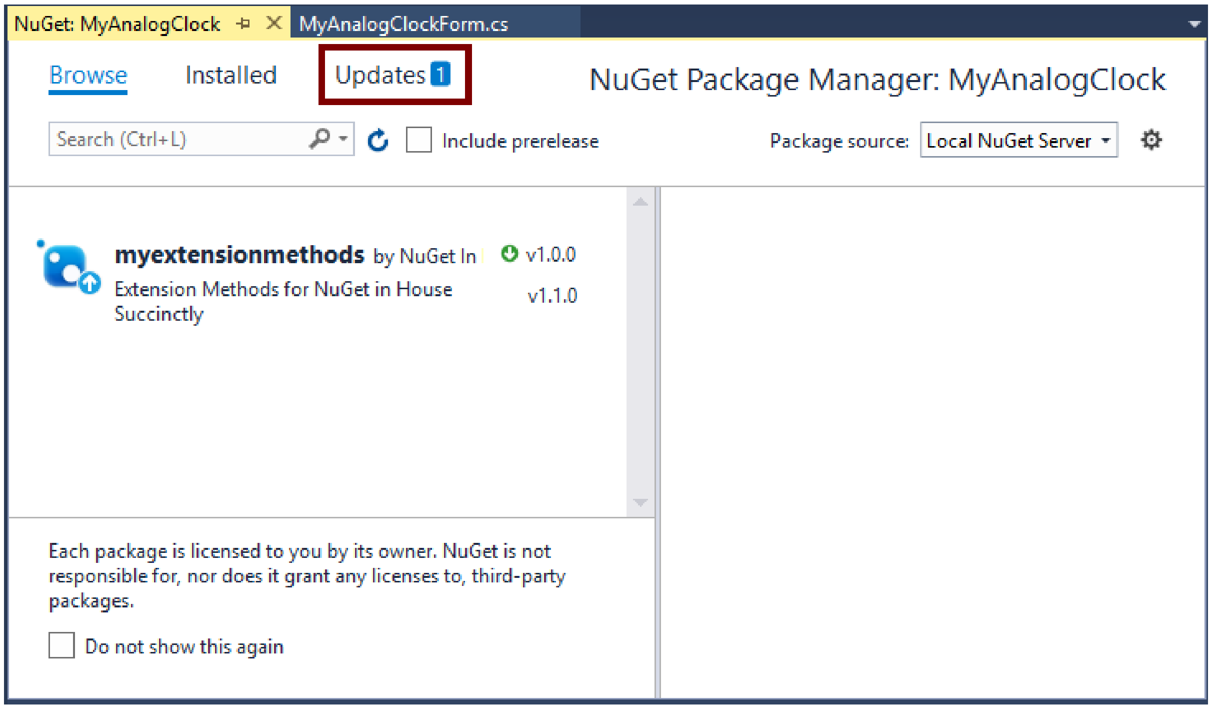
Finally, after we create the package, we should go to the packagedeploy folder in the **myextensionmethods** project, and then publish the upgrade of our package using the NuGet Command Line Interface, as shown in the following example.

Code Listing 17: Publishing the upgrade of myextensionmethods

```
nuget push myextensionmethods.1.1.0.nupkg MyNuGetServer -Source
http://localhost/mynugetserver/nuget
```

Applying the package update to the consumer project

To use the new version of **myextensionmethds**, we need to use the NuGet Package Manager as we did when we added the package for the first time. However, unlike that time, the NuGet Package Manager interface should look like the following figure.

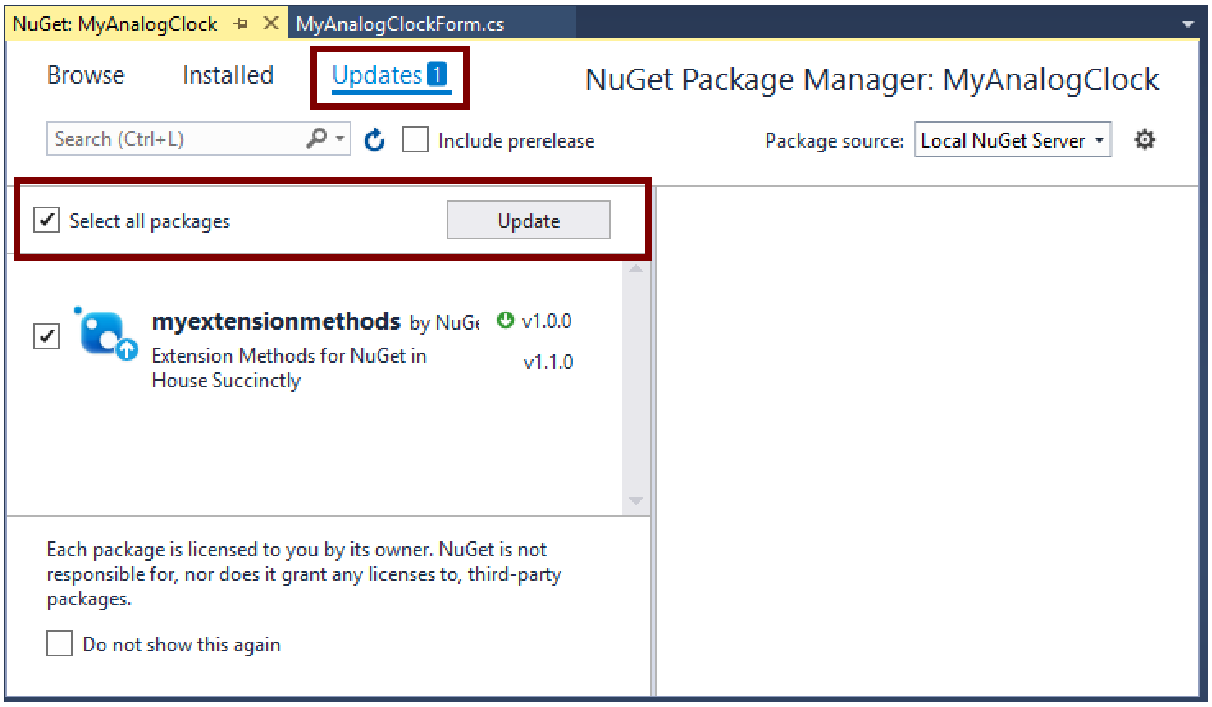


Extension Methods for NuGet in House Succinctly

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.

Figure 73: NuGet Package Manager displaying the update warning

We can see the number displayed to the right of the **Updates** section. This number tells us how many of the packages installed have upgrades. In this case, there is only one package with an available upgrade. Click **Updates** and click the **Select all packages** check box. Finally, click **Update** to install the new version of the package. After that, the **Preview Changes** dialog box will appear, as it did in the installation process. Click **OK** to finish the update.



Extension Methods for NuGet in House Succinctly

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.

Figure 74: Installing the available updates

Once NuGet Package Manager finishes the update, we should change the code for the **`MyAnalogClockForm_Paint`** event handler to reflect the modifications made to myextensionmethods. The following example displays these changes.

Code Listing 18: Code modified to reflect the upgrade

```
        /// <summary>
        /// Handles the Paint event of the form, and draws the clock.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void MyAnalogClockForm_Paint(object sender, PaintEventArgs
e)
        {
            //Clears the drawing surface of the form
            //and fills it with the form's BackColor
            e.Graphics.Clear(BackColor);

            //Sets the quality for graphics rendering
            e.Graphics.SmoothingMode = SmoothingMode.AntiAlias;

            //Sets the quality for text rendering
            e.Graphics.TextRenderingHint =
TextRenderingHint.AntiAliasGridFit;

            //Translates the graphics coordinates to
            //center the drawing.
            e.Graphics.TranslateTransform(ClientSize.Width / 2,
ClientSize.Height / 2);

            // Draws the clock face including tick marks.
            e.Graphics.DrawClockFace(ClientSize, Color.Blue, 4);

            // Draws the center of the clock.
            // This code was implemented in myextensionmethods

            // Draws the clock hands.
            e.Graphics.DrawClockHands(ClientSize, Color.Red,
Color.OrangeRed);
        }
```

Now, click **Start** in the Visual Studio toolbar to build and test our project.

Chapter summary

This chapter explained how to use packages published in our local NuGet server. For the purposes of this chapter, we created a Windows Forms application called **MyAnalogClock** to display an analog clock on the screen. Then we told Visual Studio the location of the sources for downloading and installing packages. To accomplish that, we clicked **Tools** > **NuGet Package Manager** > **Package Manager Settings** in the Visual Studio menu bar. This action displayed the **Options** dialog box. We clicked the **Package Sources** section located at the left of the dialog box, and then the plus-sign icon to add a package source. After that, we entered **Local NuGet Server** for the name of the source, and **http://localhost/mynugetserver/nuget** for the URL's package source. Then, we used the **Manage NuGet Packages** item from the project's context menu to install the **myextensionmethods** package from our local NuGet server. After that, we wrote the code for **MyAnalogClock** project and built it. Finally, we made an upgrade for the **myextensionmethods** package and applied it to the **MyAnalogClock** project using the NuGet Package Manager.

Conclusion

NuGet is a great tool that makes the developer’s life easier when working with Visual Studio projects. Possibly its biggest advantage is the automation of dependency management, as we saw throughout this book. With this automation feature, NuGet configures any project by adding references only to the necessary assemblies, and when one of these assemblies is updated by a developer, NuGet automatically updates the project with the latest version.

Unfortunately, if we try to use the NuGet Gallery (which is the repository used by NuGet to publish and download assemblies), all published code will become public. This is not a problem if we work with open-source projects, but we will face a serious problem when we deal with copyrighted code. This was the major difficulty that my development team coped with the first time we tried to use NuGet. It was frustrating at first, because NuGet represented the solution to our dependency management troubles, and this particular issue prevented us from taking advantage of NuGet features. Luckily, we found the way to deal with this problem, but not without tremendous research (and several sleepless nights and cups of coffee).

Therefore, I was encouraged to write this book to explain how to implement a "private" NuGet Gallery, and take advantage of NuGet features to keep all published code private, while helping my readers to avoid all the effort we employed to achieve it.

NuGet has definitely saved my crew many debugging hours and made us more efficient. This has had a direct influence on our delivery times, with more projects finished before due dates, and more satisfied customers.