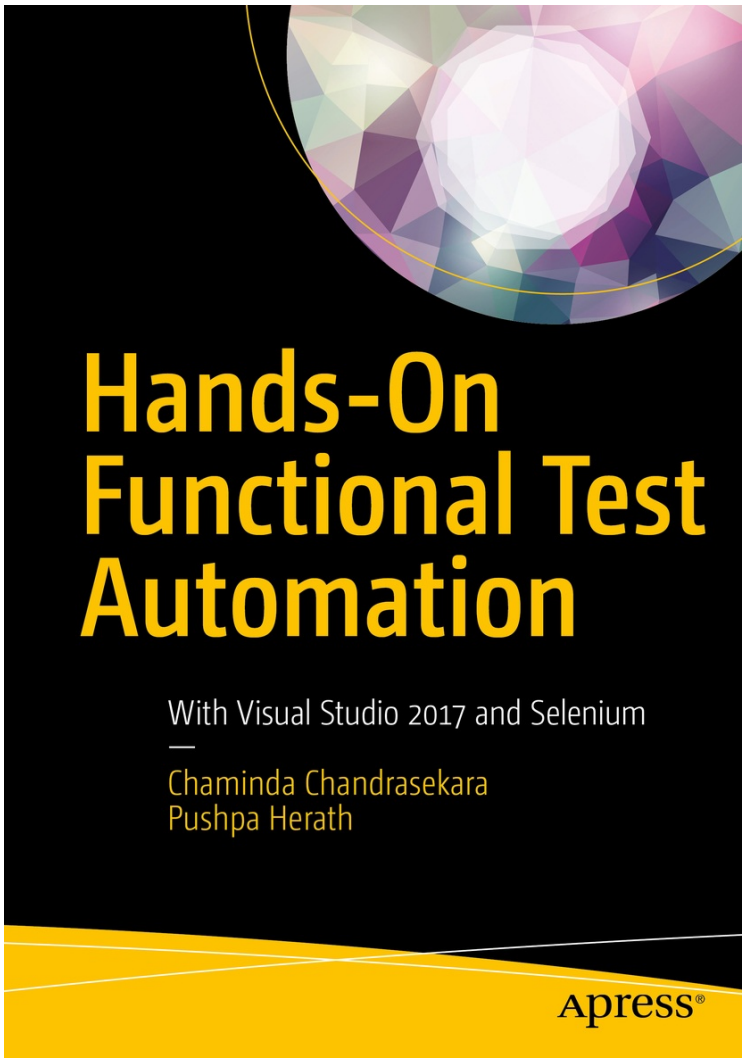
Hands-On
Functional Test
Automation
With Visual Studio 2017 and Selenium
Chaminda Chandrasekara
Pushpa Herath
Apress®

Chaminda Chandrasekara and Pushpa Herath

Hands-On Functional Test Automation
With Visual Studio 2017 and Selenium

Apress®

Chaminda Chandrasekara
Dedigamuwa, Sri Lanka

Pushpa Herath
Hanguranketha, Sri Lanka

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-4410-4 . For more detailed information, please visit http://www.apress.com/source-code .

ISBN 978-1-4842-4410-4 e-ISBN 978-1-4842-4411-1
https://doi.org/10.1007/978-1-4842-4411-1

Standard Apress

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a Delaware corporation.

Let this book be the ultimate guide for test automation engineers to make their team run the extra mile ...

Introduction

Automated testing is crucial in order to deliver quality software in modern agile development approaches by enhancing the continuous delivery. Hands-on lessons in this book will get you started with the tools around Microsoft Visual Studio, while providing good understanding of the concepts of Test Automation.

Software development demands a shorter delivery cadence with the approach of wide adoption of agility in software development. Automation of build and deployments is vital in enabling the quick delivery cycles, as is integrated software test automation into the delivery pipelines to ensure the quality of the software components is not compromised due to shorter cadence.

Test Automation with Visual Studio - Step by Step Guide will get you started with functional testing of both web apps and windows apps using different frameworks. Further, you will deep dive into getting the functional automation testing integrated with deployment pipelines.

Step-by-step lessons will give you understanding about how to do functional test automation using selenium with C# and python. You will also learn about enhancing test automation development with third-party frameworks. You can learn how to configure test clients and run functional tests through Azure DevOps release pipelines to integrate test automation with the deployment pipeline. Performance and load testing lessons will provide you with good understanding on how to do cloud-based load testing.

Each lesson will include an introduction to the related concepts to help you understand how things work. This will broaden your knowledge on implementing the test automation in the correct way. At the end of each lesson, alternative options as well as other enhancement possibilities are discussed to allow you to do further exploration.

Acknowledgments

We are thankful for all the mentors who have encouraged and helped us during our careers and who have provided us with so many opportunities to gain the maturity and the courage we needed to write this book.

We would also like to thank our friends and colleagues who have helped and encouraged us in so many ways.

Last, but in no way least, we owe a huge debt to our families. Not only because they have put up with late-night typing, research, and our permanent air of distraction, but also because they have had the grace to read what we have written. Our heartfelt gratitude is offered to them for helping us make this dream come true.

Table of Contents

About the Authors and About the Technical Reviewer

About the Authors

Chaminda Chandrasekara

is a Microsoft Most Valuable Professional (MVP) for Visual Studio ALM and Scrum Alliance Certified ScrumMaster® and focuses on and believes in continuous improvement of the software development life cycle. He works as a DevOps consultant for Jabil Circuit Sdn. Bhd. Chaminda is an active Microsoft Community Contributor (MCC) who is well recognized for his contributions in Microsoft forums, TechNet galleries, wikis, and Stack Overflow, and he contributes extensions to Azure DevOps Server and Services (former VSTS/TFS) in the Microsoft Visual Studio Marketplace. He also contributes to other open source projects in GitHub. Chaminda has published two books, *Beginning Build and Release Management with VSTS* (www.apress.com/in/book/9781484228104) and *Effective Team Management with VSTS* (https://www.apress.com/in/book/9781484235577). He blogs about technology at https://chamindac.blogspot.com and http://devopsbeyondms.blogspot.com/ .

Pushpa Herath

is a Senior Test Automation Engineer at Datavail Lanka (Pvt) Ltd. She has many years of experience in QA automation and Azure DevOps Server and Services (former VSTS/TFS). She is an expert on functional test automation using Selenium and Coded UI. Pushpa blogs about technology at https://devopsadventure.blogspot.com/ . Pushpa has experience with Microsoft tools (C#, VSTS/TFS, SQL Server, and Azure) and open source tools (MAQS open framework and Sikuli).

About the Technical Reviewer

TR Mittal Mehta

has a total of 15 years of IT experience. Currently, he is working as a configuration manager and is MCP in TFS 2012. He also has experience working in TFS, VSTS, c#, Navision, build-release, DevOps, automation and configuration areas over his last 8 years in Microsoft Technologies.

Chaminda Chandrasekara and Pushpa Herath, *Hands-On Functional Test Automation*
https://doi.org/10.1007/978-1-4842-4411-1_1

1. Understanding the Concepts

Chaminda Chandrasekara[1] and Pushpa Herath[2]

(1) Dedigamuwa, Sri Lanka
(2) Hanguranketha, Sri Lanka

With the wide adoption of Agile methodologies in the software development industry, delivery of software within a shorter time period has become the norm. Automation is a buzzword in today's software world, as it is an essential part of successful Agile implementation. Every possible aspect of software development and delivery needs to be automated as much as possible to meet the demand for getting high-quality, state of the art software solutions to the market on time.

Maintaining quality standards while delivering software in shorter cadence is a challenge for any software vendor, as verifying each and every aspect of the developed software is a time-consuming effort. Validating only recent modifications and putting software components into production could result in disasters if critical functionality is broken. Automating the testing aspect of software as much as possible and handing over the testing job to computers, with a minimal need for human interaction, is the key to success in assuring high-quality software.

In this book, you will be provided with hands-on experience to get you started with automation testing using Microsoft Visual Studio in combination with Selenium and other test automation tools and frameworks. Lessons will start with simple steps, and as you read through the chapters, you will be diving into different aspects of testing, such as load and performance testing, cloud-based load testing, and test data management options. Additionally, the book will take you through simple implementation of test automation executions using containerized test execution clients.

Before moving into the lessons-based hands-on learning, this chapter will give you a broader understanding of the need for software test automation, where and when test automation is applicable, and getting the software test automation integrated with software delivery automation. Further, we will discuss the Return on Investment (ROI) aspect of test automation, which is essential to getting the buy-in from the management of any company to get the required support and sponsorship to implement software test automation.

Examining Different Software Testing Types

Software testing can be divided into two broader categories—namely, Functional Testing and Non-Functional Testing. Each of these categories has different types of testing included in them.

Functional Testing

A system is tested for functional requirements to ensure it is implemented as per the functional specifications.

- ***Unit Testing***: Testing of an individual module or software component that is generally performed by the developer/programmer of such module or component. This type of testing requires deeper understanding of the design and implementation of the software component or module that is being tested.
- ***Integration Testing***: Testing the combined functionality of modules integrated together in software is referred to as integration testing. The modules can be individual applications or code modules that have interdependencies to perform a function.
- ***Smoke Testing***: Smoke tests are used to validate a system after a new build to ensure there are no showstoppers for execution of the system, covering all general functional scenarios of the software system. If smoke testing fails, further testing of the system generally is not carried out until the build is fixed so that the smoke tests can be run without failures.
- ***Sanity Testing***: A new software build or version is validated for a minimal level of successful execution of its functionality to allow further testing to be carried out on the new version. If sanity tests fail, no further testing is executed until the software system version passes all sanity tests.
- ***API Testing***: As a part of integration testing, sometimes application programming interfaces (APIs) are directly tested for input and output.
- ***Regression Testing***: Testing a software system as a whole, covering all modules, is termed as Regression testing. Before delivering a version upgrade to production, software is expected to go through a regression testing cycle to ensure no functionality in any part of the system is broken.
- ***Acceptance Testing***: The software system is tested in production-equivalent environments to verify all the business requirements are satisfied with end-to-end flows of the system. This validation is performed with involvement of the client of the software system, and this testing is called User Acceptance Testing (UAT).
- ***Exploratory Testing***: Exploring the software system functionality without following a specific flow defined in a test case and identifying any issues and potential test cases/flows is referred to as exploratory testing.

Non-Functional Testing

Non-functional testing helps to evaluate a system for its non-functional requirements. The ways a system operates, such as its performance, scalability, and reliability, are tested in non-functional testing.

- ***Performance Testing***: Verification of a system to ensure it meets the performance expectations. This is comprised of stress and load testing. Performance testing determines whether a system can meet performance demands while under stress or

load.

- ***Stress Testing***: Stress testing is performed to check the system's ability to handle data volume and traffic beyond its requirements or expectations so that how and when it fails can be identified.
- ***Load Testing***: Load testing verifies how much load a system can handle without a performance degrade. Unlike stress testing, the maximum load applied in load testing is generally the maximum limit of the specification or slight additional load beyond maximum load specification. The performance of system and the infrastructure of the system are monitored during the load testing to identify the bottlenecks.
- ***Volume Testing***: Verification of the system behavior with a high volume of data is performed in volume testing. The next level of volume testing is increasing the volume to stress the system in order to perform stress testing.
- ***Reliability Testing***: Reliability testing verifies whether the system can perform without any failures for a given period.
- ***Usability Testing***: Usability testing verifies the capability of a new user to easily understand the flow of the system and use it without any difficulty. The availability of proper help or documentation, such as user guides, is also validated.
- ***Security/Vulnerability Testing***: Weaknesses in software, hardware, network, etc. are verified in vulnerability testing to prevent hackers or malicious software from causing issues or controlling the systems. Systems used in military, air traffic control, and space programs, etc. are highly tested for vulnerabilities.
- ***Recovery Testing***: The capability of the system to recover from crashes or disasters is validated in recovery testing. For example, a system should be able to complete its operations without failures to its functional flow, even in a situation of sudden network failure or in a server restart, once the relevant network or server comes back online. Not even one server should be visible or impact the flow of the system.
- ***Compliance Testing***: Validation of whether a system meets the organization- or client-specified standards is known as compliance testing.
- ***Compatibility Testing***: Verification of a system's behavior on different platforms, hardware, networking, browsers, etc. is identified as compatibility testing. This includes previous version support as well, which is known as backward compatibility testing.
- ***Install/Uninstall Testing***: Capability of a system to set up and be removed from different hardware and networks without having any issues is tested under install/uninstall testing.
- ***Localization/Globalization Testing***: A software system's ability to work for specific culture and locale settings is verified in localization testing. Globalization testing checks whether a system is able to work in any culture or locale settings.

Of these various types of testing, this book focuses mainly on implementation of functional test automation with Visual Studio-related tools and frameworks, including Selenium. Some non-functional test areas are also explored in the book, especially focusing on Visual Studio and Azure DevOps Services to support load and performance testing.

The Importance of Testing Software Systems

Testing a software system assures its quality and confirms it meets the requirements or specifications essential to the system. It is worth identifying each aspect of importance of software testing to determine the real need of testing.

- ***Meeting the functional requirement specifications***: Functional testing helps to ensure the system is developed as per the requirement specifications and it is helping to improve the client business process. Acceptance testing, especially where the client/end user is involved with the testing while using production equivalent environments, ensure that the system meets the needs of the business once it is put into production use. Additionally, testing helps to ensure all components and applications in a given system are working together to provide the required functionality.
- ***Support platforms and other compatibilities***: Testing is required to make sure the system is compatible with all platforms, components, browsers, operating systems, etc., so as to ensure all those needs are verified before reaching production environments. It is vital to find any compatibility issues well before reaching production to avoid unexpected situations once the system is put to use in production.
- ***Minimize critical bugs from reaching production***: Extensive testing and performing regression testing on a system before sending it to production minimizes the chance of a critical bug reaching production. Bugs in different software stages have differing costs, and the highest cost occurs when a bug discovered in production (see Figure 1-1). A critical bug in production may even result in closure of project engagement with a client and could potentially lead to legal action against the software system vendors. So, performing functional and non-functional testing is required to ensure no issues are in the system when it is put into use in production.

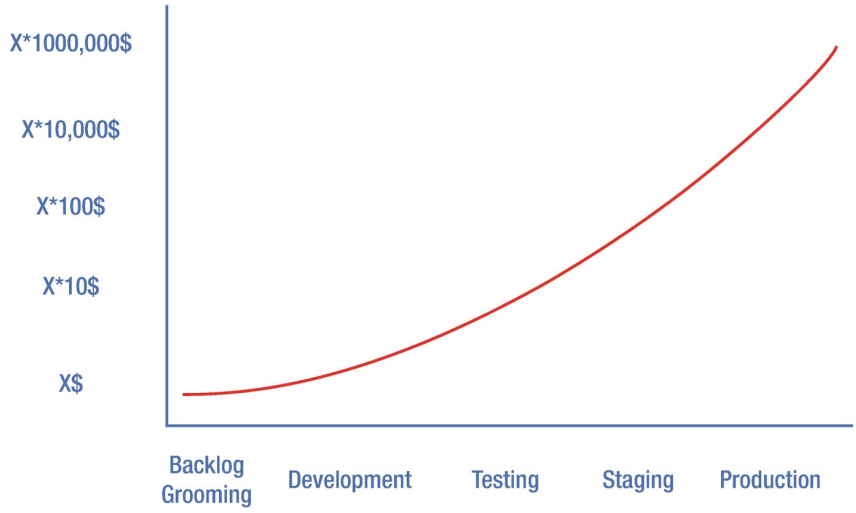

Figure 1-1 Cost of a bug

- ***Ensuring a system is capable of handling production data volumes and traffic***: A system performing non-functional testing for performance, load, stress, etc., as identified in the previous section, helps the system development teams fix the potential issues before shipping the products. This validation of system capability to cope with production data loads in a stable manner is vital for any software system to be effectively used for its intended purposes. Additionally, recoverability testing allows the system to be implemented in a way to make it work without sudden failures and to gracefully recover from unexpected situations.
- ***Preventing disruptions for the system by hackers or malicious software***: Testing is required to find any security holes or vulnerabilities in the system to prevent any hackers or malicious software from exploiting them. This is critical as, for example, a banking system hack can cause financial crisis in a country. Further, imagine if a hack to a defense system or military missile control system could cause chaos to entire world.
- ***Ensure system is usable***: Testing helps to determine if the system is user-friendly and the experience of the users of the system is pleasant and smooth. Helping the users to improve their way of work to increase productivity is the purpose of introducing software systems, and having usability is vital for any system to be effective.

Automating Testing

Automation of software provides several benefits that should be evaluated to identify why we need to invest in automating.

- ***Return on Investment (ROI)***: Even though initially the investment is higher for automation testing when compared to manual testing, in the long run automation test costs will be less than manual testing costs (see Figure 1-2). This allows more benefits against the investments made for automation as opposed to the manual testing. Manual testing will always be available, but the lesser need for manual tests will result in higher ROI with regards to testing of software.

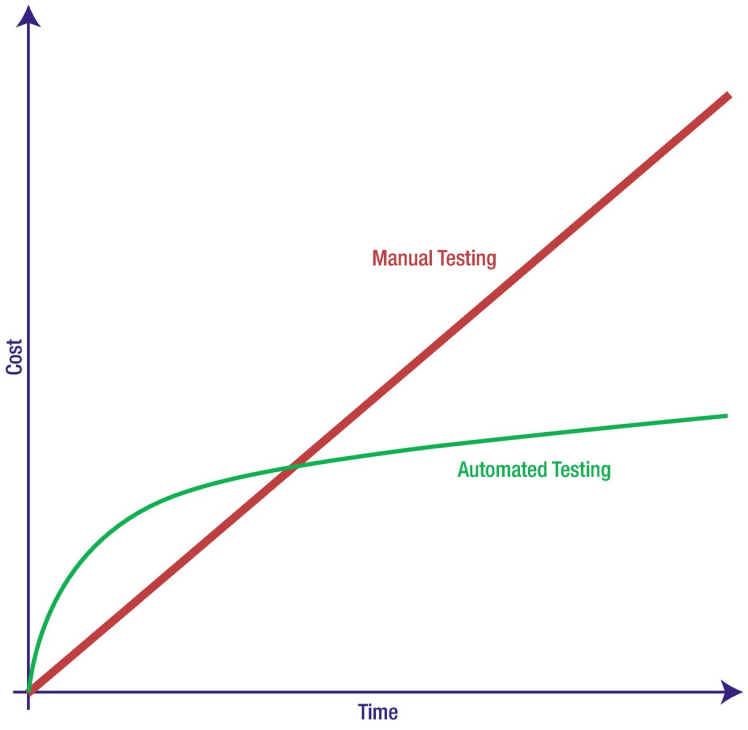

Figure 1-2 Cost of automated vs manual testing over time

- ***Capability to run on any schedule***: Automated tests can run day and night and provide results, as when they run against the systems under testing. There is no need to have human resource interaction, and you can enable them to run on their own 24/7 or on any given schedule of your preference. This capability is handy when the agility is adopted and deployment automations are in use. Once the system is deployed to a given environment, automation testing can kick in and do all the preliminary tests, such as sanity, smoke, integration, or regression tests, to ensure all previous functionalities are intact.
- ***Need for fewer human resources and interactions***: Once test automations can take over automatically testing the system for previous functionalities, the manual testing is only required for exploring and testing new functionalities. This lessens the need for human resources, which would help with cost savings.
- ***Detecting bugs quickly***: The capability to run automation tests more often and frequently allows identification of any defects quickly in the development process, which allows them to be fixed with less cost impact.
- ***Reliability***: Automations would not skip tests or test steps based on assumptions or fatigue from running the same routines over and over again, which can easily happen with human resources. This makes automated tests more reliable, as they are always performing the same steps in testing and will be using the same set of different data values.
- ***Reusable***: Automated test codes can be reusable to test similar software systems with minor changes, depending on the specifics of a particular software. This saves time and money required in the long run for any software development organization.
- ***Continuation***: Developers of test automations can utilize existing automations in development of new ones and identify what parts of functionality are already covered with automations by utilizing the clear reporting available. This, in turn, helps them to identify new or not covered areas of functionality for further automations.
- ***Simultaneous***: Test automations can run on multiple machines and distribute and run parallel tests to achieve faster execution times compared to utilizing human resources. This would enable teams discovering any issues to fix the issues quicker and provide cost savings.
- ***High volume***: Automated tests can run on many instances—for example, a mobile application being tested can run on thousands of devices parallel and faster when automation is used, which would be an impossible task to achieve with manual testing.
- ***Hitting the limits of systems***: It is impossible to stress a system or to verify the system behavior while under heavy loads and traffic with manual testing. Automations can achieve these load, performance, and volume testing requirements in an effective manner.

Considering all of these benefits, it is obvious that the implementation of software automation would be vital for any software development organization to sustain in business. The ability to meet the demand of delivering high-quality software, with state-of-the-art technological capabilities, on time to market, cannot be made a reality without using test automations.

The Impact of Future Industry Trends on Software Testing

Technology is evolving really fast in the modern world, and the software industry is changing rapidly, which demands changes about the thinking of software testing. It is worth analyzing which aspects of software testing are affected by the future trends in the software and technology industry as a whole to understand the demands of the future for test automation.

- **Adoption of Agile methodology and DevOps** : The recent trend of adoption of DevOps and agility in software development practices is in for a long run in the future. This causes high demand for test automation for different aspects of testing. It would be mandatory for software testers to obtain additional skills and work closely with developers to meet the demand for automation as much as possible. Software testers will have to become “Test Development Engineers” and will have to acquire skills to code and write scripts. With DevOps siloed, thinking in the software development process will disappear, and job role lines will be blurred. Testers may have to come out of testing siloes and explore other aspects of software systems, such as operational activities like deployment testing.
- **Cloud computing** : Software systems now run everywhere, distributed in various geo regions thanks to Cloud-based computing. These systems deliver content and data depending on the end-user’s location. Testing of such distributed systems requires moving out of traditional approaches to testing and adopting methodologies to test such distributions. Adopting automation would help to produce testing solutions to cater to these distributed needs, as parallel run capability and ability to run with less human interaction would be helpful to implement testing of all geo region testing in shorter time-frames.
- **Rise of Artificial Intelligence (AI) and Machine Learning (ML)** : The Internet of Things (IOT) is becoming part of day-to-day life in society, with wider adoption of such solutions in a smart world. Having smart phones, smart cars, and even smart houses will become a trend in the future, making testing of such scenarios critical. Automation will be vital to meeting these trends, as testing the enormous number of devices utilized in these technological areas is impossible manually.
- **Engineering for performance**: Modern software has to be intelligent, should effectively analyze the end-user behaviors, and should be able to provide customized experience for the user. This requires high performance in application systems, which have to undergo a lot of stress and performance testing, requiring automation of such testing.

It is obvious that automation of testing would play a significant role in the successful implementation of any software system. Investment in automation of testing would be rewarding for any software company to be competitive.

Summary

In this chapter, you have explored the different types of tests, the importance of software testing, and the need for automating the testing in detail. The future industry trends and the role of software testing was also described to give more insight and help you prepare for future demands.

In the next chapter, you will learn how to set up and get ready to implement test automation with Visual Studio and related frameworks.

Chaminda Chandrasekara and Pushpa Herath, *Hands-On Functional Test Automation*
https://doi.org/10.1007/978-1-4842-4411-1_2

2. Getting Started with Selenium and Visual Studio

Chaminda Chandrasekara[1] and Pushpa Herath[2]

(1) Dedigamuwa, Sri Lanka
(2) Hanguranketha, Sri Lanka

The objective of this chapter and the lessons that it encompasses is to guide you step by step to prepare your development environment to start test automation developments with Selenium and C#. You will be identifying important packages you need to add to your test automation projects and the purpose of each of those packages as well as the use of the packages. Additionally, you will explore the required setup to enable developing Selenium-based test automations with Python language in Visual Studio. Preparing your development environment with open source test automation framework MAQS and SpecFlow would pave the way to exploring capabilities of these frameworks to enhance test automations development in the coming chapters.

Lesson 2.01: Set Up a Test Project with Selenium in Visual Studio

This lesson will guide you in how to set up Visual Studio project for functional UI test automation with Selenium using the available NuGet packages. You will be writing a sample test to verify whether you have set up your test project with all the required packages.

Prerequisites: You are running Visual Studio 2017 on Windows 10 or on Windows Server 2012 R2 or a newer version of the Windows server. You should have Google Chrome version 67 or later installed. You should have intermediate level of C# language proficiency.

Setting Up Visual Studio Test Project

Let's begin setting up the first Visual Studio functional test project with C# and Selenium following the steps described here:

1. In Visual Studio 2017, select Files ➤ New ➤ Project.
2. In the New Project pop-up window, select Test under Visual Studio C# and select Unit Test Project from the test project list. Give a **Name** for the project, specify a **Solution Name**, select **Location** and click on the OK button. Leaving **Create directory for solution** checked will allow you to have a new directory created for the new solution in the selected location (see Figure 2-1).

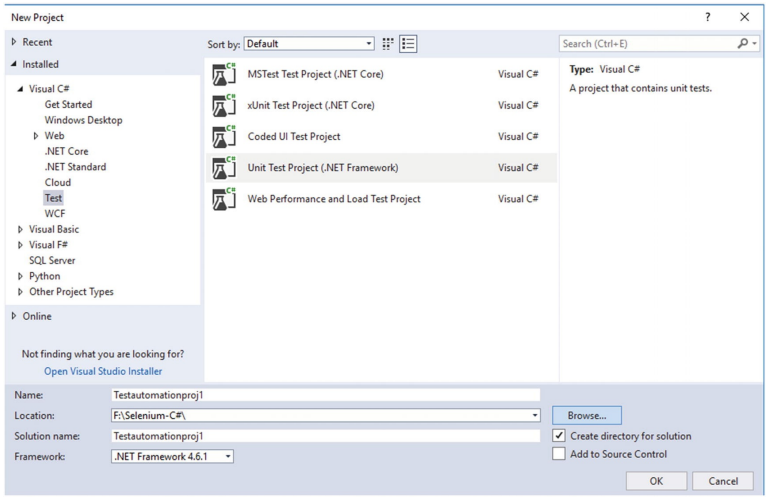

Figure 2-1 Creating a new test project

3. Next, we need to install a few required packages to enable writing Selenium base UI test automation code. Let's identify each of these NuGet packages.
   - ***Selenium.WebDriver:*** A web automation framework that comes as a NuGet package, which allows execution of UI tests against different browsers, and supports different programming languages.
   - ***Selenium.Support:*** The NuGet package contains supportive methods and classes required to handle explicit waits, which are used to wait for activities to be completed in the application being tested.
   - ***Selenium.WebDriver.ChromeDriver:*** The NuGet package contains the executable ChromeDriver that is required to run UI automation tests using the Google Chrome browser.
   - ***DotNetSeleniumExtras.WaitHelpers:*** The NuGet package contains the expected conditions implementation with .NET binding, which we will explore in detail in Chapter 3. The expected conditions implementation with .NET binding is deprecated in the latest ***Selenium.Support*** NuGet package. This portion of the code has been migrated to the ***DotNetSeleniumExtras.WaitHelpers*** package.
4. Let's see how to add the aforementioned NuGet packages to the test project created in the earlier step of this lesson, using the ***Selenium.WebDriver*** NuGet package installation as an example. Open the NuGet Package Manager by clicking Tools ➤ NuGet Package Manager ➤ Manage NuGet Packages for the Visual Studio Solution (see Figure 2-2).

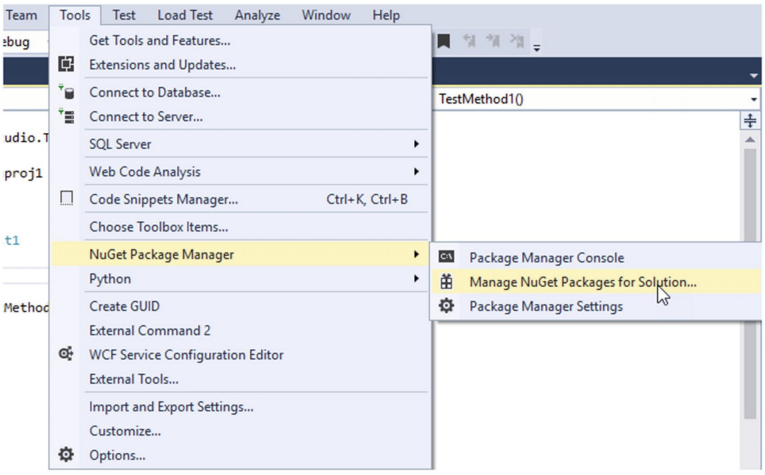

Figure 2-2 Manage NuGet packages

5. In the Browse tab of the Manage Packages for Solution window, search for ***Selenium.WebDriver***. Select the test project created in the earlier step and click on Install to get the ***Selenium.WebDriver*** NuGet package installed to the test project (see Figure 2-3).

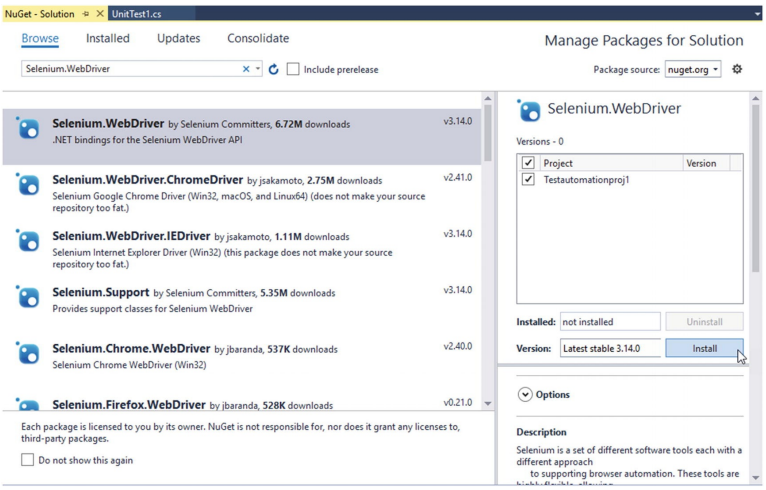

Figure 2-3 Adding a NuGet package to a project

6. After clicking the Install button, there may be a preview changes pop-up displayed. If the preview changes to pop-up is displayed, click on OK in the pop-up window to proceed with installation.
7. Next, we need to install ***Selenium.Support*** and ***Selenium.WebDriver.ChromeDriver*** packages. You can skip installing the ***DotNetSeleniumExtras.WaitHelpers*** NuGet package for now, as we are not using the expected conditions scenarios in this chapter. We can install the ***DotNetSeleniumExtras.WaitHelpers*** NuGet package later when we need it in Chapter 3. To install any NuGet package, follow the steps described for ***Selenium.WebDriver*** installation.
8. Click on the Installed tab of **Manage Packages for Solution** window and you will find all installed packages listed there (see Figure 2-4).

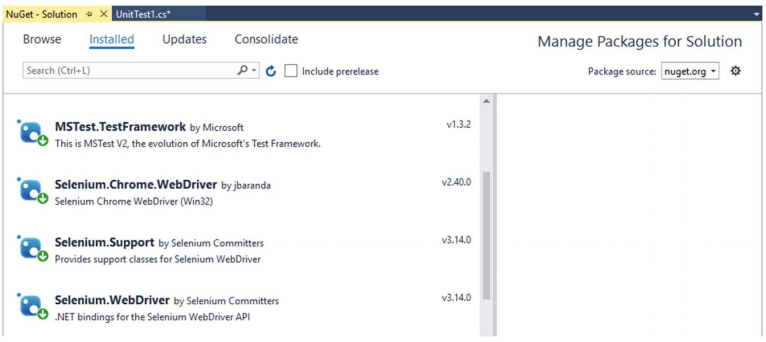

Figure 2-4 Installed packages

9. In Solution Explorer, expand project references to find installed Selenium packages. You can see installed packages are available as references in the project (see Figure 2-5).

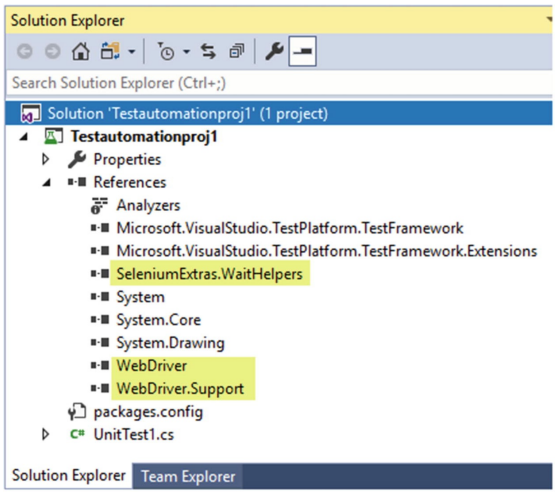

Figure 2-5 Packages in project references

Following the steps mentioned earlier, we have completed setting up the initial requirements to start writing functional UI tests with Visual Studio using Selenium.

Verifying the Test Project

We are now all set with basic requirements of the functional UI test project in Visual Studio using Selenium. Let's add some simple test code to see if the test project is able to execute functional UI tests. Code for this project can be found in `https://github.com/chamindac/Book-Test-Autmation-VS/tree/master/Chapter%202/Lesson%202.01`

1. In Solution Explorer, find the UnitTest1.cs file and double-click on it to open. Inside this class you can find a method named TestMethod1 with [TestMethod] attribute applied to it. MSTest is the default test runner and framework of Visual Studio. These attributes in the test class file are used to communicate with MSTest framework. Following are the few attributes you can use in a test class.
   - ***[TestInitialize]:*** Used to identify the method that allocates and configures resources needed by all tests in the test class.
   - ***[TestCleanup]:*** Used to identify the method that is used to free resources obtained by the tests in the test class.
   - ***[TestProperty]:*** Specify the test-specific property on a method.
   - ***[TestClass]:*** Used to identify classes that contain the test method.
   - ***[TestMethod]:*** Used to identify the test method.
2. Add the following lines of code in the method named TestMethod1. The first line initializes ChromeOption instance, and the second line adds an argument to maximize the browser. It is a good practice to maximize the browser before executing tests since a browser open with a small display area can affect the web element finding process.

   ```
   ChromeOptions option = new ChromeOptions();
   option.AddArgument("—start-maximized");
   ```

3. Then initialize the Chrome web driver instance and pass option as a parameter to maximize the web browser.

   ```
   IWebDriver driver = new ChromeDriver(option);
   ```

4. Set up the **WebDriverWait** instance where you can specify the time period for explicit wait. Explicit wait will make the web driver wait for expected conditions to become true. If the conditions are not satisfied, it will wait for the maximum timeout period (specified as 30 seconds in the following statement) before throwing a "NoSuchElementException". The usage of condition with **WebDriverWait** is explained in a later step of the method we are implementing.

   ```
   WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30));
   ```

5. Next, you need to use the web driver and navigate to a given url. For this example, let's use google.com as our test website.

   ```
   driver.Navigate().GoToUrl("https://www.google.lk/");
   ```

6. Now we have launched the Chrome browser and navigated to the web page; next, we need to perform actions on UI elements on the web page. How can you identify elements in a web page with Selenium? Selenium identifies elements using eight locators. We will discuss on all those locators in Chapter 3.
7. To identify the locator for this simple test, navigate to the Google search page with the Chrome browser. Then right-click on the search text box and select inspect, or press F12 and then click on the search text box.

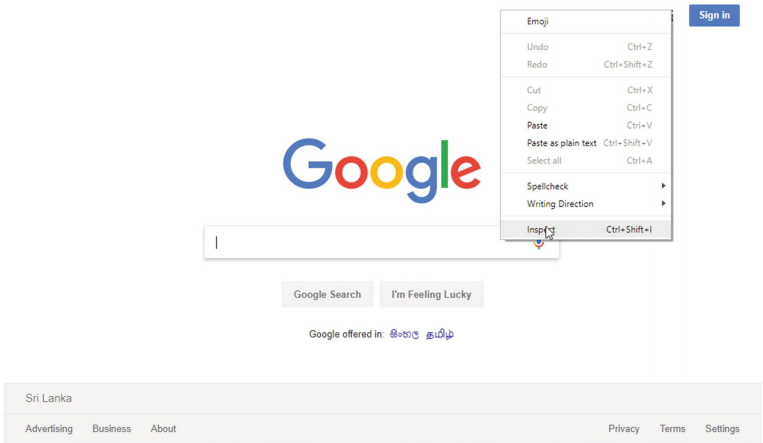

Figure 2-6 Inspecting element

8. This will open the Developer Tools window, and in it you can identify one of the locators, such as element ID of the search text box inside the highlighted area, as shown in Figure 2-7 .

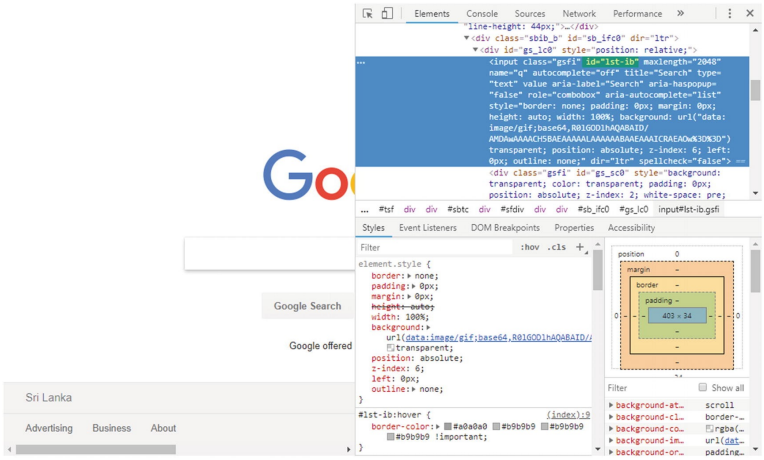

Figure 2-7 Element ID as a locator

9. Add the following code line to the method that says web driver to wait until the Google search box becomes clickable. Here we have used the **ElementToBeClickable** condition to achieve that and used element ID "**lst-ib**" as the locator. This statement will make the web driver wait until the search test box is available to type text in it, once we navigate to the google.com web page. Wait will timeout in 30 seconds if the search text box does not become available and clickable, which throws an exception making the test fail.

```
wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.ElementToBeClickable(By.Id("lst-ib")));
```

10. Once the search text box becomes clickable, you can find it with search text element ID as the locator. Remember there are other locators that we are going to discuss in Chapter 3.

```
   IWebElement textField = driver.FindElement(By.Id("lst-ib"));
```

11. Now we have identified the search text box (web element) and we can perform actions on it. The following code segment shows how to type text in a Google search text box using the SendKeys method in a web element.

```
textField.SendKeys("Selenium");
```

12. The next three lines of code identify the search button element and click on it. To locate the search button in the following code, we have used a different locator named CssSelector.

```
wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.ElementToBeClickable(By.CssSelector("input[value='Google Search']
[class='lsb']")));
IWebElement searchButton = driver.FindElement(By.CssSelector("input[value='Google Search']"));
   searchButton.Click();
```

13. Next we need to verify the action has succeed. To do that, we can use assertions. In this test we are typing the word "Selenium" in the Google search text box and click on the Google search button. Clicking on the search button page should navigate to the search results page. Hence, we have to verify whether the search result page opened or not and this can be done by verifying the search result page title using the following code line:

```
Assert.AreEqual(true, wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.TitleContains("Selenium - Google Search")));
```

14. After all the test steps are completed, you need to close the browser. In this sample code you can use the Dispose method to close all open browser windows and safely end the session as you are reaching the end of the method.

```
driver.Dispose();
```

15. The completed method should look the following:

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.Support.UI;

namespace Testautomationproj1
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestMethod1()
        {
            ChromeOptions option = new ChromeOptions();
            option.AddArgument("—start-maximized");
            IWebDriver driver = new ChromeDriver(option);
            WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30));
            driver.Navigate().GoToUrl("https://www.google.lk/");
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.ElementToBeClickable(By.Name("q")));
            IWebElement textField = driver.FindElement(By.Name("q"));
            textField.SendKeys("Selenium");
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.ElementToBeClickable(By.CssSelector("input[value='Google
Search']")));
            IWebElement searchButton = driver.FindElement(By.CssSelector("input[value='Google Search']"));
            searchButton.Click();
            Assert.AreEqual(true, wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.TitleContains("Selenium - Google
Search")));
            driver.Dispose();
        }
    }
}
```

16. Now build the test project in Visual Studio. Then, open the Test Explorer window by clicking Test ➤ Windows ➤ Test Explorer in the Visual Studio menu. The test we created will appear in Test Explorer, as shown in Figure 2-8, and you can execute it by clicking Run All. Or you can right-click on the test and run it or run with debugging.

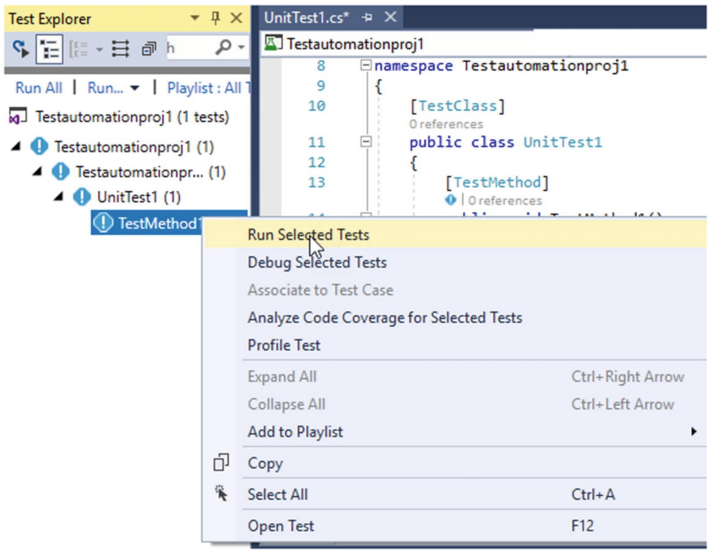

Figure 2-8 Running a test

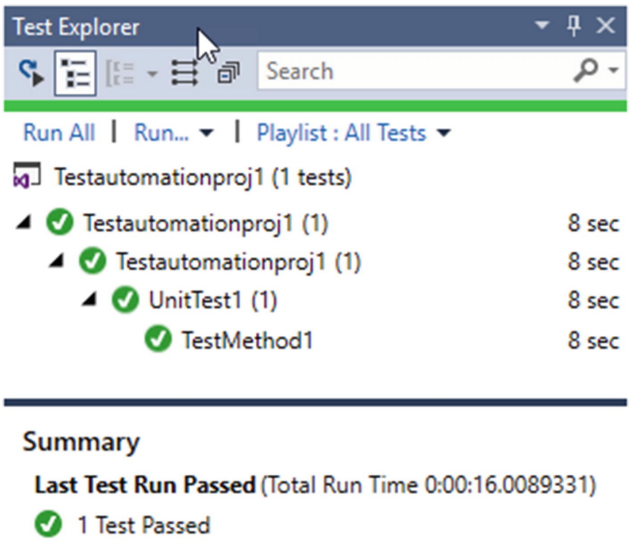

Figure 2-9 Executed test

17. When you run the test method, the Chrome browser will open, navigate to the Google search page, and perform each step of the test method.
18. After execution, the completed Test Explorer shows test method status and execution time. If the test method failed, the Test Explorer displays the failure reason, the code line that has an issue, and the cause of the issue. Test Explorer in Visual Studio is very helpful to diagnose the issues with test code, which we will discuss further in Chapter 3.

With the aforementioned sample test, we have verified the test project is equipped with all NuGet packages to execute functional UI tests. Further, you now understand how to identify web elements using locators, how to use explicit waits, and how to perform basic action on each element.

In this lesson, you have learned how to set up Visual Studio 2017 to do functional test automation using Selenium web driver and how to create a simple test project to execute UI tests using Test Explorer.

Lesson 2.02: Set Up a Test Project with Selenium and Python in Visual Studio

You can learn how to set up Visual Studio for test automation with Selenium using Python as the language in this lesson. Further, you will be able you write a sample test to verify that you have set up Python with Selenium test project correctly.

***Prerequisites**:* You must have Visual Studio 2017 or later and the Chrome browser installed on your machine. You must be familiar with setting up Visual Studio and its workloads.

Setting up Visual Studio to Work with Python

We need to install Python development workload to Visual Studio to enable writing tests with Python language. You can modify the installed workloads using Visual Studio Installer.

1. Open Visual Studio Installer in the Start menu of Windows (see Figure 2-10).

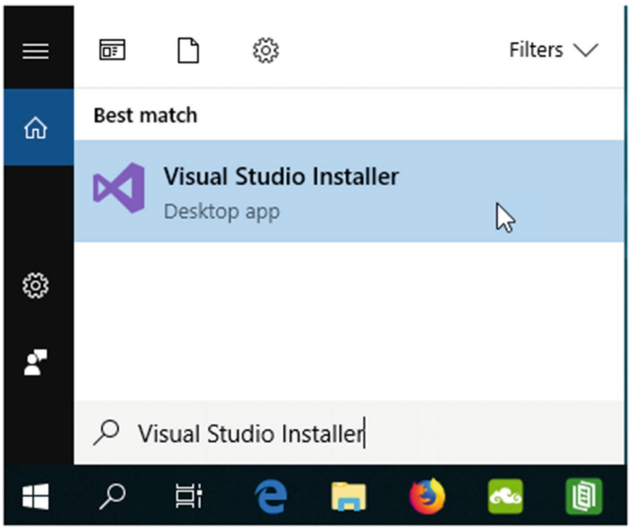

Figure 2-10 Open Visual Studio Installer

2. Click on the Modify button in the Visual Studio edition you have installed on your machine in the opened Visual Studio Installer window. Select the check box for Python development workload and click on the modify button to install it (see Figure 2-11).

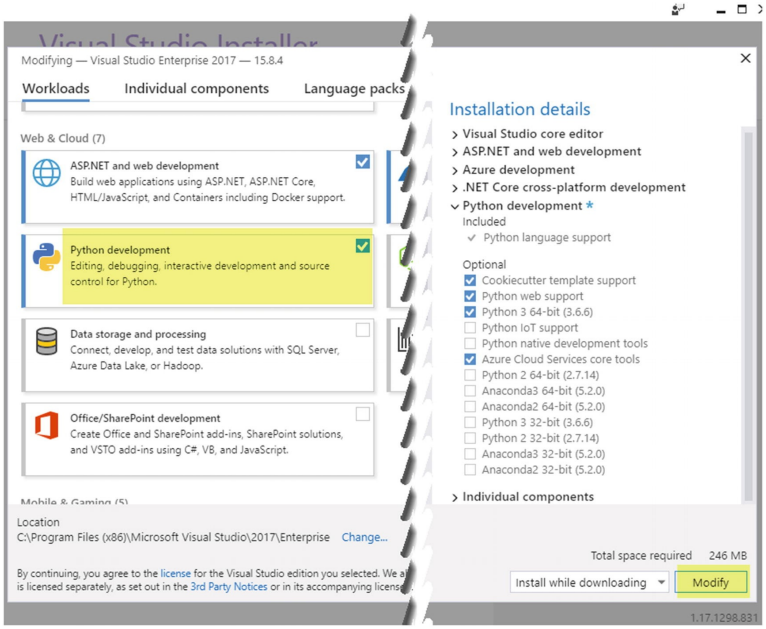

Figure 2-11 Installing Python workload

3. Once the installation is completed, launch the Visual Studio.
4. To create the Python project in Visual Studio, select Files ➤ New ➤ Project.
5. In the New Project pop-up window, select Python and Select Python Application from the project list. Give a **Name** for the Project, specify a **Solution Name**, select **Location,** and click on the OK button. Leaving **Create directory for solution** checked will allow you to have a new directory created for the new solution in the selected location (see Figure 2-12).

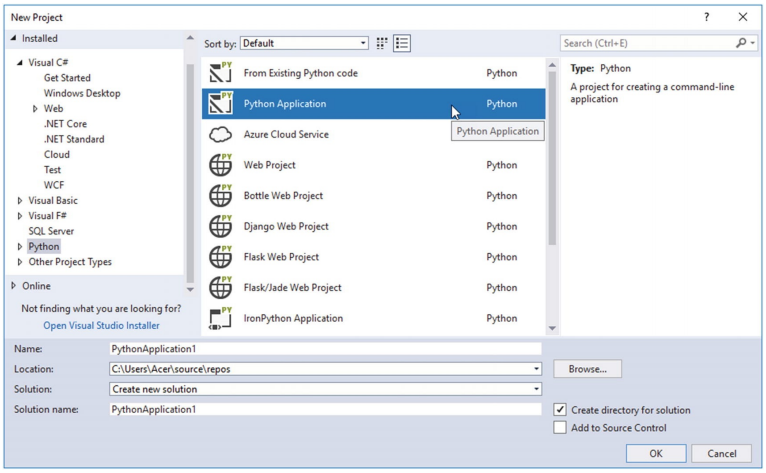

Figure 2-12 Creating a Python project

6. Now we need to add unit test class to the project in order to start writing our tests using Selenium and Python. In Solution Explorer, right-click on project and select Add ➤ New Item.
7. Select the Python Unit Test item from the file list displayed in the pop-up window. Give a name for the test class and click on the Add button to add the unit test file to the project (see Figure 2-13).

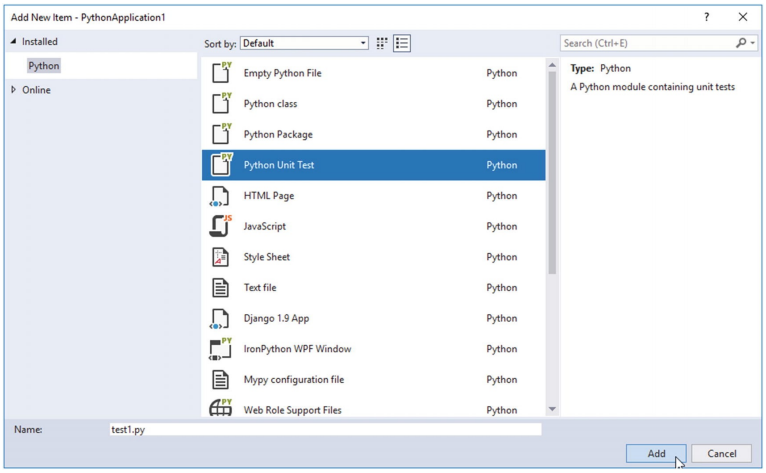

Figure 2-13 Adding Python Unit Test class

8. If you inspect the newly added Python Unit Test class file, it contains code, as shown in Figure 2-14. **unittest** is a module that is imported to allow you to create a derived class from unittest.TestCase, which you can see was created with the name Test_test1 by default. Further, a sample test case was created, which was set to throw a "Not Implemented" exception.

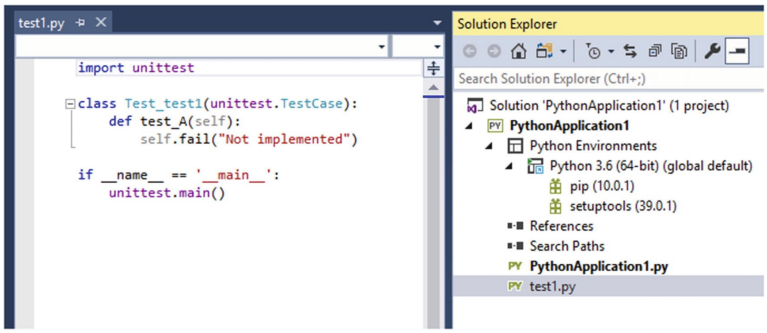

Figure 2-14 Python Unit Test class

9. Next, we need to add the required packages to develop tests. You can install packages by right-clicking on Python Interpreter and clicking on **Install Python Packages**. Python Interpreter ("Python 3.6 (64 bit) (global default)" in Figure 2-15) can be found by expanding the Python Environments node in Solution Explorer. By default, two packages are already available in the Python Interpreter.

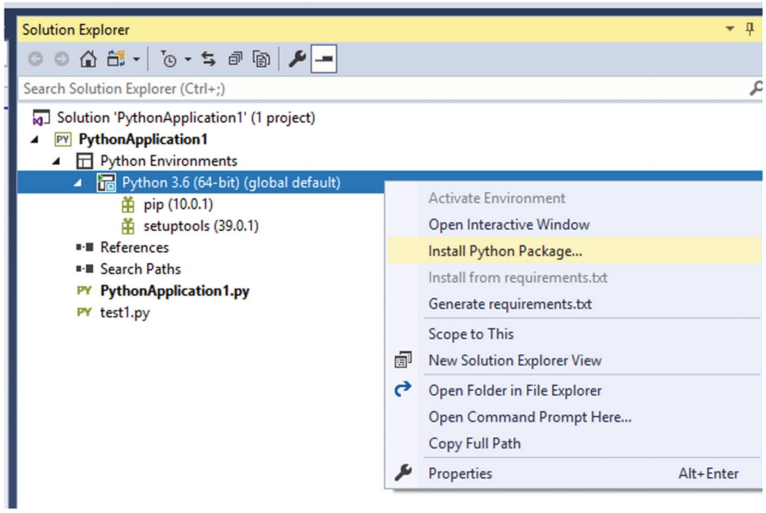

Figure 2-15 Open Python environment

10. In the search box that appears under the drop-down, which is selected as "Packages (PyPI)" in the Python Environments window, search for Selenium. From the search results displayed, you can click on Install Selenium to get the Selenium added to the project we created.

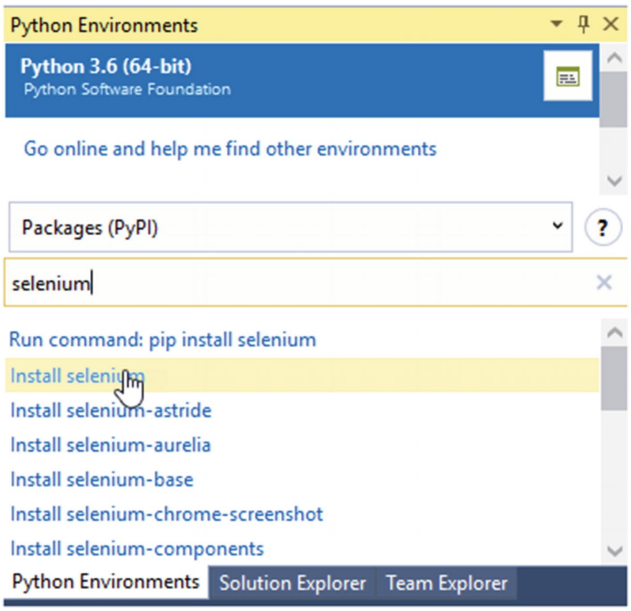

Figure 2-16 Installing the Selenium package for Python

11. You can download the Chrome driver and add chrome.exe to the project folder. You can download chromedriver version 2.41 here: `https://chromedriver.storage.googleapis.com/index.html?path=2.41`
    Download chromedriver_win32.zip file and extract the file. You can find the chromedriver.exe file inside the extracted folder. Copy and paste this chromedriver.exe to the project folder.

    We have set up the project with all packages and now the project is ready for writing test code with Selenium using Python.

Executing Sample Python Test Code

Let's try to implement a similar type of Google search, click, and verify test case, which is implemented in Lesson 2.01 with C#, by now using Python with Selenium in Visual Studio. Code for this project can be found here: `https://github.com/chamindac/Book-Test-Autmation-VS/tree/master/Chapter%202/Lesson%202.02` .

1. Open the Unit Test class file you created earlier and add the following code lines just below the `import unit test` statement:

```
from selenium import webdriver
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as ExpectedCondition
```

    Each import enables the following mentioned functionality:

- ***webdriver:*** Enables writing tests that use Selenium web driver.
- ***WebDriverWait:*** Allows handling waits for test conditions.
- ***By:*** Allows searching for web elements in a web page.
- ***ExpectedCondition:*** Allows implementation of expected conditions for waits to locate elements in a web page.

Figure 2-17 shows a basic outline of test code in python. If you are wondering about the `If __name__ == '__main__'` and the code line `unittest.main()`, it is the code that allows the python files to behave as a stand-alone program or reusable module. For further information, read the article "What is 'if_name_=="_main_'" for?" (`http://effbot.org/pyfaq/tutor-what-is-if-name-main-for.htm`) to get demystified.

Figure 2-17 Imports

2. Replace the code line `self.fail("Not implemented")` (highlighted in Figure 2-17) with following two code lines:

```
driver=webdriver.Chrome()
driver.get('https://www.google.lk/')
```

    If chromedriver.exe is not inside the project folder, you need to give the chromedriver.exe location path as a parameter when creating the chrome driver instance, which is done in the first code line. We have added chromedriver.exe to the project folder previously. Therefore, the driver path is not given in the sample code. However, if you have chromedriver.exe in a different location other than the project, you can specify the first line similar to following with the chromedriver.exe path:

```
driver=webdriver.Chrome("F:\Selenium-python\ChromeDriver\chromedriver.exe")
```

    The second code line from earlier allows navigation to a given web page, where we are using the google.com URL in this instance.
3. Below the first two code lines with the same indentation (indentation is used to define method, code block, or class scope in Python), add the following line of code to enable explicit wait time to wait until the page elements meet the given expected condition. If this time-out passes, the test case will fail with the TimeoutException. This sample code uses 60 seconds as the wait time:

```
wait=WebDriverWait(driver,60)
```

4. Let's implement clicking of the "I'm Feeling Lucky" button of the Google search page, which should open the "Google Doodle" page. The "I'm Feeling Lucky" button element locators need to be identified to be able to click it. As explained in Lesson 2.01, you can use developer tools by pressing F12 or right-clicking on an element and click

Inspect. The “Name” Locator of the “I’m Feeling Lucky” button is used with the following code. The second code line clicks on the button.

```
feelingButton=wait.until(ExpectedCondition.element_to_be_clickable((By.NAME,"btnI")))
feelingButton.click()
```

5. Then click on the button; if the button is functioning correctly, then page should navigate to the “Google Doodle” page. The following line verifies the page has successfully navigated to the correct page by checking the page title:

```
self.assertEqual(True,wait.until(ExpectedCondition.title_contains("Google Doodles")))
```

6. At the last statement of the test case, use the “quit” method of the web driver to close the browser window.

```
driver.quit()
```

The fully implemented code should be similar to the following code. It is important to note the code indentation, as it defines the scope of code.

```
import unittest
from selenium import webdriver
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as ExpectedCondition

class Test_test1(unittest.TestCase):
  def test_A(self):
    driver=webdriver.Chrome()
    driver.get('https://www.google.lk/')
    wait=WebDriverWait(driver,60)
    feelingButton=wait.until(ExpectedCondition.element_to_be_clickable((By.NAME,"btnI")))
    feelingButton.click()
    self.assertEqual(True,wait.until(ExpectedCondition.title_contains("Google Doodles")))
    driver.quit()
if __name__ == '__main__':
  unittest.main()
```

Similarly to the test case implemented with C# and Selenium, the test case implemented in this lesson with Python and Selenium will be available in the Test Explorer window, which will allow you to run or debug the test case.

In this lesson, you have learned how to set up a Visual Studio Python project to work with Selenium and Python to develop functional UI test case.

Lesson 2.03: Set Up Test Project with Visual Studio and MAQS Open Framework

Let’s explore how to set up Visual Studio for test automation with MAQS (Magenic’s Automation Quick Start) open framework. MAQS helps you to quickly set up test automation project and provides utilities to enhance the test automation writing experience. In this lesson, you will be able you write a sample test to verify that you have set up your test project successfully with the MAQS framework. You can learn more about MAQS here: https://github.com/Magenic/MAQS .

Prerequisites: You must have Visual Studio 2017 or later and the Chrome browser installed on your machine.

Setting Up Visual Studio for Work with MAQS Open Framework

Let’s create a test project in Visual Studio using the MAQS framework.

1. You must install .NET Framework version 4.7.1 before setting up a test project with MAQS 5. To install .NET Framework 4.7.1, you can use Visual Studio installer and select the .NET Framework 4.7.1 components from the individual component list (see Figure 2-18).

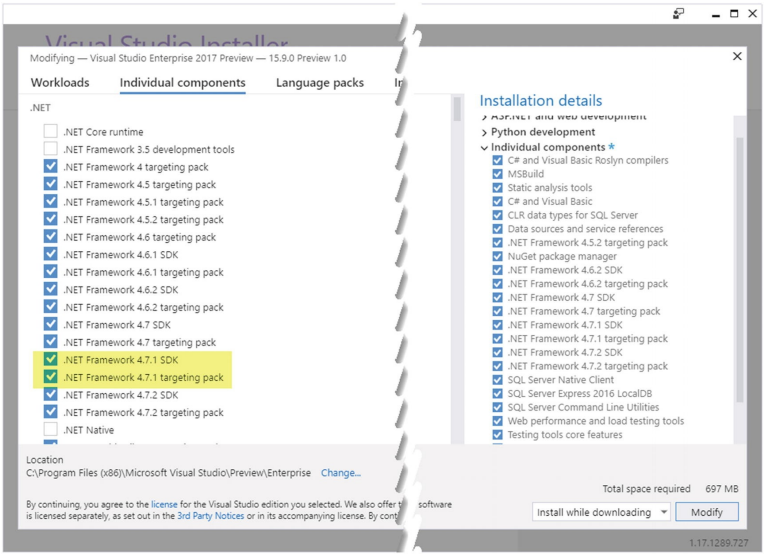

Figure 2-18 Install .NET Framework 4.7.1

2. Go to Visual Studio Market Place (https://marketplace.visualstudio.com/). In the search tab, search for the MAQS Open Framework and click on MAQS Framework from the search results.

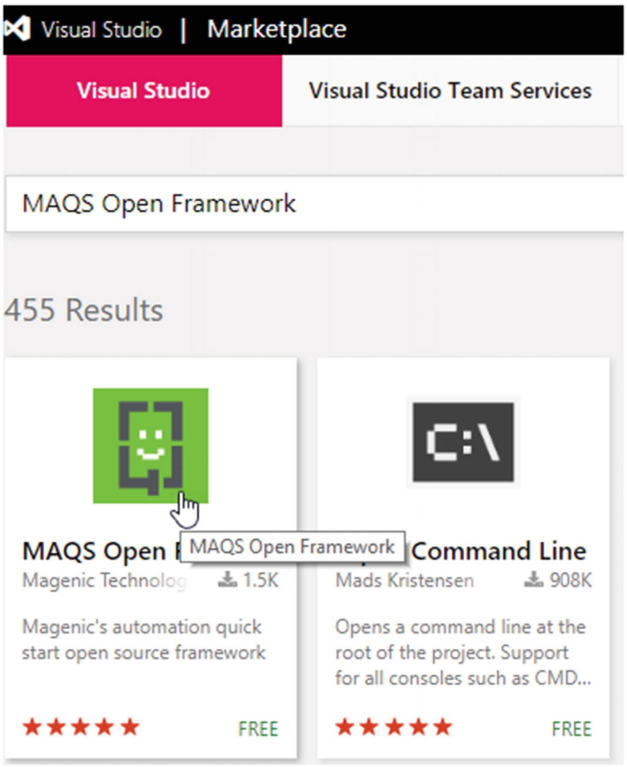

Figure 2-19 MAQS in Visual Studio marketplace

3. Click the download button to download MAQS.
4. Open the Maqs.vsix file and run this file to install MAQS Framework for Visual Studio. Instead of going to Visual Studio marketplace directly, you can install MAQS from within Visual Studio by Clicking on Tools ➤ Extensions and Updates (if you use Visual Studio to download the extension to get it installed, you need to close the Visual Studio, and once the extension is installed, you can open it again). Then you have to search for MAQS and download it (see Figure 2-20).

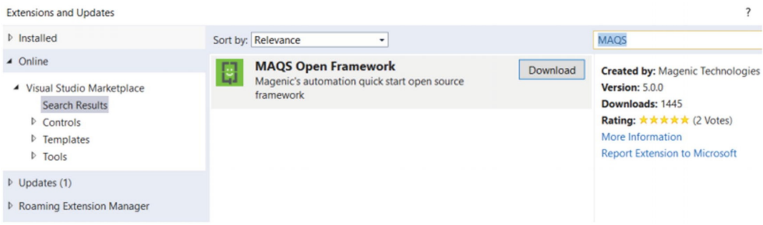

Figure 2-20 Download MAQS from VS

5. After installing the MAQS extension for Visual Studio, create a new project by clicking Files ➤ New ➤ Project in Visual Studio.
6. In the New Project pop-up window select **Magenic's Open Test**. In the project window you can see the **Maqs Framework – Selenium** project is selected. Give a Name for the Project, specify a Solution Name, select Location and click on the OK button. Leaving Create directory for solution checked will allow you to have a new directory created for the new solution in the selected location (see Figure 2-21).

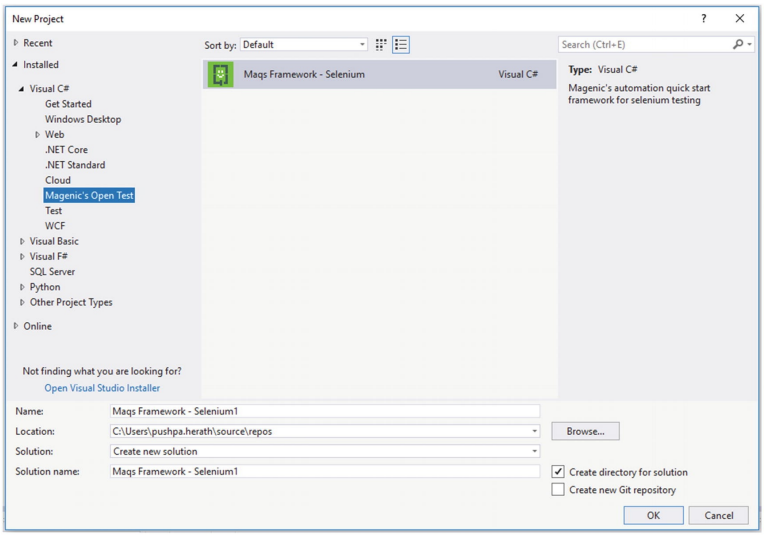

Figure 2-21 Create test project with MAQS

7. Build the project so that it restores the MAQS NuGet package to retrieve all needed dependencies and compile the project.

With these steps, we have successfully set up the MAQS Open Framework for Visual Studio and created a test project using it.

Executing Sample Test Code with MAQS Open Framework in Visual Studio 2017

The MAQS-based test protect includes pre-written tests samples that you can execute to verify whether the project has been configured correctly. Code for this project can be found here: https://github.com/chamindac/Book-Test-Autmation-VS/tree/master/Chapter%202/Lesson%202.03

1. In Solution Explorer, under the test project, the SeleniumTests.cs file can be found. Open this file to view pre-written test cases. Build the solution in Visual Studio and you will see test cases appear in the Test Explorer window. Run these tests to verify whether the MAQS-based project is set up correctly (see Figure 2-22).

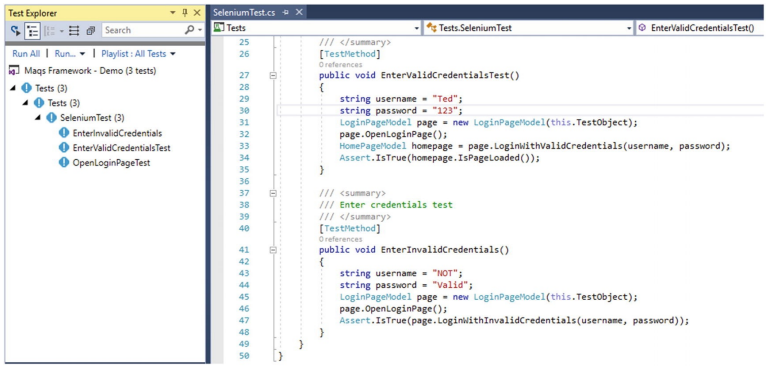

Figure 2-22 Sample tests by MAQS

If the tests are executed successfully, you have set up the test project correctly in Visual Studio with the MAQS framework. We will discuss the benefits of using MAQS and the utilities provided by it in detail in Chapter 3.

Lesson 2.04: Set Up A Test Project with Visual Studio and SpecFlow

SpecFlow is an open source testing framework that supports behavior-driven development (BDD). It lets us define application behavior in English text using a language called Gherkin. This will enhance collaboration of technical and business staff involved in a software project on the project requirements implementation and verification. You can find more information here: https://specflow.org/ .

This lesson will guide you on how to set up Visual Studio for test automation with the SpecFlow Framework. Then you will write a sample test to verify whether you have set up your test project correctly with SpecFlow.

Prerequisites: You need Visual Studio 2017 or later, and since this lesson demonstrates using the Chrome driver, the Chrome browser should be installed on your working machine.

Setting Up Visual Studio to Work with SpecFlow and Selenium

Let's get the project set up with SpecFlow following the steps below:

1. As the first step, it is recommended to add the SpecFlow Visual Studio integration to Visual Studio. It is not mandatory to do so, but when the integration is added, it requires less effort to configure SpecFlow for Visual Studio. However, adding this extension does not provide you with a specific project template. To add the SpecFlow integration to Visual Studio, you can use the Extensions and Updates window in Visual Studio by clicking Tools ➤ Extensions and Update menu item. In the Extensions and Updates window, search for SpecFlow and download the extension (see Figure 2-23). You have to close Visual Studio to get the extension installed and, once done, you can open Visual Studio again.

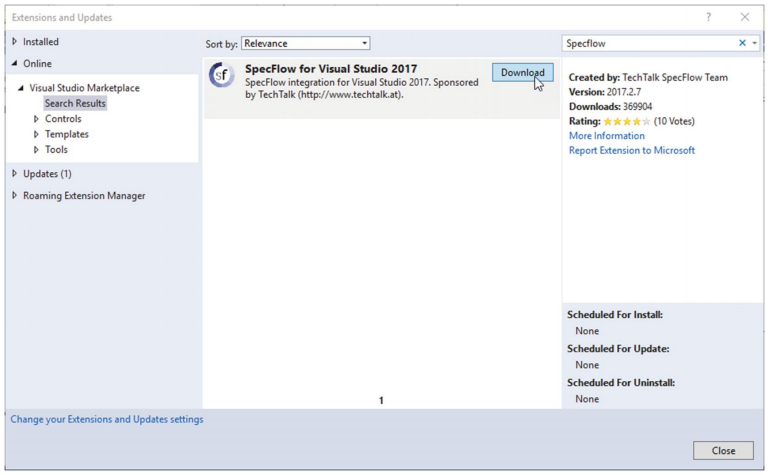

Figure 2-23 SpecFlow extension for Visual Studio

2. Once you are done setting up the extension, create a test project in Visual Studio by clicking File ➤ New ➤ Project.
3. In the New Project pop-up window, select Windows Desktop under Visual C#. Select Class Library from the list of projects in the project window. Give a Name for the Project, specify a Solution Name, select Location, and click on the OK button. Leaving **Create directory for solution** checked will allow you to have a new directory created for the new solution in the selected location (see Figure 2-24) .

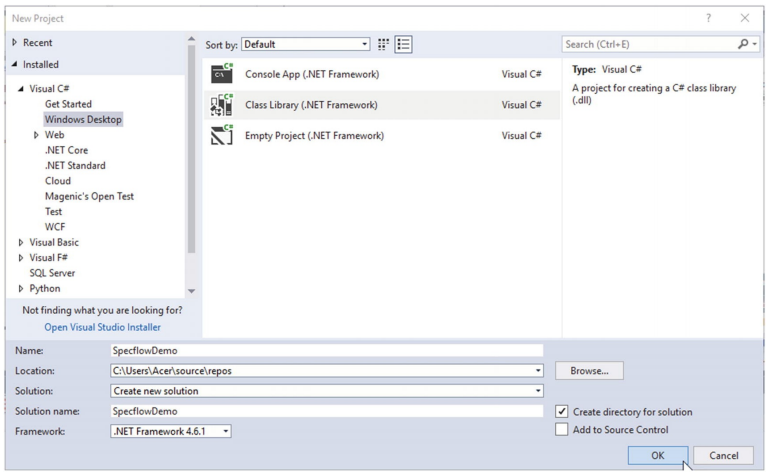

Figure 2-24 Creating project

4. Few NuGet packages are required to set up the SpecFlow for the project we created in step 3. Open the NuGet package management window for the solution by right-clicking on the solution in Solution Explorer and clicking "Manage NuGet Packages for solution" in the pop-up menu. Search for SpecFlow in the Browse tab of the NuGet package management window, select SpecFlow NuGet package (which contains techtalk.SpeckFlow.dll, which is required to implement SpecFlow-based tests). Install the SpecFlow NuGet package to the project you created (see Figure 2-25).

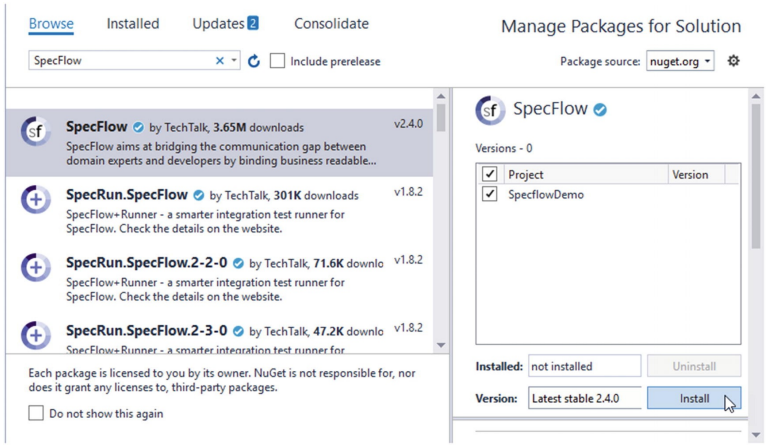

Figure 2-25 Installing SpecFlow NuGet package

5. Next you have to install NUnit for the project. Why NUnit? In reality, SpecFlow needs a test framework, and it supports few test frameworks, such as NUnit, xUnit, MbUnit, and MSTest. These test frameworks facilitate running tests with SpecFlow in Visual Studio using the Test Explorer. However, you can use SpecFlow+Runner (`https://specflow.org/plus/runner/`), which comes with a cost but provides integration with Test Explorer without having to use other test frameworks. Further, SpecFlow Runner provides detailed reporting features integrated with Visual Studio. But, let's skip the SpecFlow runner and try to deal with free stuff in the lessons in this book. To set up the NUnit framework, go to the NuGet package manager, search NUnit in the Browse tab, and install NUnit for the project (see Figure 2-26).

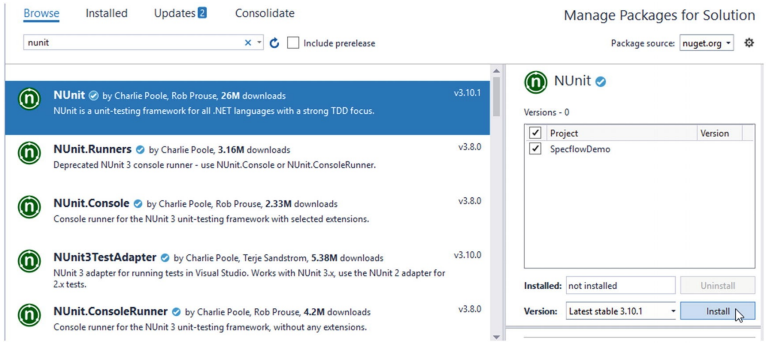

Figure 2-26 Installing NUnit package

6. Then install NUnit3TestAdapter using the NuGet Package Manager. This is a supportive package required to work with NUnit.

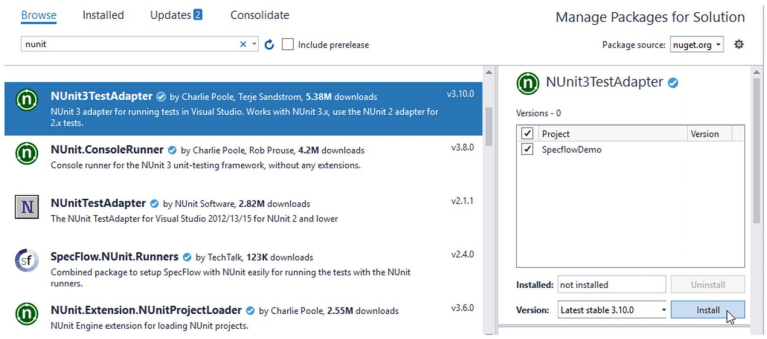

Figure 2-27 Installing NUnit3TestAdapter package

7. Install Selenium.WebDriver, Selenium.Support, Selenium.Chrome.WebDriver, and DotNetSeleniumExtras.WaitHelpers packages using the NuGet package manager.

With these steps, we have set up the project for writing SpecFlow-based tests in Visual Studio.

Execute Sample Test with the SpecFlow Framework

As Visual Studio is ready to work with Selenium and SpecFlow, let's try to execute sample tests with the project we set up in the previous section of this lesson.

1. We have to add a feature file to the project that will be used by SpecFlow to define acceptance criteria for features, such as use-cases or user stories. The feature file uses Gherkin language syntax, which was introduced by Cucumber (`https://cucumber.io/`). It is a good practice to create a separate folder inside the project to add feature files. Hence, create a new folder inside the project to add feature files. Add feature file to the project by right-clicking on the feature folder in Solution Explorer and select Add ➤ New Item. Select the SpecFlow feature file from the new items pop-up by clicking on Visual C# Items. Give a meaningful name and click on Add (see Figure 2-28). You can see all SpecFlow items in Visual C# items due to the SpecFlow extension we installed for Visual Studio.

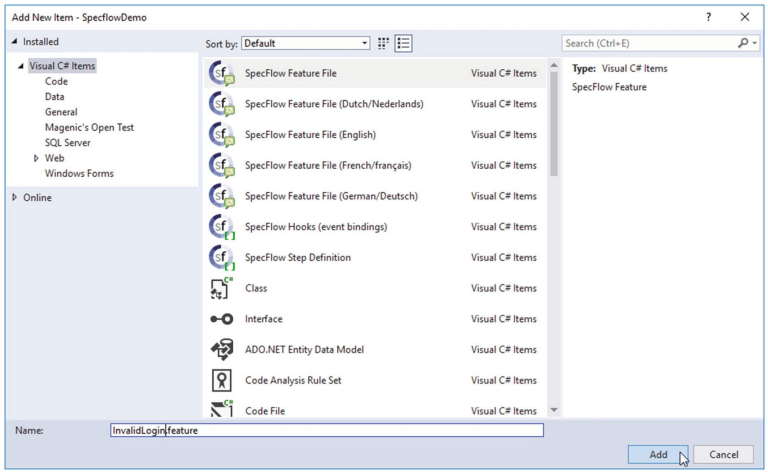

Figure 2-28 Adding Feature file

2. Feature file is where we describe the scenario. Gherkin language syntax will let anybody understand the software behavior easily since it is written in business domain-specific English language. This language has little syntax. The following are the main syntax items and uses:

- ***Feature:*** used to specify the description of the feature
- ***Scenario:*** used to specify the description of the scenario
- ***Background:*** used to specify the action to be taken before each scenario in the feature file is executed
- ***Given:*** used to specify the precondition of the scenario
- ***And:*** used to specify the additional preconditions and actions
- ***When:*** used to specify the action
- ***Then:*** used to specify the results of the action

3. In the new Feature file added, you can add scenarios that need to be tested, as shown in the following sample feature file, which tests invalid login. The scenario explained here is a user trying to log in to the web application by giving an invalid username and password. Because a preconditioned user is at the home page of the web site and next action, navigate to the login page is given as an additional precondition. Then enter an incorrect username and password and click on the login button. Finally, verify the validation message as a result of these actions.

```
Feature: Invalid Login
    In order to restrict the access to the userprofile
    as a non registered user of web site i should not be able to log into the website
@mytag
Scenario: Unsuccessful Login with invalid credentials
    Given User is at the home page
    And Navigate to login page
    When User enter incorrect user name and password
    And Click on the signin button
    Then Validation message should display and browser should close
```

4. After you generate the Feature file, you need the step definition file to do automation scripting. Step definition is a code with a pattern attached to it. To generate a step definition file, right-click on feature file code and select Generate step definition (see Figure 2-29).

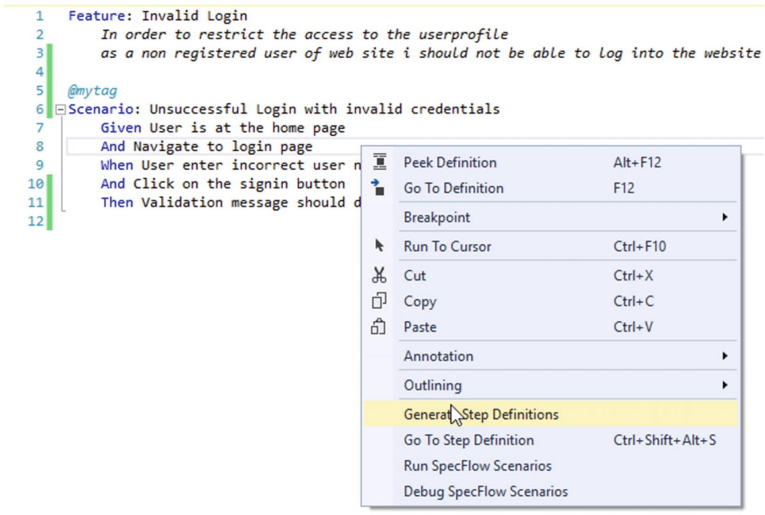

Figure 2-29 Generate Step Definition file

5. The “Generate Step Definition Skeleton” pop-up window will be displayed with all given steps of the feature file scenario. You can decide which steps should be included in the step definition file. Then click the Generate button (see Figure 2-30).

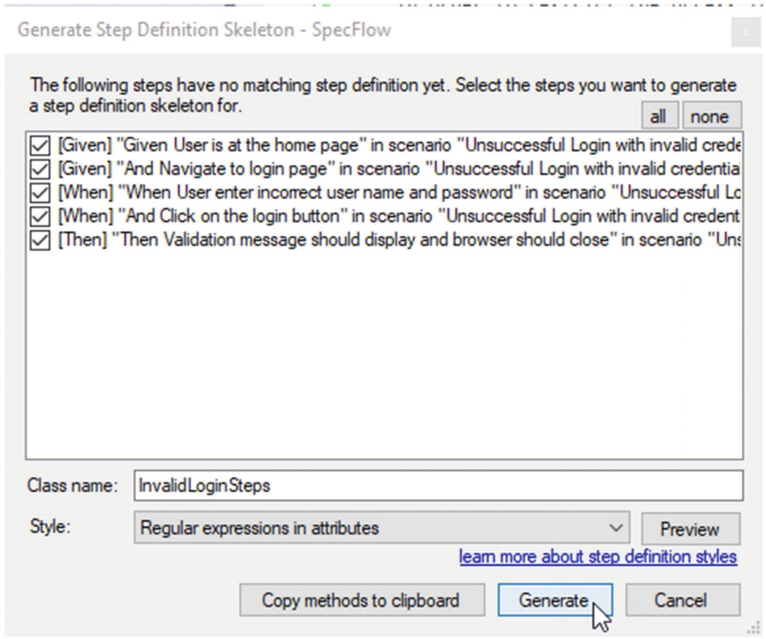

Figure 2-30 Select steps for Step Definition

6. After clicking on the Generate button, another pop-up will be displayed where you can give location of the definition file. Select the default project folder and click on the Save button (see Figure 2-31).

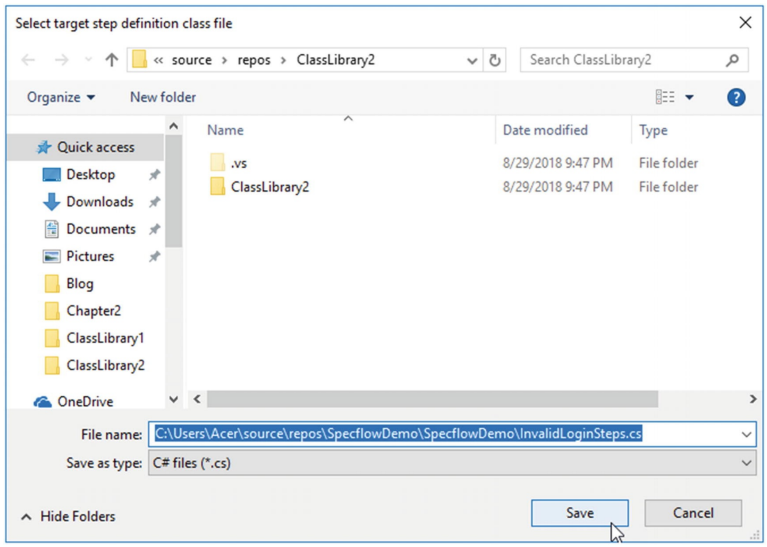

Figure 2-31 Save Step Definition file in the project folder

7. In the Feature file, each step explains an action that needs to be taken while testing the scenario. In step definition, sample code with method structure will be created according to the Feature file scenario steps and you have to fill the method body.
8. The following code shows the completed step definition code. You can download all the source code for this project from https://github.com/chamindac/Book-Test-Autmation-VS/tree/master/Chapter%202/Lesson%202.04 :

   a. Method1: **GivenUserIsAtTheHomePage()** - Open the Chrome browser, maximize the browser, and navigate to home page of web site.
   b. Method2: **GivenNavigateToLoginPage() -** Navigate to the Login page (GitHub is used here).
   c. Method3: **WhenUserEnterIncorrectUserNameAndPassword() -** Enter invalid username and password.
   d. Method4: **WhenClickOnTheLoginButton()** - Click on the sign in button of the login page.
   e. Method5: **ThenValidationMessageShouldDisplayAndBrowserShouldClose()** - Verify validation message and close the browser:

```
using NUnit.Framework;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.Support.UI;
using System;
using TechTalk.SpecFlow;

namespace SpecflowDemo
{
  [Binding]
  public class InvalidLoginSteps
  {
    IWebDriver driver;
    WebDriverWait wait;

    [Given(@"User is at the home page")]
    public void GivenUserIsAtTheHomePage()
    {
      ChromeOptions option = new ChromeOptions();
      option.AddArgument("—start-maximized");
      driver = new ChromeDriver(option);
      driver.Navigate().GoToUrl("https://github.com");
    }

    [Given(@"Navigate to login page") ]
    public void GivenNavigateToLoginPage()
    {
      wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30));
      driver.FindElement(By.LinkText("Sign in")).Click();
      wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.UrlContains("https://github.com/login"));
    }

    [When(@"User enter incorrect user name and password")]
    public void WhenUserEnterIncorrectUserNameAndPassword()
    {
      driver.FindElement(By.Id("login_field")).SendKeys("ABC");
      driver.FindElement(By.Id("password")).SendKeys("123");

    }

    [When(@"Click on the signin button")]
    public void WhenClickOnTheSigninButton()
    {
      driver.FindElement(By.Name("commit")).Click();
    }

    [Then(@"Validation message should display and browser should close")]
    public void ThenValidationMessageShouldDisplayAndBrowserShouldClose()
    {
      IWebElement validationMsg = wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.ElementIsVisible(By.Id("js-
flash-container")));
      StringAssert.Contains("Incorrect username or password", validationMsg.Text);
      driver.Dispose();
    }
  }
}
```

9. Build the project in Visual Studio after filling the methods with code as shown in step 8. Test Explorer should appear in the test after you build the project.
10. Sometimes tests may not appear in Test Explorer even after building the project successfully. In this case, you may have to update the App.config file of the project. This App.config file gets added to the project when we initially add the SpecFlow NuGet package.
11. Open the App.config file and modify the configuration file by adding line **<unitTestProvider name="NUnit"/>** as shown here:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
 <configSections>
 <section name="specFlow" type="TechTalk.SpecFlow.Configuration.ConfigurationSectionHandler, TechTalk.SpecFlow" />
 </configSections>
 <specFlow>
 <unitTestProvider name="NUnit" />
 <!-- For additional details on SpecFlow configuration options see http://go.specflow.org/doc-config --></specFlow>
</configuration>
```

12. Now the test method should appear in the Test Explorer window. Right-click on method and select Run Selected Tests.

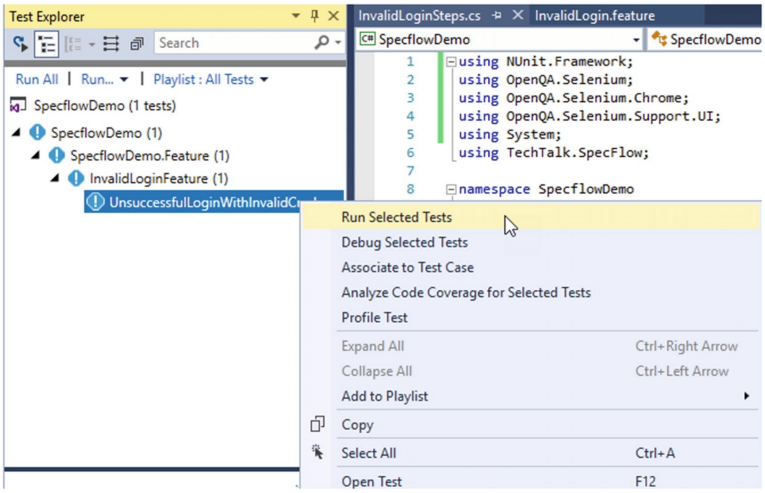

Figure 2-32 Running SpecFlow-based tests

You have learned how to set up Visual Studio to work with SpecFlow and Selenium within this lesson by setting up a project and then writing and executing a sample scenario.

Summary

In this chapter, you have obtained hands-on experience with setting up Visual Studio for Selenium with C# and Python languages. Additionally, you have learned how to use MAQS and SpecFlow frameworks inside Visual Studio. These lessons will be helpful as you continue on to the next chapter, where we will be deep-diving into different needs and scenarios of implementing automation testing for web and windows applications.

Chaminda Chandrasekara and Pushpa Herath, *Hands-On Functional Test Automation*
https://doi.org/10.1007/978-1-4842-4411-1_3

3. Functional Testing for Web Applications

Chaminda Chandrasekara[1] and Pushpa Herath[2]

(1) Dedigamuwa, Sri Lanka
(2) Hanguranketha, Sri Lanka

The objective of this chapter is to guide you to start automation development using different languages and Selenium. Through the first few lessons you will be identifying important of web element locators and commands. Then you can find lessons with code examples of how to automate web application with Selenium, C#, Python, SpecFlow, and the open source automation framework MAQS.

Source code of this chapter can be found in https://github.com/chamindac/Book-Test-Autmation-VS/tree/master/Chapter%203.

Lesson 3.01: Create Test Project with C#

You already learned how to set up Visual Studio project for functional UI test automation with Selenium using the available NuGet packages in Chapter 2. Within this chapter we are going to discuss web element locators, commands with sample codes to improve your ability to write automation tests with Selenium. You will be writing a sample test to verify each web element locator and command, which are explained in the next two lessons.

Prerequisites: You must be running Visual Studio 2017 on Windows 10 or on Windows Server 2012 R2 or newer version of Windows server. You must have Google Chrome version 67 or later installed. You must have intermediate level of C# language proficiency.

Let's begin setting up Visual Studio functional test project with C# and Selenium by following the steps described below:

1. In Visual Studio 2017, select Files ➤ New ➤ Project.
2. In the New Project pop-up window, select Test under Visual Studio C# and Select Unit Test Project from the test project list. Give a **Name** for the Project, specify a **Solution Name**, select **Location** and Click on the OK button. Leaving **Create directory for solution** checked will allow you to have a new directory created for the new solution in the selected location.
3. Next install the following NuGet packages to enable writing Selenium-based UI test automation code.

- `Selenium.WebDriver`
- `Selenium.Support`
- `Selenium.WebDriver.ChromeDriver`
- `DotNetSeleniumExtras.WaitHelpers`

With this project created, let's move on to the next lesson to start exploring the using different locators.

Lesson 3.02: How to Capture Web Elements

Identifying web elements uniquely within the webpage is an important requirement of test automation. Locators are the HTML properties that act as a unique address of the web elements that help to identify the web elements uniquely. You will explore the Selenium locators in detail by performing the steps described in this lesson.

Following are the most common web elements that can be found in almost all the web applications:

- Text box
- Button
- Drop Down
- Hyperlink
- Check box
- Radio button

Suppose there is more than one text box on a web page and we need to add values to each text box. To do that, we need to identify each text box uniquely. Selenium has eight types of locators that we can use to uniquely identify web elements.

Following are the eight Selenium web element locators:

- ID
- Name
- Class Name
- CSS Selector
- Link Text
- Partial Link Text
- Tag Name
- XPath

Prerequisites: You must be running Visual Studio 2017 on Windows 10 or on Windows Server 2012 R2 or a newer version of Windows server. You must have Google Chrome version 67 or later installed. You need an intermediate level of C# language proficiency. We will be using sample application available on GitHub (https://github.com/dotnet-architecture/eShopOnWeb) to perform steps in this lesson. Set up the application as an Azure web app or host it in IIS following the instructions available in GitHub.

You must identify web element locators available for each UI element. So, you can find available locator values with Developer tools as follows:

1. Open the Chrome web browser and navigate to the eShop application web page.
2. Click on the F12 key.
3. Click on the Inspector icon. (See Figure 3-2).

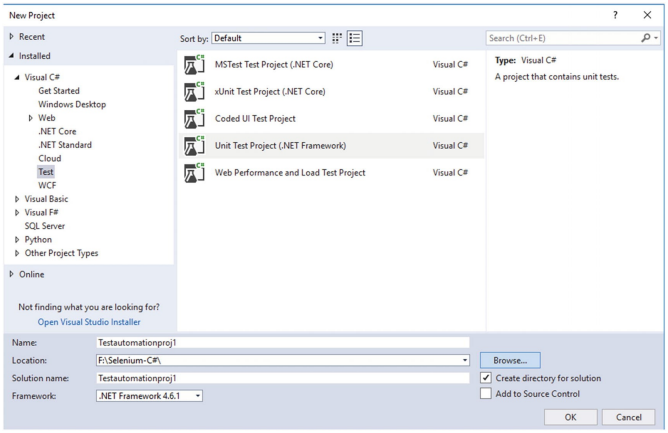

Figure 3-1 Creating a new test project

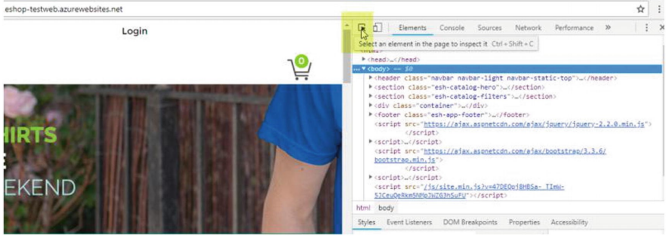

Figure 3-2 Click on the Inspector icon of Developer tools

4. Hover and click on the web element that needs to know the locator values. Figure 3-3 shows how to inspect locators of the “Login” link on the web page. You can find HTML properties of the “Login” link in the highlighted area of the DOM explorer, as shown in Figure 3-3.

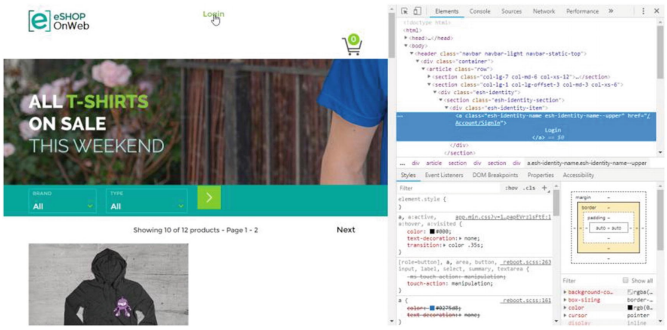

Figure 3-3 Inspect element locators of the Login link

Now we know what the Selenium locators are and how to identify locator values using the Developer Tools window. Let’s look at an explanation of each of the locators with examples.

Locator: ID

The ID locator is considered the most famous and accurate locator. Let’s try to identify element using ID and to perform action on it.

1. Open the web browser and navigate to the Account/SignIn page of the eShop web application.
2. Inspect locator of e-mail text field. Inside the highlighted area you can find the ID property of the e-mail text field. (See Figure 3-4).

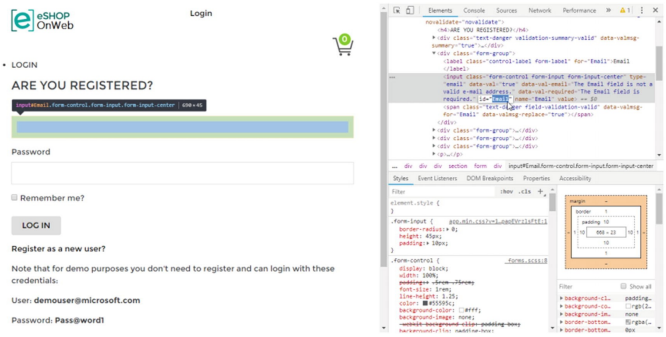

Figure 3-4 Inspect element of the e-mail field

Figure 3-5 image shows a magnified image of the highlighted area.

```
<input class="form-control form-input form-input-center"
type="email" data-val="true" data-val-email="The Email
field is not a valid e-mail address." data-val-required=
"The Email field is required." id="Email" name="Email"
value>
```

Figure 3-5 HTML properties of the e-mail field

You can see the input field has `id` locator. Let’s try to write a small test script to enter value to the e-mail field.
The following steps describe the code given below:

1. Open the Chrome browser.
2. Navigate to the login page. (Replace `http://eshop-testweb.azurewebsites.net/Account/SignIn` with the web app URL of your eShop web app.)
3. Find the e-mail field using its ID value and type an e-mail address to the field using the SendKeys method.

```
[TestMethod]
public void TestLocatorID()
{
        IWebDriver driver = new ChromeDriver();
        driver.Navigate().GoToUrl("http://eshop-testweb.azurewebsites.net/Account/SignIn");
        IWebElement email = driver.FindElement(By.Id("Email"));
        email.SendKeys("demouser@microsoft.com");

}
```

Locator: Name

This locator is also similar to `id`. You can identify name property value within HTML pages, and we can use the name attribute to locate the web elements.
You can see the web element values of the password input field highlighted in Figure 3-6.

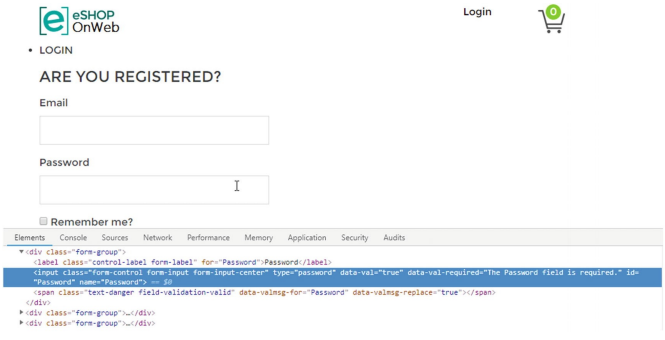

Figure 3-6 Inspect password field

Let’s understand how the Name locator works with a sample code.

1. Open the Chrome browser.
2. Navigate to the login page. (Replace `http://eshop-testweb.azurewebsites.net/Account/SignIn` with the web app URL of your eShop web app.)

3. Then enter password value by identifying input field using the Name locator.

```
[TestMethod]
Public void TestLocatorName()
{
            IWebDriver driver = new ChromeDriver();
            driver.Navigate().GoToUrl("http://eshop-testweb.azurewebsites.net/Account/SignIn");
            IWebElement password = driver.FindElement(By.Name("Password"));
            password.SendKeys("Pass@word1");
}
```

Locator: Class Name

This locator gives the element that matches the values specified in the attribute name `class`.

Figure 3-7 shows how to inspect values of shopping basket image. You can see `class` attribute in the highlighted area.

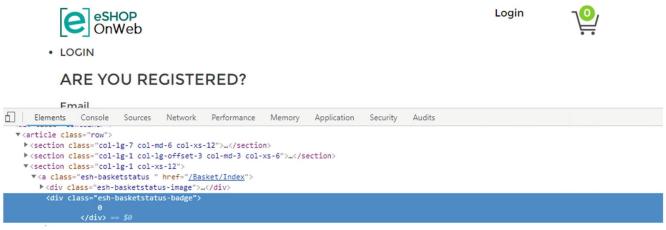

Figure 3-7 Inspect element values of the basket icon

You can see a magnified image of the highlighted area in Figure 3-8.

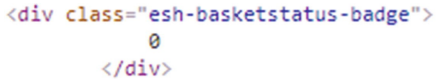

Figure 3-8 HTML element values of basket icon

The following code explains how to locate elements using the Class Name locator:

1. Open the Chrome browser.
2. Navigate to the home page of eShop web application.
3. Then click on the basket image on the home page.

```
[TestMethod]
public void TestLocatorClassName()
{
            IWebDriver driver = new ChromeDriver();
            driver.Navigate().GoToUrl("http://eshop-testweb.azurewebsites.net/");
            IWebElement basketImage = driver.FindElement(By.ClassName("esh-basketstatus-badge"));
            basketImage.Click();
}
```

Locator: CSS Selector

When an element does not have ID or name, using the CSS selector is the best alternative. CSS selectors are patterns that match against elements in the element hierarchy.

Let's try to identify different types of CSS Selectors.

Direct Child

Let's try to identify the CSS Selector of `<a>` HTML tag in Figure 3-9 using a sample code. You can see it is a direct child of a `<div>` and the `<div>` has a value defined for `class` attribute.

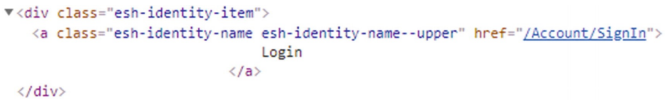

Figure 3-9 HTML code sample to identify CSS Selector: direct child concept

When we write CSS Selectors we can use different attribute values. A dot (`.`) represents `class` attribute and `#` represents an `id`. This `<div>` has the `class` attribute value. We can create a CSS selector with a dot attribute. In the CSS Selector, `>` represents a direct child in following code.

```
IWebElement loginLink = driver.FindElement(By.CssSelector(".esh-identity-item>a"));
```

Sub-Child

There are situations where we need to access sub-child of an element.

In this HTML (see Figure 3-10), you can see `<section>` tag and inside it a `<div>` tag. The anchor tag is inside the `<div>` tag. This makes `<a>` a sub-child of `<section>`. In this type of situation, you can write CSS Selector as follows. As you can see in this code that segment space is used to represent a sub-child, instead of `>`, which was used to identify a direct child with CSS Selector.

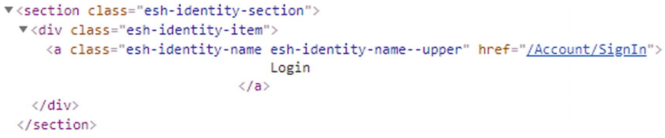

Figure 3-10 HTML code sample to identify CSS Selector: sub-child concept

```
IWebElement loginLink = driver.FindElement(By.CssSelector(".esh-identity-section a"));
```

ID

Both of these examples used class value when defining CSS Selector. With this example, you can learn how to use `id` as a CSS Selector.

This HTML code has the `<input>` tag, and it has the `id` attribute value (see Figure 3-11). We can write CSS Selector of the input field as follows:

Figure 3-11 HTML code sample to identify CSS Selector: id

```
IWebElement email = driver.FindElement(By.CssSelector("#Email"));
```

Next Sibling

We can use the next sibling concept to find the next adjacent element inside same parent. The HTML code in Figure 3-12 has three `<sections>` inside the `<article>` tag.

Figure 3-12 HTML code sample to identify CSS Selector: next sibling

If we need to locate the `<a>` tag inside the third `<select>` tag, we can write CSS Selector as follows. We can see there are three sections and we need to access the third section. So we can give value `section:nth-of-type(3)` to find the third section. The space before and after > does not matter here, as with > we are specifying to locate the child. The flowing locator says, in class row, find the child section tag that is in the third position and then locate a child `<a>` tag.

```
IWebElement basketIcon = driver.FindElement(By.CssSelector(".row > section:nth-of-type(3) > a"));
```

Attribute value

You can select elements using attribute values when `id` or `class` value is not available.

The HTML in Figure 3-13 has a `<button>` with `type` attribute. Let's try to write CSS Selector with this tag value.

Figure 3-13 HTML code sample to identify CSS Selector: Attribute

You can write CSS Selector using `type` attribute as follows:

```
IWebElement submitButton = driver.FindElement(By.CssSelector("button[type='submit']"));
```

Sub-String Matches

This allows you to match part of the string when defining CSS Selectors.

- To match prefix use `^`.
  Let's consider there are anchor tags with id value starting with `txt_ipval` and the rest of `id` is a dynamic value. You can define CSS selector as follows:

  ```
  CSS: a [id^='txt_ipval_ ']
  ```

- To match a suffix use `$`.
  Suppose there is an anchor tag whose `id` ends with `txt_endval`. You can write CSS Selector as follows:

  ```
  CSS: a[id$='_txt_endval']
  ```

- To match substring use `*`.
  Suppose there is an anchor tag with an `id` value that contains `txt_val` as part of `id`. We can write CSS Selector as follows:

  ```
  CSS: a[id*='txt_val']
  ```

Match with Inner Text

You can use the text value of element when defining CSS Selector. Let's consider there is HTML code that has an anchor tag with text "Log Out". You can write the CSS Selector as follows:

```
CSS: a: contains ('Log Out')
```

So far you have learned on different types of CSS Selectors. The following section describes how to inspect elements and write sample code using CSS Selector.

Using CSS Selectors

Let's try to find CSS Selector of the Login button of the eShop web application. After inspecting the element, you can see there are multiple classes used for the single button. In such situation we have to join classes and find accurate value. The Login button has four different classes. Let's use all four classes as CSS Selector value. See Figure 3-14.

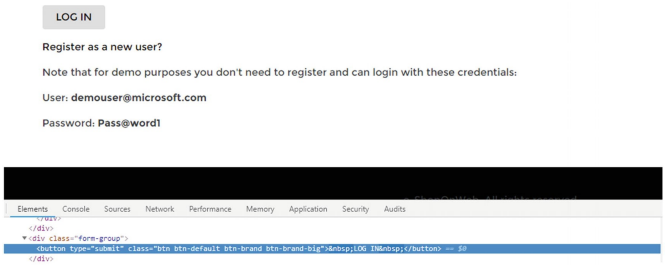

Figure 3-14 Inspect element of the Login button

You can see HTML elements of the Login button clearly in Figure 3-15.

Figure 3-15 HTML element value of the Login Button

The following code explains how to perform click action on the Login button after locating the Login button using the CSS Selector:

1. Open the Chrome browser.
2. Navigate to the Login page of the eShop web application.
3. Then click on the Login button.

```
[TestMethod]
```

```
public void TestCSSSelector()
{
            IWebDriver driver = new ChromeDriver();
            driver.Navigate().GoToUrl("http://eshop-testweb.azurewebsites.net/Account/SignIn");
            IWebElement submitButton = driver.FindElement(By.CssSelector(".btn.btn-default.btn-brand.btn-brand-big"));
            submitButton.Submit();
}
```

Locator: Link Text

You can use Link Text to select the link web elements that contain the matching text.

You can clearly see the anchor tag elements of the Login Link from Figure 3-16 magnified in Figure 3-17.

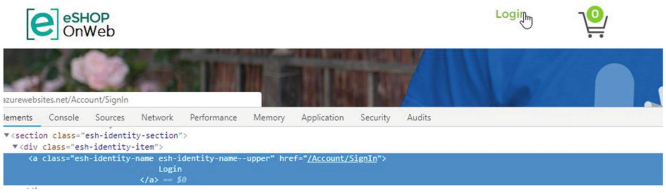

Figure 3-16 Inspect element value of the Login Link

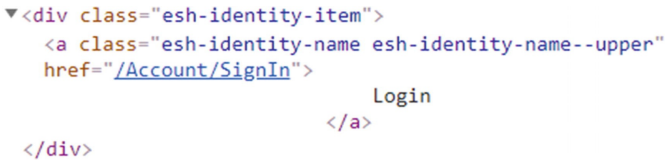

Figure 3-17 HTML element values of the Login Link

Click on the Login button by using Link Text as a finding mechanism. This is shown in the following sample code:

1. Open the Chrome browser.
2. Navigate to the home page of eShop web application.
3. Click on the Login link at the top of the page after locating it with link text “Login”.

```
[TestMethod]
public void TestLinkText()
{
            var options = new ChromeOptions();
            options.AddArgument("--start-maximized");
            IWebDriver driver = new ChromeDriver(options);
            driver.Navigate().GoToUrl("http://eshop-testweb.azurewebsites.net");
            IWebElement loginLink = driver.FindElement(By.LinkText("Login"));
            loginLink.Click();
}
```

Locator: Tag Name

This locator is used to find the web element by matching the specified Tag Name.

Figure 3-18 shows how to inspect elements of drop-down field.

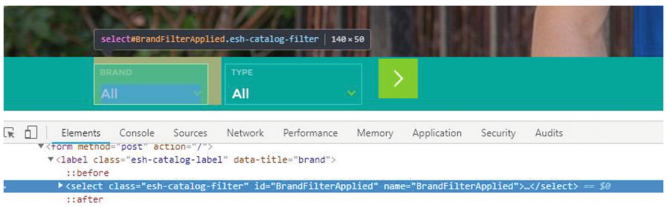

Figure 3-18 Inspect element values of brands drop-down field

You can identify element values of drop-down field clearly (see Figure 3-19).

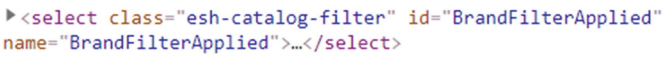

Figure 3-19 HTML element value of” brand” drop-down field

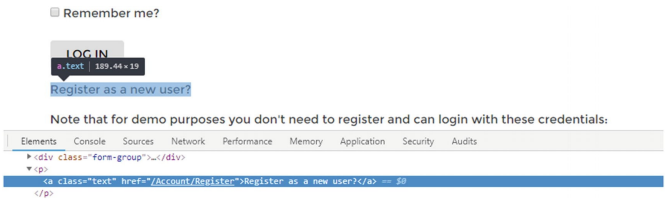

Figure 3-20 Inspect element values of the “Register as new user” Link

The drop-down field has the `select` tag. In the following sample code, you can see the drop-down field is identified using the tag value, and a click action on it is performed.

1. Open the Chrome browser.
2. Navigate to the home page of the eShop web application.
3. Click on the first drop-down of the page. If there is more than one `select` tag on the page, the following code Clicks on the first `select` tag element.

```
[TestMethod]
public void TestTagName()
        {
            var options = new ChromeOptions();
            options.AddArgument("--start-maximized");
            IWebDriver driver = new ChromeDriver(options);
            driver.Navigate().GoToUrl("http://eshop-testweb.azurewebsites.net");
            IWebElement tagElement = driver.FindElement(By.TagName("select"));
            tagElement.Click();
}
```

Locator: Partial Link Text

Web element can be found using part of the link text. This allows you to identify an element that contains text that matches the specified partial link text.

Inspect element values of the “Register as new user” Link. (see Figure 3-21).

```
<a class="text" href="/Account/Register">Register as a new user?</a>
```

Figure 3-21 HTML element of” Register as new user” Link

The following sample code shows how to find element using the partial link text locator and perform click action on it.

1. Open the Chrome browser.
2. Navigate to the Login page of the eShop web application.
3. Click on the “Register as new user” link after finding it with the partial link text.

```
[TestMethod]
public void TestPartialLinkText()
{
            var options = new ChromeOptions();
            options.AddArgument("--start-maximized");
            IWebDriver driver = new ChromeDriver(options);
            driver.Navigate().GoToUrl("http://eshop-testweb.azurewebsites.net/Account/SignIn");
            IWebElement registerNewUser = driver.FindElement(By.PartialLinkText("Register as a"));
            registerNewUser.Click();
}
```

Locator: XPath

There are two types of XPaths: Relative XPath and absolute XPath.

Absolute XPath

- It is a direct way to locate and element.
- It starts from the root.
- It starts with a single slash (/).

```
/html/body/div/div[2]/div[3]/form/input[1]
```

Relative XPath

- It starts with a double slash (//)
- It starts a search from anywhere in the HTML DOM.

```
//*[@id="logoutForm"]/section[2]/a[2]/div
```

XPath Example

Figure 3-22 shows how you can find XPath easily with chrome driver.

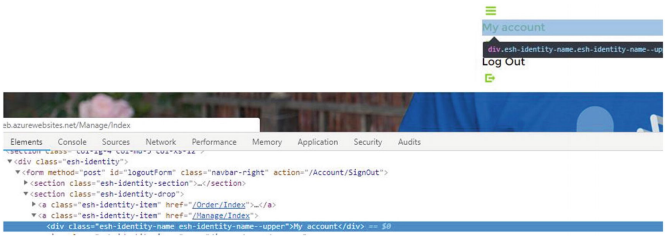

Figure 3-22 Inspect element value of” My account” link

1. First inspect the element.
2. Then right-click on the highlighted area of DOM and select copy ➤ Copy XPath.
3. When you perform paste on the notepad you can see XPath of the element and can use it to locate an element.

Let’s try to find the relative XPath of “My account” of the following HTML DOM (see Figure 3-23). logoutForm is a unique id in this page. We can start searching from this unique ID and provide the entire path up to div. In the following HTML code, you can see there are two sections and we need to access the second section. Inside that section, you can see two anchor tags, and we need to access the second one. Inside that <a> tag you can see , which we need to access. So we can write XPath as follows:

```
▼<form method="post" id="logoutForm" class="navbar-right" action="/Account/SignOut">
  ▶<section class="esh-identity-section">…</section>
  ▼<section class="esh-identity-drop">
    ▶<a class="esh-identity-item" href="/Order/Index">…</a>
    ▼<a class="esh-identity-item" href="/Manage/Index">
       <div class="esh-identity-name esh-identity-name--upper">My account</div>
```

Figure 3-23 HTML code of logout form

```
//[@id='Logoutform']/section[2]/a[2]/div
```

We identified the relative XPath of , and you can try to perform an action of the identified element as shown in following sample code:

1. Open the Chrome browser.
2. Navigate to the home page of the eShop web application.
3. Locate and click on the Login link and navigate to the login page.
4. Enter user e-mail and password.
5. Click on the submit button.
6. Click on the My account Link located using XPath.

```
[TestMethod]
public void TestXPath()
{
            var options = new ChromeOptions();
            options.AddArgument("--start-maximized");
            IWebDriver driver = new ChromeDriver(options);
            driver.Navigate().GoToUrl("http://eshop-testweb.azurewebsites.net");
            IWebElement loginLink = driver.FindElement(By.LinkText("Login"));
            loginLink.Click();
            IWebElement email = driver.FindElement(By.Id("Email"));
            email.SendKeys("demouser@microsoft.com");
            IWebElement password = driver.FindElement(By.Name("Password"));
            password.SendKeys("Pass@word1");
            IWebElement submitButton = driver.FindElement(By.CssSelector(".btn.btn-default.btn-brand.btn-brand-big"));
            submitButton.Submit();
            IWebElement myAccount = driver.FindElement(By.XPath("//*[@id='logoutForm']/section[2]/a[2]/div"));
            myAccount.Click();
}
```

We discussed all eight locators in detail within this lesson, and by now you should have a good understanding about web element locators and how to use them.

Lesson 3.03: Web Elements Commands

While we use web applications, we perform different actions on web elements. So, when we do test automation, we have to automate every action a user performs on web elements. In the previous lesson, we talked about web element locators. Let’s talk about web element commands within this lesson.

We will be using the eShop application and (https://jqueryui.com/droppable) to demonstrate commands in this lesson. First, let’s explore built-in web element commands.

Command: Click

Clicking is the most common way to interact with elements. Click action can perform on many elements such as buttons, links, check boxes, and many more.

```
Click (): void
```

accepts nothing as parameters and returns nothing.

```
IWebElement loginLink = driver.FindElement(By.LinkText("Login"));
loginLink.Click();
```

Command: Send Keys

`SendKeys` command is used to enter text to input fields and to text areas.

```
SendKeys(string text):void
```

accepts text values as parameter and returns nothing. Here we input an e-mail address to text box using the `SendKeys` method:

```
IWebElement email = driver.FindElement(By.Id("Email"));
email.SendKeys("demouser@microsoft.com");
```

Command: Clear

Clears the content of a given element. Can be used on text elements such as text boxes and text areas.

```
Clear (): void
```

accepts nothing as parameters and returns nothing.

```
IWebElement email = driver.FindElement(By.Id("Email"));
email.Clear();
```

Command: Find Element

Find the first element with the given web element locator:

```
FindElement(By by) : IWebElement
```

accepts the locating mechanism as parameter and returns first found `IWebElement` in the current context.
Here we find the element that has `id = "Email"`.

```
IWebElement email = driver.FindElement(By.Id("Email"));
```

Command: Find Elements

Find all the `IWebElements` on the current context with the given locator.

```
FindElements(By by) : <IWebElement>
```

accepts locating mechanism as parameter and returns all `IWebElements` matching the current criteria or returns an empty list if nothing matches.
Here we get the list of elements matching the given values:

```
IList<IWebElement> ProductList = driver.FindElements(By.CssSelector(".esh-catalog-item.col-md-4"));
```

Command: GetAttribute

Get the values of the specified attribute of the element.

```
GetAttribute(string attributename) : string
```

accepts attribute name as parameter and returns attribute values or returns null if value is not set.
The HTML code in Figure 3-24 has an input field that has the attribute called `type`. If we need to read the value of the attribute from the code, we can use the `GetAttribute` method.

```
<div class="form-group">
    <label class="control-label form-label" for="Email">Email
    </label>
    <input class="form-control form-input form-input-center"
    type="email" data-val="true" data-val-email="The Email
    field is not a valid e-mail address." data-val-required=
    "The Email field is required." id="Email" name="Email"
    value>
    <span class="text-danger field-validation-valid" data-
    valmsg-for="Email" data-valmsg-replace="true"></span>
</div>
```

Figure 3-24 HTML code sample

The following sample code shows how to read attribute value using `GetAttribute` method:

```
[TestMethod]
public void TestGetAttribute()
{
            var options = new ChromeOptions();
            options.AddArgument("--start-maximized");
            IWebDriver driver = new ChromeDriver(options);
            driver.Navigate().GoToUrl("http://eshop-testweb.azurewebsites.net/Account/SignIn");
            IWebElement email = driver.FindElement(By.Id("Email"));
            var value = email.GetAttribute("type");
            Console.WriteLine(value);
}
```

You can check the output of test case to see attribute value.
To check test case output, select test case from the Test Explorer and click on the Output link. Then test output will be shown (see Figure 3-25).

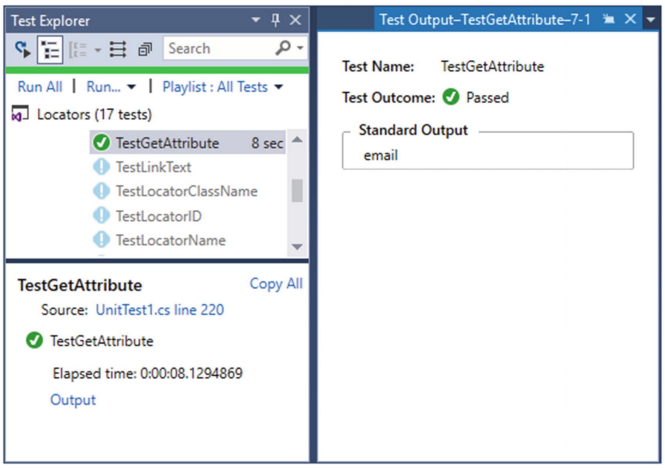

Figure 3-25 Test Output

Command: GetCssValue

Get the value of CSS property of the current element.

```
GetCssValue(string CSSPropertyName):string
```

accepts name of the CSS property as a parameter and returns the value of the specified CSS property.
The following sample code describes how to get background color of the specified element. You can go to test outputs to check the value of the property.

```
[TestMethod]
public void TestGetCssValue()
{
            var options = new ChromeOptions();
            options.AddArgument("--start-maximized");
            IWebDriver driver = new ChromeDriver(options);
            driver.Navigate().GoToUrl("http://eshop-testweb.azurewebsites.net/Account/SignIn");
            IWebElement email = driver.FindElement(By.Id("Email"));
            var value = email.GetCssValue("background-color");
            Console.WriteLine(value);
}
```

Command: Submit

Submit a web page or form or part of web page back to the server.

```
Submit (): void
```

accepts nothing as parameter and returns nothing.
You can see in Figure 3-26 that the HTML form has button with `type = "submit"`. For a button with `type= "submit"` you can use the `Submit` method.

```
<div class="form-group">
    <button type="submit" class="btn btn-default btn-brand
    btn-brand-big"> LOG IN </button>
</div>
```

Figure 3-26 HTML code with submit button

You can use the submit method as follows to submit the form:

```
IWebElement submitButton = driver.FindElement(By.CssSelector(".btn.btn-default.btn-brand.btn-brand-big"));submitButton.Submit();
```

Action Commands on IWebElements

Advance user interaction API contains an `Actions` class to support Webdriver in handling keyboard and mouse events such as drag and drop, clicking multiple elements with the control key, etc. Let's explore few Action commands available with Selenium Webdriver in the following sections.

Command: ClickAndHold

It will click-and-hold at the location of the source element and then move to the location of the target element.
You can see in Figure 3-27 that the web page has `<div>` that needs to drag and drop into another `<div>`.

```
<iframe src="/resources/demos/droppable/default.html" class=
"demo-frame">
  #document
    <!doctype html>
    <html lang="en">
      <head>…</head>
      <body>
        <div id="draggable" class="ui-widget-content ui-
        draggable ui-draggable-handle" style="position:
        relative;">…</div>
        <div id="droppable" class="ui-widget-header ui-
        droppable">…</div>
      </body>
    </html>
</iframe>
```

Figure 3-27 HTML code sample of drag element and drop area

This HTML code sample has a drag element ID and drop area ID inside an iframe. We cannot locate elements inside the iframe from outside the iframe. So before identifying element locators, we need to move into the iframe first.
You can switch over the elements in iframes in one of three ways:

- By Index
- By Name or Id
- By Web Element

The following code shows how to switch to iframe using index of the iframe:

1. Open the Chrome browser.
2. Navigate to the web page (`https://jqueryui.com/droppable`)

3. Switch to iframe using index.
4. Click and hold, then drag the element.
5. Move to the drop area and drop the item.

```
[TestMethod]
public void TestClickAndHold()
{
            IWebDriver driver = new ChromeDriver();
            driver.Navigate().GoToUrl("https://jqueryui.com/droppable/");
            driver.SwitchTo().Frame(0);
            IWebElement drag = driver.FindElement(By.Id("draggable"));
            IWebElement drop = driver.FindElement(By.Id("droppable"));
            Actions action = new Actions(driver);
            action.ClickAndHold(drag).Build().Perform();
            action.MoveToElement(drop).Build().Perform();
}
```

Command: DragAndDrop

This command will drag the element from one location and drop to a given location.

You can see sample code for drag and drop here:

1. Open the Chrome browser.
2. Navigate to the web page (`https://jqueryui.com/droppable`).
3. Switch to iframe using index.
4. Drag element.
5. Drop element to given area.

```
[TestMethod]
public void TestDragAndDrop()
{
            IWebDriver driver = new ChromeDriver();
            driver.Navigate().GoToUrl("https://jqueryui.com/droppable/");
            driver.SwitchTo().Frame(0);
            IWebElement drag = driver.FindElement(By.Id("draggable"));
            IWebElement drop = driver.FindElement(By.Id("droppable"));
            Actions action = new Actions(driver);
            action.DragAndDrop(drag, drop).Build().Perform();
}
```

Command: DragAndDropToOffSet

Drag element from one location to a given offset. In the following example, the draggable element will move to point 100,100 in the web page body.

You can see sample code for drag and drop for offset as follows:

1. Open the Chrome browser.
2. Navigate to the web page (`https://jqueryui.com/droppable`).
3. Switch to iframe using index.
4. Drag element.
5. Drop element to given offset.

```
[TestMethod]
public void TestDragAndDropOffset()
{
            IWebDriver driver = new ChromeDriver();
            driver.Navigate().GoToUrl("https://jqueryui.com/droppable/");
            driver.SwitchTo().Frame(0);
            IWebElement drag = driver.FindElement(By.Id("draggable"));
            Actions action = new Actions(driver);
            action.DragAndDropToOffset(drag, 100, 100).Build().Perform();
}
```

Lesson 3.04: Handling Web Driver Waits

We know each web application page has its own page loading time. Sometimes this time varies due to different reasons, including the following:

- Database calls made by website
- Slow browsers
- Slow internet connection

Because of different page loading times, we need a way to wait until elements are available and can be located in DOM to detect them in the test automation code. Selenium has different types of wait handling mechanisms. Let's discuss these wait handling methods in this lesson.

First let's try to understand what would happen if the code is not handling waits properly. See the following sample code:

1. Open the Chrome browser.
2. Navigate to the Google search page (`https://www.google.com/`).
3. Type text "Selenium" in search fields.
4. Select list item 2 from the list.

```
[TestMethod]
public void GoogleSearchWithoutWait()
{
            IWebDriver driver = new ChromeDriver();
            driver.Navigate().GoToUrl("https://www.google.com");
            IWebElement searchField = driver.FindElement(By.Id("lst-ib"));
            searchField.SendKeys("Selenium");
            IWebElement listValue = driver.FindElement(By.XPath("//div[@class='sbsb_a']//ul/li[2]"));
            listValue.Click();
}
```

After executing this code, you will see the browser opens; type text "Selenium" in the Google search field. Then the test fails with `NoSuchElementException` . This error is either due to locator in the script being incorrect or the specified element not being loaded by the time the code tries to detect it. See Figure 3-28.

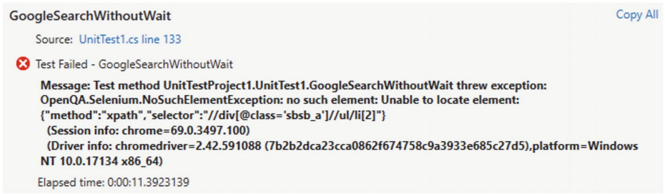

Figure 3-28 Test fail message

Let's add wait time and try to execute the same test steps.

The Selenium web driver provides two types of waits.

Implicit Wait

Implicit wait tells the web driver to wait a certain amount of time in each find element step across the script, until the element is found. If element is found within the specified implicit wait time, then execution continues. Otherwise the execution fails with a timeout. Implicit wait is easy and simple to apply.

The following code shows sample code with implicit wait:

```
[TestMethod]
public void GoogleSearchImplicitWait()
{
            IWebDriver driver = new ChromeDriver();
            driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(30);
            driver.Navigate().GoToUrl("https://www.google.com");
            IWebElement searchField = driver.FindElement(By.Id("lst-ib"));
            searchField.SendKeys("Selenium");
            IWebElement listValue = driver.FindElement(By.XPath("//div[@class='sbsb_a']//ul/li[2]"));
            listValue.Click();
}
```

You can see this sample code has a 30-second implicit wait time. After executing this code, you can see the test passes.

Note You may have seen automation scripts that have used Thread.sleep() waits. But this is not a good practice for high-quality automation script, because Thread.sleep freezes test execution for specific amount of time. The code will freeze for a specified time duration no matter whether elements are loaded or not in the DOM. This unnecessary wait time will increase the overall test execution time and reduce the effectiveness of test automation usage.

Let's try to write code to perform the same Google search steps with thread sleep using 5000 milliseconds of sleep time.

```
[TestMethod]
public void GoogleSearchWithThreadWait()
{
            IWebDriver driver = new ChromeDriver();
            driver.Navigate().GoToUrl("https://www.google.com");
            IWebElement searchField = driver.FindElement(By.Id("lst-ib"));
            searchField.SendKeys("Selenium");
            Thread.Sleep(5000);
            IWebElement listValue = driver.FindElement(By.XPath("//div[@class='sbsb_a']//ul/li[2]"));
            listValue.Click();
}
```

Explicit Wait

Explicit wait demands waiting for certain conditions to occur before proceeding further in the code, with a specified timeout. If the finding element does not meet the specified condition within a given time period, it throws a `TimeOutException`.

Let's modify the Google search code with explicit wait and try to execute the code.

```
[TestMethod]
public void GoogleSearchWithWait()
{
            IWebDriver driver = new ChromeDriver();
            WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30));
            driver.Navigate().GoToUrl("https://www.google.com");
            IWebElement searchField = driver.FindElement(By.Id("lst-ib"));
            searchField.SendKeys("Selenium");
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.ElementToBeClickable(By.XPath("//div[@class='sbsb_a']//ul/li[2]")));
            IWebElement listValue = driver.FindElement(By.XPath("//div[@class='sbsb_a']//ul/li[2]"));
            listValue.Click();
}
```

You can see the execution time difference of each method in Figure 3-29. You can see how `Thread.Sleep` of 5 seconds affects the execution time when comparing the time without using thread sleep. You can imagine how much it will slow down the test execution by using many thread sleep statements just by looking at the effect of using one.

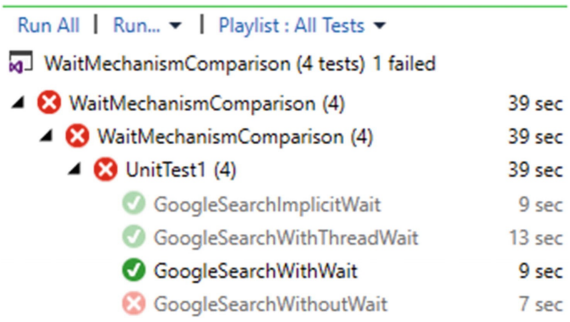

Figure 3-29 Test execution time comparison

Let's try to identify a few available conditions of Selenium to be used with explicit waits in the next section of this lesson.

ElementExists

`ElementExists` is an expectation for checking that the element is present in the DOM of a page. This does not mean that the element is visible.

```
Parameters: The locator
Returns : IWebElement once it is located
```

Here is an example:

```
IWebElement loginIcon = wait.Until(ExpectedConditions.ElementExists(By.LinkText("Login")));
```

ElementIsVisible

`ElementIsVisible` is an expectation for checking that element is present in the DOM of a page and is visible. The element is also displayed and has height and width that is greater than 0.

- Parameters: The locator
- Returns: `IWebElement` once it is located and visible

Here is an example:

```
IWebElement loginLink= wait.Until(ExpectedConditions.ElementIsVisible(By.LinkText("Login")));
```

ElementToBeClickable

This is an expectation for checking an element is visible and enabled such that you can click it.

- Parameters: The element
- Returns: IWebElement once it is clickable

Now an example:

```
IWebElement emailField= wait.Until(ExpectedConditions.ElementToBeClickable(By.Id("Email")));
```

InvisibilityOfElementLocated

This is an expectation for checking that an element is either invisible or not present/available in the DOM.

- Parameters: The locator
- Returns: True if the element is not displayed; otherwise, false.

Suppose after clicking on a save button, you need to wait until the busy indicator disappears to indicate save operation is done. In such situations, you can use this expectation, `InvisibilityOfElementLocated`.

InvisibilityOfElementWithText

`InvisibilityOfElementWithText` is an expectation for checking that an element with a given text is either invisible or not present/available in the DOM.

- Parameters: The locator, Text of the element.
- Returns: True if the element is not displayed; otherwise, false.

TextToBePresentInElement

This is an expectation for checking if the given text is present in the specified element.

- Parameters: The WebElement, Text to be present in the element.
- Returns: True once the element contains the given text; otherwise, false.

Here is an example:

```
Assert.AreEqual(true, wait.Until(ExpectedConditions.TextToBePresentInElement(loginButton, "LOG IN")));
```

TextToBePresentInElementLocated

An expectation for checking if the given text is present in the element that matches the given locator.

- Parameters: The Locator, Text to be present in the element.
- Returns: True once the element contains the given text; otherwise, false.

Here is an example:

```
Assert.AreEqual(true, wait.Until(ExpectedConditions.TextToBePresentInElementLocated(By.CssSelector(".btn.btn-default.btn-brand.btn-brand-big"),
"LOG IN")));
```

TextToBePresentInElementValue

This is an expectation for checking if the given text is present in the specified element's `value` attribute .

- Parameters: The locator, Text to be present in the element.
- Returns: True once the element contains the given text; otherwise, false.

Here is an example:

```
Assert.AreEqual(true, wait.Until(ExpectedConditions.TextToBePresentInElementValue(By.Id("Email"), "demouser@microsoft.com")));
```

TitleContains

An expectation for checking that the title of a page contains a case-sensitive substring.

- Parameters: The fragment of title expected.
- Returns: True when the title matches; otherwise, false.

Here is an example:

```
Assert.AreEqual(true, wait.Until(ExpectedConditions.TitleContains("Log in")));
```

TitleIs

An expectation for checking the title of a page.

- Parameters: The expected title, which must be an exact match.
- Returns: true when the title matches; otherwise, false.

Here is an example:

```
Assert.AreEqual(true,wait.Until(ExpectedConditions.TitleIs("Catalog - Microsoft.eShopOnWeb")));
```

UrlContains

An expectation for the URL of the current page to contain the specified text (portion of the URL).

- Parameters: The fraction of the URL that the page should be on.
- Returns: True when the URL contains the text; otherwise, false.

Here is an example:

```
Assert.AreEqual(true, wait.Until(ExpectedConditions.UrlContains("http://eshop-testweb")));
```

UrlToBe

An expectation for the URL to be a specified exact URL including any arguments.

- Parameters: The URL that the page should be on.
- Return: True when the URL is what it should be; otherwise, false.

Here is an example:

```
Assert.AreEqual(true, wait.Until(ExpectedConditions.UrlToBe("http://eshop-testweb.azurewebsites.net/")));
```

Using Expected Conditions

Now you know expected conditions available in Selenium. The following sample code uses the earlier described expected conditions and waits for these conditions to occur during the test execution. The following test method can be used as an example for the use of expected conditions:

```
[TestMethod]
public void TestMethod1()
{
            ChromeOptions option = new ChromeOptions();
            option.AddArgument("--start-maximized");
            IWebDriver driver = new ChromeDriver();
            driver.Navigate().GoToUrl("http://eshop-testweb.azurewebsites.net/");
            WebDriverWait wait = new WebDriverWait(driver,TimeSpan.FromSeconds(30));
            Assert.AreEqual(true,wait.Until(ExpectedConditions.TitleIs("Catalog - Microsoft.eShopOnWeb")));
            IWebElement loginLink= wait.Until(ExpectedConditions.ElementIsVisible(By.LinkText("Login")));
            loginLink.Click();
            Assert.AreEqual(true, wait.Until(ExpectedConditions.TitleContains("Log in")));
            Assert.AreEqual(true,wait.Until(ExpectedConditions.UrlToBe("http://eshop-testweb.azurewebsites.net/Account/SignIn")));
            IWebElement emailField= wait.Until(ExpectedConditions.ElementToBeClickable(By.Id("Email")));
            emailField.SendKeys("demouser@microsoft.com");
            Assert.AreEqual(true, wait.Until(ExpectedConditions.TextToBePresentInElementValue(By.Id("Email"), "demouser@microsoft.com")));
            IWebElement passwordField = wait.Until(ExpectedConditions.ElementExists(By.Id("Password")));
            passwordField.SendKeys("Pass@word1");
            IWebElement loginButton = wait.Until(ExpectedConditions.ElementToBeClickable(By.CssSelector(".btn.btn-default.btn-brand.btn-brand-
big")));
            Assert.AreEqual(true, wait.Until(ExpectedConditions.TextToBePresentInElementLocated(By.CssSelector(".btn.btn-default.btn-brand.btn-
brand-big"), "LOG IN")));
            loginButton.Submit();
            Assert.AreEqual(true, wait.Until(ExpectedConditions.UrlContains("http://eshop-testweb")));
            Assert.AreEqual(true, wait.Until(ExpectedConditions.UrlToBe("http://eshop-testweb.azurewebsites.net/")));
}
```

You learned different type of waits and effect of each mechanism on overall automation test performance. Also, you explored different types of wait conditions available in Selenium.

Lesson 3.05: C# Automation Code Example

Now you have good understanding on Locators, Selenium Webdriver commands, and Selenium wait handling. Let's try to write a simple code with Selenium and C#. Following is the scenario we are going to test in this lesson using Selenium and C#.

eShop web application visitors can register on an online web site by giving an e-mail and password and confirming the password. They then click on my account link to navigate to manage user account page. The user can see added user info on that page.

1. First you need to add the required references to the test class.

```
using System;using Microsoft.VisualStudio.TestTools.UnitTesting;using OpenQA.Selenium;using OpenQA.Selenium.Chrome;using OpenQA.Selenium.Support.U
ExpectedConditions = SeleniumExtras.WaitHelpers.ExpectedConditions;
```

2. The following code part is used to launch the Chrome driver and navigate to the eShop web application home page. Then it defines 30 seconds as wait time.

```
string userEmail = "xyz@gmail.com";ChromeOptions option = new ChromeOptions();option.AddArgument("--start-maximized");IWebDriver driver = new
ChromeDriver(option);driver.Navigate().GoToUrl("http://eshop-testweb.azurewebsites.net/");
WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30));
```

3. Next click on the Login link on the home page and make sure it waits a maximum of 30 seconds to locate the Login link. In the event that the Login link does not become visible within 30 seconds, code w with a timeout exception.

```
IWebElement loginLink = wait.Until(ExpectedConditions.ElementIsVisible(By.LinkText("Login")));
loginLink.Click();
```

4. Then click on the “Register New user” link to navigate to the user registration page.

```
IWebElement registerNewUser = wait.Until(ExpectedConditions.ElementToBeClickable(By.LinkText("Register as a new
user?")));            registerNewUser.Click();
```

5. Enter the user e-mail and password and confirm password values. Click on REGISTER to register new user.

```
IWebElement emailField = wait.Until(ExpectedConditions.ElementToBeClickable(By.Id("Email")));
emailField.SendKeys(userEmail);
IWebElement passWordField = wait.Until(ExpectedConditions.ElementToBeClickable(By.Id("Password")));            passWordField.SendKeys("Abc@123");
IWebElement confirmPassWordField =
wait.Until(ExpectedConditions.ElementToBeClickable(By.Id("ConfirmPassword")));            confirmPassWordField.SendKeys("Abc@123");            IWe
registerButton = wait.Until(ExpectedConditions.ElementToBeClickable(By.CssSelector(".btn.btn-default.btn-brand.btn-brand-big")));
            registerButton.Submit();
```

6. After successfully registering, a user can move to their user account by clicking on the “my account” link.

```
IWebElement myAccountLink = wait.Until(ExpectedConditions.ElementToBeClickable(By.LinkText("My account")));
myAccountLink.Click();
```

When clicking on the “my account” link, you should be moved to the Manage user account page. You can verify the user has moved to the user management page successfully by checking that the e-ma user is available on that page.

```
Assert.AreEqual(true, wait.Until(ExpectedConditions.TextToBePresentInElementValue(By.Id("Email"), userEmail)));
```

You can find sample code as follows:

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.Support.UI;
using ExpectedConditions = SeleniumExtras.WaitHelpers.ExpectedConditions;

namespace UnitTestProject1
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void AddNewUser()
        {
            string userEmail = "pbc@gmail.com";
            ChromeOptions option = new ChromeOptions();
            option.AddArgument("--start-maximized");
            IWebDriver driver = new ChromeDriver(option);
            driver.Navigate().GoToUrl("http://eshop-testweb.azurewebsites.net/");
            WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30));
            IWebElement loginLink = wait.Until(ExpectedConditions.ElementIsVisible(By.LinkText("Login")));
            loginLink.Click();
            IWebElement registerNewUser = wait.Until(ExpectedConditions.ElementToBeClickable(By.LinkText("Register as a new user?")));
            registerNewUser.Click();
            IWebElement emailField = wait.Until(ExpectedConditions.ElementToBeClickable(By.Id("Email")));
            emailField.SendKeys(userEmail);
            IWebElement passWordField = wait.Until(ExpectedConditions.ElementToBeClickable(By.Id("Password")));
            passWordField.SendKeys("Abc@123");
            IWebElement confirmPassWordField = wait.Until(ExpectedConditions.ElementToBeClickable(By.Id("ConfirmPassword")));
            confirmPassWordField.SendKeys("Abc@123");
            IWebElement registerButton = wait.Until(ExpectedConditions.ElementToBeClickable(By.CssSelector(".btn.btn-default.btn-brand.btn-
brand-big")));
            registerButton.Submit();
            IWebElement myAccountLink = wait.Until(ExpectedConditions.ElementToBeClickable(By.LinkText("My account")));
            myAccountLink.Click();
            Assert.AreEqual(true, wait.Until(ExpectedConditions.TextToBePresentInElementValue(By.Id("Email"), userEmail)));
        }
    }
}
```

You learned how to use locators, commands, and wait conditions when automating a test scenario using Selenium with C#.

Lesson 3.06: Python Automation Code Example

You tried sample code with Selenium and C# in the previous lesson. Let’s try to write sample code with Selenium and Python.

Let’s write a test case to test following scenario.

As a registered user of an online shopping site, I want to login to the site using my user name and password and then add a product to the basket. Then I should be able to view a list of products in the basket.

You have learned how to set up Visual Studio to work with Python and create a project in a previous the second chapter. Keep in mind that Python code heavily relies on indentation to identify scopes, unlike C#, where `{` and `}` are used for the same purpose.

1. First create a Python application project and add Python unit test file to project.
2. Open unit test file and add the following code lines just below the `import unittest` statement:

```
from selenium import webdriverfrom selenium.webdriver.support.wait
import WebDriverWaitfrom selenium.webdriver.common.by import Byfrom selenium.webdriver.support
import expected_conditions as ExpectedConditionfrom selenium.webdriver.support.select
import Select
```

3. The following code launches the Chrome browser and maximizes window. It then navigates to the eShop web application. The wait time is defined as 30.

```
driver=webdriver.Chrome()
driver.maximize_window()
driver.get('http://eshop-testweb.azurewebsites.net')        wait=WebDriverWait(driver,30);
```

4. The following line is used to verify the web page has navigated to the application home page correctly by checking the page title.

```
self.assertEqual(True,wait.until(ExpectedCondition.title_is("Catalog - Microsoft.eShopOnWeb")))
```

5. After navigating to the home page, click on the Login link.

```
LoginLink=wait.until(ExpectedCondition.element_to_be_clickable((By.LINK_TEXT,"Login")))
LoginLink.click()
```

6. Type in user e-mail and password to login to the website.

```
emailField=wait.until(ExpectedCondition.element_to_be_clickable((By.ID,"Email")))        emailField.send_keys("demouser@microsoft.com")        pas
default.btn-brand.btn-brand-big")))
submitButton.submit()
```

After logging in, navigate to the home page. Then select brand from the drop-down and select the first item from the search results.

```
selectListItem=Select(driver.find_element_by_id(("BrandFilterApplied")))selectListItem.select_by_visible_text((".NET"))driver.find_element_by_cs
s_selector((".esh-catalog-items.row > div:nth-of-type(1) > form > input")).click()
```

Then verify the browser has navigated to the page where you can view the list of products in the basket.

```
self.assertEqual(True,wait.until(ExpectedCondition.url_to_be("http://eshop-testweb.azurewebsites.net/Basket/Index")))
```

You can find the complete code as follows:

```
import unittest
from selenium import webdriver
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as ExpectedCondition
from selenium.webdriver.support.select import Select
from selenium.webdriver.common.keys import Keys

class Test_TestCases(unittest.TestCase):
    def test_A(self):
        driver=webdriver.Chrome()
        driver.maximize_window()
        driver.get('http://eshop-testweb.azurewebsites.net')
        wait=WebDriverWait(driver,30);
        self.assertEqual(True,wait.until(ExpectedCondition.title_is("Catalog - Microsoft.eShopOnWeb")))
        LoginLink=wait.until(ExpectedCondition.element_to_be_clickable((By.LINK_TEXT,"Login")))
        LoginLink.click()
        emailField=wait.until(ExpectedCondition.element_to_be_clickable((By.ID,"Email")))
        emailField.send_keys("demouser@microsoft.com")
        password=wait.until(ExpectedCondition.visibility_of_element_located((By.ID,"Password")))
        password.send_keys("Pass@word1")
        submitButton=wait.until(ExpectedCondition.element_to_be_clickable((By.CSS_SELECTOR,".btn.btn-default.btn-brand.btn-brand-big")))
        submitButton.submit()
        selectListItem=Select(driver.find_element_by_id(("BrandFilterApplied")))
        selectListItem.select_by_visible_text((".NET"))
        driver.find_element_by_css_selector((".esh-catalog-items.row > div:nth-of-type(1) > form > input")).click()
        self.assertEqual(True,wait.until(ExpectedCondition.url_to_be("http://eshop-testweb.azurewebsites.net/Basket/Index")))

if __name__ == '__main__':
    unittest.main()
```

You were able to get an idea of how to work with different locators, commands, and wait handling mechanisms while working with Selenium and Python code in this lesson.

Lesson 3.07: MAQS Framework with C# Automation Code Example

You learned how to set up Visual Studio to work with MAQS in Chapter 2. As you already know, one of the main advantages of the MAQS framework is we do not need to spend time performing initial set up of package installation and configurations. After creating a project with MAQS, you may have to change configuration values using `App.config` file under the Test project according to your project requirements.

Page Object Model is used in the project created with MAQS. Page Object Model is an object design pattern where web pages are represented as classes. The elements of the page are represented as variables in the class.

Let's try to write automation script to test the following scenario. Complete code can be found at https://github.com/chamindac/Book-Test-Autmation-VS/tree/master/Chapter%203/Lesson%203.07/MAQS .

Scenario: First navigate to the home page of the eShop web application and click on the Login link on the home page to go to the login page. Enter user e-mail and password to login to the user account. After logging in to the account you will find the My orders link. Click on the My orders link and verify the page has navigated to the My orders page.

1. Let's start by changing configuration values. Open `App.config` file and change `WebSiteBase` value to URL of the eShop web application you have hosted in IIS or in Azure. Also, you can see the browser configuration area where you can select different browsers. By default, the Chrome browser is selected. All our lessons are presented with the Chrome browser. So, we can keep the default value as it is. See Figure 3-30.

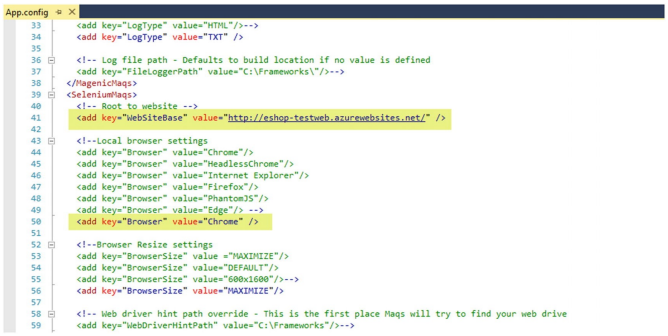

Figure 3-30 App.config file

We have to create class files for each page mentioned in this scenario. We can create the following classes under `PageModel` folder.

- `HomePage.cs`
- `UserLoginPage.cs`
- `MyOrders.cs`

2. To add new class files to the `PageModel` folder, right-click on the `PageModel` folder in the Solution Explorer and select Add ➤ New Item.
3. The New Item pop-up will be opened. Select Magenic's Open Test, then select Maqs Selenium Page Model Class. Give the name as `HomePage.cs` and click on the Add button. See Figure 3-31.

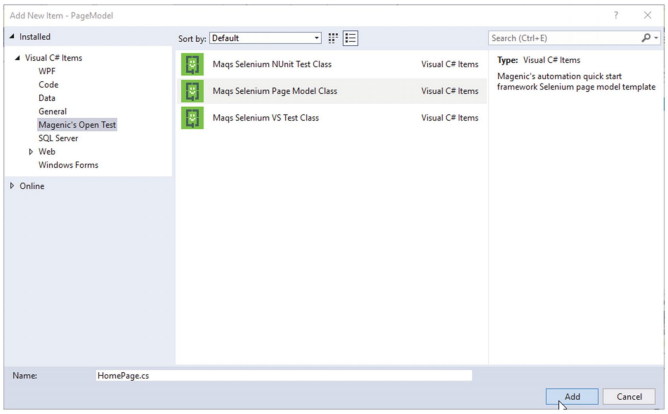

Figure 3-31 Add MAQS Selenium Model Class

Follow the same steps to add UserLoginPage.cs and MyOrders.cs.
After going through the following section, you will be able to understand methods in each class and use of them.

HomePage.cs

Below are the methods in the `HomePage.cs` with an explanation of each method action.

- IsPageLoaded(): Check that Login link is displayed on the page to verify the page has loaded.
- NavigateToHomePage(): Navigate to home page.
- ClickonLoginLink(): Click on Login link to navigate to Login page.
- SelectMyOrdersLink(): Click on My orders link to navigate to My orders page.

```
using Magenic.Maqs.BaseSeleniumTest;
using Magenic.Maqs.BaseSeleniumTest.Extensions;
using Magenic.Maqs.Utilities.Helper;
using OpenQA.Selenium;

namespace PageModel
{
    /// <summary>
    /// Page object for HomePage
    /// </summary>
    public class HomePage
    {
        /// <summary>
        /// The page url
        /// </summary>
        private static readonly string PageUrl = SeleniumConfig.GetWebSiteBase();

        /// <summary>
        /// Selenium test object
        /// </summary>
        private SeleniumTestObject testObject;

        /// <summary>
        /// Initializes a new instance of the <see cref="HomePage" /> class.
        /// </summary>
        /// <param name="testObject">The selenium test object</param>
        public HomePage(SeleniumTestObject testObject)
        {
            this.testObject = testObject;
        }

        /// <summary>
        /// Gets the Login Link element
        /// </summary>
        private LazyElement LoginLink
        {
            get { return new LazyElement(this.testObject, By.LinkText("Login"), "Login Link in home page"); }
        }

        /// <summary>
        /// Gets the myorders Link element
        /// </summary>
        private LazyElement MyOrdersLink
        {
            get { return new LazyElement(this.testObject, By.LinkText("My orders"), "MyOrders Link disply in home page after login to user
account"); }
        }
        /// <summary>
        /// Check if the page has been loaded
        /// </summary>
        /// <returns>True if the page was loaded</returns>
        public bool IsPageLoaded()
        {
            return this.LoginLink.Displayed;
        }

        ///<summary>
        ///Navigate to home page
        /// </summary>
        public void NavigateToHomePage()
        {
            this.testObject.WebDriver.Navigate().GoToUrl(PageUrl);
        }

        ///<summary>
        ///Click on LoginLink to move to login page
        /// </summary>
        public void ClickonLoginLink()
        {
            LoginLink.Click();
        }

        ///<summary>
        ///Navigate to myorders
        /// </summary>
        public void SelectMyOrdersLink()
        {
            this.testObject.WebDriver.Wait().ForClickableElement(By.LinkText("My orders"));
            MyOrdersLink.Click();
        }
    }

}
```

UserLoginPage.cs

Following are the methods on UserLoginPage.cs with an explanation of each method action.

IsPageLoaded(): Check that the e-mail field is displayed to verify that the Login Page is loaded.
LogInToUserAccount(): Login to the user account by providing valid user name and password of registered user.

```
using Magenic.Maqs.BaseSeleniumTest;
using Magenic.Maqs.BaseSeleniumTest.Extensions;
using Magenic.Maqs.Utilities.Helper;
using OpenQA.Selenium;

namespace PageModel
{
    /// <summary>
```

```
        /// Page object for LoginToSite
        /// </summary>
        public class UserLoginPage
        {
            /// <summary>
            /// The page url
            /// </summary>
            private static readonly string PageUrl = SeleniumConfig.GetWebSiteBase() + "Account/SignIn";

            /// <summary>
            /// Selenium test object
            /// </summary>
            private SeleniumTestObject testObject;

            /// <summary>
            /// Initializes a new instance of the <see cref="UserLoginPage" /> class.
            /// </summary>
            /// <param name="testObject">The selenium test object</param>
            public UserLoginPage(SeleniumTestObject testObject)
            {
                this.testObject = testObject;
            }

            /// <summary>
            /// Gets the email field element
            /// </summary>
            private LazyElement EmailField
            {
                get { return new LazyElement(this.testObject, By.CssSelector("#Email"), "User email fileld"); }
            }

            /// <summary>
            /// Gets the password field element
            /// </summary>
            private LazyElement PassowrdField
            {
                get { return new LazyElement(this.testObject, By.CssSelector("#Password"), "User password fileld"); }
            }

            /// <summary>
            /// Gets the Login field element
            /// </summary>
            private LazyElement LoginButton
            {
                get { return new LazyElement(this.testObject, By.CssSelector(".btn.btn-default.btn-brand.btn-brand-big"), "Lonin button"); }
            }

            /// <summary>
            /// Check if the page has been loaded
            /// </summary>
            /// <returns>True if the page was loaded</returns>
            public bool IsPageLoaded()
            {
                return this.EmailField.Displayed;
            }

            ///<summary>
            ///SignIn
            /// </summary>
            public void LogInToUserAccount()
            {
                EmailField.SendKeys("demouser@microsoft.com");
                PassowrdField.SendKeys("Pass@word1");
                LoginButton.Click();
            }
        }

    }
```

MyOrders.cs

The method `VerifyMyOrderPage()` checks “My Order History” text to verify the user has navigated to the my order page.

```
using Magenic.Maqs.BaseSeleniumTest;
using Magenic.Maqs.BaseSeleniumTest.Extensions;
using Magenic.Maqs.Utilities.Helper;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium;

namespace PageModel
{
    /// <summary>
    /// Page object for MyOrders
    /// </summary>
    public class MyOrders
    {
        /// <summary>
        /// The page url
        /// </summary>
        private static readonly string PageUrl = SeleniumConfig.GetWebSiteBase() + "/Order/Index";

        /// <summary>
        /// Selenium test object
        /// </summary>
        private SeleniumTestObject testObject;

        /// <summary>
        /// Initializes a new instance of the <see cref="MyOrders" /> class.
        /// </summary>
        /// <param name="testObject">The selenium test object</param>
        public MyOrders(SeleniumTestObject testObject)
        {
            this.testObject = testObject;
        }

        /// <summary>
        /// Gets the sample element
        /// </summary>
        private LazyElement Sample
        {
            get { return new LazyElement(this.testObject, By.CssSelector("#CSS_ID"), "SAMPLE"); }
        }

        /// <summary>
        /// Check if the page has been loaded
        /// </summary>
        /// <returns>True if the page was loaded</returns>
        public bool IsPageLoaded()
        {
            return this.Sample.Displayed;
```

```
        }

        ///<summary>
        ///
        /// </summary>
        public void VerifyMyOrderPage()
        {
            this.testObject.WebDriver.Wait().UntilPageLoad();
            Assert.AreEqual(true, this.testObject.WebDriver.FindElement(By.CssSelector(".container > h1")).Text.Equals("My Order History"));
        }
    }

}
```

Adding the Test Class

We have added all three classes to the project and have written methods inside each class to perform different tasks. Next, we have to add a test class and access these PageModel classes from there.

To add a new test class, right-click on Tests in Solution Explorer and select Add ➤ New Item.

The new item pop-up will displayed. Select Magenic's Open Test then select Maqs Selenium VS Test Class. Give it the name `TestClass` and click on Add. See Figure 3-32.

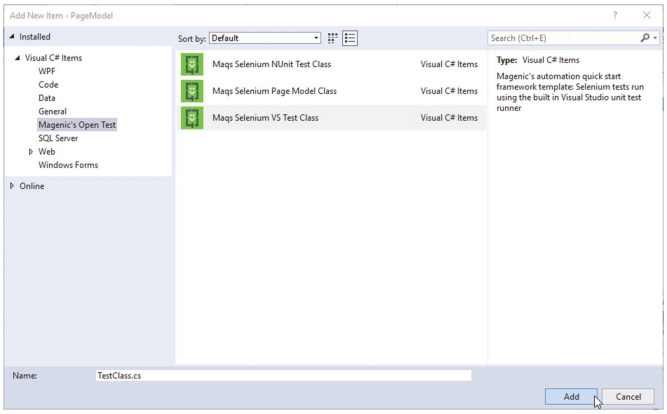

Figure 3-32 Add Maqs Selenium VS Test Class

Now that we have added a test class, we can write a test method as shown in Figure 3-32.

```
using Magenic.Maqs.BaseSeleniumTest;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using PageModel;

// TODO: Add reference to object model
// using PageModel;

namespace Tests
{
    /// <summary>
    /// TestClass test class
    /// </summary>
    [TestClass]
    public class TestClass : BaseSeleniumTest
    {
        /// <summary>
        /// Registered user login
        /// </summary>
        [TestMethod]
        public void LoginToUserAccount()
        {
            HomePage homePage = new HomePage(this.TestObject);
            homePage.NavigateToHomePage();
            Assert.AreEqual(true,homePage.IsPageLoaded());
            homePage.ClickonLoginLink();
            UserLoginPage loginToSite = new UserLoginPage(this.TestObject);
            loginToSite.LogInToUserAccount();
            homePage.SelectMyOrdersLink();
            MyOrders myOrders = new MyOrders(this.TestObject);
            myOrders.VerifyMyOrderPage();
        }
    }
}
```

You can see inside the test method class that instances of the page object model classes are initialized and methods are called to perform the test steps.

Lesson 3.08: SpecFlow Framework with C# Automation Code Example

We have discussed SpecFlow in the previous chapter and have created a sample project. Let's try to write simple test with SpecFlow and Selenium using C#. The sample code discussed here can be found at https://github.com/chamindac/Book-Test-Autmation-VS/tree/master/Chapter%203/Lesson%203.08 .

1. Create a New Windows Desktop project, as described in Chapter 2.
2. Then Add Feature file to the project that describes the scenario you are going to test. (see Figure 3-33).

```
Feature: OnlineShoppingUserRegistration
    In order to do online transactions
    As an unregisterd user of the web site
    I want to register to website

@mytag
Scenario: Register new user
    Given user is at home page
    And navigate to registration page
    When user enter valid email password and confirm password
    And click on the Signin button
    Then user navigate to user account
    When user logout from the user account
    Then myaccount link should not be displayed
```

Figure 3-33 Feature File

3. After adding the feature file to the project, you can add a step definition file where we write test methods.

   Let's try to identify each section of step definition.

At the top of the class file, you can find all the required references getting added.

```
using NUnit.Framework;using OpenQA.Selenium;using OpenQA.Selenium.Chrome;using OpenQA.Selenium.Support.UI;using System;using
TechTalk.SpecFlow;using ExpectedConditions = SeleniumExtras.WaitHelpers.ExpectedConditions;
```

When you go through the step definition class file, you can see there are separate methods for each step-in feature file. You will be able to find `Webdriver` and `WebDriverWait` has initialized at the beginning.

```
IWebDriver driver;
WebDriverWait wait;
```

Then the first method is used to launch the web browser and navigate to the home page of the eShop web application.

```
[Given(@"user is at home page")]public void GivenUserIsAtHomePage()
{
ChromeOptions options = new ChromeOptions();
options.AddArgument("--start-maximized");
driver = new ChromeDriver(options);
driver.Navigate().GoToUrl("http://eshop-testweb.azurewebsites.net");
wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30));
 }
```

The second method is used to navigate to the login page by clicking on the Login link on the home page. Then it clicks on the “Register as a new user?” link to move on to the new user registration page.

```
[Given(@"navigate to registration page")]public void GivenNavigateToRegistrationPage()
{
IWebElement loginLink= wait.Until(ExpectedConditions.ElementToBeClickable(By.LinkText("Login")));loginLink.Click();IWebElement registerNewUser =
wait.Until(ExpectedConditions.ElementExists(By.LinkText("Register as a new user?")));registerNewUser.Click();    }
```

The next step is to provide user details to register with the eShop website.

```
[When(@"user enter valid email password and confirm password")]public void WhenUserEnterValidEmailPasswordAndConfirmPassword()      {
IWebElement emailField= wait.Until(ExpectedConditions.ElementToBeClickable(By.Id("Email")));
emailField.SendKeys("BNLL@kk.com");
IWebElement passWord = wait.Until(ExpectedConditions.ElementExists(By.Id("Password"))); passWord.SendKeys("Pass@123");
IWebElement confirmPassWord = wait.Until(ExpectedConditions.ElementExists(By.Id("ConfirmPassword")));
confirmPassWord.SendKeys("Pass@123");                }
```

After providing the details it is required to click on sign in button to submit new user details to the eShop website.

```
[When(@"click on the Signin button")]public void WhenClickOnTheSigninButton()
{
IWebElement registerButton = wait.Until(ExpectedConditions.ElementToBeClickable(By.CssSelector(".btn.btn-default.btn-brand.btn-brand-big")));
registerButton.Click();
}
```

Next you can check the My account link availability to verify a new user has been successfully added.

```
[Then(@"user navigate to user account")] public void ThenUserNavigateToUserAccount()
{
Assert.AreEqual(true, driver.FindElement(By.LinkText("My account")).Displayed);
}
```

Finally, logout from user account can be performed.

```
[When(@"user logout from the user account")]
 public void WhenUserLogoutFromTheUserAccount(
{
IWebElement logoutLink = driver.FindElement(By.LinkText("Log Out"));
logoutLink.Click();
}
```

You can check the My account link availability to verify the user has successfully logged out from the user account.

```
[Then(@"myaccount link should not be displayed")]public void ThenMyaccountLinkShouldNotBeDisplayed() {
Assert.AreEqual(false, driver.FindElements(By.LinkText("My account")).Count>0);
}
```

The complete code for these steps is shown here:

```
using NUnit.Framework;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.Support.UI;
using System;
using TechTalk.SpecFlow;
using ExpectedConditions = SeleniumExtras.WaitHelpers.ExpectedConditions;

namespace OnlineShoppingAppTest
{
    [Binding]
    public class OnlineShoppingUserRegistrationSteps
    {
        IWebDriver driver;
        WebDriverWait wait;
        [Given(@"user is at home page")]
        public void GivenUserIsAtHomePage()
        {
            ChromeOptions options = new ChromeOptions();
            options.AddArgument("--start-maximized");
            driver = new ChromeDriver(options);
            driver.Navigate().GoToUrl("http://eshop-testweb.azurewebsites.net");
            wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30));
        }

        [Given(@"navigate to registration page")]
        public void GivenNavigateToRegistrationPage()
        {
           IWebElement loginLink= wait.Until(ExpectedConditions.ElementToBeClickable(By.LinkText("Login")));
            loginLink.Click();
            IWebElement registerNewUser = wait.Until(ExpectedConditions.ElementExists(By.LinkText("Register as a new user?")));
            registerNewUser.Click();
        }

        [When(@"user enter valid email password and confirm password")]
        public void WhenUserEnterValidEmailPasswordAndConfirmPassword()
        {
            IWebElement emailField= wait.Until(ExpectedConditions.ElementToBeClickable(By.Id("Email")));
            emailField.SendKeys("BNLL@kk.com");
            IWebElement passWord = wait.Until(ExpectedConditions.ElementExists(By.Id("Password")));
            passWord.SendKeys("Pass@123");
            IWebElement confirmPassWord = wait.Until(ExpectedConditions.ElementExists(By.Id("ConfirmPassword")));
            confirmPassWord.SendKeys("Pass@123");
        }

        [When(@"click on the Signin button")]
        public void WhenClickOnTheSigninButton()
        {
            IWebElement registerButton = wait.Until(ExpectedConditions.ElementToBeClickable(By.CssSelector(".btn.btn-default.btn-brand.btn-
```

```
brand-big")));
                registerButton.Click();
            }
            [Then(@"user navigate to user account")]
            public void ThenUserNavigateToUserAccount()
            {
                Assert.AreEqual(true, driver.FindElement(By.LinkText("My account")).Displayed);
            }

            [When(@"user logout from the user account")]
            public void WhenUserLogoutFromTheUserAccount()
            {
                IWebElement logoutLink = driver.FindElement(By.LinkText("Log Out"));
                logoutLink.Click();
            }

            [Then(@"myaccount link should not be displayed")]
            public void ThenMyaccountLinkShouldNotBeDisplayed()
            {
                Assert.AreEqual(false, driver.FindElements(By.LinkText("My account")).Count>0);
            }
        }
    }
```

In this lesson you learned how to work with different locators, commands, and wait handling mechanisms, while working with SpecFlow and Selenium using C#.

Summary

In this chapter, you explored different locators available in Selenium and gained good understanding on different commands. Further, you learned wait handling mechanisms and their use in test automation scripts. With sample scenario implementations using various tools, you were provided with required experience and knowledge on how and when to use different types of locators, commands, and waits.

In the next chapter, we will explore possibilities of using test automation to test Windows applications.

Chaminda Chandrasekara and Pushpa Herath, *Hands-On Functional Test Automation*
https://doi.org/10.1007/978-1-4842-4411-1_4

4. Functional Testing for Windows Apps

Chaminda Chandrasekara[1] and Pushpa Herath[2]

(1) Dedigamuwa, Sri Lanka
(2) Hanguranketha, Sri Lanka

The objective of this chapter is to guide you on development of functional test automation for windows applications using different technologies. You will find lessons with code examples on how to automate windows applications with Coded UI, Sikuli, and Winium.

Lesson 4.01: Create a Test Project with Coded UI and C#

Let's begin by setting up a Coded UI test project to start writing automated functional testing for Windows applications.

Prerequisites: You must be running Visual Studio 2017 on Windows 10 or on Windows Server 2012 R2 or a newer version of Windows server. You must have intermediate level of C# language proficiency.

Install the Coded UI test component in Visual Studio 2017

Here are the steps:

1. Launch the Visual Studio Installer.
2. In the Visual Studio Installer, go to the Individual components tab and select the Coded UI test under Debugging and testing section (see Figure 4-1).

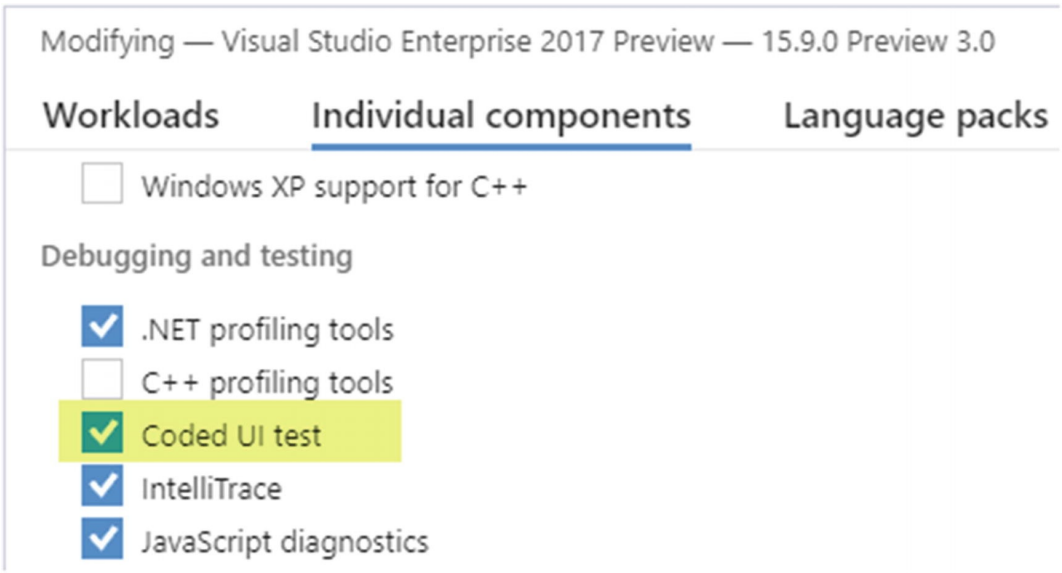

Figure 4-1 Coded UI component in Visual Studio Installer

3. Click on Modify to apply the changes to Visual Studio.

Setting Up the Visual Studio Test Project

Let's begin setting up the Visual Studio test project with C# and Coded UI following the steps described here:

1. In Visual Studio 2017, select Files ➤New ➤ Project.
2. In the New Project pop-up window, select Test under Visual Studio C# and select Coded UI Test Project from the test project list. Give a **Name** for the Project, specify a **Solution Name**, select **Location,** and click on the OK button. Leaving **Create directory for solution** checked will allow you to have a new directory created for the new solution in the selected location (see Figure 4-2).

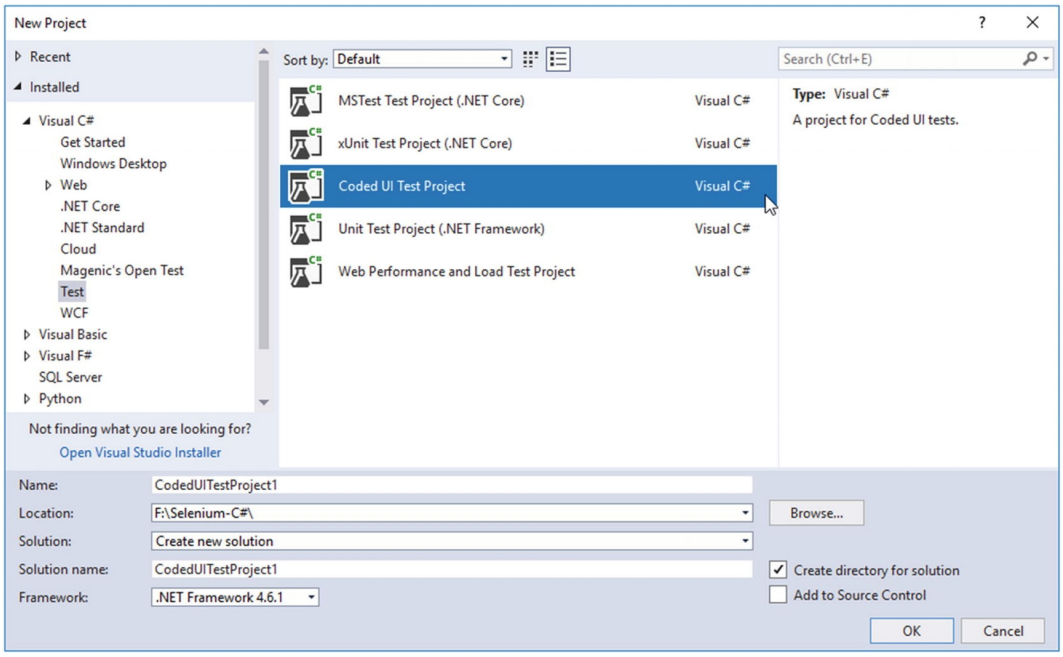

Figure 4-2 Create new Coded UI project

3. After adding a new Coded UI project, the Generate Code for Coded UI Test dialog will pop up. Select Record actions, edit UI map or add assertions, and select OK (see Figure 4-3).

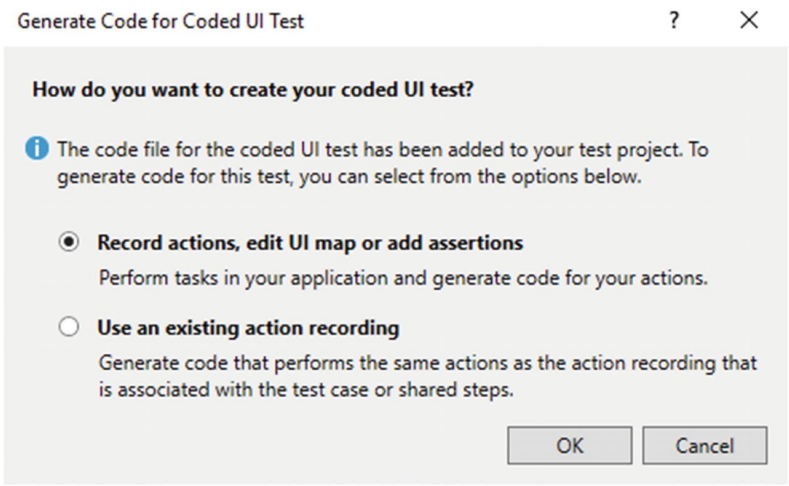

Figure 4-3 Generate code for Coded UI test pop-up dialog

The Coded UI Test Builder will appear (see Figure 4-4).

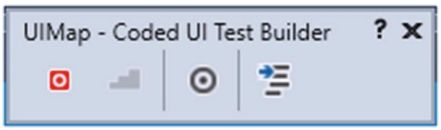

Figure 4-4 Coded UI Test Builder

Coded UI Test Builder can be opened this way only for newly created projects. Afterward you can open Coded UI Test Builder using following methods.

Method 1

Go to Test ➤ Generate Code for Coded UI Test ➤ Use Coded UI Test Builder (See Figure 4-5).

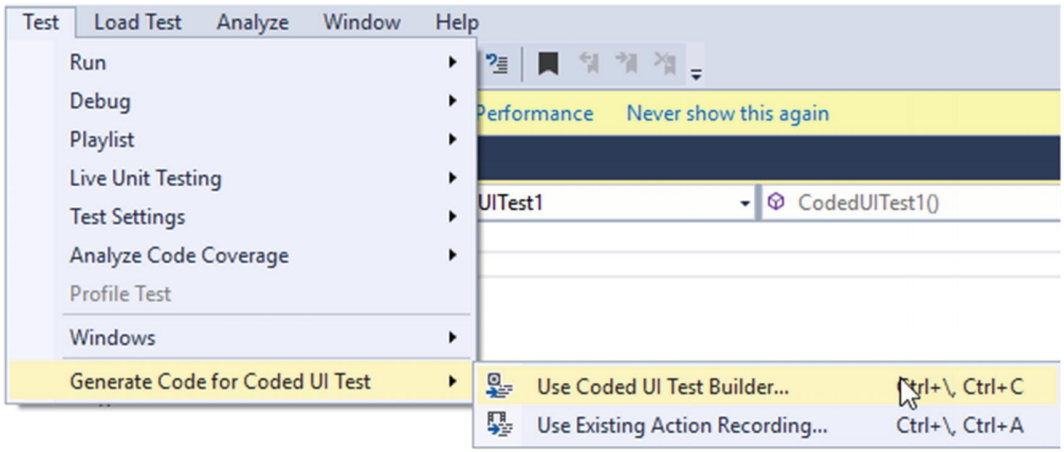

Figure 4-5 Open Coded UI Test Builder Method 1

Method 2

Open CodedUITest1.cs file from Solution Explorer.

1. Right-click within either a test method or property of Coded UI test class.
2. Select Generate Code for Coded UI Test ➤ Use Coded UI Test Builder (see Figure 4-6).

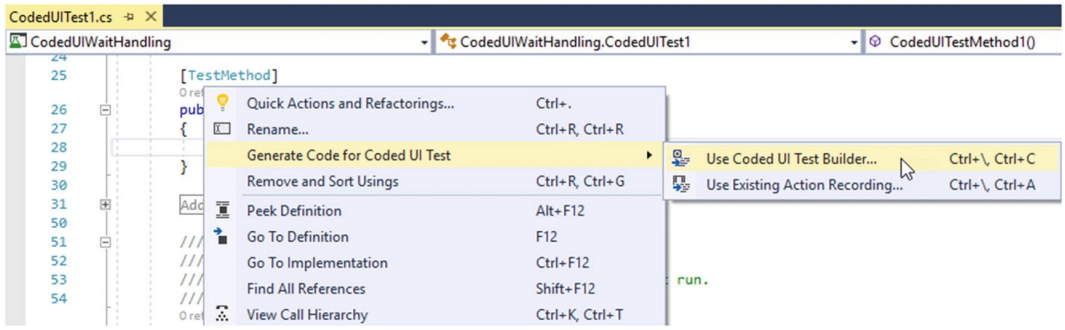

Figure 4-6 Open Coded UI Test Builder Method 2

Within this lesson you learned how to create a Coded UI test project and launch the Coded UI Test Builder. In the next lesson, we will discuss how to handle windows elements.

Lesson 4.02: How to Capture Windows Elements

You know while we work with web elements or windows elements, those elements need to be identified uniquely before performing any action on them. You may remember we discussed different locators in Chapter 2. Similarly, windows elements can be identified uniquely using various windows attributes. Let's discuss how to capture Windows elements in this lesson.

We know developer tools in web browsers can be used to inspect web elements. When it comes to Windows elements, the Coded UI test builder can be used to identify Windows elements.

Let's identify the Coded UI test builder first (See Figure 4-7).

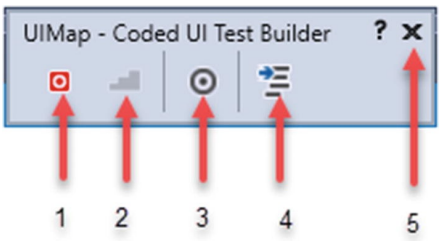

Figure 4-7 Coded UI Test Builder Features

1. Record Icon
   To start recording, click on the record icon and perform the scenarios you want to test. Click the same record icon to pause and resume the recording.
2. Recorded Actions
   This will display all the steps recorded so far.
3. Add Assertions
   You can find all the available property values of windows elements and add assertions to verify different actions.
4. Generate Code
   After recording the actions needed to be performed in the test, you can generate code to perform those actions by using

this option.

5. Coded UI Test Builder Close Button
   Can be used to close Coded UI Test Builder window.

We have a fair understanding of Coded UI Test Builder icons now. You will be able to learn more about these options in a future lesson in this chapter.

As we have discussed, the Add Assertion window displays property values of Windows elements. We can use this option and find attributes of each Windows element. Let's see how we can find attribute values of the Windows File Explorer icon in the Windows task bar.

Double-click on the Add Assertion icon of the Coded UI Test Builder and hold the mouse button. Then move the cursor over the File Explorer icon in the taskbar. You can see the mouse cursor has turned into a crosshair, and the highlighting option is enabled. It highlights controls after you hover the mouse over the control. When the cursor is on the control, release the mouse. Then the Add Assertion window will open with property values (see Figure 4-8).

Figure 4-8 Identify controls of the highlighted area

Now you can check property values in the Add Assertions window as follows (see Figure 4-9).

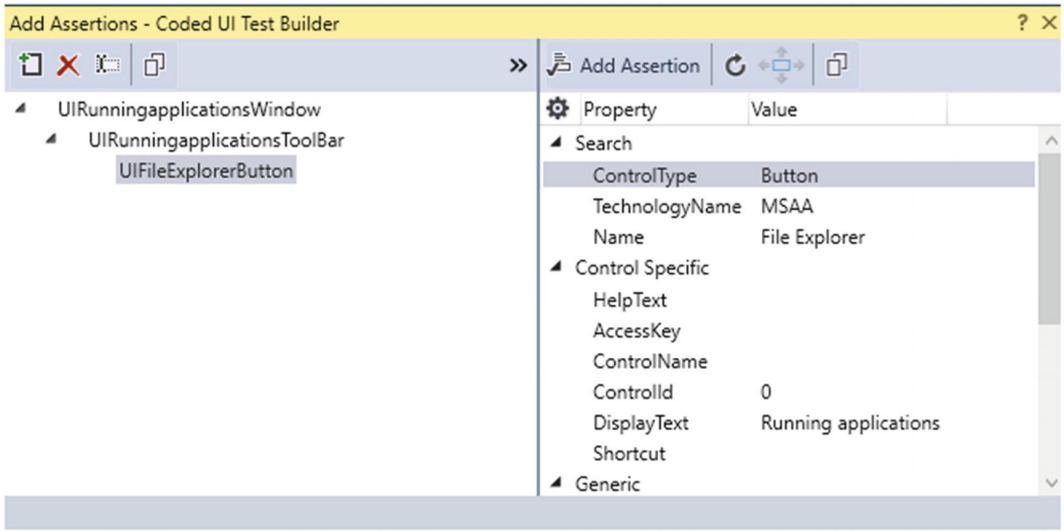

Figure 4-9 Add Assertion window with property values

So far, we have learned how to identify a control. Let's try to write a simple code to click on the File Explorer icon on the Windows taskbar.

The Add Assertion window displays the control structure. If you examine Figure 4-9, you can identify `UIFileExplorerButton` is inside the `UIRunningapplicationToolBar` and the toolbar control is inside the `UIRunningapplicationWindow`. So, when locating the controls, first we need to identify the window control and then need to identify the matching toolbar control inside the window control. Finally, we need to find the button control inside the toolbar control.

Select `UIRunningapplicationWindow` and read control property values (see Figure 4-10).

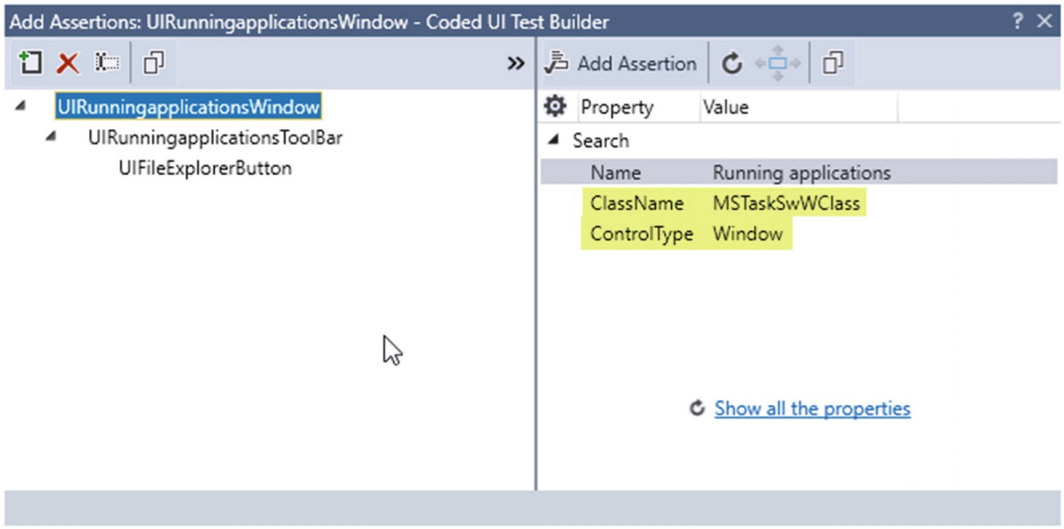

Figure 4-10 UIRunningapplicationsWindow properties

Let's use `ClassName` and `ControlType` values to identify this control. Next let's find property values of the `UIRunningapplicationsToolBar` control (see Figure 4-11).

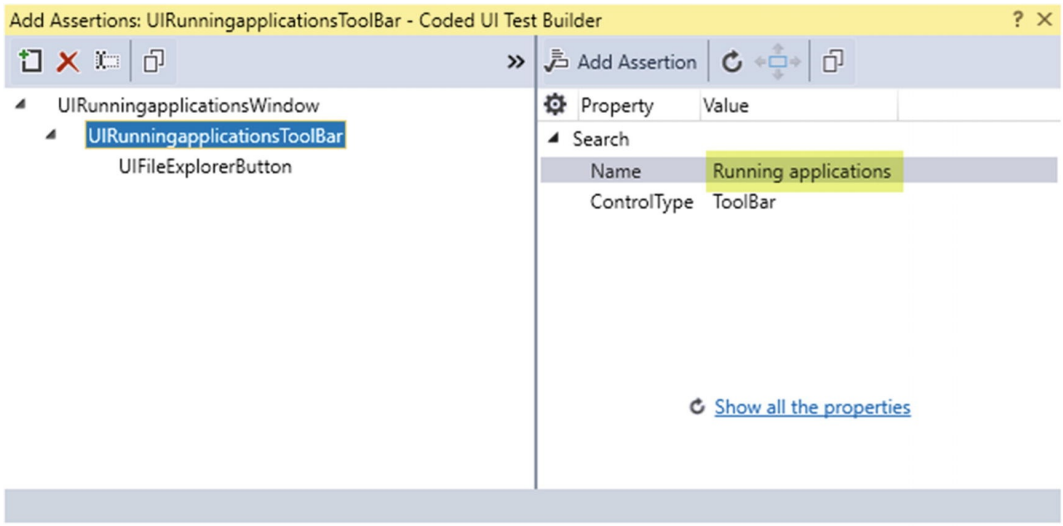

Figure 4-11 UIRunningapplicationsToolBar control properties

Let's use the `Name` attribute to identify this control. `UIFileExplorerButton` property values can be identified as shown in Figure 4-12 .

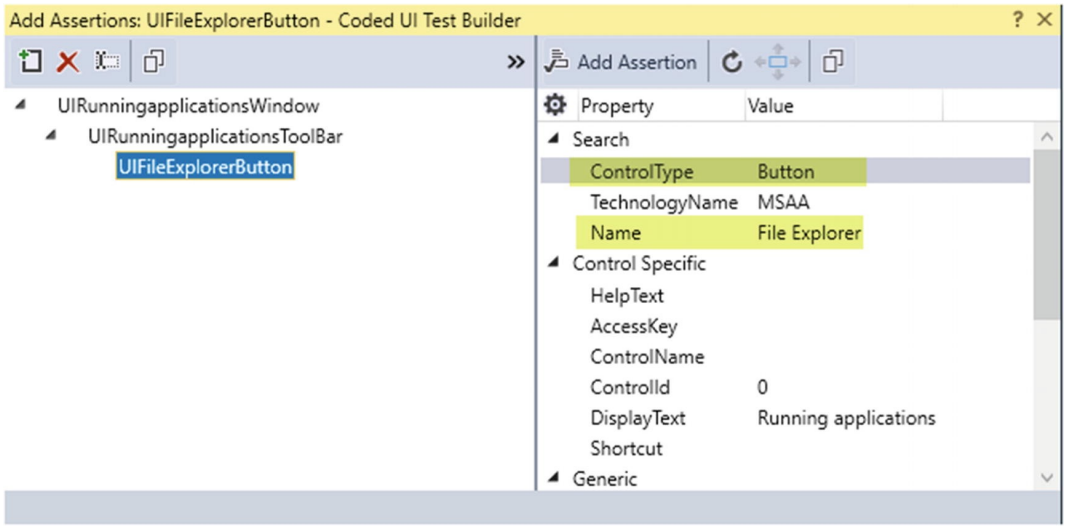

Figure 4-12 UIFileExplorerButton control properties

Now we know property values of each control. Let's try to write a test method to click on the File Explorer icon.

Add `Microsoft.VisualStudio.TestTools.UITesting.WinControls` as a reference to the project. Then use the namespace in the code to work with the Windows controls (see Figure 4-13).

```
using System;
using System.Collections.Generic;
using System.Text.RegularExpressions;
using System.Windows.Input;
using System.Windows.Forms;
using System.Drawing;
using Microsoft.VisualStudio.TestTools.UITesting;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Microsoft.VisualStudio.TestTools.UITest.Extension;
using Keyboard = Microsoft.VisualStudio.TestTools.UITesting.Keyboard;
using Microsoft.VisualStudio.TestTools.UITesting.WinControls;
```

Figure 4-13 Reference files

Let's move to Test Method and add the code as follows.

We first need to find the window control. So initialize the new instance of `WinWindow` class. Then give the property values of window control that need to be found.

```
WinWindowparentWindow=newWinWindow();
parentWindow.SearchProperties["ClassName"]="MSTaskSwWClass";
parentWindow.SearchProperties["ControlType"]="Window";
```

Next we need to give search property values of the toolbar inside the window control. So, initialize the new instance of `WinToolBar` class while providing the parent control as parameter.

```
WinToolBartoolBar=newWinToolBar(parentWindow);
toolBar.SearchProperties["Name"]="Runningapplications";
```

Finally, initialize the `WinButton` instance and provide toolbar instance as a parent control.

```
WinButtonbutton=newWinButton(toolBar);
button.SearchProperties["Name"]="File Explorer";
button.SearchProperties["ControlType"]="Button";
```

Steps to identify controls are completed now. Next you can perform click on button control.

```
Mouse.Click(button);
```

You can find complete test method code as follows:

```
[TestMethod]
public void CodedUITestMethod1()
{
    WinWindow parentWindow = new WinWindow();
    parentWindow.SearchProperties["ClassName"] = "MSTaskSwWClass";
    parentWindow.SearchProperties["ControlType"] = "Window";
    WinToolBar toolBar = new WinToolBar(parentWindow);
    toolBar.SearchProperties["Name"] = "Running applications";
    WinButton button = new WinButton(toolBar);
    button.SearchProperties["Name"] = "File Explorer";
    button.SearchProperties["ControlType"] = "Button";
    Mouse.Click(button);
}
```

You can execute the Coded UI test method by right-clicking on the test method in Test Explorer and select Run Selected Tests (see Figure 4-14) .

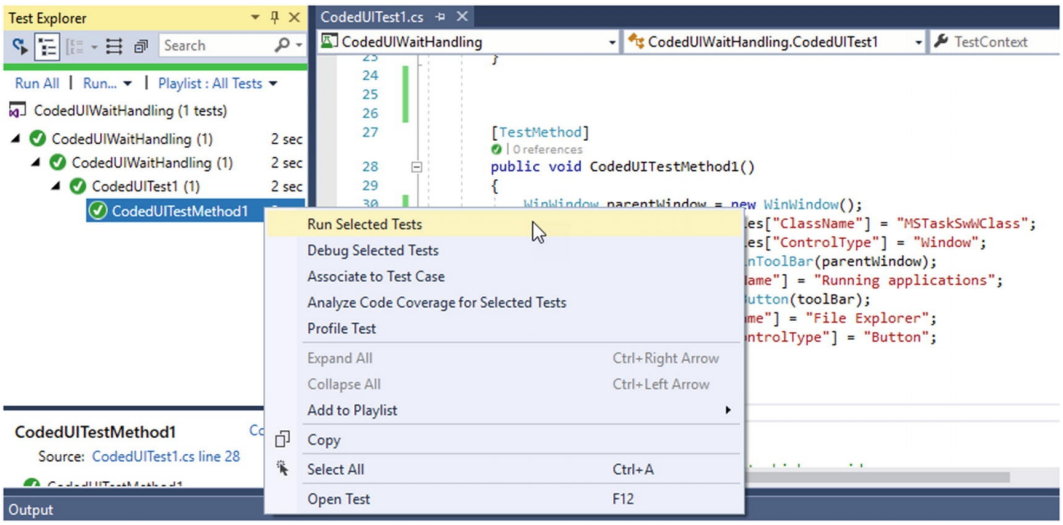

Figure 4-14 Run a test method using Test Explorer

You learned about the Coded UI Test Builder and how to find window control property values in this lesson. Additionally you learned how to perform action on controls by using a sample code.

Lesson 4.03: Element Commands

This lesson will show you the mouse actions and keyboard actions on window controls. Source code for this lesson can be found at https://github.com/chamindac/Book-Test-Autmation-VS/tree/master/Chapter%204/Lesson%204.03/CodedUIWaitHandling

Click( )

The Coded UI `Click()` method can be used to simply click on window controls or perform various complicated click actions. Let's discuss different ways of using the `Click()` method.

Mouse.Click( )

Click on the current position where the cursor is located.

```
[TestMethod]
public void Click()
{
Mouse.Click();
}
```

Mouse.Click(UITestControl Control)

Click on the given control. In the following example, the File Explorer icon in the taskbar is the window control to click on.

```
[TestMethod]
public void MouseClickonControl()
{
            WinWindow parentWindow = new WinWindow();
            parentWindow.SearchProperties["ClassName"] = "MSTaskSwWClass";
            parentWindow.SearchProperties["ControlType"] = "Window";
            WinToolBar toolBar = new WinToolBar(parentWindow);
            toolBar.SearchProperties["Name"] = "Running applications";
            WinButton button = new WinButton(toolBar);
            button.SearchProperties["Name"] = "File Explorer";
            button.SearchProperties["ControlType"] = "Button";
            Mouse.Click(button);
}
```

Click(ModifierKeys modifierKeys)

When we need to use modifier keys such as Shift, Control, or Alter keys, we can pass the modifier key value as the parameter and perform a click action. The following example shows how to click while pressing the Windows modifier key.

```
[TestMethod]
public void ClickModifierKey()
{
    Mouse.Click(ModifierKeys.Windows);
}
```

Click(Point ScreenCoordinate);

Click on the given screen coordinates provided as parameter values. The following example gives (x,y) values as (0,0).

```
[TestMethod]
public void ClickOnPoint()
{
    Mouse.Click(Point.Empty);
}
```

Click(UITestControl Control,MouseButtons Button);

Click a given mouse button on a given control. You can click on different mouse buttons. For example, it allows clicking on the left, right, or middle mouse button. The following sample code shows how to right-click on the File Explorer icon in the task bar.

```
[TestMethod]
public void RightClickonControl()
{
    WinWindow parentWindow = new WinWindow();
    parentWindow.SearchProperties["ClassName"] = "MSTaskSwWClass";
    parentWindow.SearchProperties["ControlType"] = "Window";
    WinToolBar toolBar = new WinToolBar(parentWindow);
    toolBar.SearchProperties["Name"] = "Running applications";
    WinButton button = new WinButton(toolBar);
    button.SearchProperties["Name"] = "File Explorer";
    button.SearchProperties["ControlType"] = "Button";
    Mouse.Click(button, MouseButtons.Right);

}
```

Click(UITestControl Control,Point relativeCoordinate);

Click on a given control on a given point relative to the control. The following sample code explains how to click on a given point of a given control.

```
[TestMethod]
public void ClickonControlPoint()
{
    WinWindow parentWindow = new WinWindow();
    parentWindow.SearchProperties["ClassName"] = "MSTaskSwWClass";
    parentWindow.SearchProperties["ControlType"] = "Window";
    WinToolBar toolBar = new WinToolBar(parentWindow);
    toolBar.SearchProperties["Name"] = "Running applications";
    WinButton button = new WinButton(toolBar);
    button.SearchProperties["Name"] = "File Explorer";
    button.SearchProperties["ControlType"] = "Button";
    var buttonPosition = button.BoundingRectangle;
    Point relativePoint = new Point(buttonPosition.X + 5, buttonPosition.Y - 5);
    Mouse.Click(button, relativePoint);
}
```

Click(UITestControl Control, MouseButtons Button, ModifierKeys, Point relativeCoordinate)

You can click a mouse button on given coordinates of the control along with modifier keys. Relative coordinates help to click on a position relative to the control.

Click(MouseButtons Button, ModifierKeys, Point screenCoordinate)

You can click on the given coordinates of the control along with given modifier key. Screen coordinates allows clicking on a position in relation to the whole screen.

Click(UITestControl Control, ModifierKeys modifierKeys);

Click on a given control while pressing a given modifier key.

DoubleClick( )

In Coded UI, there are more options to perform double-click on the control. Let’s see different type of double-clicks.

DoubleClick( );

Double-click at the location of the mouse cursor.

```
[TestMethod]
public void DoubleClick()
{
    Mouse.DoubleClick();
}
```

DoubleClick(UITestControl Control)

Double-click on the given control. The following sample code shows how to double-click on the File Explorer icon in the taskbar.

```
[TestMethod]
public void DoubleClickControl()
{
    WinWindow parentWindow = new WinWindow();
    parentWindow.SearchProperties["ClassName"] = "MSTaskSwWClass";
    parentWindow.SearchProperties["ControlType"] = "Window";
    WinToolBar toolBar = new WinToolBar(parentWindow);
    toolBar.SearchProperties["Name"] = "Running applications";
    WinButton button = new WinButton(toolBar);
    button.SearchProperties["Name"] = "File Explorer";
    button.SearchProperties["ControlType"] = "Button";
    Mouse.DoubleClick(button);
}
```

DoubleClick(MouseButtons Button)

Double-click the given mouse button at the current mouse pointer position. The following sample code is for double-clicking the left mouse button.

```
[TestMethod]
public void DoubleClickMouseButton()
{
    Mouse.DoubleClick(MouseButtons.Left);
}
```

DoubleClick(ModifierKeys modifierKeys)

Double-click at the current mouse pointer position while pressing the given modifier key. The following code is for double-clicking while pressing the windows key.

```
[TestMethod]
public void DoubleClickModifier()
{
    Mouse.DoubleClick(ModifierKeys.Windows);
}
```

DoubleClick(Point screenCoordinate)

Double-click on given screen coordinates. The following sample code shows double-click on (0,0) location.

```
[TestMethod]
public void DoubleClickPoint()
{
    Mouse.DoubleClick(Point.Empty);
}
```

1. `DoubleClick(MouseButtons button, ModifierKeys, Point screenCoordinates)`: Will double click the given mouse button while pressing a given modifier key at given screen coordinates.
2. `DoubleClick(UITestControl control, MouseButtons button, ModifierKeys, Point relativeCoordinates)`: Will double-click the given mouse button while pressing a given modifier key at given coordinates relative to a given UI control.
3. `DoubleClick(UITestControl control, ModifierKeys modifierKeys)`: Will double-click while pressing a given modifier key on a given UI control.
4. `DoubleClick(UITestControl control, MouseButtons button)`: Will double-click the given mouse button on a given UI control.
5. `DoubleClick(UITestControl control, Point relativeCoordinate)`: Will double-click on given coordinates of a given UI control.

These are the available DoubleClick methods. You can use these according to your requirements.

Hover( )

Hover allows the mouse pointer hover operation simulation in the Coded UI tests.

Hover(Point screenCoordinate)

Hover the mouse over given screen coordinates. The following sample code is for hovering the mouse over the (0,0) point.

```
[TestMethod]
public void MouseHoverPoint()
{
    Mouse.Hover(Point.Empty);
```

```
}
```

Hover(UITestControl control)

Hover the mouse over a given control. In the following code, hover the mouse over the File Explorer icon in the taskbar.

```
[TestMethod]
public void MouseHoverToControl()
{
    WinWindow parentWindow = new WinWindow();
    parentWindow.SearchProperties["ClassName"] = "MSTaskSwWClass";
    parentWindow.SearchProperties["ControlType"] = "Window";
    WinToolBar toolBar = new WinToolBar(parentWindow);
    toolBar.SearchProperties["Name"] = "Running applications";
    WinButton button = new WinButton(toolBar);
    button.SearchProperties["Name"] = "File Explorer";
    button.SearchProperties["ControlType"] = "Button";
    Mouse.Hover(button);
}
```

Hover(Point screenCoordinate, int millisecondsDuration)

Hover the mouse over a given screen coordinate and hold the execution for the given time period before moving to the next code line.

```
[TestMethod]
public void MouseHoverDuration()
{
    Mouse.Hover(Point.Empty, 2000);
}
```

Hover(UITestControl control, Point relativeCoordinate);

Hover the mouse over the coordinates relative to a given control.

Hover(UITestControl control, Point relativeCoordinate, int millisecondDuration)

Hover the mouse over the given coordinates relative to a given control and hold the execution for the given period of time before moving to the next code line.

MoveScrollWheel

Moving the scroll wheel of the mouse can be simulated in the Coded UI tests with this method.

MoveScrollWheel(int wheelMoveCount)

Move the scroll wheel a given number of times. In the following sample code, scroll 20 times.

```
[TestMethod]
public void MoveScrollwithScrollCount()
{
    Mouse.MoveScrollWheel(20);
}
```

1. `MoveScrollWheel(int wheelMoveCount, ModifierKeys modifierKeys)`: Will move scroll wheel given number of times while pressing a given modifier key on the current focused control.
2. `MoveScrollWheel(UITestControl control, int wheelMoveCount)`: Will move scroll wheel a given number of times on the given UI control.
3. `MoveScrollWheel(UITestControl control, int wheelMoveCount, ModifierKeys modifierKeys)`: Will move the scroll wheel a given number of times on a given UI control while pressing a given modifier key.

StartDragging

`StartDragging` methods allow us to start dragging a control from a given location to a location that should be specified with a `StopDragging` method.

- `StartDragging(UITestControl control)`: Will start dragging the given UI control.
- `StartDragging(UITestControl control, MouseButtons button)`: Will start dragging the given UI control using the given mouse button.
- `StartDragging(UITestControl control, Point relativeCoordinate)`: Will start dragging the given UI control in the given coordinates of the UI control.
- `StartDragging()`: Will start dragging the control in the current mouse pointer position.
- `StartDragging(UITestControl control, Point relativeCoordinate, MouseButtons button, ModifierKeys modifierKeys)`: Will start dragging the given UI control in the coordinates relative to the given UI control, using the given mouse button, while pressing the given modifier key.

StopDragging

After start dragging there should be the end point to stop dragging action. The `StopDragging` method is used to tell the end-point. The following are the `StopDragging` methods available.

- `StopDragging(Point pointToStop)`: Will stop dragging a control on the given coordinates.
- `StopDragging(int moveByX, int moveByY)`: Will stop dragging a control after moving a given number of pixels horizontally and vertically.
- `StopDragging(UITestControl control, int moveByX, int moveByY)`: Will stop dragging a control say A to control B, after moving a given number of pixels horizontally and vertically relative to the given UI control B, which would be the control that is the place holder of the control A that is being dragged and dropped.
- `StopDragging(UITestControl control)`: Will stop dragging the control that is being dragged, after dragging it to the given UI control.
- `StopDragging(UITestControl control, Point relativeCoordinate)`: Will stop dragging the control that is being dragged, after moving it to the given coordinates relative to the given UI control.

The following sample code shows how to use `StartDragging` and `StopDragging` methods. In the following code, the File Explorer icon in the taskbar is moved a given number of horizontal and vertical number of pixels.

```
[TestMethod]
public void StartDragAndDrop()
 {
     WinWindow parentWindow = new WinWindow();
     parentWindow.SearchProperties["ClassName"] = "MSTaskSwWClass";
     parentWindow.SearchProperties["ControlType"] = "Window";
     WinToolBar toolBar = new WinToolBar(parentWindow);
     toolBar.SearchProperties["Name"] = "Running applications";
     WinButton button = new WinButton(toolBar);
     button.SearchProperties["Name"] = "File Explorer";
     button.SearchProperties["ControlType"] = "Button";
     Mouse.StartDragging(button);
     Rectangle desktopWindow = WinWindow.Desktop.BoundingRectangle;
     Mouse.StopDragging(desktopWindow.Width / 4, desktopWindow.Y);
 }
```

SendKeys

`SendKeys` method allows you to type text in an input control.

- SendKeys(UITestControl control, string text, ModifierKeys, bool isEncoded, bool isUnicode): Will type the provided text in the given UI control while pressing the given modifier key. Can provide if text should be encoded and if text is in Unicode.
- SendKeys(string text, ModifierKeys, bool isEncoded, bool isUnicode): Will type the provided text in the currently focused UI control while pressing the given modifier key. Can provide if text should be encoded and if text is in Unicode.
- SendKeys(UITestControl control, string text, ModifierKeys, bool isEncoded): Will type the provided text in the given UI control while pressing the given modifier key. Can provide if text should be encoded.
- SendKeys(string text, ModifierKeys, bool isEncoded): Will type the provided text in the currently focused UI control while pressing the given modifier key. Can provide if text should be encoded.
- SendKeys(string text, ModifierKeys modifierKeys): Will type the provided text in the currently focused UI control while pressing the given modifier key.
- SendKeys(UITestControl control, string text, bool isEncoded): Will type the provided text in the given UI control while pressing the given modifier key. Can provide if text should be encoded.
- SendKeys(string text, bool isEncoded): Will type the provided text in the currently focused UI control. Can provide if text should be encoded.
- SendKeys(UITestControl control, string text): Will type the provided text in the given UI control.
- SendKeys(string text): Will type the provided text in the currently focused UI control.
- SendKeys(UITestControl control, string text, ModifierKeys modifierKeys): Will type the provided text in the given UI control while pressing the given modifier key.

The following sample code shows how to open the File Explorer window and type the word “currency” in the search field.

```
[TestMethod]

public void TypeValues()
{
    WinWindow dialog = new WinWindow();
    dialog.SearchProperties["ClassName"] = "MSTaskSwWClass";
    dialog.SearchProperties["ControlType"] = "Window";
    WinToolBar toolBar = new WinToolBar(dialog);
    toolBar.SearchProperties["Name"] = "Running applications";
    WinButton button = new WinButton(toolBar);
    button.SearchProperties["Name"] = "File Explorer";
    button.SearchProperties["ControlType"] = "Button";
    Mouse.Click(button);

    WinWindow dialog2 = new WinWindow();
    dialog2.SearchProperties["ClassName"] = "CabinetWClass";
```

```
    dialog2.SearchProperties["ControlType"] = "Window";
    WinEdit winSearch = new WinEdit(dialog2);
    winSearch.SearchProperties["Name"] = "Search Box";
    Keyboard.SendKeys(winSearch, "Currency");
}
```

We have learned possible mouse and keyboard actions on Windows controls in this lesson.

Lesson 4.04: Handle Element Waits

In web apps test automation, we learned about implicit and explicit waits in Selenium. Likewise, windows apps also need wait handling mechanisms. When it comes to the Coded UI framework, there are various wait methods. Let's try to identify those methods in this lesson. You can use them similarly to what you have done with the Selenium waits, which we have discussed in detail in Chapter 3 on web-based test automation.

- `WaitForControlEnabled`
  Wait for the control to be enabled. This can be used to check the status of the control and wait until control is enabled before performing any action on it.
- `WaitForControlReady`
  Wait for the control to be ready to accept mouse and keyboard inputs. When performing actions on UI elements, those elements take different time durations to appear on the UI. This method can be used to hold execution until the UI control is fully loaded and ready to accept actions on it.
- `WaitForControlExist`
  Waits for the control to appear in the UI. While doing UI test automation, there are situations where we need to verify pop-ups, validations, and message boxes. But these controls appear in the UI after performing a specific action and it may take different time durations for them to appear in the UI. For this type of a situation, WaitForControlExist method can be used.
- `WaitForControlNotExist`
  Wait for controls to disappear from the UI. As an example, you can consider a situation where we get a window pop-up in the middle of the test and we need to close that and continue testing. The close button of the window pop-up is clicked to close the window and we need a small amount of time to change focus from the window pop-up to the application. We can use the WaitForControlNotExist method to check whether the pop-up window has closed.
- `WaitForControlPropertyEqual`
  Wait for the specified property of the control to have the given value. As an example, in a situation where we do calculations and we need to verify that the calculated value is equal to a given value, we can use the WaitForControlPropertyEqual method.
- `WaitForControlPropertyNotEqual`
  Wait for a specified property of the control to have the opposite value. As an example, you can consider a condition that normally returns a value of "false" but in a specific situation returns "true." We can use this wait method to verify a returned value.
- `WaitForControlCondition`
  Wait for a specified predicate return to be true. As an example, a method returns a Boolean value when several conditions are true. You can use this method to wait till the method returns true.
  In this lesson, we have explored the available wait methods in the Coded UI test framework.

Lesson 4.05: Coded UI Test Builder–Record and Playback

After going through this lesson, you will be able to get an idea on how to work with the Coded UI record and playback feature, which allows you to record your actions in a manual testing and create a test automation script from it using Coded UI, which can be replayed to execute the tests next time. Source code of this lesson can be found at
https://github.com/chamindac/Book-Test-Autmation-VS/tree/master/Chapter%204/Lesson%204.05/CodedUIRecordPlayback

1. Create new coded UI project. You can see UIMap.cs file in Solution Explorer.
2. Open Coded UI test builder.
3. Click on the Start Recording button and perform actions you want to record (see Figure 4-15).

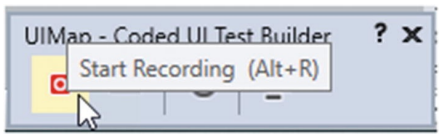

Figure 4-15 Click on the Start Recording button

4. Perform steps in the application under testing. After performing actions you want to record, you can click on the Pause Recording button, in the same location as the Start Recording button, to pause the recording.
5. The next step is to generate code to perform actions we recorded. To do that, click on the Generate Code icon of the Coded UI Test Builder. You can give a method name and add a description. After that, click on Add and Generate button to generate code (see Figure 4-16).

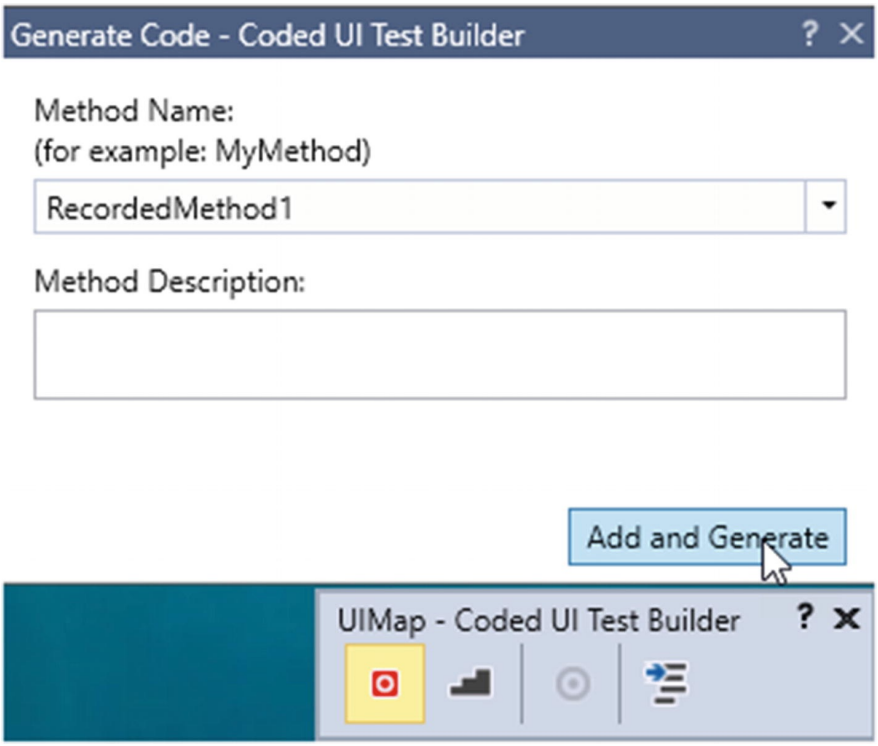

Figure 4-16 Generate Code

6. Now go and check Solution Explorer and you will find that UIMap.Designer.cs file has been added. Open this file and you will find recorded steps inside this class (see Figure 4-17).

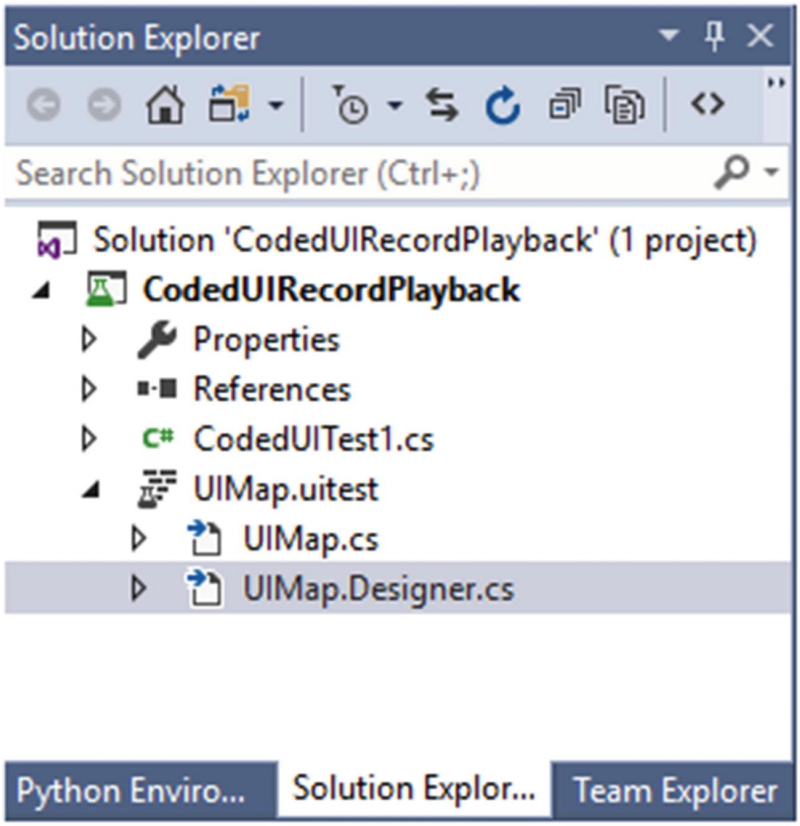

Figure 4-17 UIMap.Designer.cs file

Open the CodedUITest1.cs file and you will find a new line has been added to the test method.

```
[TestMethod]
public void CodedUITestMethod1()
{
```

```
        this.UIMap.RecordedMethod1();
    }
```

You can open UIMap.Designer class to see the complete recorded code. Execute this method to test the recorded scenario.

Even though record and playback of tests this way seems easy, it is not a recommended way to develop test automation code. Tests generated this way contain unwanted code steps and are not clean or easily understandable, causing issues in maintenance of test code.

Lesson 4.06: Automation Code Example with Coded UI with C#

So far, we have discussed controls and commands of Coded UI in previous lessons. With this lesson, you will be able to learn sample test code using Coded UI and C#. Source code of this lesson can be found at https://github.com/chamindac/Book-Test-Autmation-VS/tree/master/Chapter%204/Lesson%204.06/CodedUIDemo .

1. Create a text file in the pictures section of file explorer and give it the name “Currency.”
2. Create a folder in the pictures section of File Explorer and give it the name “Camera Roll.”

The following explained test method will open up the File Explorer and move to the pictures section. Then it will verify if the folder Camera Roll exists. Next it will search that the file named `Currency` is available by performing a search in Windows File Explorer.

All required reference assemblies should be added to the project. Then refer to the namespaces in the top of the class file.

```
using Microsoft.VisualStudio.TestTools.UITesting;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Keyboard = Microsoft.VisualStudio.TestTools.UITesting.Keyboard;
using Microsoft.VisualStudio.TestTools.UITesting.WinControls;
```

We are now ready to write the test code.

Initialize a new instance of the `WinWindow` class and provide property values of the controls that need to be found before performing actions. The first step should be to open the File Explorer window. To open the File Explorer, click on the File Explorer icon in the taskbar.

```
WinWindow win1 = new WinWindow();
win1.SearchProperties["ClassName"] = "MSTaskSwWClass";
win1.SearchProperties["ControlType"] = "Window";
WinToolBar toolBar = new WinToolBar(win1);
toolBar.SearchProperties["Name"] = "Running applications";
WinButton button = new WinButton(toolBar);
button.SearchProperties["Name"] = "File Explorer";
button.SearchProperties["ControlType"] = "Button";
Mouse.Click(button);
```

After opening the File Explorer window, click on the pictures tab of the side menu.

```
WinWindow win2 = new WinWindow();
win2.SearchProperties["ClassName"] = "CabinetWClass";
win2.SearchProperties["ControlType"] = "Window";
WinTreeItem winTreeItem = new WinTreeItem(win2);
winTreeItem.SearchProperties["ControlType"] = "TreeItem";
winTreeItem.SearchProperties["Name"] = "Pictures";
Mouse.Click(winTreeItem);
```

Verify the folder called “Camera Roll” is available.

```
WinListItem winListItem = new WinListItem(win2);
winListItem.SearchProperties["ControlType"] = "ListItem";
winListItem.SearchProperties["Name"] = "Camera Roll";
Assert.AreEqual(winListItem.DisplayText, "Camera Roll");
```

Type string “Currency” in the search field.

```
WinWindow searchArea = new WinWindow(win2);
searchArea.SearchProperties["ClassName"] = "DirectUIHWND";
WinPane quichAccess = new WinPane(searchArea);
quichAccess.SearchProperties["ControlType"] = "Pane";
quichAccess.SearchProperties["Name"] = " Search Quick access";
WinEdit winSearch = new WinEdit(quichAccess);
winSearch.SearchProperties["Name"] = "Search Box";
Keyboard.SendKeys(winSearch, "Currency");
```

Find the complete test code as follows.

```
[TestMethod]
```

```
public void CodedUITestMethod1()
{
    WinWindow win1 = new WinWindow();
    win1.SearchProperties["ClassName"] = "MSTaskSwWClass";
    win1.SearchProperties["ControlType"] = "Window";
    WinToolBar toolBar = new WinToolBar(win1);
    toolBar.SearchProperties["Name"] = "Running applications";
    WinButton button = new WinButton(toolBar);
    button.SearchProperties["Name"] = "File Explorer";
    button.SearchProperties["ControlType"] = "Button";
    Mouse.Click(button);
    WinWindow win2 = new WinWindow();
    win2.SearchProperties["ClassName"] = "CabinetWClass";
    win2.SearchProperties["ControlType"] = "Window";
    WinTreeItem winTreeItem = new WinTreeItem(win2);
    winTreeItem.SearchProperties["ControlType"] = "TreeItem";
    winTreeItem.SearchProperties["Name"] = "Pictures";
    Mouse.Click(winTreeItem);
    WinListItem winListItem = new WinListItem(win2);
    winListItem.SearchProperties["ControlType"] = "ListItem";
    winListItem.SearchProperties["Name"] = "Camera Roll";
    Assert.AreEqual(winListItem.DisplayText, "Camera Roll");
    WinWindow searchArea = new WinWindow(win2);
    searchArea.SearchProperties["ClassName"] = "DirectUIHWND";
    WinPane quichAccess = new WinPane(searchArea);
    quichAccess.SearchProperties["ControlType"] = "Pane";
    quichAccess.SearchProperties["Name"] = " Search Quick access";
    WinEdit winSearch = new WinEdit(quichAccess);
    winSearch.SearchProperties["Name"] = "Search Box";
    Keyboard.SendKeys(winSearch, "Currency");
}
```

In this lesson, we have learned how to write a simple Coded UI test to work with Windows applications.

Lesson 4.07: Using Sikuli for Windows App Testing

Sikuli is a GUI automation tool. It can be used to automate anything you see in the UI with image recognition mechanism. This can be used to automate both web and Windows applications. Mostly Sikuli is used to automate flash images and websites. We can introduce Sikuli as follows:

- It is an open source tool.
- Can automate Windows applications
- Can automate Desktop applications
- Can automate flash objects

Let's try to learn Sikuli with the sample code provided in this lesson. Source code of this lesson can be found at https://github.com/chamindac/Book-Test-Autmation-VS/tree/master/Chapter%204/Lesson%204.07/SikuliDemo .

1. In Visual Studio 2017, select Files ➤ New ➤ Project.
2. In the New Project pop-up window, select Test under Visual Studio C# and select Unit Test Project from the test project list. Give a **Name** for the Project, specify a **Solution Name**, select **Location,** and click on the OK button. Leaving **Create directory for solution** checked will allow you to have a new directory created for the new solution in the selected location.
3. Go to Tools ➤ NuGet Package Manager ➤ Manage NuGet Packages for Solution.
4. Install Sikuli4Net package (see Figure 4-18).

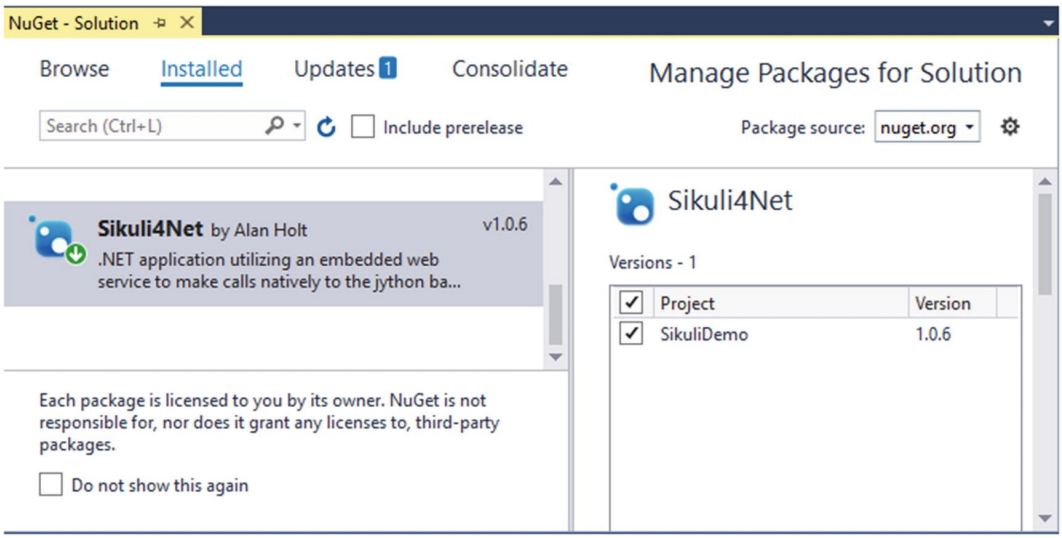

Figure 4-18 Install the Sikuli4Net package

Now we can write a sample code. With this code we are going to search for Notepad; select Notepad from the search results and verify Notepad is opened.

1. Open the UnitTest1.cs file.
2. Add reference files to the project as follows:

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Sikuli4Net.sikuli_REST;
```

3. Then create a `Screen` class object to access classes.

```
Screen screen = new Screen();
```

4. Then identify the patterns going to match. For example:

```
Pattern pattern1 = new Pattern(imagepath);
```

We have four patterns that we are going to match within this sample test. We can keep those patterns as follows. You can find the images mentioned in the following code at https://github.com/chamindac/Book-Test-Autmation-VS/tree/master/Chapter%204/Lesson%204.07/SikuliDemo . Download them and put them on your local drive folder and change the path of following code accordingly.

```
Pattern pattern_SearchImage = new Pattern("F:\\SikuliImages\\Sikuli-search.png");
Pattern pattern_searchTextField = new Pattern("F:\\SikuliImages\\sikuli-
searchfield.png");
Pattern pattern_searchResult = new Pattern("F:\\SikuliImages\\sikuli-searcresult.png");
Pattern pattern_notepad = new Pattern("F:\\SikuliImages\\sikuli-notepad.png");
```

The first pattern to match is the Windows search icon in the taskbar. Find the pattern and click on it using the following code lines:

```
screen.Find(pattern_SearchImage);
screen.Click(pattern_SearchImage);
```

Then enter a value to search.

```
screen.Find(pattern_searchTextField);
screen.Type(pattern_searchTextField,"NotePad");
```

Select the Notepad image among the search results.

```
screen.Find(pattern_searchResult);
screen.Click(pattern_searchResult);
```

Finally, verify the Notepad application is visible.

```
Assert.AreEqual(true,screen.Exists(pattern_notepad));
```

Find the complete code as follows:

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
```

```
using Sikuli4Net.sikuli_REST;

namespace SikuliDemo
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestMethod1()
        {
            Screen screen = new Screen();
            Pattern pattern_SearchImage = new Pattern("F:\\SikuliImages\\Sikuli-
search.png");
            Pattern pattern_searchTextField = new Pattern("F:\\SikuliImages\\sikuli-
searchfield.png");
            Pattern pattern_searchResult = new Pattern("F:\\SikuliImages\\sikuli-
searcresult.png");
            Pattern pattern_notepad = new Pattern("F:\\SikuliImages\\sikuli-
notepad.png");
            screen.Find(pattern_SearchImage);
            screen.Click(pattern_SearchImage);
            screen.Find(pattern_searchTextField);
            screen.Type(pattern_searchTextField,"NotePad");
            screen.Find(pattern_searchResult);
            screen.Click(pattern_searchResult);
            Assert.AreEqual(true,screen.Exists(pattern_notepad));
        }
    }
}
```

In this lesson, we have used the Sikuli framework to execute test automations, which is based on image matching.

Lesson 4.08: Using Winium for Windows App Testing

Winium is a Selenium-based open source automation framework for the Windows platform. This tool is similar to Selenium. You can find more info here: https://github.com/2gis/Winium .

You can download Winium.Desktop.Driver from this location: https://github.com/2gis/Winium.Desktop/releases .

Source code of this lesson can be found at https://github.com/chamindac/Book-Test-Autmation-VS/tree/master/Chapter%204/Lesson%204.08/Winium-Demo .

Let's try to test the Windows application using Winium with C# in this lesson.

1. In Visual Studio 2017, select Files ➤ New ➤ Project.
2. In the New Project pop-up window, select Test under Visual Studio C# and select Unit Test Project from the test project list. Give a **Name** for the Project, specify a **Solution Name**, select **Location** and Click on OK button. Letting **Create directory for solution** checked will allow you to have a new directory created for the new solution in the selected location.
3. Go to Tools ➤ NuGet Package Manager ➤ Manage NuGet Packages for Solution.
4. Add the following NuGet package, which is the only one required for the Desktop operations. The other two packages in Figure 4-19 are related to mobile applications.

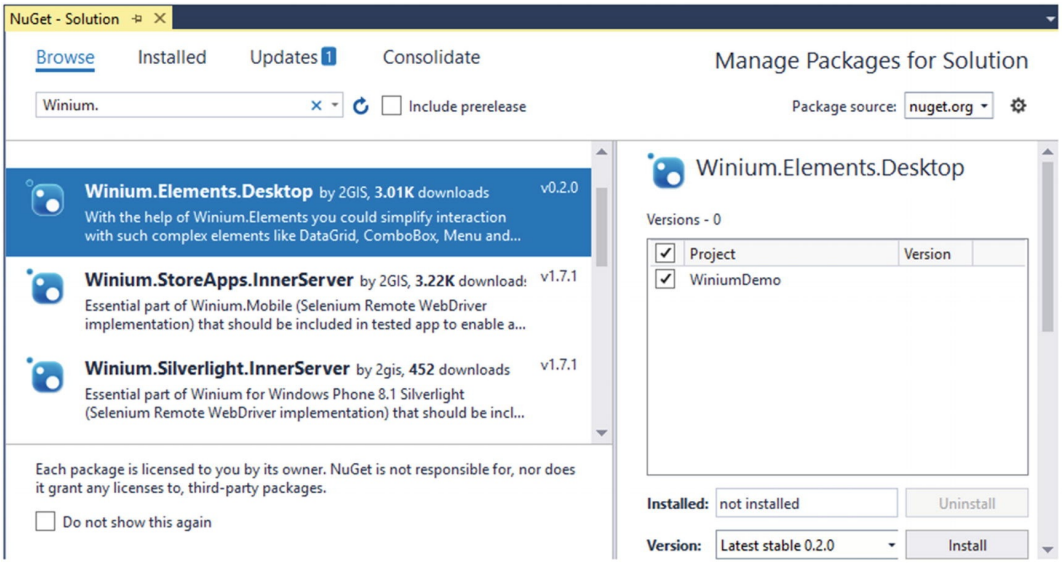

Figure 4-19 Install Winium.Elements.Desktop NuGet

Winium.Elements.Desktop (see Figure 4-19)

Now Visual Studio is ready to work with Winium. Let's write a simple code to learn how automate Windows apps with Winium.

To write an automation code we need to identify property values of each control. We can use Coded UI to identify UI property values.

In this lesson, let's try to write a simple test code. As a pre-requisite, create a folder called `BookWork` in the desktop. The test scenario is to open the File Explorer and select Desktop tab, then search for `BookWork` folder.

Add references to the test class file.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium.Winium;
```

Initiate the `DesktopOptions` instance and give the location of the application that we are going to test.

```
DesktopOptions opt = new DesktopOptions();
opt.ApplicationPath = @"C:\Windows\explorer.exe";
```

Then initiate the `WiniumDriver` instance by giving driver path and option. You have to copy the `WiniumDriver` downloaded as a prerequisite of this lesson to a local drive folder and provide the path in the following code instead of the hardcoded path used:

```
WiniumDriver driver = new
WiniumDriver(@"C:\Users\Acer\source\repos\UnitTestProject2\UnitTestProject2\bin\Debug",
opt);
```

Now File Explorer is opened. Select Desktop from the quick access area to search for it.

```
driver.FindElementByName("Desktop").Click();
```

Then type the word "BookWork" in the search field:

```
driver.FindElementByName("Search Box").Click();
driver.FindElementByName("Search Box").SendKeys("BookWork");
```

You can find the complete code as follows:

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium.Winium;

namespace Winium_Demo
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestMethod1()
        {
            DesktopOptions opt = new DesktopOptions();
            opt.ApplicationPath = @"C:\Windows\explorer.exe";
            WiniumDriver driver = new WiniumDriver(@"F:\Selenium-C#\Winium-Demo\Winium-
Demo\bin\Debug", opt);
            driver.FindElementByName("Desktop").Click();
            driver.FindElementByName("Search Box").Click();
            driver.FindElementByName("Search Box").SendKeys("BookWork");
        }
    }
}
```

In this lesson, you learned how to set up Visual Studio to work with Winium and start automation testing with Winium. Instead of Winium, it is recommended that you use a new Microsoft tool based on Selenium for Windows application test automation. This Windows application driver can be found at https://github.com/Microsoft/WinAppDriver and can be used similarly to Winium following the documentation available at GitHub.

Summary

This chapter mainly focused on introducing the Windows app test automation tools to readers. You explored how to start working with different tools that can be used to automate windows UI.

In the next chapter, we will discuss data management options for test automations.

Chaminda Chandrasekara and Pushpa Herath, *Hands-On Functional Test Automation*
https://doi.org/10.1007/978-1-4842-4411-1_5

5. Test Data Management in Functional Testing

Chaminda Chandrasekara[1] and Pushpa Herath[2]

(1) Dedigamuwa, Sri Lanka
(2) Hanguranketha, Sri Lanka

We have explored several options to write functional test automations in previous chapters. It is important to understand why we need to think about test data management strategy in designing test automations to understand the test data vitality in making the functional testing more effective.

- To avoid dependencies between tests: For example, to run the delete or update user test, you have to execute the Create User test first. This prevents execution of tests independently and blocks the ability to execute tests parallel and in distributed mode using multiple test clients. It is important to run automated tests independently and in parallel to allow fast execution and increased quality of the test results.
- To avoid using prerequisite test case code to create/manage required data: Creating test data using test code may increase test execution time significantly. If the test data is not created as planned, whole test suite may fail in the execution.
- Avoid the complexity in test code: Trying to implement test data creation with another set of tests cause complexity in tests and even in analysis of results. Test code would be messy and distinguishing between data creation tests and actual tests may become challenging over time.

Understanding what data are required to execute a system is important in defining a strategy for adding them as a prerequisite data to the system before executing test automations. The following types of data can be identified as required for functional test automations.

- Configuration data: Data that is required for the system to properly function. These data may be connection settings or system-level parameters defining or affecting the behavior of a given system.
- Reference data: Data that the system needs to enable the functionality to be executed. For example, this can be list of country names, states in a given country, etc.
- Scenario data: Data that is required to execute each individual test. For example, an opened and activated savings account should be available in a banking system to test the first cash deposit functionality in a test.
- Test Data: Data to be used in execution of the tests. For example, to test create user operation in a system, you may need collection of usernames and passwords.

In this chapter, let's explore the possibilities of using test data in a more robust way through hands-on lessons.

Lesson 5.01: Using External Data Sources to Keep Test Data

For keeping data that is required to execute tests, you can use external data sources such as an SQL database, an Excel file, or a csv file, etc. A configuration file can be used to keep the connection information of the data source and define the data source—for example, the table to be used.

Prerequisites: You must have completed Chapter 3 Lesson 3.05 and have the source code available to you from that lesson. However, complete source code from this lesson is available here: `https://github.com/chamindac/Book-Test-Autmation-VS/tree/master/Chapter%205/Lesson%205.01/UnitTestProject1/UnitTestProject1` . We will be using a sample application available on GitHub (`https://github.com/dotnet-architecture/eShopOnWeb`) to perform steps in this lesson. Set up the application as an Azure web app or host it in IIS following the instructions available on GitHub.

1. Open the Solution from Chapter 3 Lesson 3.05 in Visual Studio. The test written in that example is adding a new user to an eShop Web Application. However, the username and password are hardcoded in the test code in this example.
2. Let's create a list of usernames and passwords in an SQL Database table to be used as test data. Open the SQL Server Object Explorer in Visual Studio and connect to `(localdb)\MSSQLLocalDB`. Then create a new database named `TestData` and add a table named `Users` with two varchar (50) fields. The `UserName` field as primary key and password field should be the two fields.
3. Add a few usernames such as those shown in Figure 5-1 to the `Users` table created.

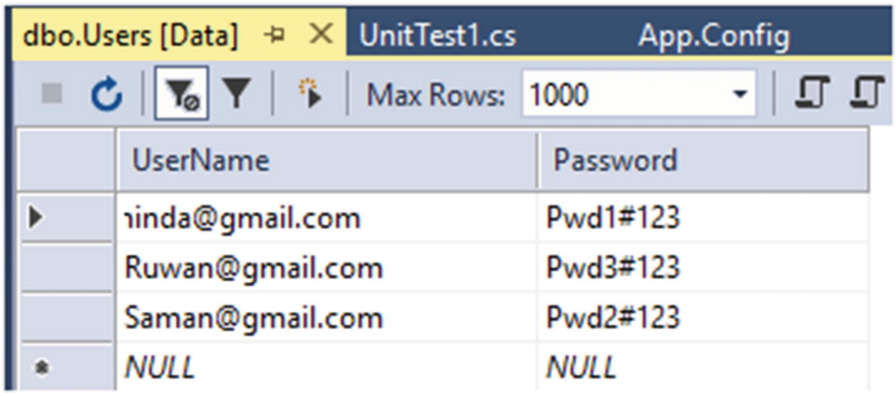

Figure 5-1 Data in SQL Table

4. Add an `App.config` file to the unit test project and add the following content to it. In the connection string, the database name is set to `TestData` and integrated security is used with the local SQL database server available in Visual Studio. In the data source, the name is given as `MyDataSource` and the table name of the `Users` table is specified as `dataTableName`. Setting `Sequential` to the data access method allows the execution of the test sequentially for each data row found in the table.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="microsoft.visualstudio.testtools"
type="Microsoft.VisualStudio.TestTools.UnitTesting.TestConfigurationSection,
Microsoft.VisualStudio.TestPlatform.TestFramework.Extensions" />
  </configSections>
  <connectionStrings>
    <add name="MyDataCon" connectionString="Data Source=(localdb)\MSSQLLocalDB;Initial
Catalog=TestData;Integrated Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=True;ApplicationIntent=ReadWrite;MultiSubnetFailover=False"
providerName="System.Data.SqlClient"  />
  </connectionStrings>
  <microsoft.visualstudio.testtools>
    <dataSources>
      <add name="MyDataSource" connectionString="MyDataCon" dataTableName="Users"
dataAccessMethod="Sequential"/>
    </dataSources>
  </microsoft.visualstudio.testtools>
</configuration>
```

5. Add the `System.Data` assembly reference to the `UnitTest` project. Open the `UnitTest1.cs` and add the following code to the `UnitTest1` class. This `TestContext` will be used to retrieve the data when the tests are executed.

```
private TestContext context;

public TestContext TestContext
{
       get { return context; }
       set { context = value; }
}
```

6. Add the new attribute `DataSource` to the `AddNewUser` method available in the `UnitTest` class .

```
[DataSource("MyDataSource")]
```

7. Replace the hardcoded username and passwords in `AddNewUser` method so that the complete code of the class are similar to following.

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.Support.UI;
using ExpectedConditions = SeleniumExtras.WaitHelpers.ExpectedConditions;

namespace UnitTestProject1
{
    [TestClass]
    public class UnitTest1
    {
        private TestContext context;

        public TestContext TestContext
        {
            get { return context; }
            set { context = value; }
        }

        [TestMethod]
        [DataSource("MyDataSource")]
        public void AddNewUser()
        {
            string userEmail = context.DataRow["UserName"].ToString();
            string userpassword = context.DataRow["Password"].ToString();
            ChromeOptions option = new ChromeOptions();
            option.AddArgument("--start-maximized");
            IWebDriver driver = new ChromeDriver(option);
            driver.Navigate().GoToUrl("http://eshop-testweb.azurewebsites.net/");
            WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30));
            IWebElement loginLink = wait.Until(ExpectedConditions.ElementIsVisible(By.LinkText("Login")));
            loginLink.Click();
            IWebElement registerNewUser =
wait.Until(ExpectedConditions.ElementToBeClickable(By.LinkText("Register as a new user?")));
            registerNewUser.Click();
            IWebElement emailField = wait.Until(ExpectedConditions.ElementToBeClickable(By.Id("Email")));
            emailField.SendKeys(userEmail);
            IWebElement passWordField =
wait.Until(ExpectedConditions.ElementToBeClickable(By.Id("Password")));
            passWordField.SendKeys(userpassword);
            IWebElement confirmPassWordField =
wait.Until(ExpectedConditions.ElementToBeClickable(By.Id("ConfirmPassword")));
            confirmPassWordField.SendKeys(userpassword);
            IWebElement registerButton =
```

```
wait.Until(ExpectedConditions.ElementToBeClickable(By.CssSelector(".btn.btn-default.btn-brand.btn-brand-
big")));
            registerButton.Submit();
            IWebElement myAccountLink = wait.Until(ExpectedConditions.ElementToBeClickable(By.LinkText("My
account")));
            myAccountLink.Click();
            Assert.AreEqual(true,
wait.Until(ExpectedConditions.TextToBePresentInElementValue(By.Id("Email"), userEmail)));

            driver.Quit();
        }
    }
}
```

8. Notice the two statements reading the data from test context at the beginning of the method, which are shown in the following code. Then the variables `userEmail` and `userPassword` are used in the rest of the code.

```
string userEmail = context.DataRow["UserName"].ToString();
            string userpassword = context.DataRow["Password"].ToString();
```

9. Build the `UnitTestProject1` and execute the test `AddNewUser` from the Test Explorer window in Visual Studio. You will see the test get executed for the number of times equal to the number of data rows in the `Users` table in the `TestData` table we have created (see Figure 5-2).

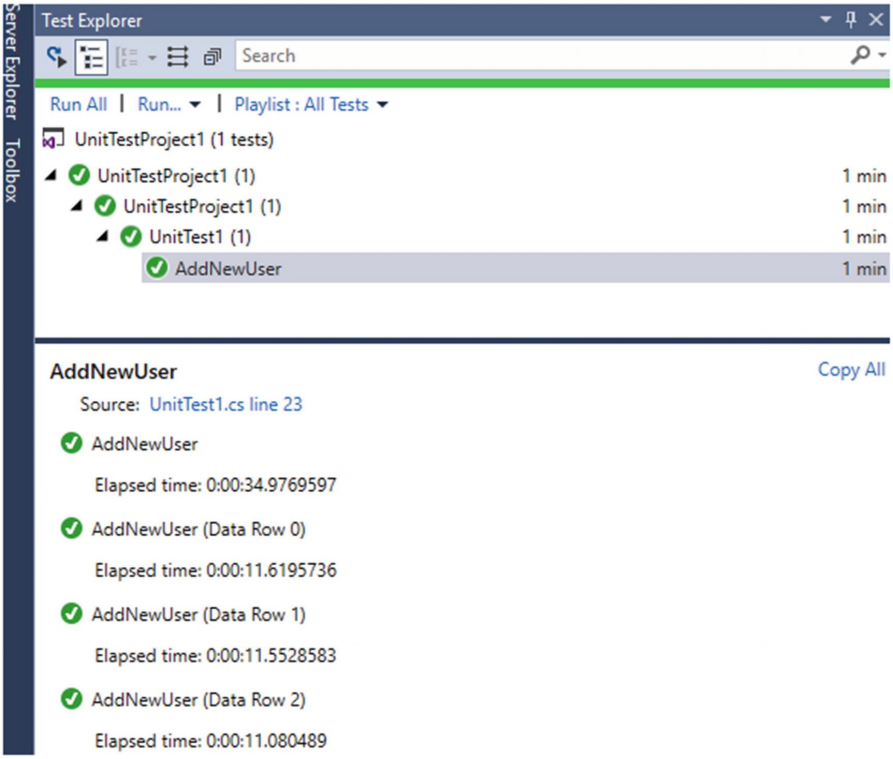

Figure 5-2 Executed test for data available in the Users table

In this lesson, we have discussed the capability of using external data sources for keeping the test data. You may use try different external data sources such as csv files or Microsoft Access Database, etc. for the purpose of keeping test data, and set up the data source similarly to what we have done in the lesson.

Lesson 5.02: Cleaning Test Data After Test Execution

In the previous lesson we discussed the capability of using external data sources to store test data. However, if you try to run the `AddNewUser` test without changing the test data for the second time, the test will fail, as it is not allowed to register the same username twice in the example eShop Web application. This shows the importance of cleaning up the test data at the end of a test execution.

To clean up the test data, you must have a good understanding of the system database structure and the data in it. An alternative option would be to use an API or the UI of the system to remove the registered users in this scenario. However, the most efficient way would be to remove the users directly from the database, which would minimize the test execution time. Depending on the complexity of the system under testing, you should decide on an option of test data clean-up using a database or APIs of the system or UI of the system. Remembering the option of using database scripts to clean up is the most efficient, but it is the riskier implementation. API is a good choice, if available. However, using UI via a set of another set of selenium test steps should be the last option and should be avoided as much as possible, as it would add additional time to test execution and make the tests messy, unmaintainable, and complex.

Prerequisites: You must have completed the previous lesson of this chapter and have the source code available to you from that lesson. However, complete source code from this lesson is available here: `https://github.com/chamindac/Book-Test-Autmation-VS/tree/master/Chapter%205/Lesson%205.02/UnitTestProject1` . We will be using a sample application available in GitHub (`https://github.com/dotnet-architecture/eShopOnWeb`) to perform steps in this lesson. Set up the application as an Azure web app or host it in IIS following the instructions available on GitHub.

Let's try to remove the newly created user just after the `AddNewUser` test is executed to support execution of it using the same test data any number of times.

1. Open the Solution from the previous lesson of this chapter in the Visual Studio.
2. Modify the `App.Config` of the `UnitTestProject1` to include the following line in the `ConnectionsStrings` section. Make sure to replace the

database server name, user id, and the password of the SQL Server where you have set up the eShop Web Application databases.

```
<add name="SystemIdentityCon" connectionString="Data Source=databaseserver;Initial
Catalog=eshopidentity;User ID=userid;Password=userpassword;Connect
Timeout=60;Encrypt=True;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False"
providerName="System.Data.SqlClient"  />
```

3. The complete `App.App` configuration should be similar to the following:.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="microsoft.visualstudio.testtools"
type="Microsoft.VisualStudio.TestTools.UnitTesting.TestConfigurationSection,
Microsoft.VisualStudio.TestPlatform.TestFramework.Extensions" />
  </configSections>
  <connectionStrings>
    <add name="MyDataCon" connectionString="Data Source=(localdb)\MSSQLLocalDB;Initial
Catalog=TestData;Integrated Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=True;ApplicationIntent=ReadWrite;MultiSubnetFailover=False"
providerName="System.Data.SqlClient"  />
    <add name="SystemIdentityCon" connectionString="Data Source=databaseserver;Initial
Catalog=eshopidentity;User ID=userid;Password=userpassword;Connect
Timeout=60;Encrypt=True;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False"
providerName="System.Data.SqlClient"  />
  </connectionStrings>
  <microsoft.visualstudio.testtools>
    <dataSources>
      <add name="MyDataSource" connectionString="MyDataCon" dataTableName="Users"
dataAccessMethod="Sequential"/>
    </dataSources>
  </microsoft.visualstudio.testtools>
</configuration>
```

4. Create a new folder in `UnitTestProject1` with the name `CleanupScripts`. Then add a new SQL file named `RemovenewUser.sql` to that folder. Open the SQL file and add the following script to it to enable deleting a given user. In the properties window of Visual Studio for the SQL file, select Copy always in the Copy to Output Directory property.

```
delete from dbo.AspNetUsers where username = '{0}'
```

5. Add `System.Configuration` as a reference to `UnitTestProject1`. Then add the following using statements in the `UnitTest1.cs` file.

```
using System.Configuration;
using System.Data.SqlClient;
using System.IO;
```

6. Add the following method to the `UnitTest1` class to enable cleanup after each test method in the class. The attribute `TestCleanup` triggers this method after each test method execution. Method retrieves the username to delete from the test context data. Then load the script from the SQL file and replace the username to delete in the file. Using the SQL connection created referring to the connection string of the identity database of the eShop Web Application, a command is executed to delete the newly added user.

```
[TestCleanup()]
public void Cleanup_RemoveNewUser()
{
            string userToDelete = context.DataRow["UserName"].ToString();

            string deleteUserScript = File.ReadAllText(@".\CleanupScripts\RemovenewUser.sql");
            deleteUserScript = string.Format(deleteUserScript, userToDelete);

            using (SqlConnection conn = new
SqlConnection(ConfigurationManager.ConnectionStrings["SystemIdentityCon"].ConnectionString))
            {
                conn.Open();

                using (var cmd = new SqlCommand())
                {
                    cmd.Connection = conn;
                    cmd.CommandType = System.Data.CommandType.Text;
                    cmd.CommandText = deleteUserScript;
                    cmd.ExecuteNonQuery();
                }
            }
}
```

7. Complete code of the `UnitTest1.cs` file should be similar to the following:

```
using System;
using System.Configuration;
using System.Data.SqlClient;
using System.IO;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.Support.UI;
using ExpectedConditions = SeleniumExtras.WaitHelpers.ExpectedConditions;

namespace UnitTestProject1
```

```
{
    [TestClass]
    public class UnitTest1
    {
        private TestContext context;

        public TestContext TestContext
        {
            get { return context; }
            set { context = value; }
        }

        [TestMethod]
        [DataSource("MyDataSource")]
        public void AddNewUser()
        {
            string userEmail = context.DataRow["UserName"].ToString();
            string userpassword = context.DataRow["Password"].ToString();
            ChromeOptions option = new ChromeOptions();
            option.AddArgument("--start-maximized");
            IWebDriver driver = new ChromeDriver(option);
            driver.Navigate().GoToUrl("http://eshop-testweb.azurewebsites.net/");
            WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30));
            IWebElement loginLink = wait.Until(ExpectedConditions.ElementIsVisible(By.LinkText("Login")));
            loginLink.Click();
            IWebElement registerNewUser =
wait.Until(ExpectedConditions.ElementToBeClickable(By.LinkText("Register as a new user?")));
            registerNewUser.Click();
            IWebElement emailField = wait.Until(ExpectedConditions.ElementToBeClickable(By.Id("Email")));
            emailField.SendKeys(userEmail);
            IWebElement passWordField =
wait.Until(ExpectedConditions.ElementToBeClickable(By.Id("Password")));
            passWordField.SendKeys(userpassword);
            IWebElement confirmPassWordField =
wait.Until(ExpectedConditions.ElementToBeClickable(By.Id("ConfirmPassword")));
            confirmPassWordField.SendKeys(userpassword);
            IWebElement registerButton =
wait.Until(ExpectedConditions.ElementToBeClickable(By.CssSelector(".btn.btn-default.btn-brand.btn-brand-
big")));
            registerButton.Submit();
            IWebElement myAccountLink = wait.Until(ExpectedConditions.ElementToBeClickable(By.LinkText("My
account")));
            myAccountLink.Click();
            Assert.AreEqual(true,
wait.Until(ExpectedConditions.TextToBePresentInElementValue(By.Id("Email"), userEmail)));

            driver.Quit();
        }

        [TestCleanup()]
        public void Cleanup_RemoveNewUser()
        {
            string userToDelete = context.DataRow["UserName"].ToString();

            string deleteUserScript = File.ReadAllText(@".\CleanupScripts\RemovenewUser.sql");
            deleteUserScript = string.Format(deleteUserScript, userToDelete);

            using (SqlConnection conn = new
SqlConnection(ConfigurationManager.ConnectionStrings["SystemIdentityCon"].ConnectionString))
            {
                conn.Open();

                using (var cmd = new SqlCommand())
                {
                    cmd.Connection = conn;
                    cmd.CommandType = System.Data.CommandType.Text;
                    cmd.CommandText = deleteUserScript;
                    cmd.ExecuteNonQuery();
                }
            }
        }
    }
}
```

8. Now you can run the `AddNewUser` test multiple times with the same data in the Users table of the `TestData` database, as the code is making sure the test data used in the test is immediately removed after the test execution.

This lesson helped you to understand the usage of the `TestCleanup` method to effectively use the test data and get them cleaned up after the test execution in functional test automation.

Lesson 5.03: Making the Test Data Cleanup Robust

In the previous lesson, we discussed how to clean up the test data being used in a test execution. However, is that implementation a robust one? To check this and make the previous lesson test cleanup code more robust and stable is the purpose of this lesson.

Prerequisites: You must have completed the previous lesson of this chapter and have the source code available to you from that lesson. However, complete source code of this lesson is available here: https://github.com/chamindac/Book-Test-Autmation-VS/tree/master/Chapter%205/Lesson%205.03/UnitTestProject1 . We will be using a sample application available on GitHub (

https://github.com/dotnet-architecture/eShopOnWeb) to perform the steps in this lesson. Set up the application as an Azure web app or host it in IIS following the instructions available on GitHub.

As we have discussed, the `TestCleanup` attribute will create the `Cleanup_RemoveNewUser` method called after each test method execution. If we add a test method that is not adding a new user to the system (eShop Web Application) we might have a potential failure in the test data cleanup. Let's try to add another test method to the test class in the previous lesson's code to check this scenario and find a proper solution if this problem exists.

1. Open the Solution from the previous lesson of this chapter in the Visual Studio.
2. Add the following new test method to `UnitTest1` class, which is evaluating the assert to true with the intention of passing the test on all scenarios.

```
[TestMethod]
public void TestMethod2()
{
        Assert.AreEqual(1, 1);
}
```

3. Build the project and execute all tests from the Test Explorer window.
4. As expected, the newly added test method fails in the cleanup attempt, as no new user is added, which was the expected case in the previous lesson implementation of test data cleanup. The error is thrown because the test context has no data row, as expected in the cleanup, for the newly added test method (see Figure 5-3).

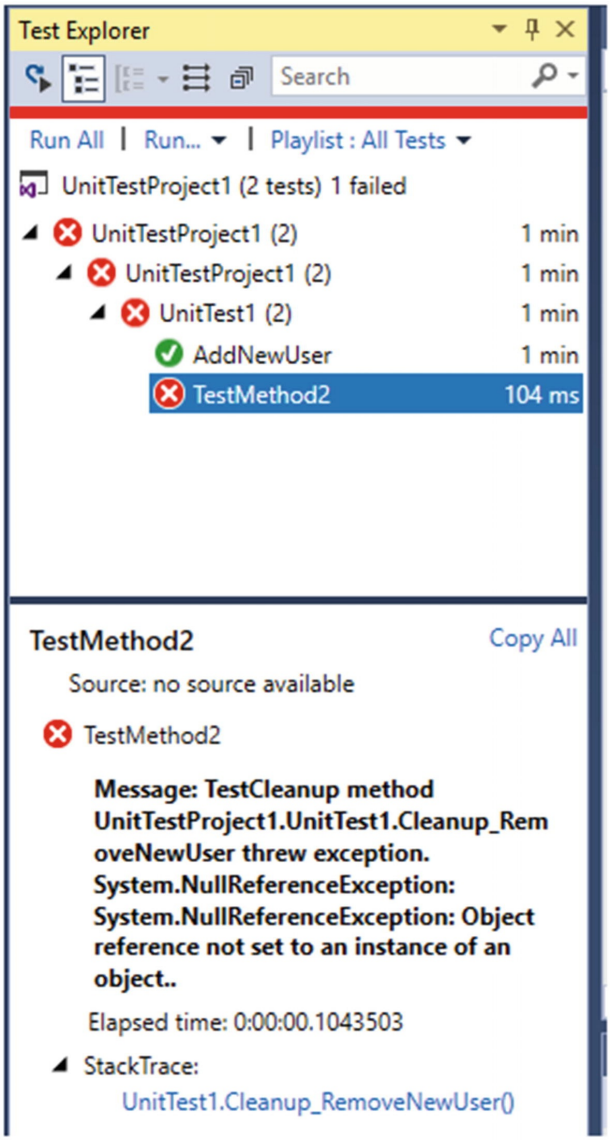

Figure 5-3 Failing in the cleanup

5. Let's get this issue fixed so that we can implement test data cleanup in a robust way.
6. As the first step, add the following using a statement to the `UnitTest1.cs` to allow the use of generic lists in C#.

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

7. Then add a new private variable type of `List<Action>` to the `Unittest1` class. We are going to add cleanup methods as an action list in the test class.

```
private List<Action> TestCleanupMethods = new List<Action>();
```

8. Remove the `TestCleanup` attribute from the `Cleanup_RemoveNewUser` method . Make the `userToDelete` a parameter for the method. Complete method code should be as shown here after modification.

```
public void Cleanup_RemoveNewUser(string userToDelete)
        {
            string deleteUserScript = File.ReadAllText(@".\CleanupScripts\RemovenewUser.sql");
            deleteUserScript = string.Format(deleteUserScript, userToDelete);

            using (SqlConnection conn = new
SqlConnection(ConfigurationManager.ConnectionStrings["SystemIdentityCon"].ConnectionString))
            {
                conn.Open();

                using (var cmd = new SqlCommand())
                {
                    cmd.Connection = conn;
                    cmd.CommandType = System.Data.CommandType.Text;
                    cmd.CommandText = deleteUserScript;
                    cmd.ExecuteNonQuery();
                }
            }
        }
```

9. Add the following statement to the `AddNewuser` test method just after retrieving test data from the test context (see Figure 5-4). The statement will add remove method with user e-mail value as a parameter to the list of actions we have created as a private variable in the UnitTest1 class. You can even consider placing this statement just after adding the new user to the eShop Web Application. The script we have written to delete a user will not return a failure even if we do not have a user to delete. Hence, placing the following statement just after retrieving test data also should be fine.

```
TestCleanupMethods.Add(() => Cleanup_RemoveNewUser(userEmail));
```

```
[TestMethod]
[DataSource("MyDataSource")]
public void AddNewUser()
{
    string userEmail = context.DataRow["UserName"].ToString();
    string userpassword = context.DataRow["Password"].ToString();

    TestCleanupMethods.Add(() => Cleanup_RemoveNewUser(userEmail));

    ChromeOptions option = new ChromeOptions();
```

Figure 5-4 Add cleanup method to list of actions

10. As shown here, you can define any number of test data cleanup methods and add them to the list, as appropriate, in the relevant test methods. Even one test method may have multiple cleanup method requirements based on the complexity of the test method.
11. Then add a cleanup all method that will be calling the cleanup methods in the action list. Reverse ordering the list helps to execute the last added cleanup method first, which would be an effective solution when the test data required to be cleaned up are in a relational database and have multiple cleanup methods with dependencies. Further, this implementation will allow calling individual cleanup methods when they are added to action list. This makes the newly added test method `TestMethod2` execute without failing, as it is not adding any cleanup method to the action list.

```
[TestCleanup]
public void Cleanup_All()
{
   TestCleanupMethods.Reverse();

   foreach (var cleanupMethod in TestCleanupMethods)
   {
      cleanupMethod();
   }
}
```

12. The complete code of `Unittest1.cs` should be as follows after completing all the aforementioned steps.

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data.SqlClient;
using System.IO;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.Support.UI;
using ExpectedConditions = SeleniumExtras.WaitHelpers.ExpectedConditions;

namespace UnitTestProject1
{
    [TestClass]
    public class UnitTest1
    {
        private TestContext context;
        private List<Action> TestCleanupMethods = new List<Action>();

        public TestContext TestContext
        {
            get { return context; }
```

```
            set { context = value; }
        }

        [TestMethod]
        [DataSource("MyDataSource")]
        public void AddNewUser()
        {
            string userEmail = context.DataRow["UserName"].ToString();
            string userpassword = context.DataRow["Password"].ToString();

            TestCleanupMethods.Add(() => Cleanup_RemoveNewUser(userEmail));

            ChromeOptions option = new ChromeOptions();
            option.AddArgument("--start-maximized");
            IWebDriver driver = new ChromeDriver(option);
            driver.Navigate().GoToUrl("http://eshop-testweb.azurewebsites.net/");
            WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30));
            IWebElement loginLink =
wait.Until(ExpectedConditions.ElementIsVisible(By.LinkText("Login")));
            loginLink.Click();
            IWebElement registerNewUser =
wait.Until(ExpectedConditions.ElementToBeClickable(By.LinkText("Register as a new user?")));
            registerNewUser.Click();
            IWebElement emailField = wait.Until(ExpectedConditions.ElementToBeClickable(By.Id("Email")));
            emailField.SendKeys(userEmail);
            IWebElement passWordField =
wait.Until(ExpectedConditions.ElementToBeClickable(By.Id("Password")));
            passWordField.SendKeys(userpassword);
            IWebElement confirmPassWordField =
wait.Until(ExpectedConditions.ElementToBeClickable(By.Id("ConfirmPassword")));
            confirmPassWordField.SendKeys(userpassword);
            IWebElement registerButton =
wait.Until(ExpectedConditions.ElementToBeClickable(By.CssSelector(".btn.btn-default.btn-brand.btn-brand-
big")));
            registerButton.Submit();
            IWebElement myAccountLink =
wait.Until(ExpectedConditions.ElementToBeClickable(By.LinkText("My account")));
            myAccountLink.Click();
            Assert.AreEqual(true,
wait.Until(ExpectedConditions.TextToBePresentInElementValue(By.Id("Email"), userEmail)));

            driver.Quit();
        }

        [TestMethod]
        public void TestMethod2()
        {
            Assert.AreEqual(1, 1);
        }

        [TestCleanup]
        public void Cleanup_All()
        {
            TestCleanupMethods.Reverse();

            foreach (var cleanupMethod in TestCleanupMethods)
            {
                cleanupMethod();
            }
        }

        public void Cleanup_RemoveNewUser(string userToDelete)
        {
            string deleteUserScript = File.ReadAllText(@".\CleanupScripts\RemovenewUser.sql");
            deleteUserScript = string.Format(deleteUserScript, userToDelete);

            using (SqlConnection conn = new
SqlConnection(ConfigurationManager.ConnectionStrings["SystemIdentityCon"].ConnectionString))
            {
                conn.Open();

                using (var cmd = new SqlCommand())
                {
                    cmd.Connection = conn;
                    cmd.CommandType = System.Data.CommandType.Text;
                    cmd.CommandText = deleteUserScript;
                    cmd.ExecuteNonQuery();
                }
            }
        }
    }
}
```

13. With this implementation, both the tests should execute fine and should clean the new user added with the `AddnewUser` test (see Figure 5-5). So, this implementation allows the tests to be executed with the test data any number of times, in a robust and stable manner.

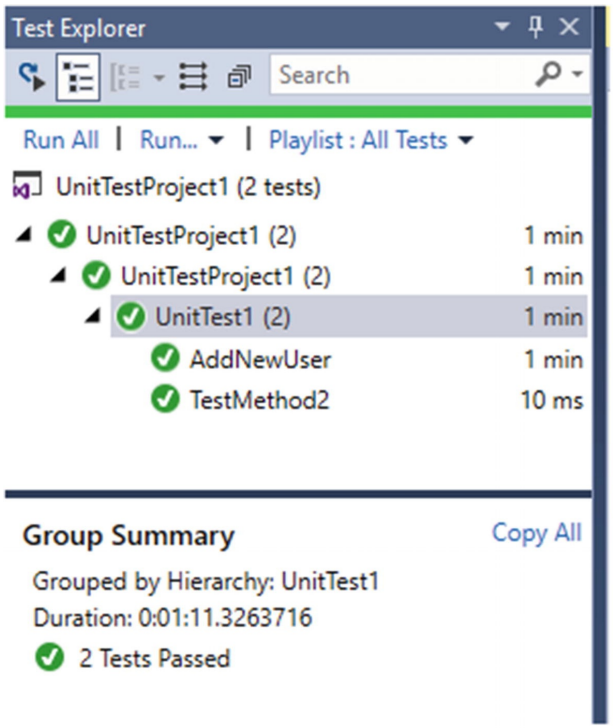

Figure 5-5 Both tests passed

We have made the test data cleanup we have implemented in the previous lesson robust and stable in this lesson.

Summary

In this chapter, we have focused on test data management and cleanup test data after execution of tests with hands-on lessons.

For handling configuration reference data, it is advisable to use database scripts, to populate them, and to have a pre-test check method implemented to validate if all data required is populated in the system databases. You could even opt to manually add these data to the system via system UI, and then keep a backup of the database and restore it before the execution to make the tests run on pre-prepared environment. However, this implementation would be challenging in a situation where the system is undergoing bigger changes even affecting the database schemas.

For handling test scenario data, it is recommended that you use scripts to add required data to the database directly or use application system APIs if available. You should try to avoid using Test Initialize to implement the call to test scenario data adding, as it will be a messy implementation if there are multiple tests written in the same test class. Rather you should use the actual test method to call the scripts to generate the test data required for the scenario. This is easier to implement in systems when they mature; however, it would be a challenging to implement scenario-based test data creation in rapidly changing applications.

In the next chapter, we will discuss the possibilities of integrating test automation code into CI/CD pipelines to make the execution of tests to happen in a fully automated manner.

Chaminda Chandrasekara and Pushpa Herath, *Hands-On Functional Test Automation*
https://doi.org/10.1007/978-1-4842-4411-1_6

6. Integrating Functional Testing to Deployment Pipelines

Chaminda Chandrasekara[1] and Pushpa Herath[2]

(1) Dedigamuwa, Sri Lanka
(2) Hanguranketha, Sri Lanka

Tests that are developed as we have discussed in previous chapters should be able to run against the application being tested. The application being tested may be deployed automatically using automated deployment pipelines. Depending on the deployed application environment (such as Development Integration, Quality Assurance, User Acceptance, Staging or Production, etc.), tests should be able to run against the relevant environment. Hence, the automated tests have to be integrated with the deployment pipelines. This integration of test automations with deployment pipelines enable execution of tests targeting the deployed application environment.

To execute the tests, test client machines should be set up, and these client machines can be virtual machines. It is effective to set up such test client machines in Cloud-based infrastructure using Cloud infrastructure as a service, such as Azure or Amazon Cloud infrastructure services. Starting the required test client machines on demand to execute tests will help to reduce costs.

In this chapter, you will be exploring setting up automated test executions with Azure DevOps pipelines. To gain more knowledge in Azure DevOps pipelines, you can refer to the book *Beginning Build and Release Management with TFS 2017 and VSTS: Leveraging Continuous Delivery for Your Business* (https://www.amazon.com/Beginning-Build-Release-Management-2017/dp/1484228103).

Lesson 6.01: Set Up Agent Pools

Agent pools in Azure DevOps help to group a pool of agents (you can understand a pool of test clients, as our focus is on test automation). This grouping helps to isolate test clients relevant to a given application or a project. You can learn more about organization and project level agent pools, roles, and permissions in the article available at the link https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/pools-queues?view=vsts . In this lesson, let's understand how we can set up an agent pool in Azure DevOps.

Prerequisites: You must have working knowledge of Azure DevOps. You need an account created with all Azure DevOps features or Azure DevOps pipelines using Azure DevOps services (https://azure.microsoft.com/en-us/services/devops/).

Setting Up Organization Agent Pools

Let's begin with setting up an organization level agent pool in Azure DevOps.

1. At the Azure DevOps homepage, click on Organization Settings. Then in the Pipelines tab, click on Agent Pools.
2. Click on New agent pool (see Figure 6-1).

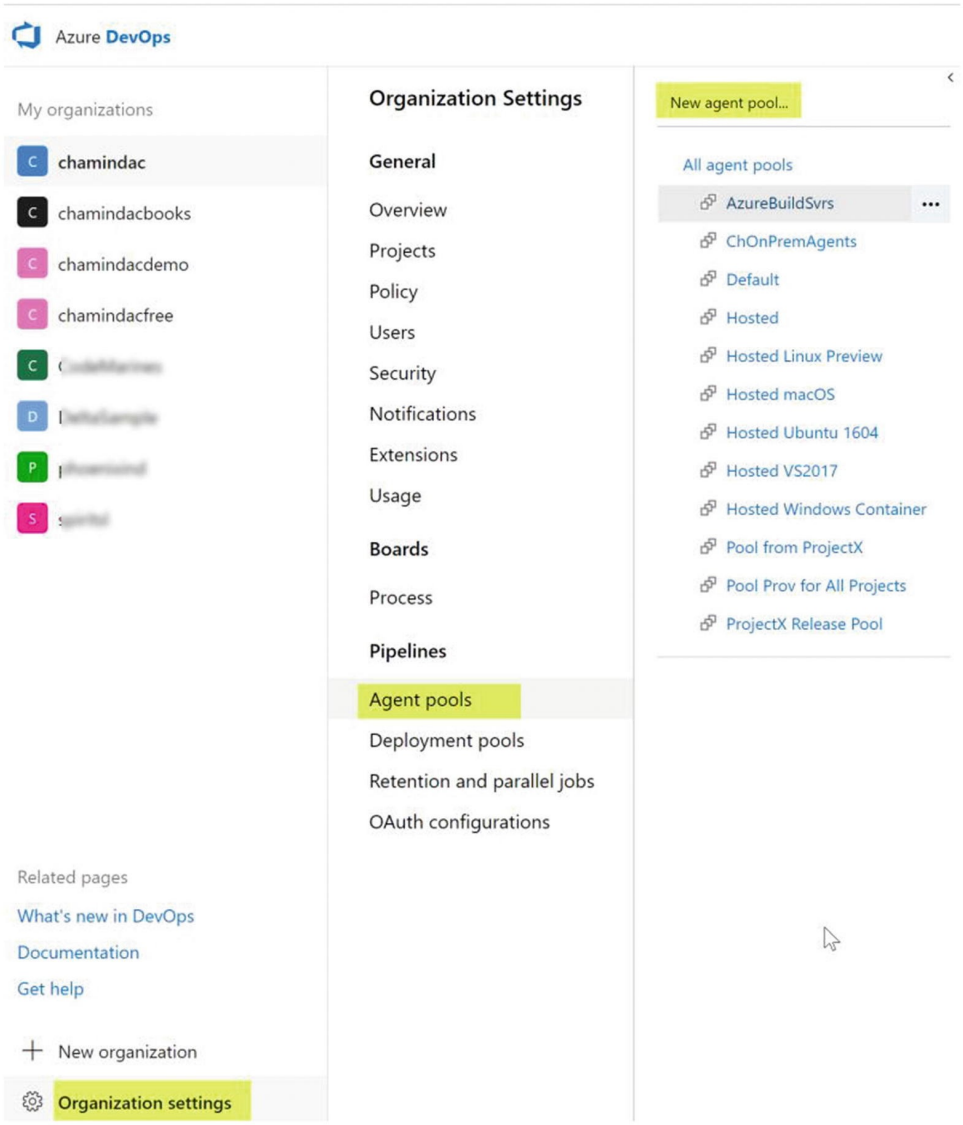

Figure 6-1 New agent pool

3. In the "Create an organization agent pool" window provide a name for the pool (see Figure 6-2). Make sure to uncheck the "Auto-provision corresponding agent pools in all projects" as we are going to create the required project level agent pools in the second part of this lesson. A queue is available for each pool, which helps to keep jobs for agents in the pool queued, until an agent is available to execute the job (see Figure 6-2).

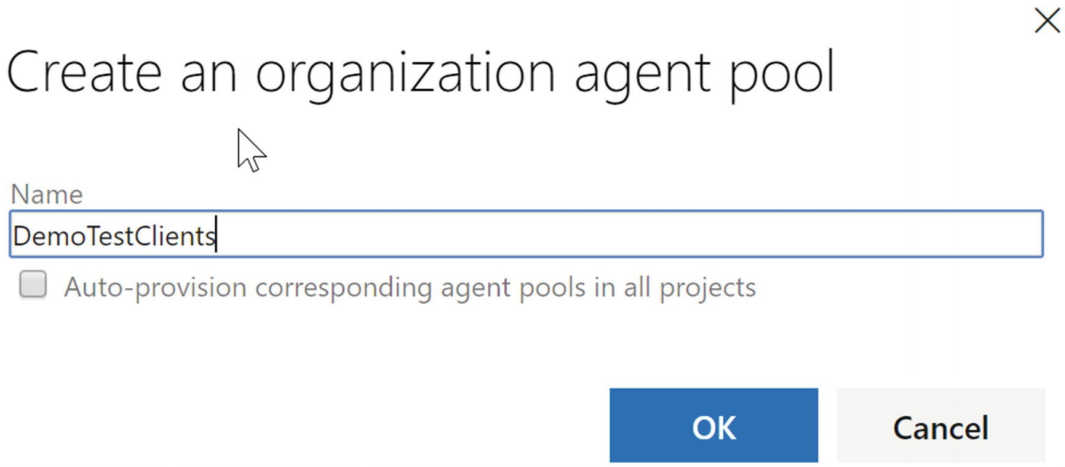

Figure 6-2 Creating a new agent pool

We have created an organization level agent pool, and as the next step, let's create a team project level agent pool using the created organization level agent pool.

Setting Up Project Agent Pools

Project agent pool helps to isolate an agent to a given Azure DevOps team project A team project in Azure DevOps allows a set of developers to work together to build applications as a team. It comes with Code repositories and planning and tracking tools as well as tools to implement continuous delivery pipelines. More information about creating a team project can be found at https://docs.microsoft.com/en-us/azure/devops/organizations/projects/create-project .

1. At the Azure DevOps homepage, click on team project to navigate to team project homepage.
2. Then click on project settings ➤ Agent pools in the Pipelines tab ➤ New agent pool (see Figure 6-3).

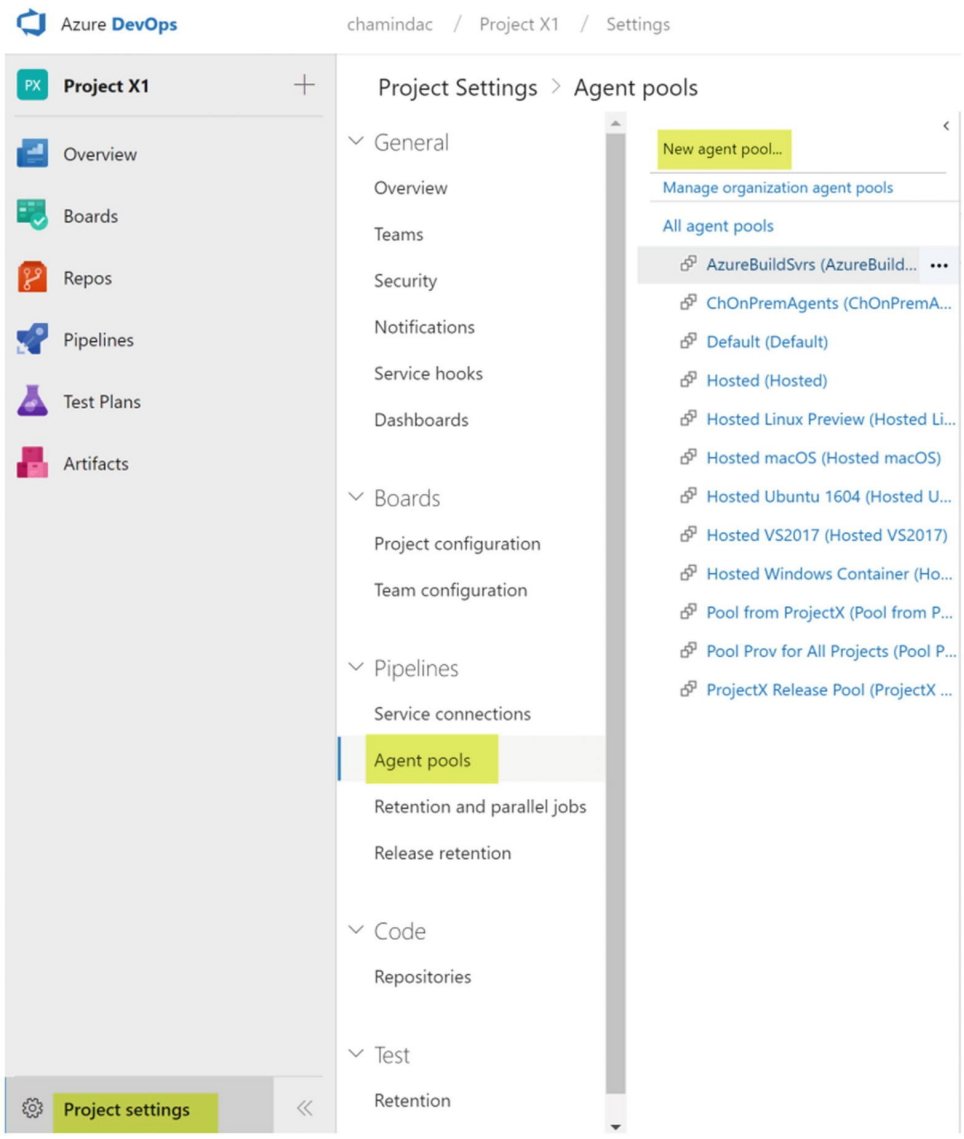

Figure 6-3 Project agent pool

3. In the pop-up window, select the "Base it on an existing organization agent pool" option and select the Organization Agent pool created earlier. Then click OK to create a project agent pool (see Figure 6-4).

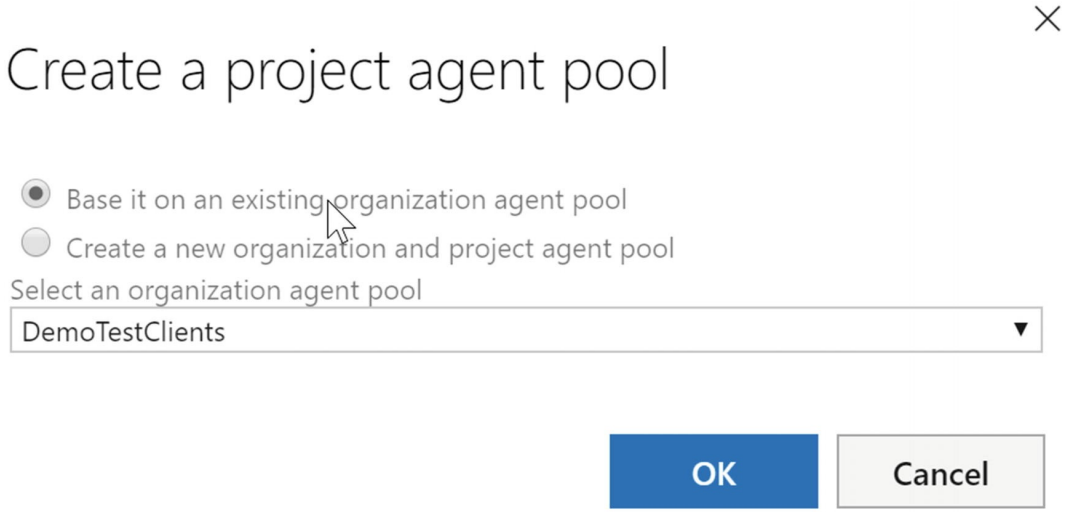

Figure 6-4 Create a project agent pool

Instead of following the two-step process mentioned earlier, you can create a project agent pool directly by selecting the option to "Create a new organization and project agent pool." A new project agent will be created with a new organization agent pool and the other projects will not get the pool created in this direct approach.

This lesson helped you to understand how to create an agent pool in Azure DevOps team project. We can use the agent pool to register test client machines as agents, which we will discuss in a future lesson in this chapter.

Lesson 6.02: Setting Up Deployment Pools

Another option to get the test client machines/agents grouped into a pool is the usage of deployment group pools . It is similar to grouping agents (test clients in this case) with agent pools.

Prerequisites: You must have working knowledge of Azure DevOps. You must have an account created with all Azure DevOps features or Azure DevOps pipelines using Azure DevOps services (`https://azure.microsoft.com/en-us/services/devops/`).

1. On the Azure DevOps homepage, click on Organization Settings. Then in the Pipelines tab, click on Deployment pools ➤ New.
2. In the pop-up window, specify a pool name. Select the required team projects to provision the deployment group pool and click Create to create and provision the pool (see Figure 6-5).

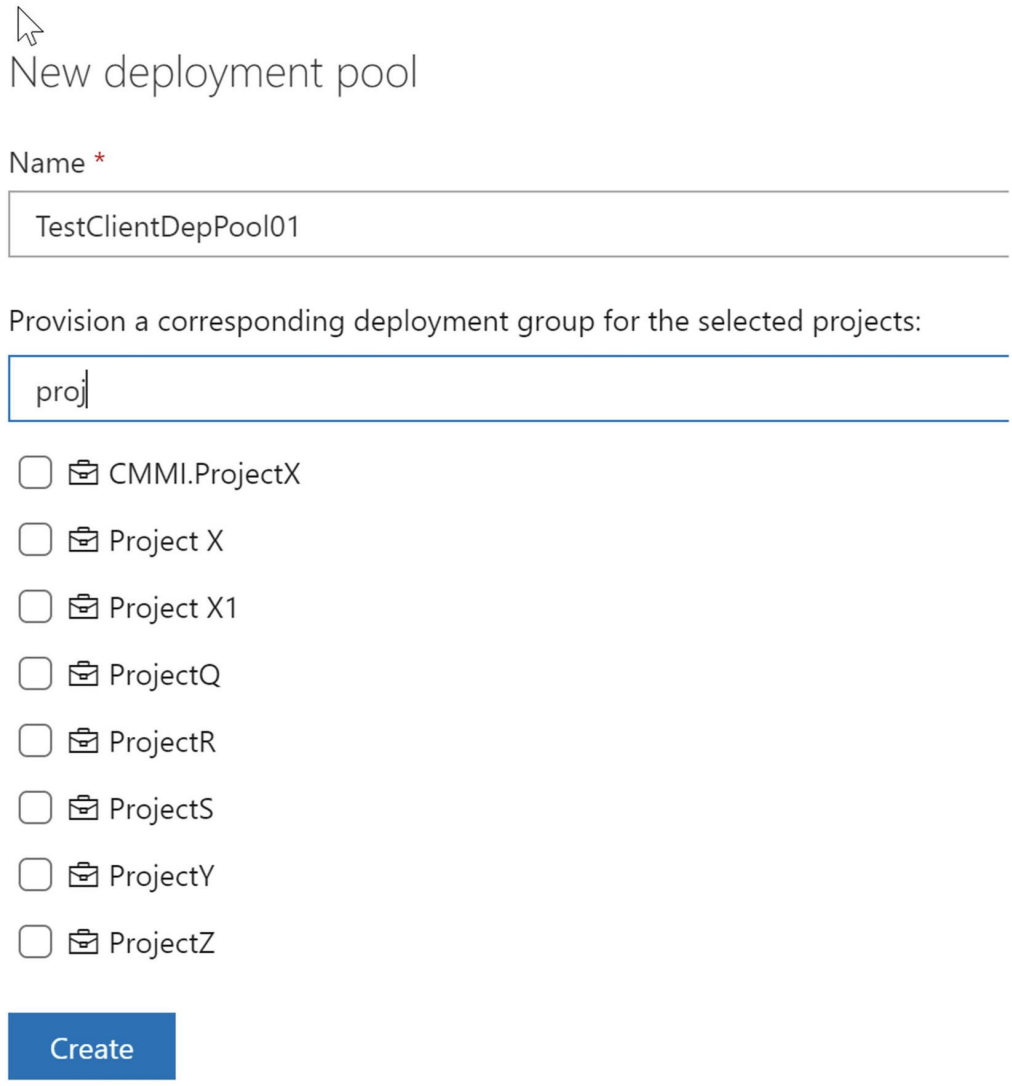

Figure 6-5 Create deployment pool

We have created a deployment pool and provisioned it for one team project in this lesson. You can use this pool in the next lessons of this chapter to group test clients .

Lesson 6.03: Set Up Test Clients with Agent Pool

The test client machine can be a virtual machine created in a Cloud platform such as Azure or Amazon or a virtual machine created in a physical server or even a physical machine.

Prerequisites: You must have a Windows 10 or Windows server 2012 or later, virtual machine, or physical machine. Make sure you have installed Google Chrome browser in your test client machine. You must have an account created with all Azure DevOps features or Azure DevOps pipelines using Azure DevOps services (`https://azure.microsoft.com/en-us/services/devops/`).

1. In the test client machine, log into your Azure DevOps account. Using your account, create a personal access token (PAT) for Azure DevOps, as instructed in `https://docs.microsoft.com/en-us/azure/devops/organizations/accounts/use-personal-access-tokens-to-authenticate?view=vsts#create-personal-access-tokens-to-authenticate-access` , with the scope of Agent pools set as read and manage. For more information on scopes, refer to `https://docs.microsoft.com/en-us/azure/devops/integrate/get-started/authentication/oauth?view=vsts#scopes` . Copy the PAT and store it in a safe location for future use, as it will be only shown once.
2. In Azure DevOps, navigate to the team project used in Lesson 6.01.
3. Then click on project settings ➤ Agent pools in the Pipelines tab and click on the Download agent button. Then select and download the Windows agent depending on whether your machine has x86 or x64. This will download a zip file to your test client machine.
4. Extract the zip file to a folder in a local drive.
5. Run a command prompt as administrator and change directory to the extracted agent content folder. Run the command config.cmd. Provide the Azure DevOps organization URL (see Figure 6-6).

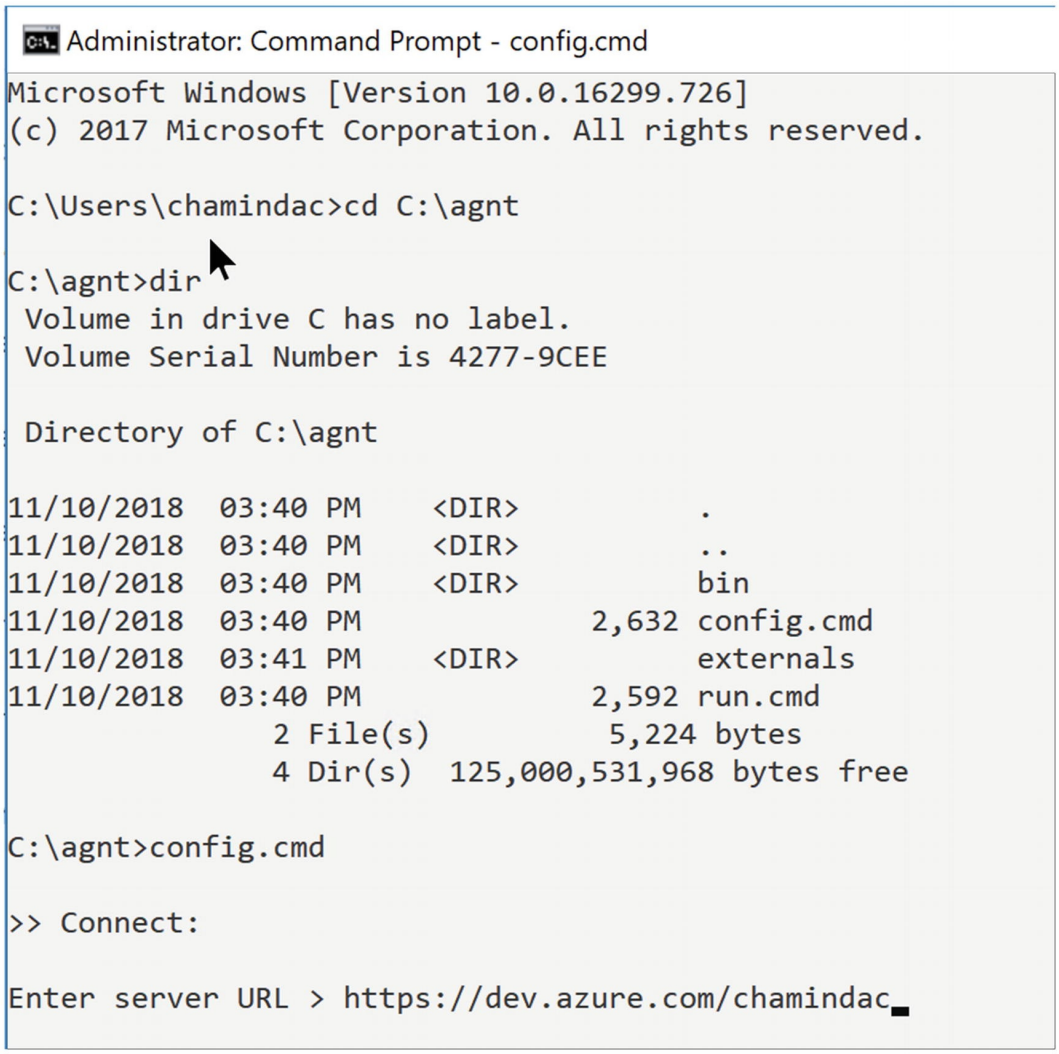

Figure 6-6 Run config.cmd

6. Press enter for PAT when prompted, and provide the PAT with agent pool read and manage scope. When prompted for agent name, provide the name of the agent pool created in Lesson 6.01. Provide or use the default name for the agent, which is the machine name (see Figure 6-7).

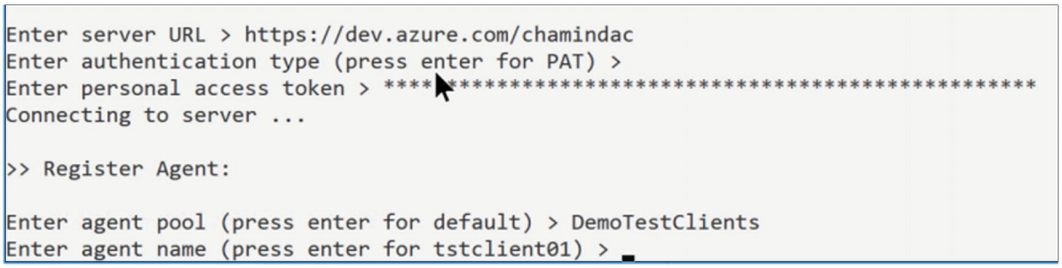

Figure 6-7 PAT and Agent pool

7. Use the default path for the agent work folder or provide a local drive folder path. When prompted to run agent as a service, press Enter to say No. It is important to note that running agent not as a service is required to enable execution of UI tests. This is referred to as running the agent in interactive mode.
8. For the auto log-on option, type Y to say yes. This is required so as to enable the test client to auto log-on to the machine with agent running user.
9. Provide name of a local admin user of the test client machine when prompted and provide that account password. The agent will be running with this user.
10. Finally, press Enter for reboot (see Figure 6-8).

```
>> Register Agent:

Enter agent pool (press enter for default) > DemoTestClients
Enter agent name (press enter for tstclient01) >
Scanning for tool capabilities.
Connecting to the server.
Successfully added the agent
Testing agent connection.
Enter work folder (press enter for _work) >
2018-11-10 16:08:30Z: Settings Saved.
Enter run agent as service? (Y/N) (press enter for N) >
Enter configure autologon and run agent on startup? (Y/N) (press enter for N) > Y
Enter User account to use for autologon > chamindac
Enter Password for the account tstclient01\chamindac > ************
Checking for policies that may prevent autologon from working correctly.
Checking for policies that may prevent screensaver from being disabled.
Restart the machine to launch agent and for autologon settings to take effect.
Enter Restart the machine at a later time? (Y/N) (press enter for N) (press enter for N) >
```

Figure 6-8 Running agent in interactive mode

11. After rebooting, the test client machine will auto log-on and the test client agent will be online (see Figure 6-9).

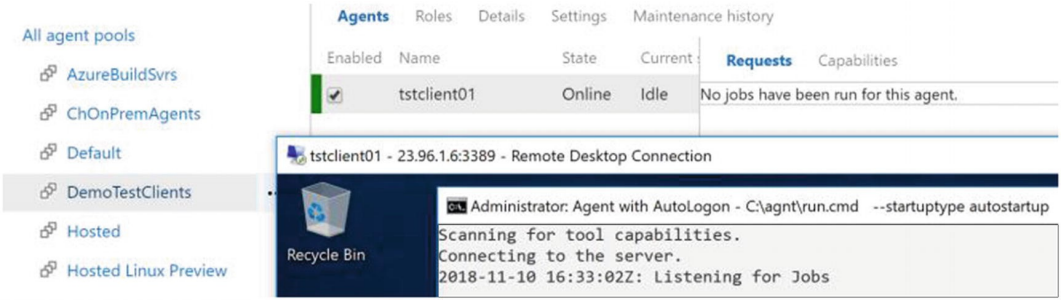

Figure 6-9 Agent online

In this lesson we have set up a test client machine with the Azure DevOps agent to enable execution of automated tests with deployment pipelines, which we will be doing in next lessons of this chapter.

Lesson 6.04: Set Up Test Clients with Deployment Group Pool

The test client machine can be a virtual machine created in a Cloud platform such as Azure or Amazon or a virtual machine created in a physical server or even a physical machine.

Prerequisites: You must have a Windows 10 or Windows server 2012 or later, virtual machine, or physical machine. Make sure you have installed Google Chrome browser on your test client machine. You have an account created with all Azure DevOps features or Azure DevOps pipelines using Azure DevOps services (`https://azure.microsoft.com/en-us/services/devops/`).

1. In Azure DevOps, navigate to the team project used in Lesson 6.02.
2. Click on Pipelines ➤ Deployment Groups in the left navigation pane. You will see the name of the deployment group pool selected for the team project in Lesson 6.02.
3. Click on the deployment group name (see Figure 6-10).

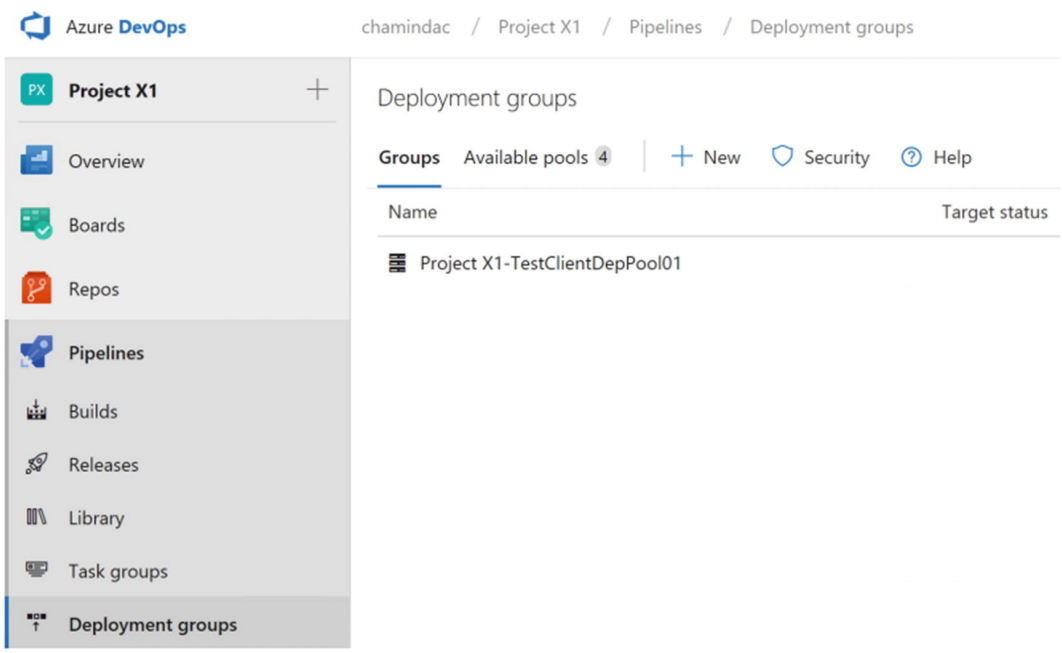

Figure 6-10 Deployment group

4. In the Details tab, select Windows as the Type of target to register. In this scenario, the target is the test client machine.
5. Check the option "Use a personal access token" in the script for authentication. This will include the PAT required to register the test client with the deployment group in the register script. The PAT generated here will be scoped to Machine groups Read and Manage permissions. Then click the Copy script to clipboard button (see Figure 6-11).

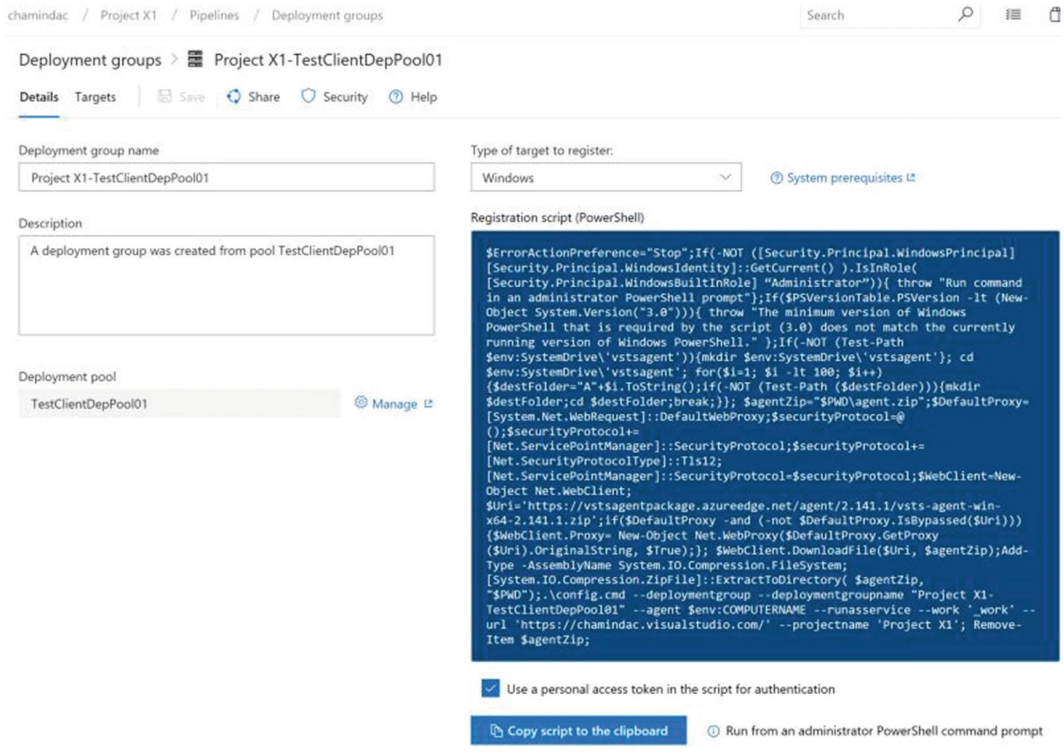

Figure 6-11 Register target script

6. Paste the copied script in a text editor and remove the --runasservice option provided in the script. We should not run test client agent as a service if we intend to run UI tests, as UI tests demand interactive mode of agent. After removing --runasservice, copy the script (see Figure 6-12).

Figure 6-12 Removing --runasservice

7. Open PowerShell window as an administrator. Paste the copied script and press Enter (see Figure 6-13) .

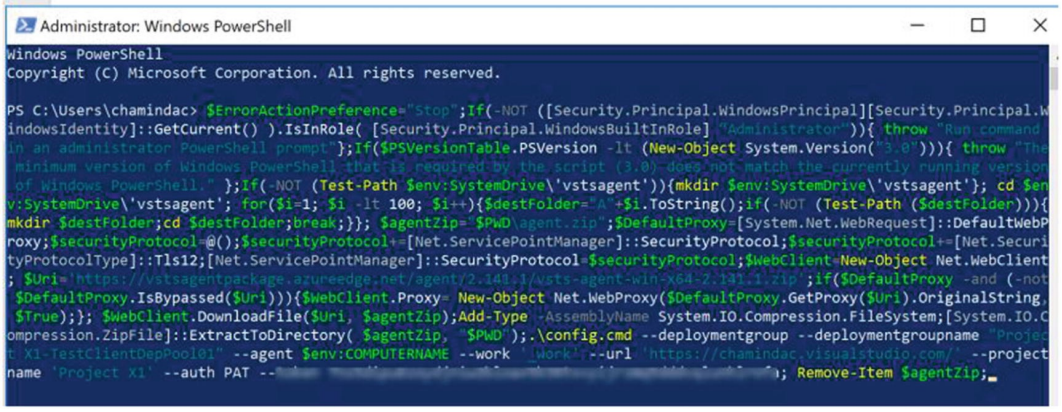

Figure 6-13 Target register script

8. Script will connect to the Azure DevOps account and a directory for agent will be created. Then the files required to set up the agent will be downloaded automatically. When prompted for whether to provide tags, type Y to do so and press Enter. Then provide a meaningful tag. A tag helps to locate or demand an agent in a given deployment group pool. For example, WebServer and DBServer can be useful tags to define the IIS web server and database server. In this instance, name the tag TestClient. You could even utilize tags such as smoketests, regressiontests, etc. (see Figure 6-14) .

Figure 6-14 Tags

9. When prompted for run agent as service, specify no by typing N and press Enter. Next the script will prompt for configure auto log-on, and for this we have to type Y. This is required to enable automatic log-on to the test client machine with the agent running user to enable listening to any test run jobs. For the username to run the agent, provide local admin user of the test client machine. Allow the machine to auto-login to take effect (see Figure 6-15).

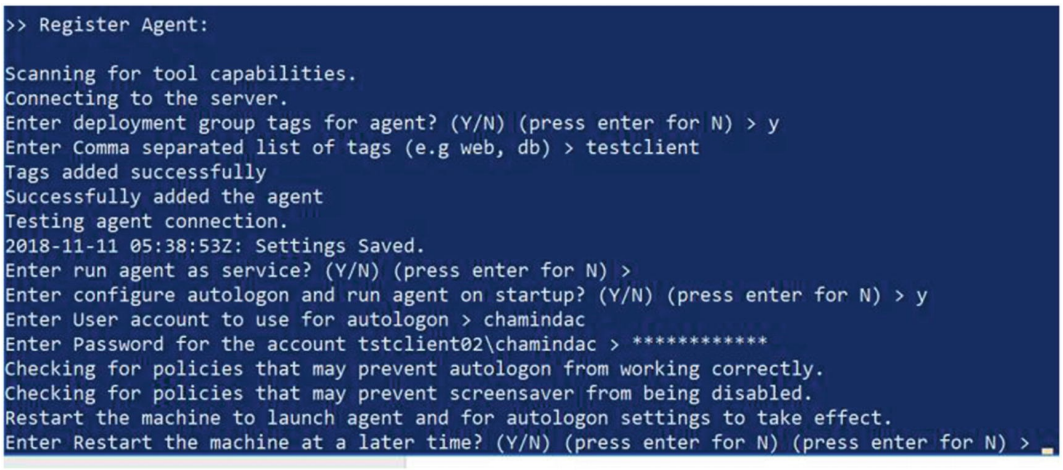

Figure 6-15 Registering test client in interactive mode

10. After the machine restarts, it will auto log-on and the test client will be online as a target in the deployment group (see Figure 6-16).

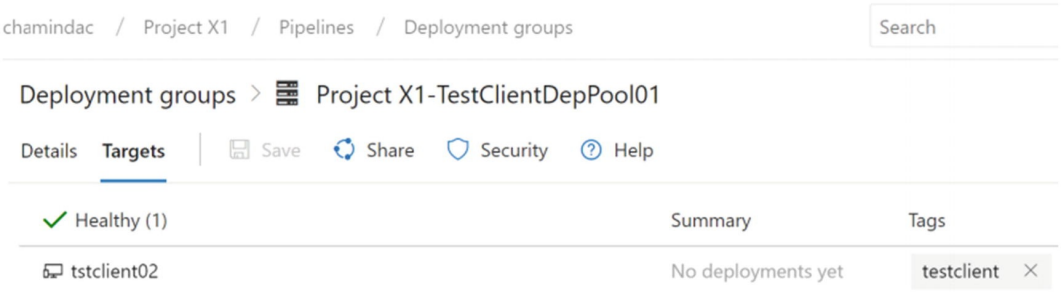

Figure 6-16 Test client in a deployment pool

In this lesson, we have registered a test client as a target in a deployment group pool. We will be able to use this test client with deployment pipelines to execute automated tests in the next lessons of this chapter.

Lesson 6.05: Create a Build Pipeline to Build Test Automation Code

We have to build and generate required assemblies as artifacts. These artifacts can then be used in deployment pipelines to execute tests.

Prerequisites: You must have an account created with all Azure DevOps features or Azure DevOps pipelines using Azure DevOps services (`https://azure.microsoft.com/en-us/services/devops/`). You must have the code used in Lesson 2.01 in your GitHub account. You can fork the code from here: `https://github.com/chamindac/Book-Test-Autmation-VS/tree/master/Chapter%202/Lesson%202.01/Testautomationproj1` .

1. In your Azure DevOps account go to Pipelines ➤ Build in the left navigation menu. Then click on New ➤ New build pipeline (see Figure 6-17).

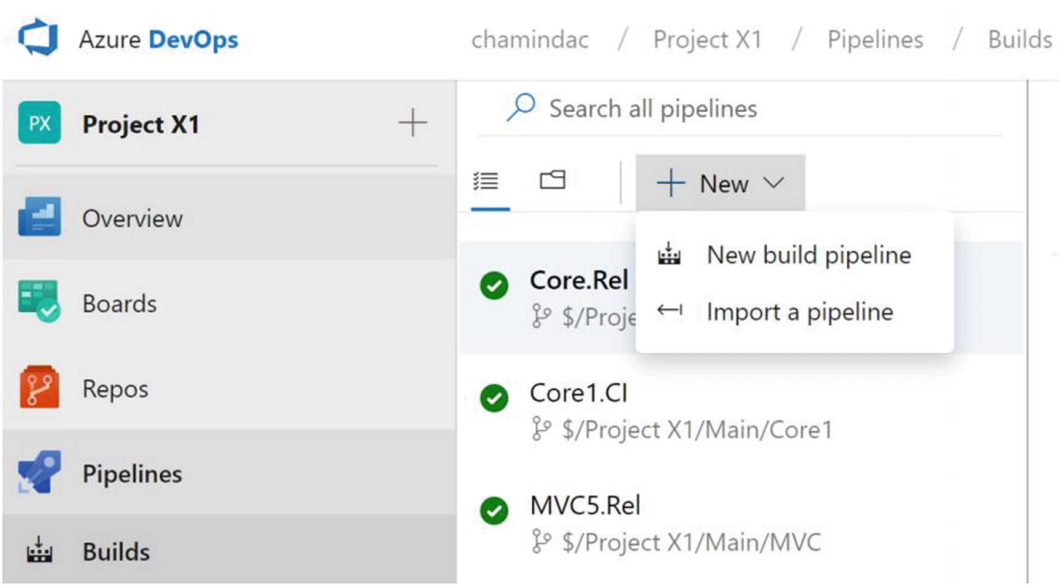

Figure 6-17 New build pipeline

2. Click on the link "Use the visual designer" to set up build with visual designer. Even though explaining YAML builds is out of the scope of this book, it is worth noting that YAML will help you to keep your build version-controlled in a repository. Simply you can have a build pipeline definition as code using YAML (see Figure 6-18).

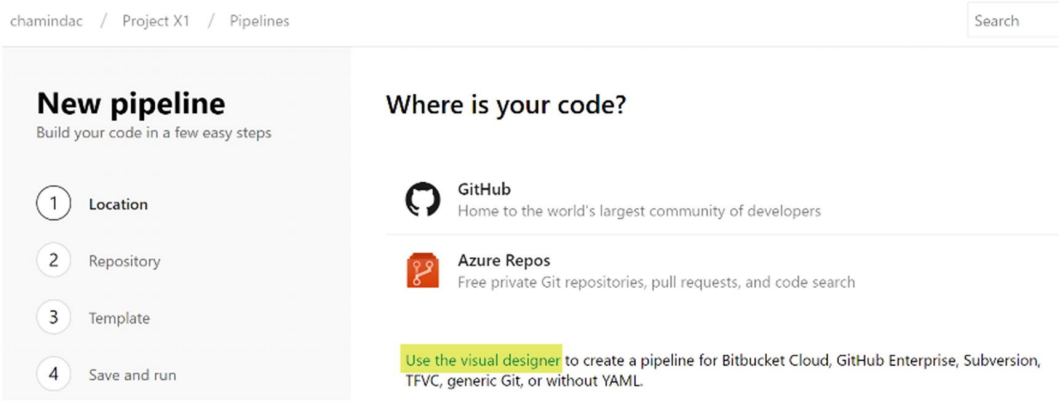

Figure 6-18 Use the visual designer

3. Select the GitHub as source control repo type and you can get establish connection by using OAuth or using a token. You will have to log onto your GitHub account to get it authorized via OAuth when prompted (see Figure 6-19).

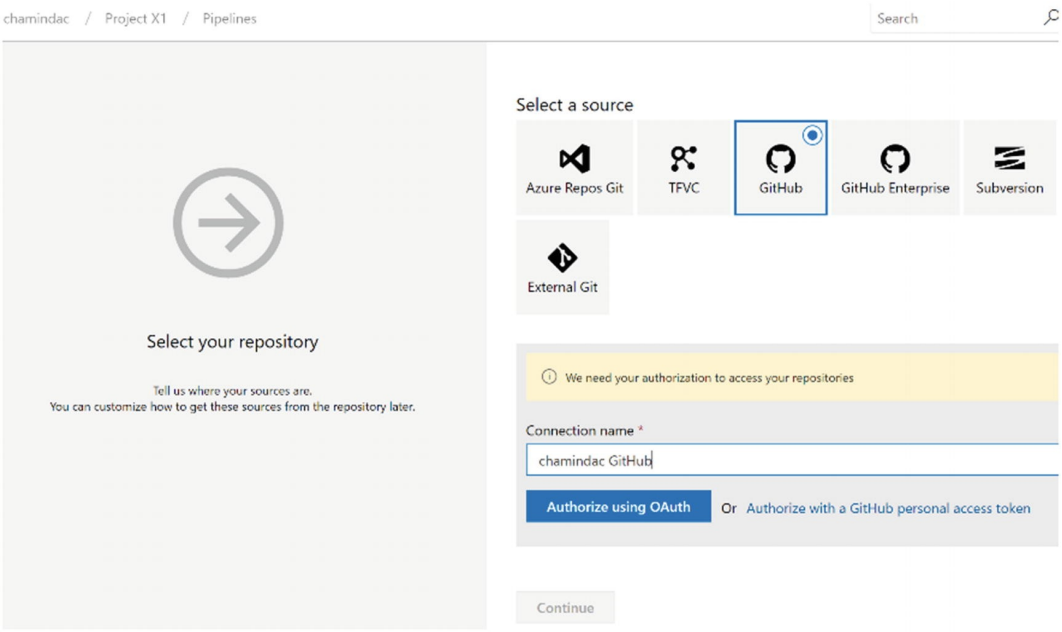

Figure 6-19 Authorize with OAuth

4. Select the forked code repo and select branch as master (see Figure 6-20).

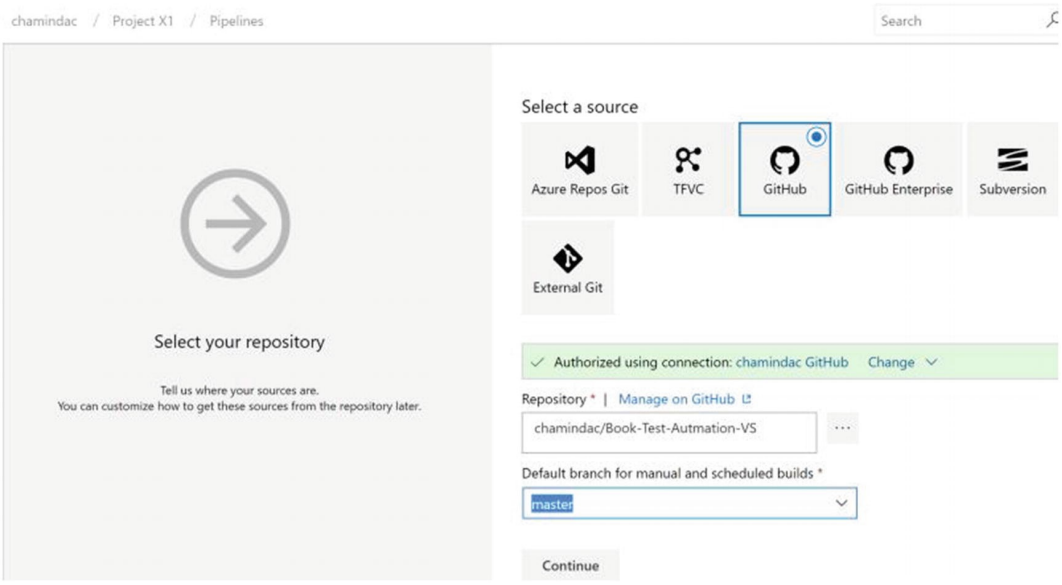

Figure 6-20 Select repo and branch

5. Click on start with an Empty job (see Figure 6-21).

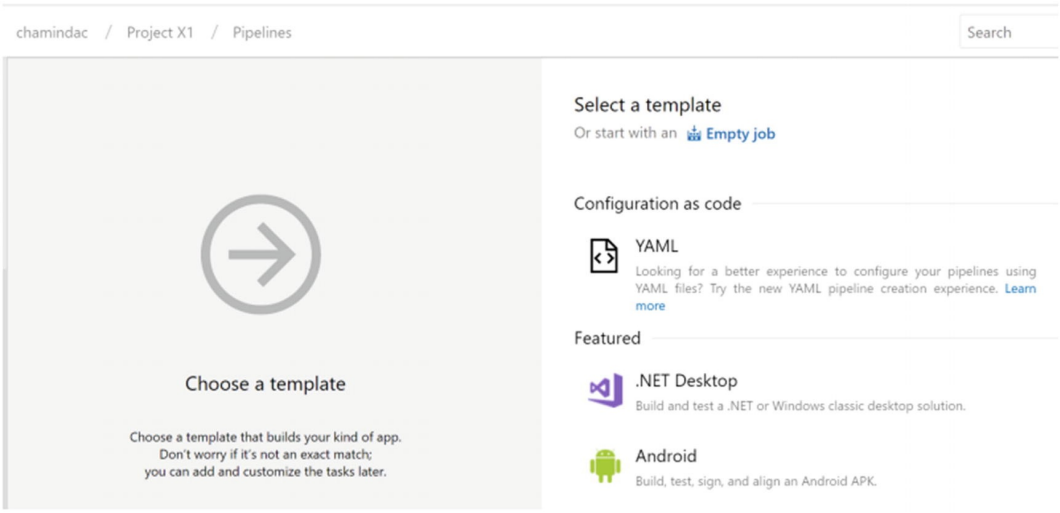

Figure 6-21 Empty job

6. Select the agent pool as Hosted Visual Studio 2017 in the build pipeline. The click the + sign in Agent Job 1 to add steps (see Figure 6-22).

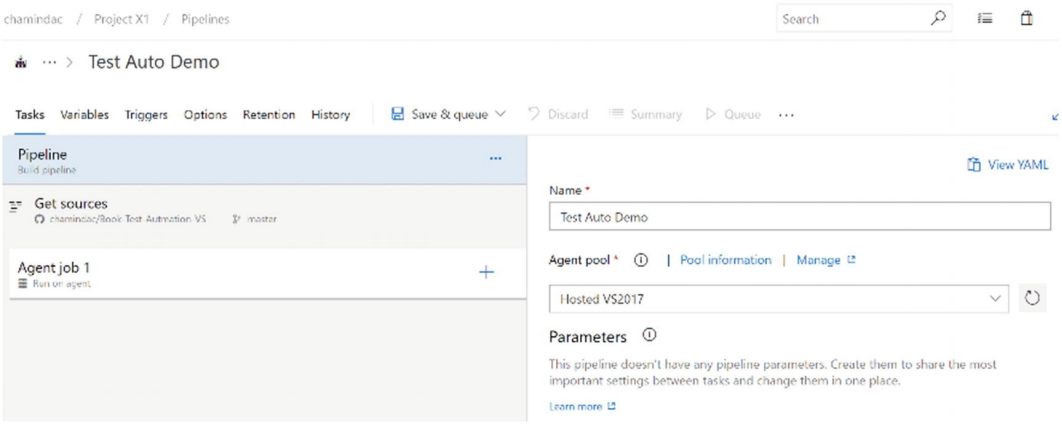

Figure 6-22 Selecting agent pool

7. Add NuGet installer task and select version 4.3.0 (see Figure 6-23).

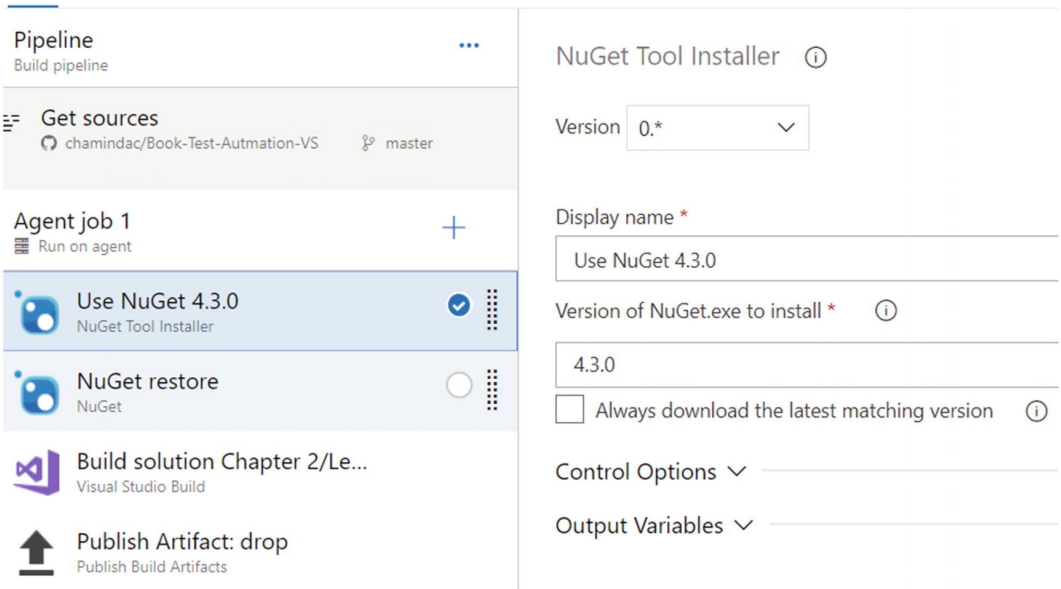

Figure 6-23 NuGet installer

8. Add a NuGet task and set the command to restore. Provide the solution path from Chapter 2 Lesson 2.01. This will restore all NuGet packages required by the solution (see Figure 6-24).

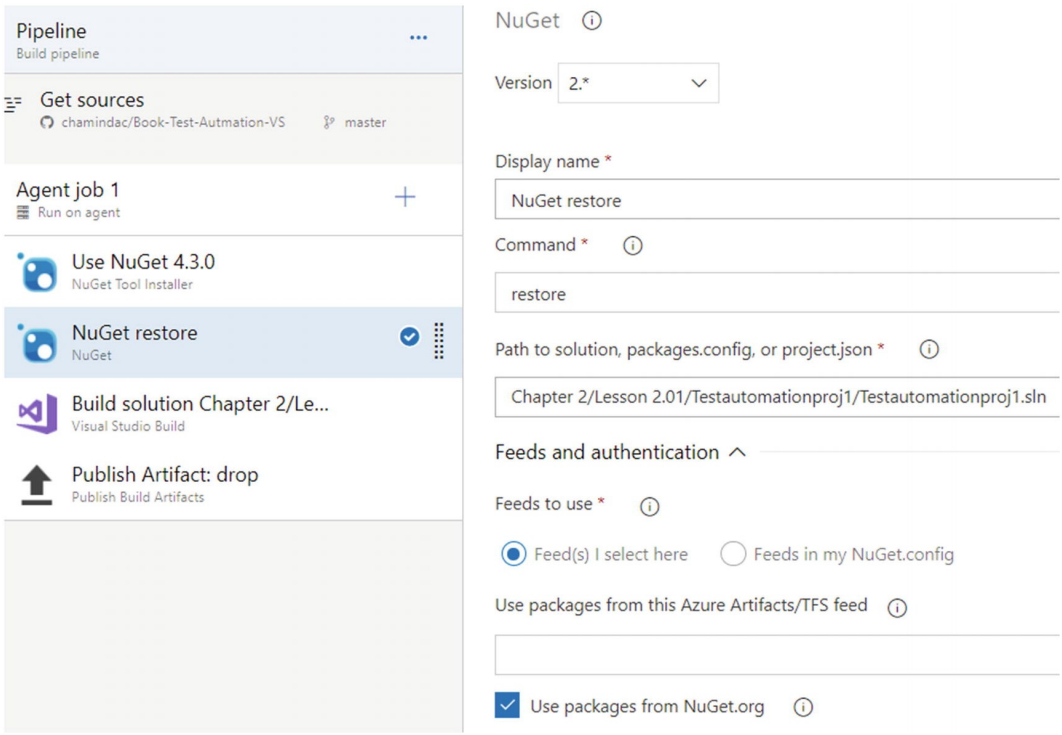

Figure 6-24 NuGet restore

9. Add a Visual Studio build step and specify the solution to build. In the Variables tab of the build definition, specify BuildPlatform as **any cpu** and BuildConfiguration as **release**. Provide MSBuild argument `/p:OutDir="$(Build.ArtifactStagingDirectory)"` to allow creation of binaries in the staging directory when the solution is built (see Figure 6-25).

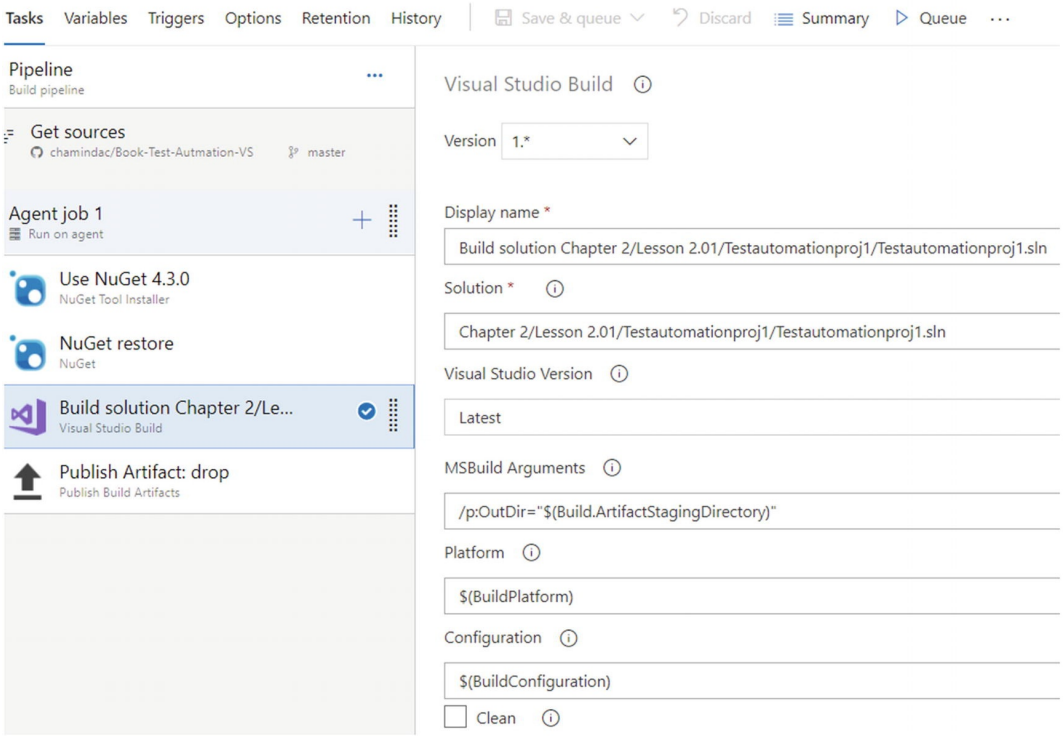

Figure 6-25 Build the solution

10. Add a publish artifact step to publish the binaries in the staging folder. The publish location should be set to Azure Pipeline/TFS (see Figure 6-26).

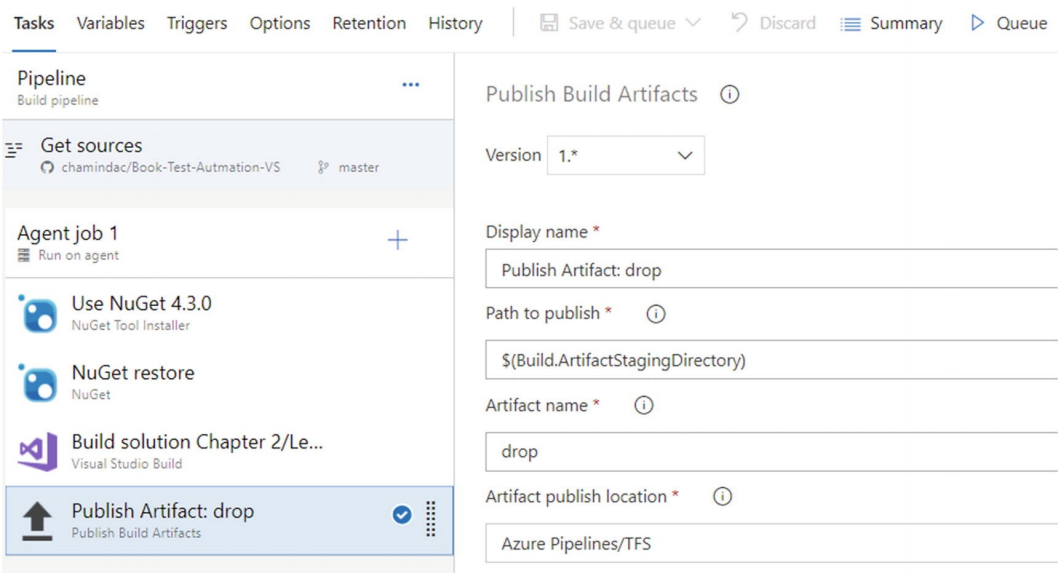

Figure 6-26 Publish build artifacts

11. Save and queue a new build and you can explore the published artifacts once the build completes (see Figure 6-27).

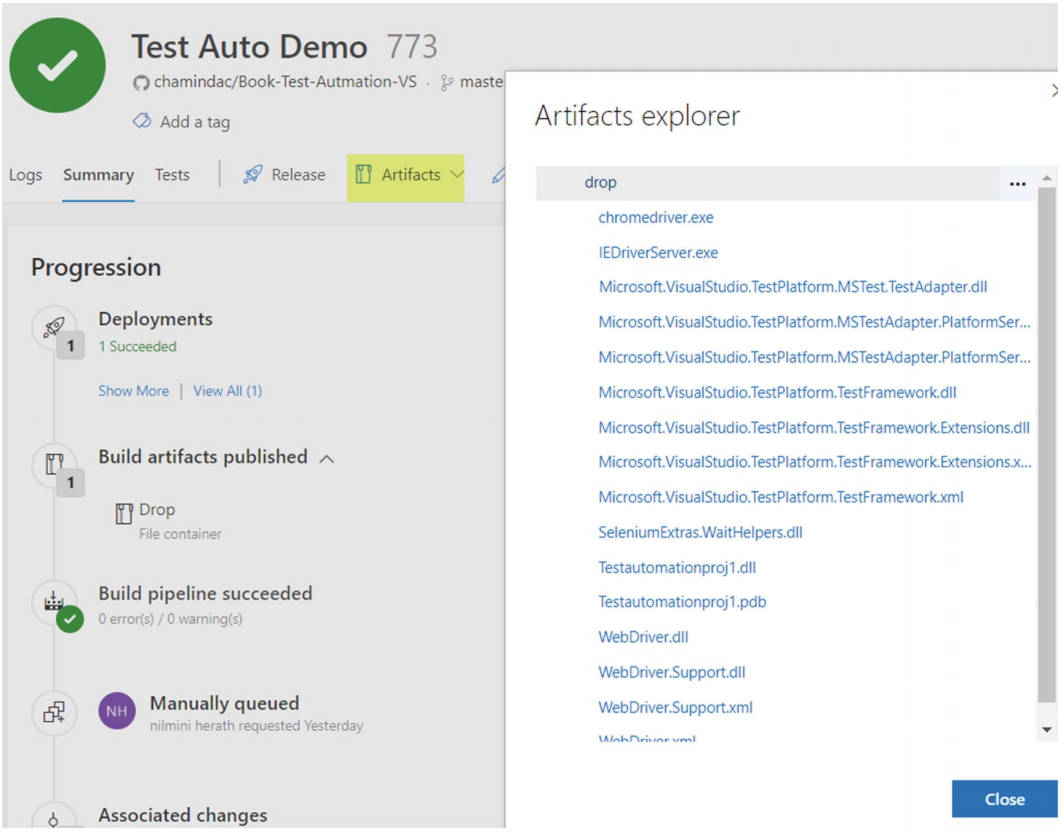

Figure 6-27 Published build artifacts

In this lesson, we have built the test automation code and published the build Binaries, which can be used in the deployment pipeline to execute tests.

Lesson 6.06: Create a Deployment Pipeline to Execute Test Automation

We have built and generated required test automation assemblies as artifacts. These artifacts can then be used in deployment pipelines to execute tests.

Prerequisites: You must have an account created with all Azure DevOps features or Azure DevOps pipelines using Azure DevOps services (`https://azure.microsoft.com/en-us/services/devops/`). You must have code forked `https://github.com/chamindac/Book-Test-Autmation-VS/tree/master/Chapter%202/Lesson%202.01/Testautomationproj1` and have built it as described in Lesson 6.05.

1. In your Azure DevOps account, go to Pipelines ➤ Releases in the left navigation menu. Then click on New ➤ New release pipeline (see Figure 6-28).

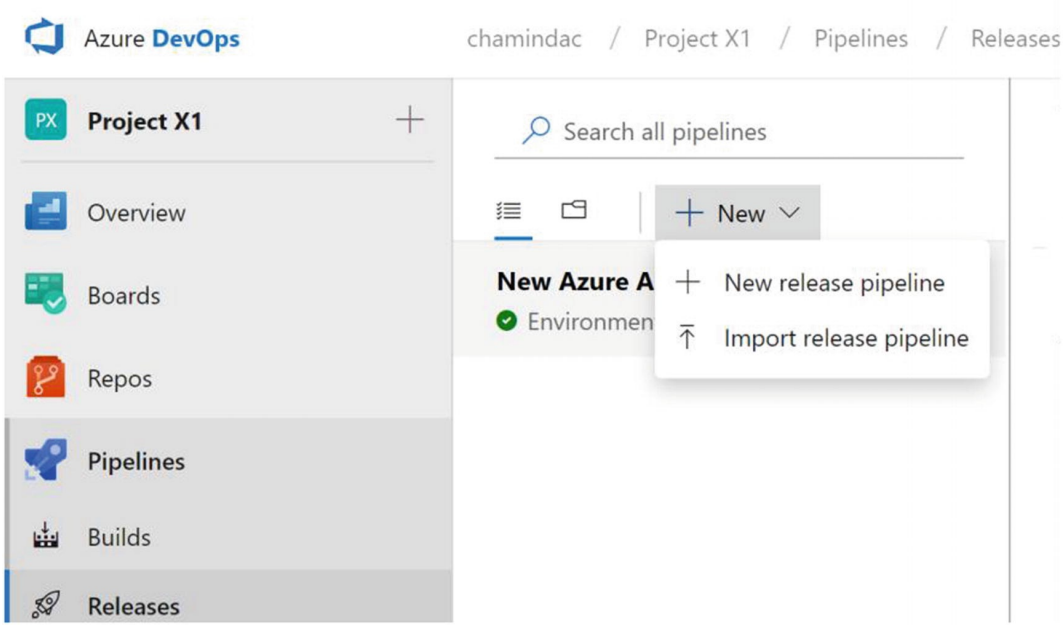

Figure 6-28 New release pipeline

2. Click Empty job (see Figure 6-29).

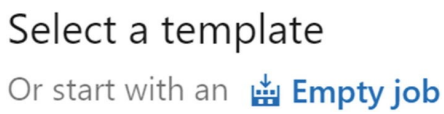

Figure 6-29 Select Empty job

3. Specify a stage name and click X to close the tab (see Figure 6-30).

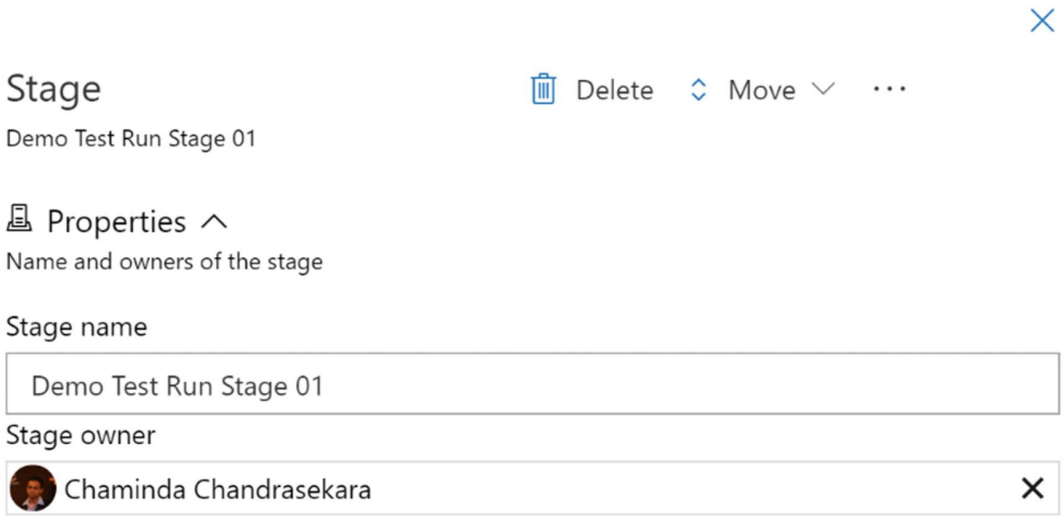

Figure 6-30 Step name

4. Click on Add Artifact (see Figure 6-31).

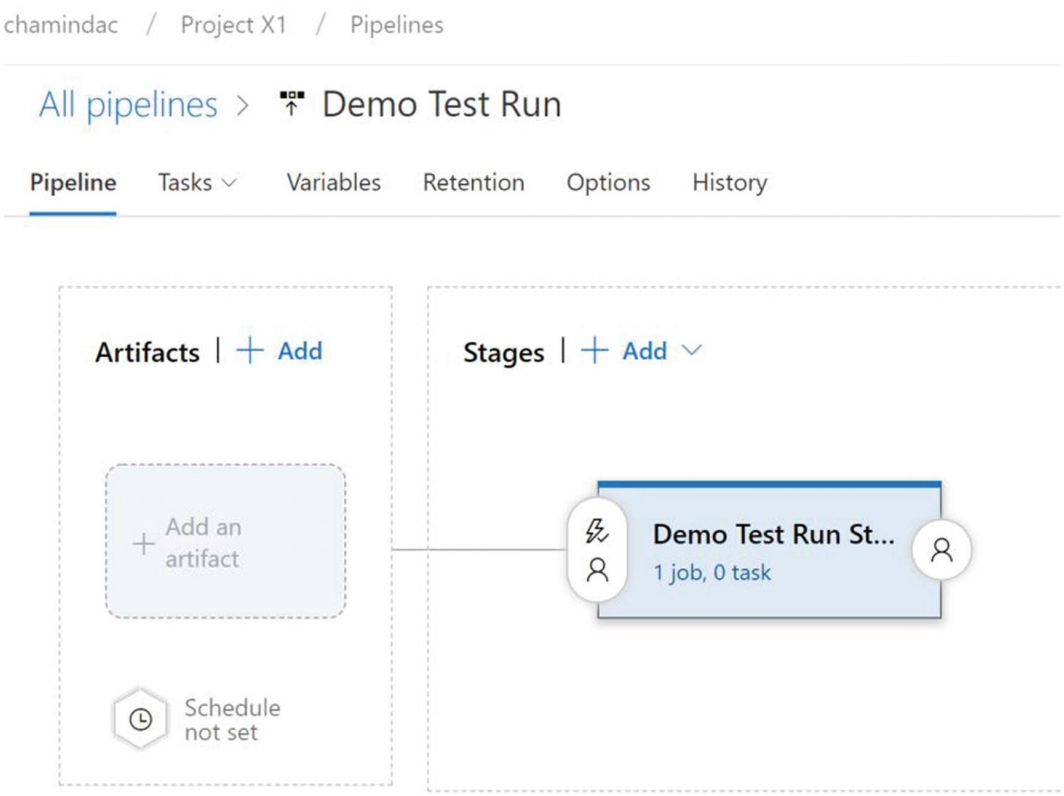

Figure 6-31 Add Artifact

5. Select the build we have created in Lesson 6.05. Select Default version as Latest (see Figure 6-32) .

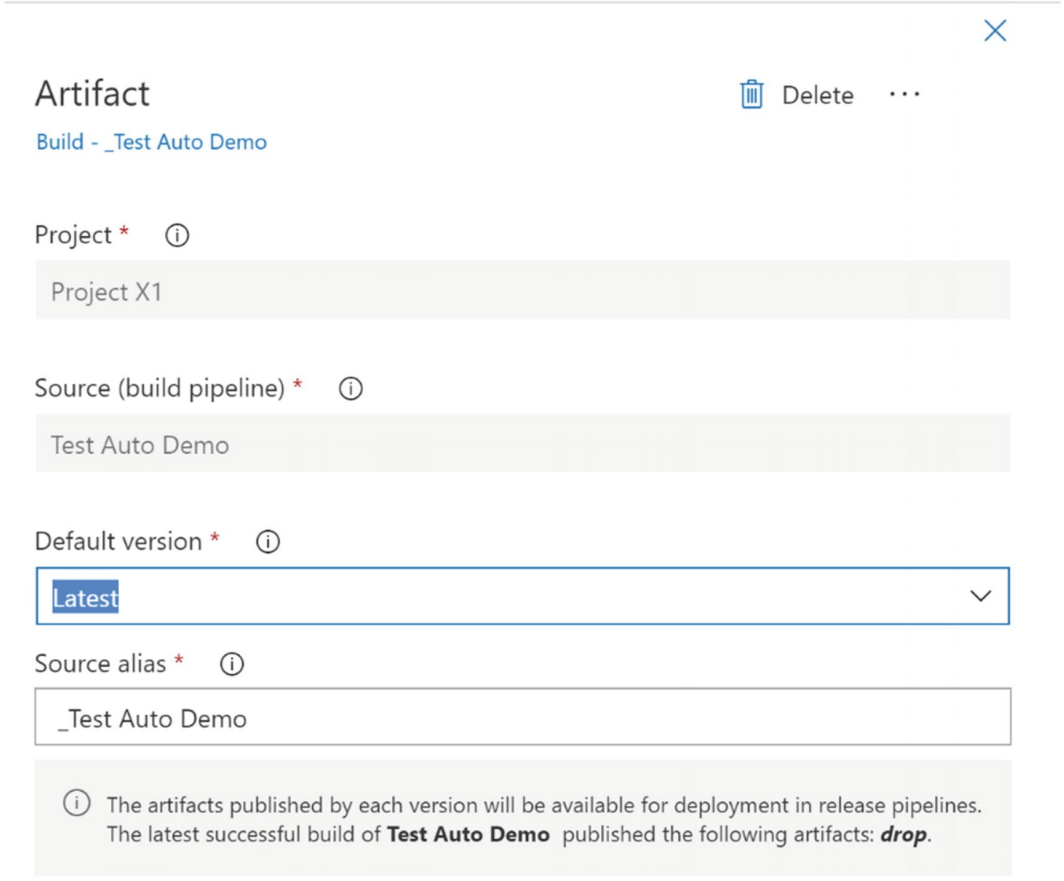

Figure 6-32 Specify the build as artifact

6. Click on Tasks and on the stage name (see Figure 6-33).

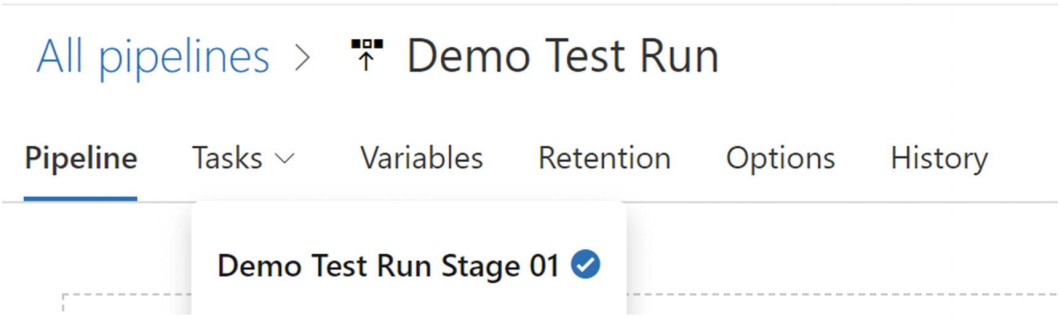

Figure 6-33 Select stage

7. Select the Agent pool as DemoTestClients, which is set up with test client in Lesson 6.03 (see Figure 6-34) .

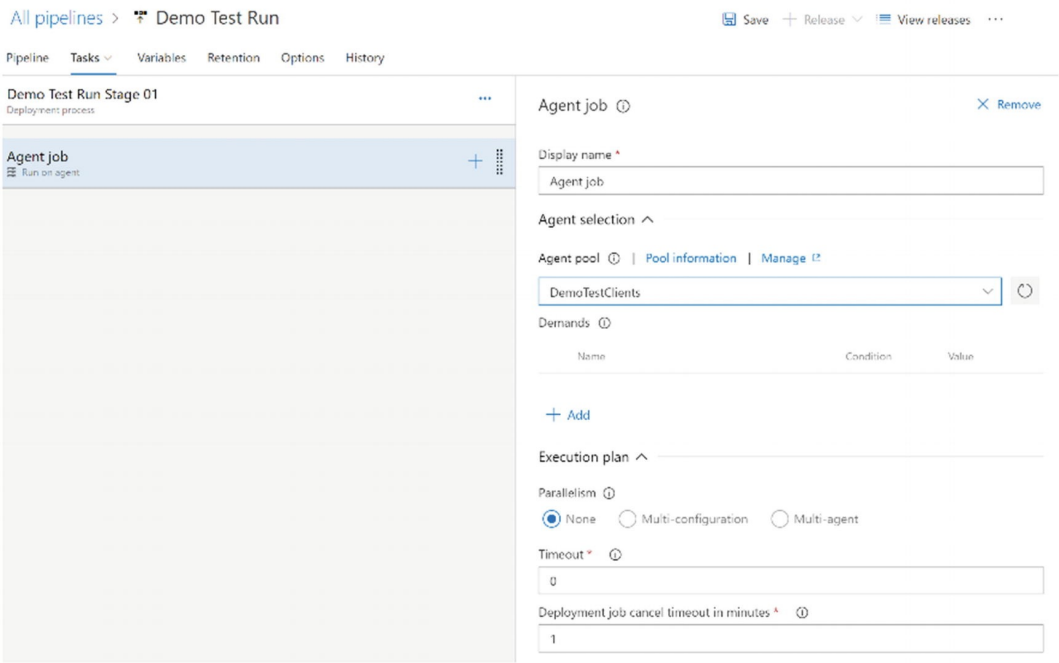

Figure 6-34 Agent job settings

8. For the agent job, specify the build artifacts to be downloaded (see Figure 6-35).

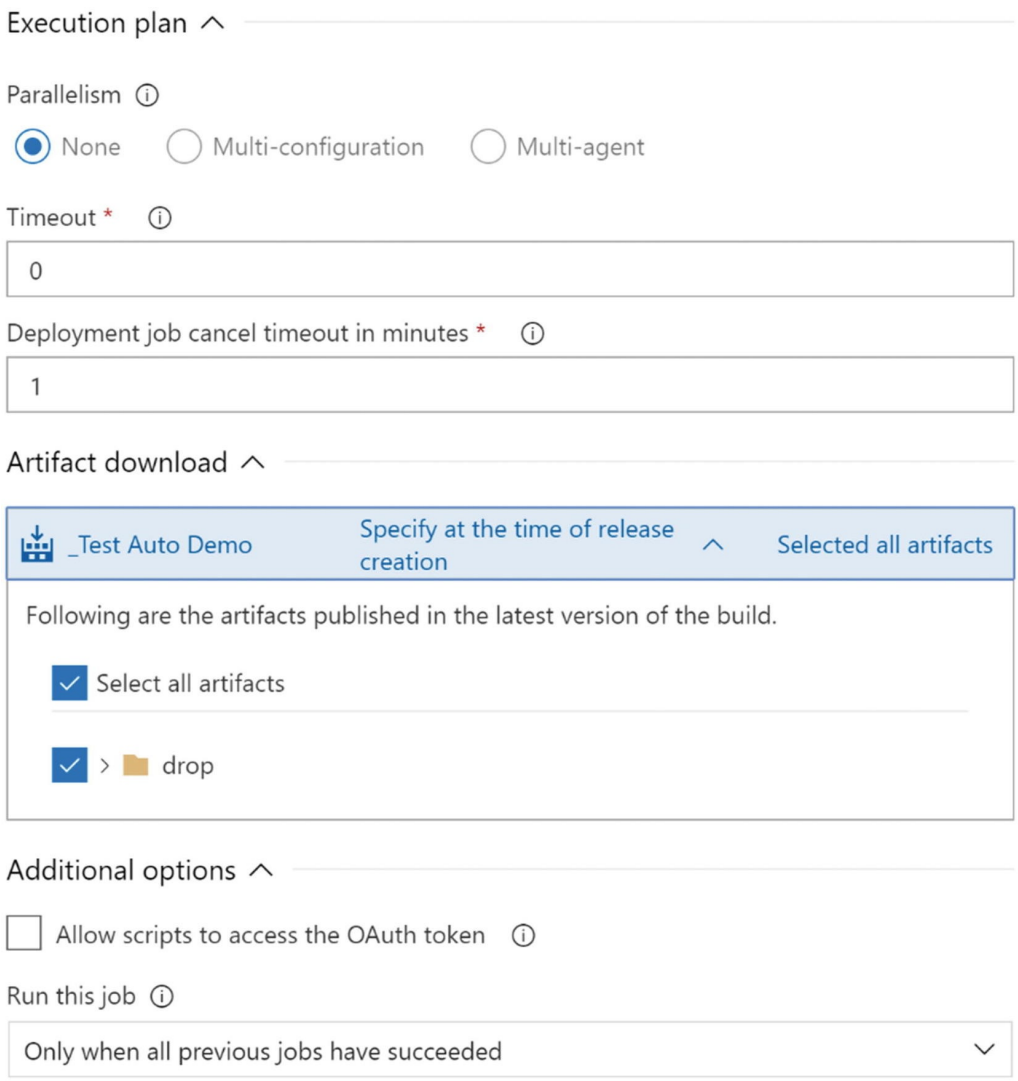

Figure 6-35 Agent job artifacts

9. Add a Visual Studio Test Platform Installer step. Select the version as Latest Stable (see Figure 6-36) .

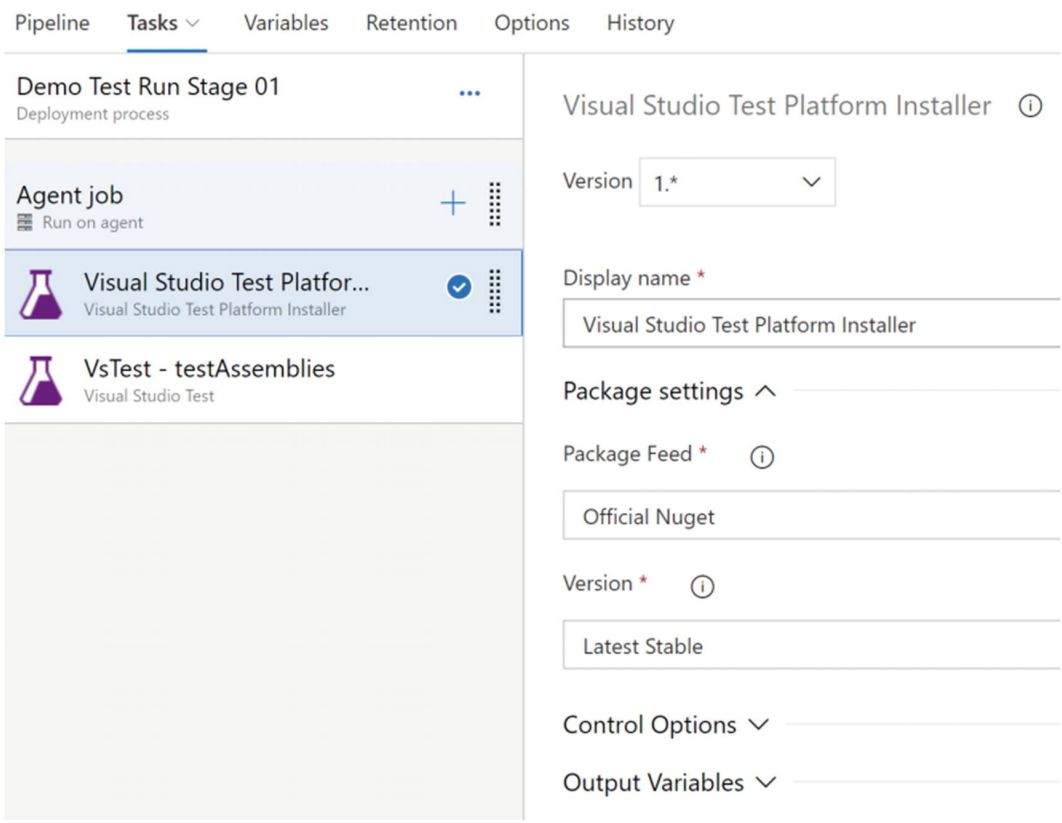

Figure 6-36 Visual Studio Test Platform Installer

10. Add a Visual Studio Step. Change the test file name to select **\Testautomationproj1.dll (see Figure 6-37).

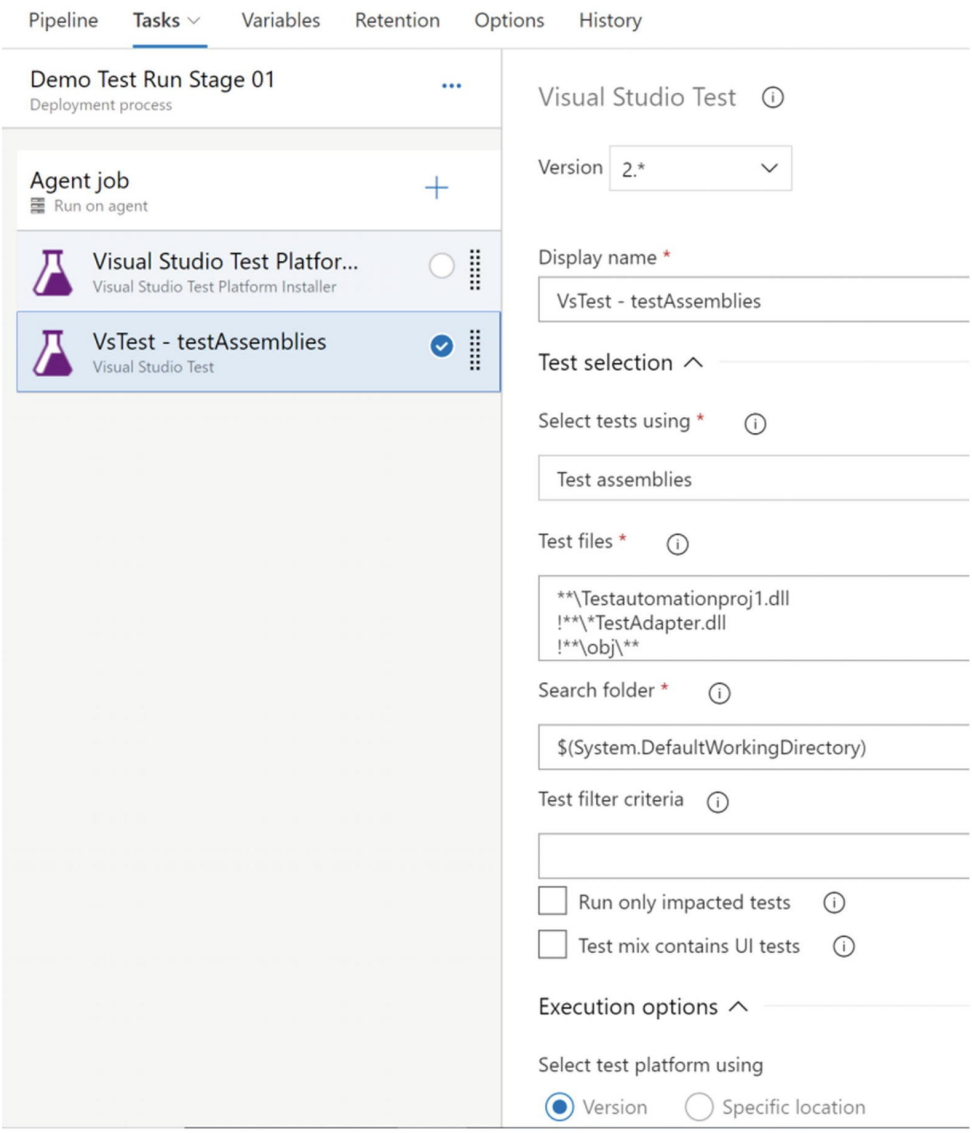

Figure 6-37 Visual Studio Test step

11. Make sure to select Test platform version as Installed by Tools Installer (see Figure 6-38).

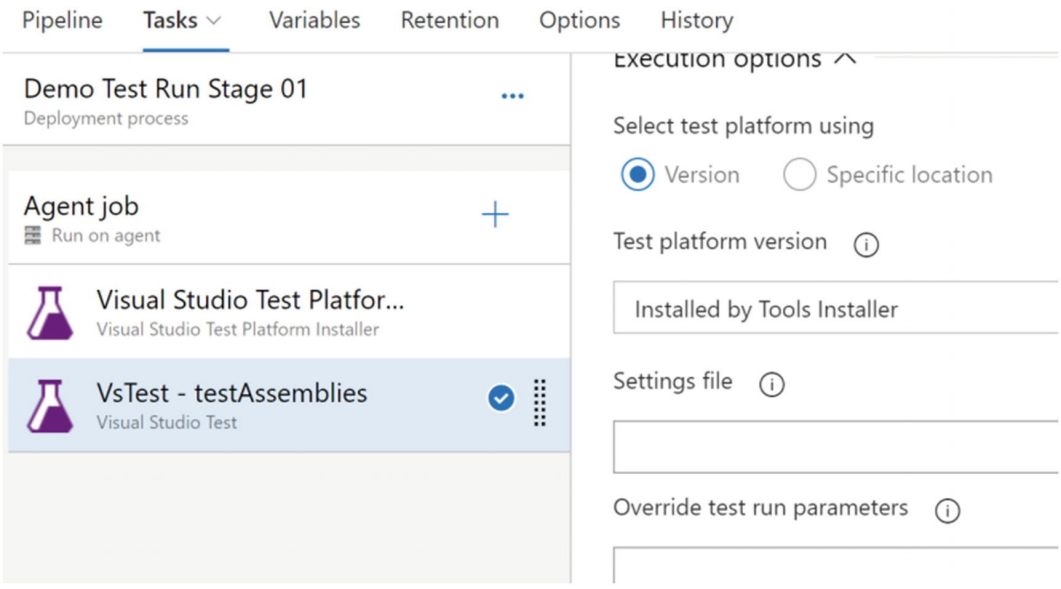

Figure 6-38 Test platform version

12. Save the release pipeline. Click Release ➤ Create Release (see Figure 6-39).

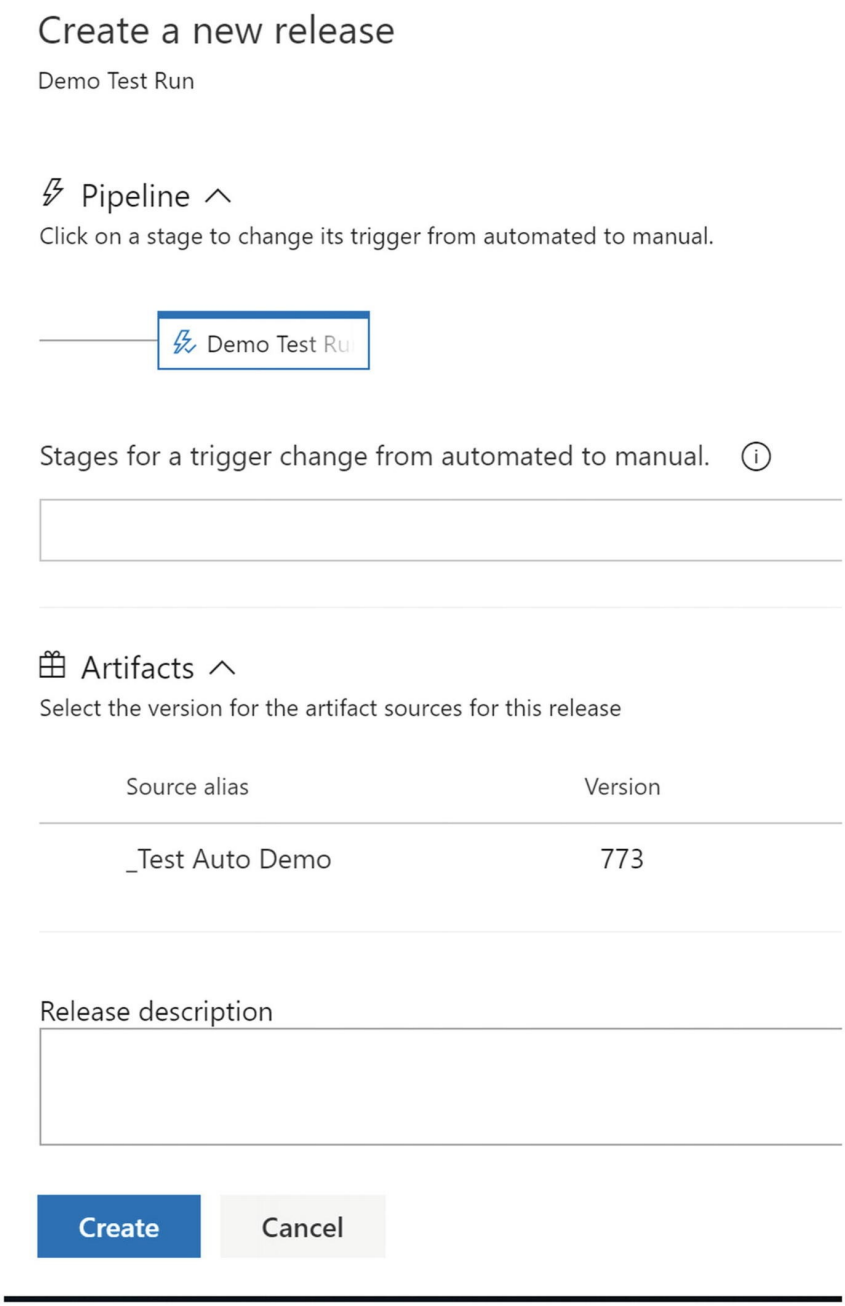

Figure 6-39 Create release

13. The release will be executed, and the automated tests will run (see Figure 6-40) .

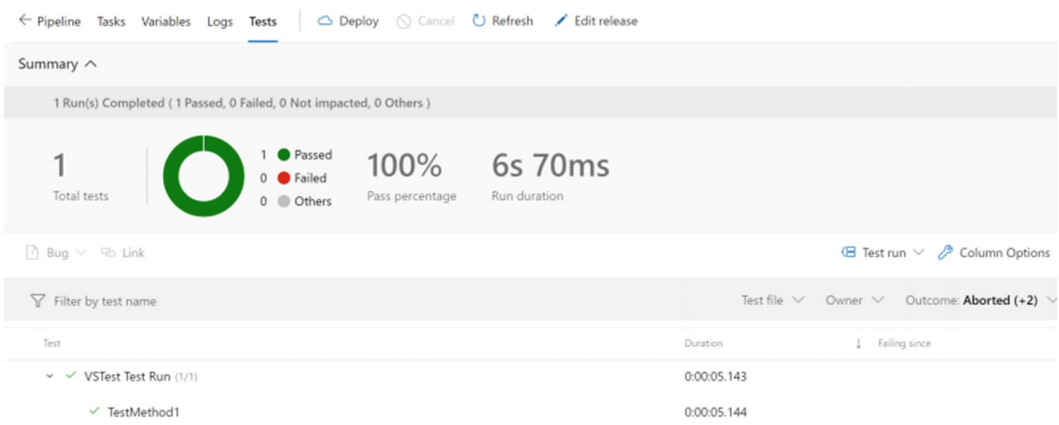

Figure 6-40 Executed test

14. You can repeat adding these two steps in the same manner to a Deployment Group Phase to understand how it works. First, add a Deployment group job (see Figure 6-41).

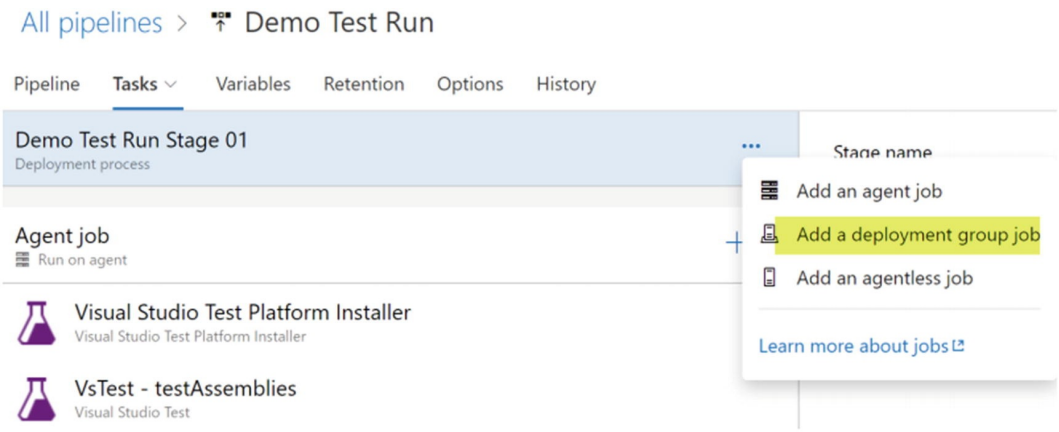

Figure 6-41 Add a deployment group job

15. Select the deployment group created in this chapter and select the tag we added to the test client registered with the deployment group (see Figure 6-42) .

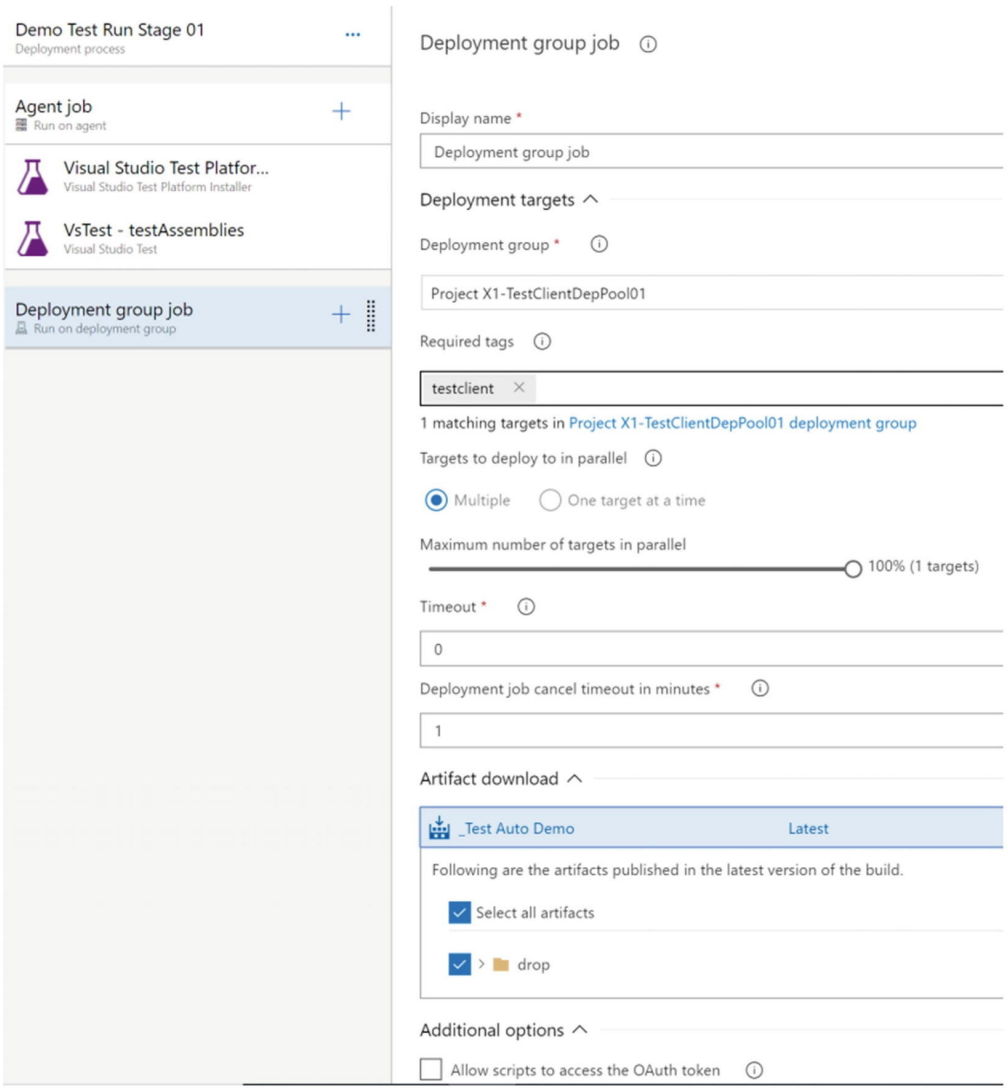

Figure 6-42 Deployment group

16. Then set up the two steps as done previously in this lesson to execute test automation in a test client machine registered with a deployment pool.

In this lesson, we have explained how to execute tests with the deployment pipeline. With this knowledge, you can enable test executions with deployments of software applications.

Summary

In this chapter, we have learned how to use Azure DevOps agent pools and deployment groups to automate test execution with deployment pipelines. We used the Visual Studio Test step for test execution in the lessons. This task still has limitations if some .Net framework assemblies or SDKs are used for test development, such as the MAQS framework we discussed previous chapters. In these situations, you will be required to set up Visual Studio and required SDKs in the test client machines.

The Functional Test step does not require Visual Studio in test execution. However, it requires set up of a separate test agent based on Microsoft Test Management using the Visual Studio Test Agent Deployment task. It is very slow in performance and you would not want to proceed with deprecated tasks.

In the next chapter, we will explore options to perform load and performance testing.

Chaminda Chandrasekara and Pushpa Herath, *Hands-On Functional Test Automation*
https://doi.org/10.1007/978-1-4842-4411-1_7

7. Load and Performance Testing

Chaminda Chandrasekara[1] and Pushpa Herath[2]

(1) Dedigamuwa, Sri Lanka
(2) Hanguranketha, Sri Lanka

In the previous chapters of the book we discussed automation of functional testing using Selenium and Visual Studio and getting the test automations running with CI/CD pipelines. For any application, it is essential to go through load and performance testing, before getting it to the production. Ideally these tests should be performed in an environment similar to the production environment in order to identify any issues in the pre-production stage.

For load and performance testing, Selenium can only be used with some open source and paid third-party tools and frameworks. One such tool is Selenium Grid (`https://github.com/SeleniumHQ/selenium/wiki/Grid2`) and another limited free and paid option to use is WebLoad (`https://www.radview.com/selenium-performance-testing/`).

However, the focus of this chapter is to guide you to get you started on the load and performance testing in cloud. For authoring and executing these tests, we would be using Visual Studio, Azure, and Azure DevOps. Furthermore, you can learn how to generate load from multiple regions and test your application by using real world scenarios.

Lesson 7.01: Load Test with Visual Studio and Azure DevOps

In this lesson, let's author the load tests with Visual Studio and run them as cloud-based load testing using Azure DevOps.

Prerequisites: You need to run Visual Studio 2017 on Windows 10 or on Windows Server 2012 R2 or a newer version of the Windows server. You must have an Azure DevOps account. You must be familiar with Azure DevOps and using the Team Explorer in Visual Studio.

Install Web Performance and Load Testing Tools Components in Visual Studio 2017

You can install web performance and load test components while you Install Visual Studio. If you haven't installed this component yet, you can install as described here:

1. Launch Visual Studio Installer.
2. In the Visual Studio Installer, move to Individual components and select web performance and load testing tools under the Debugging and testing section (see Figure 7-1).

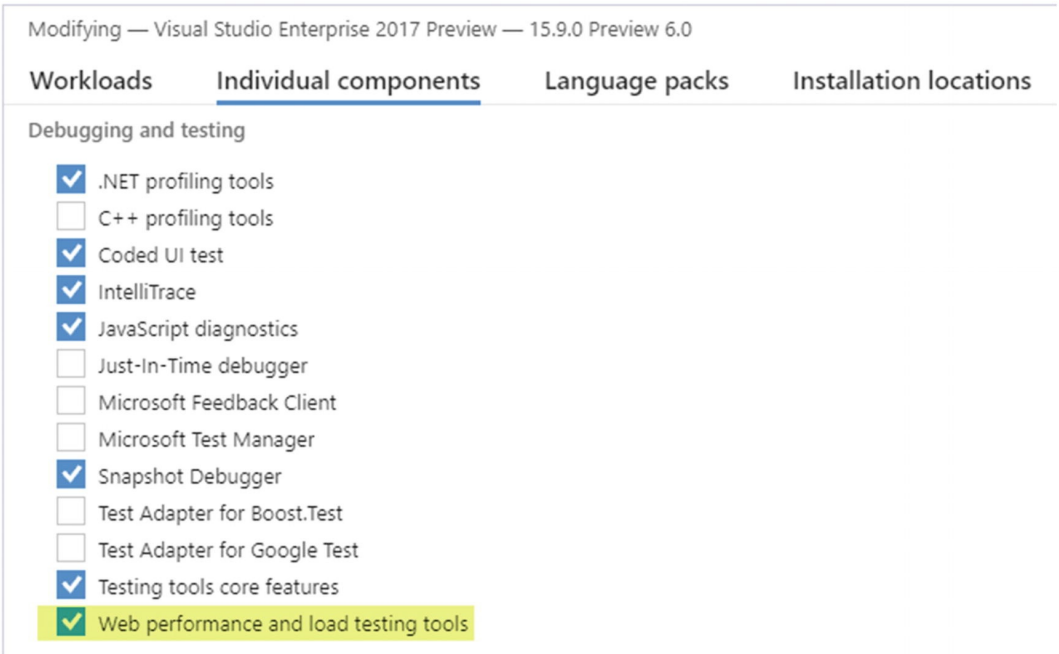

Figure 7-1 Web performance and load testing tool component in Visual Studio Installer

3. Click on the Modify button to apply the changes to the Visual Studio.

Setting Up Visual Studio Web Performance and Load Test Project

Let's begin with setting up Visual Studio web performance and loading the test project following the steps described here.

1. In Visual Studio 2017, select Files ➤ New ➤ Project
2. In the New Project pop-up window, select Test under Visual Studio C#, and select Web Performance and Load Test project from the test project list. Give a **Name** to the Project, specify a **Solution Name**, select **Location,** and click on the OK button. Leaving **Create directory for solution** checked will allow you to have a new directory created for the new solution in the selected location (see Figure 7-2).

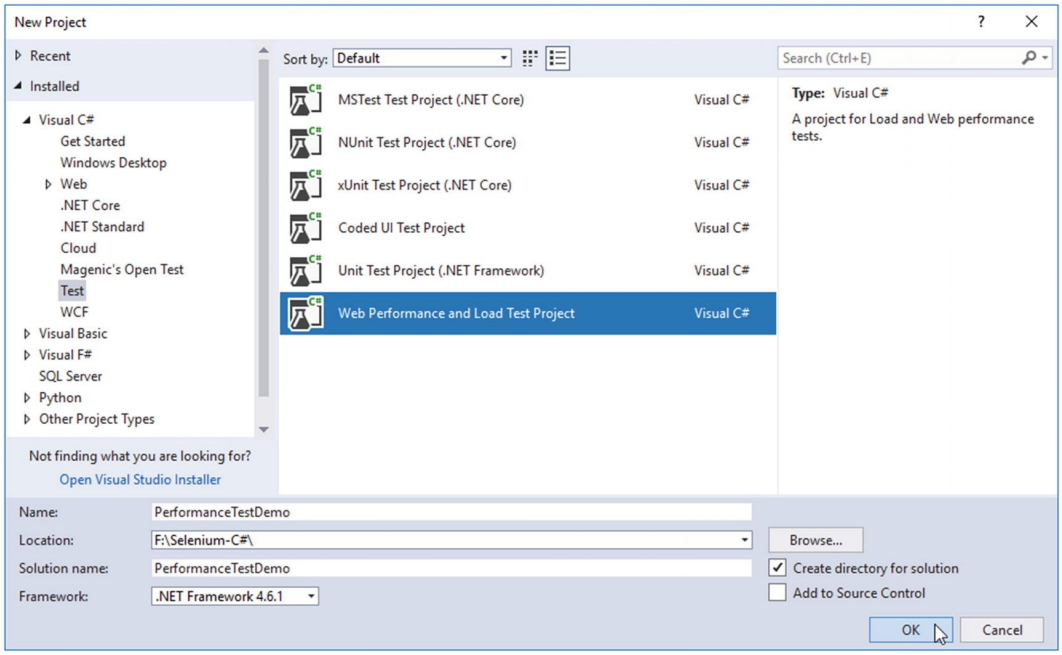

Figure 7-2 Create new web performance test project

3. After creating a new web performance test project, you can find project items in the Solution Explorer as follows (see Figure 7-3).

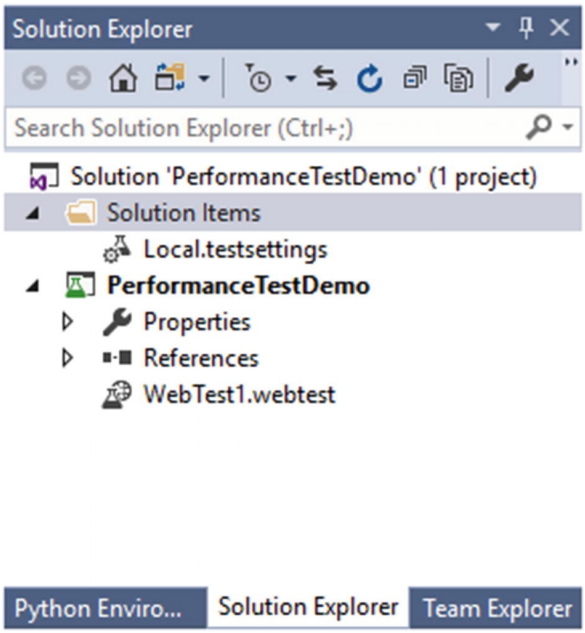

Figure 7-3 Solution Explorer view of web performance test project

4. Now we can record a web performance test. To record the performance of the web application, the Internet Explorer browser is configured as the default browser. If the Web Test Recorder add-on in the browser is disabled, recording action cannot be performed. So, let's enable the Web Test Recorder add-on in the browser before we proceed into the next steps. Open the Internet Explorer browser and open the Manage add-ons in browser settings (see Figure 7-4).

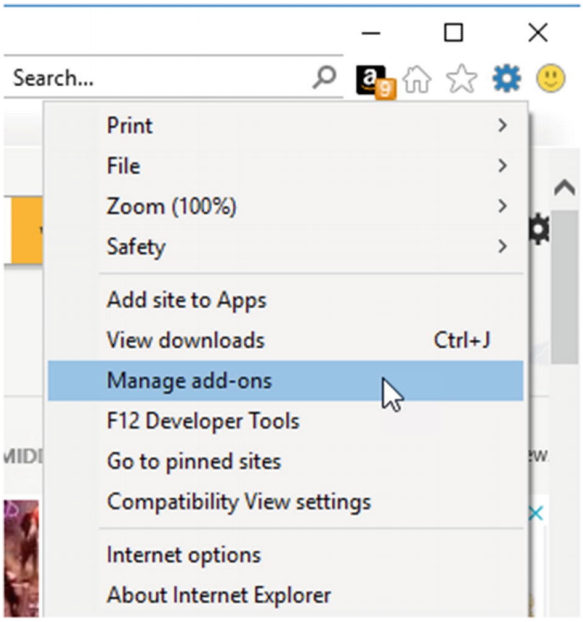

Figure 7-4 Select Manage add-ons from Internet Explorer settings

5. The Manage Add-on window will be displayed. Select the web test recorder and enable it by clicking the Enable button (see Figure 7-5).

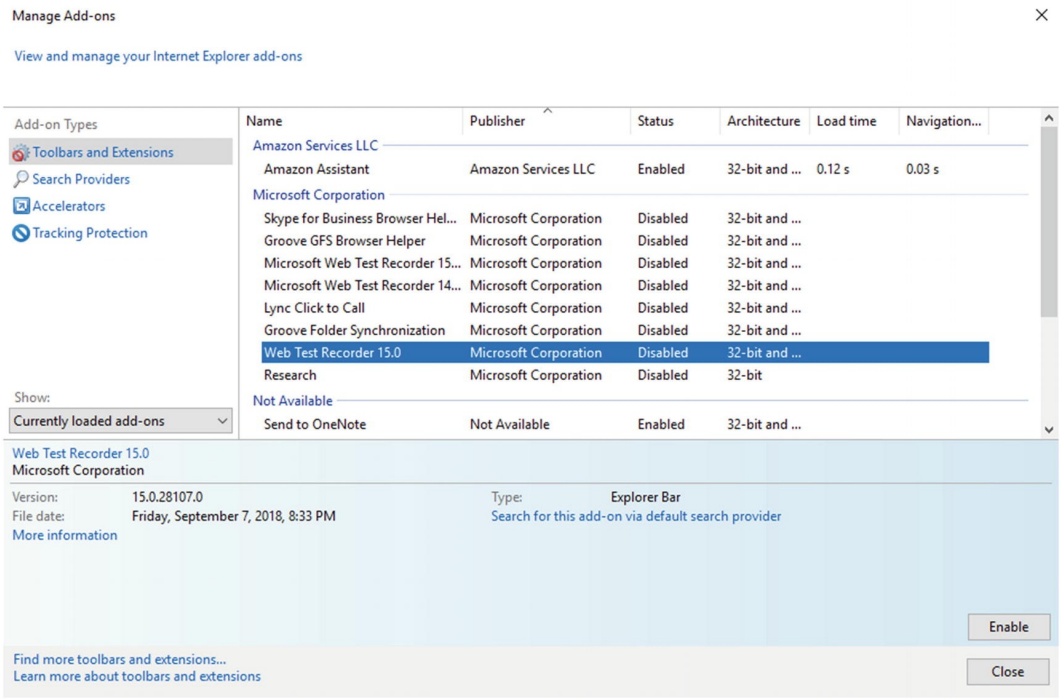

Figure 7-5 Enable the Web Test Recorder add-on

6. With the Web Test Recorder enabled, let's try to record user actions in the web application. To do that, open Web Test1.webtest file.
7. Click on the Add Recording icon (see Figure 7-6).

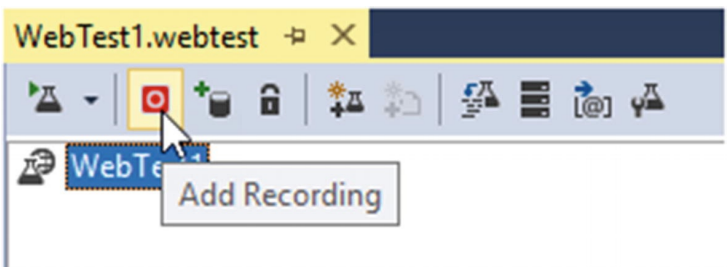

Figure 7-6 Click on Add Recording icon

After you click on the Add Recording icon, the browser will open and give the URL of the web application that needs to be tested.

8. Perform the test steps and click on the stop icon to stop recording (see Figure 7-7).

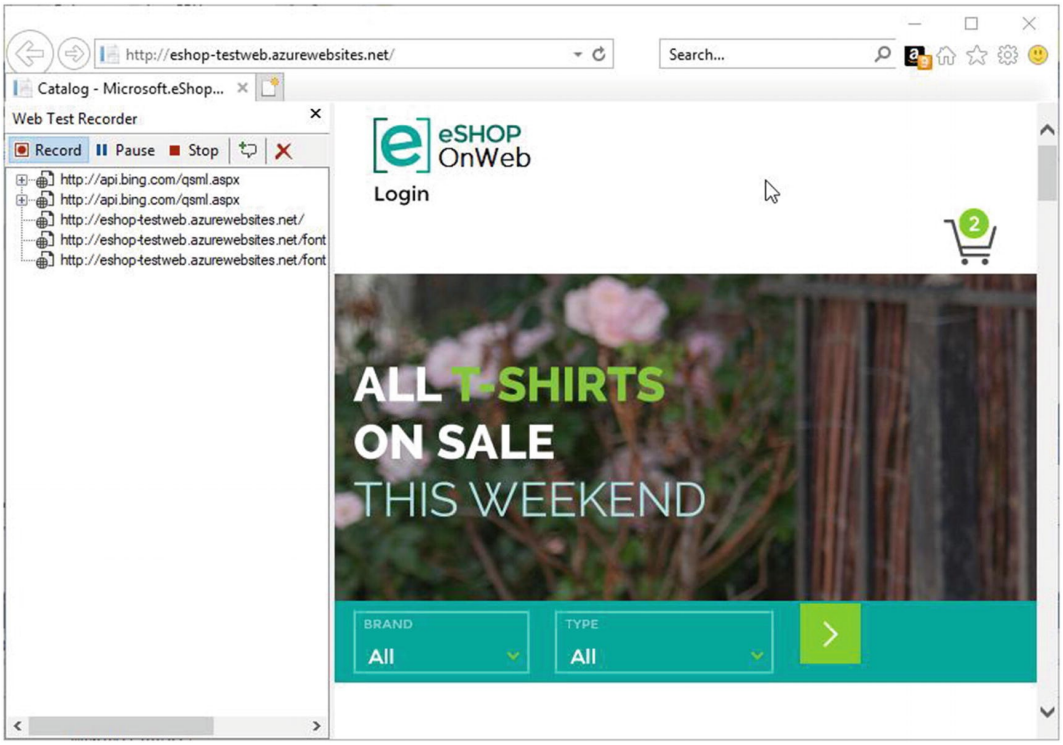

Figure 7-7 Recording actions performed on web application

9. After you click on the stop icon, you can see the web test is created with a list of http requests (see Figure 7-8).

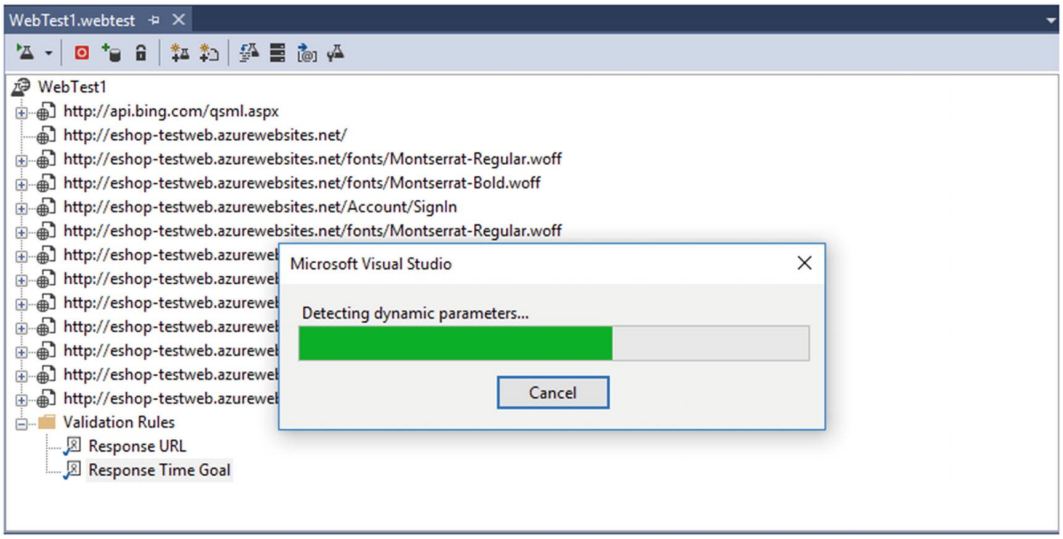

Figure 7-8 Web Test1.webtest file with a list of web requests

10. Now you can change the web properties according to performance goals and save properties (see Figure 7-9). Note that we can set the response time goals as well if they are required to be set (see Figure 7-9).

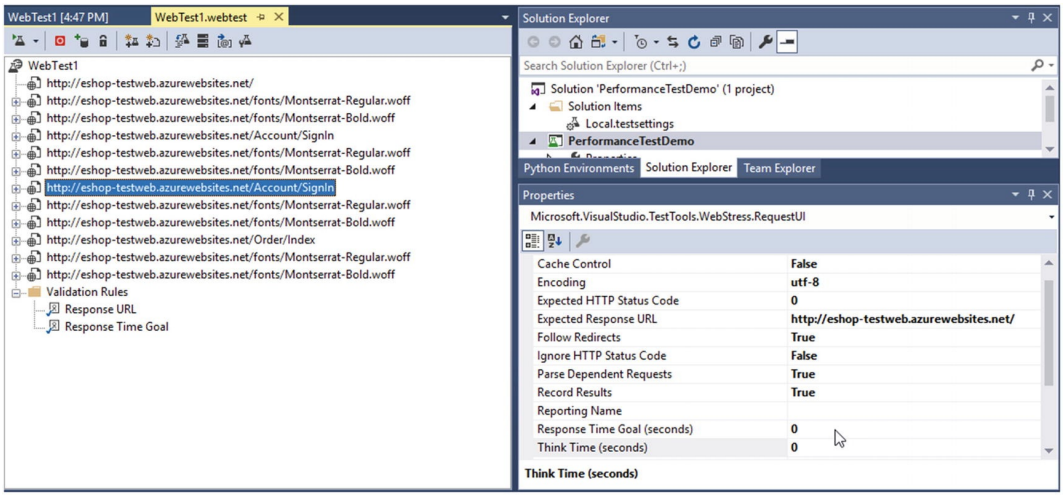

Figure 7-9 Edit test properties

11. After setting the performance goals, run the web test (see Figure 7-10).

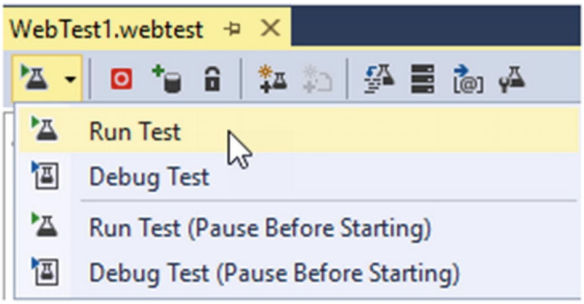

Figure 7-10 Run the web performance test

In this sample web test :

- We should set the Response time of a specific request to be 2 seconds to achieve performance goals.
- After the test execution is complete, the test result status is given as fail. But it still indicates the web request found the requested page.
- You can select failed request and see all the relevant details. In the Details tab, you would be able to find the reason for failure as **The response time (3.070 seconds) exceeded the response time goal of 2 seconds** (see Figure 7-11).

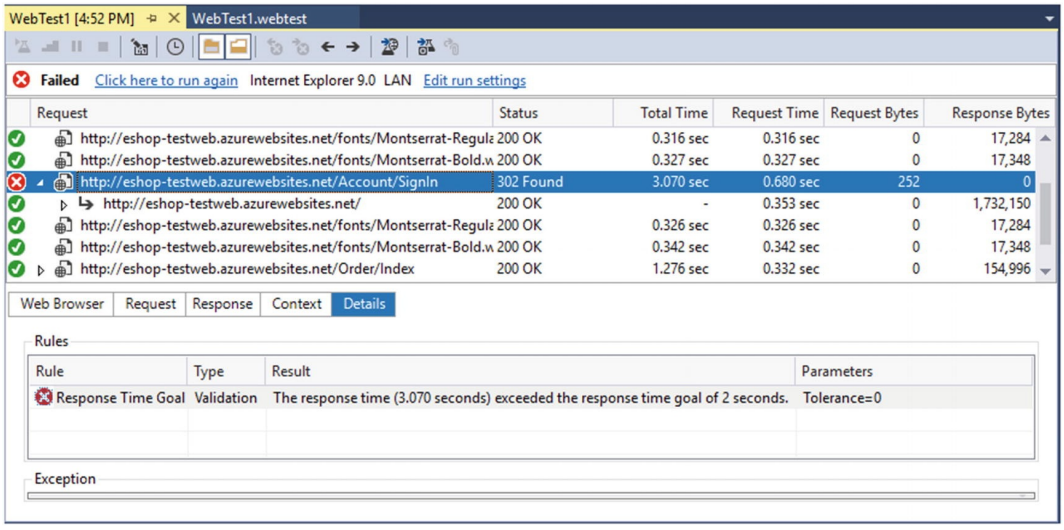

Figure 7-11 Web test run results

- It indicates that this test has failed because it took more time than our expected performance time goal.

Now you know how to create a web performance test and set performance goals. Likewise, you can change different properties and identify performance problems with the application.

Let's add a load test file to this project. To add a load test file to the project, right-click on the project in the Solution Explorer and select Add ➤ Load Test (See Figure 7-12).

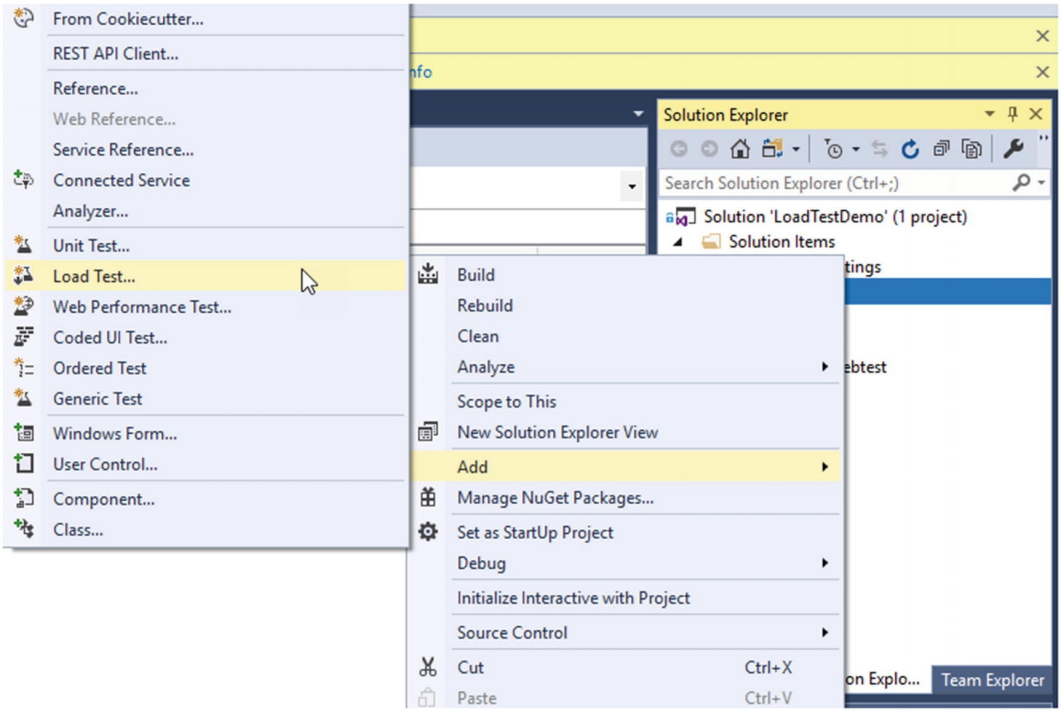

Figure 7-12 Add a load test file

The load test wizard will be opened. Select Cloud-based load test and click on the Next button (see Figure 7-13). It is important for you to have the connection made to the required Team Project in Azure DevOps in the Visual Studio Team Explorer window. Based on the selected team Project in a given Azure DevOps organization, the Azure DevOps organization URL is auto-selected as Account in this Wizard step.

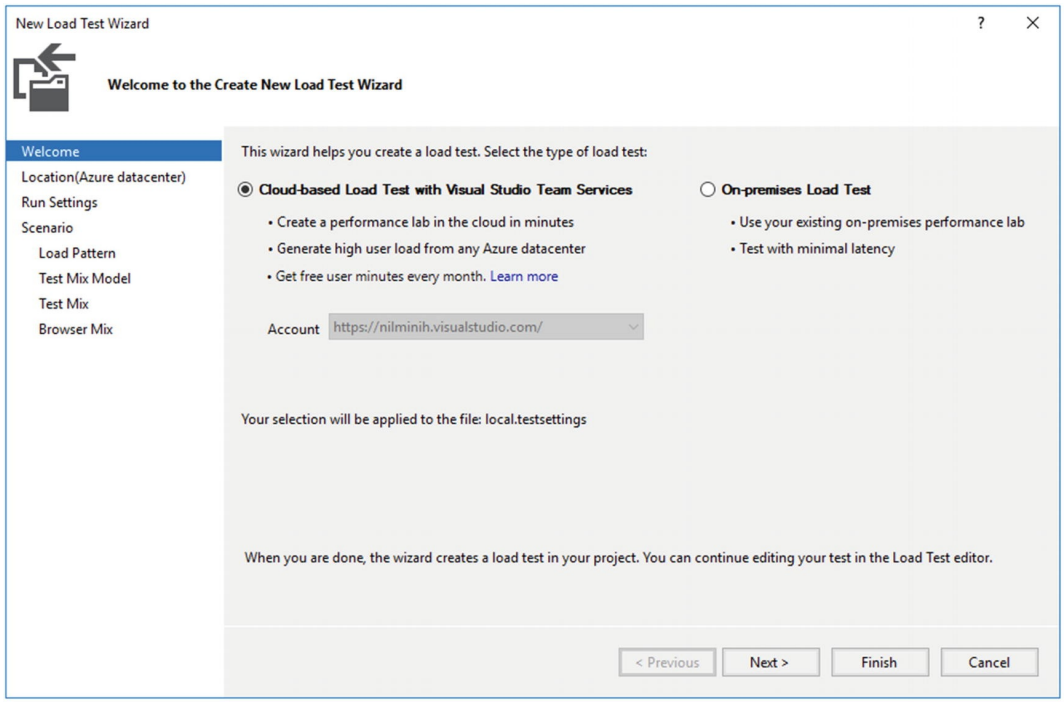

Figure 7-13 Select On-premises option

Note	On-premises load testing is deprecated in Visual Studio.

Then you can select the geographical location of the data center that is going to generate load in the tests. This is especially useful when you want to generate the load to your application from the geographical location of your application users, so that you measure the performance that would be experienced by your application end-users (see Figure 7-14).

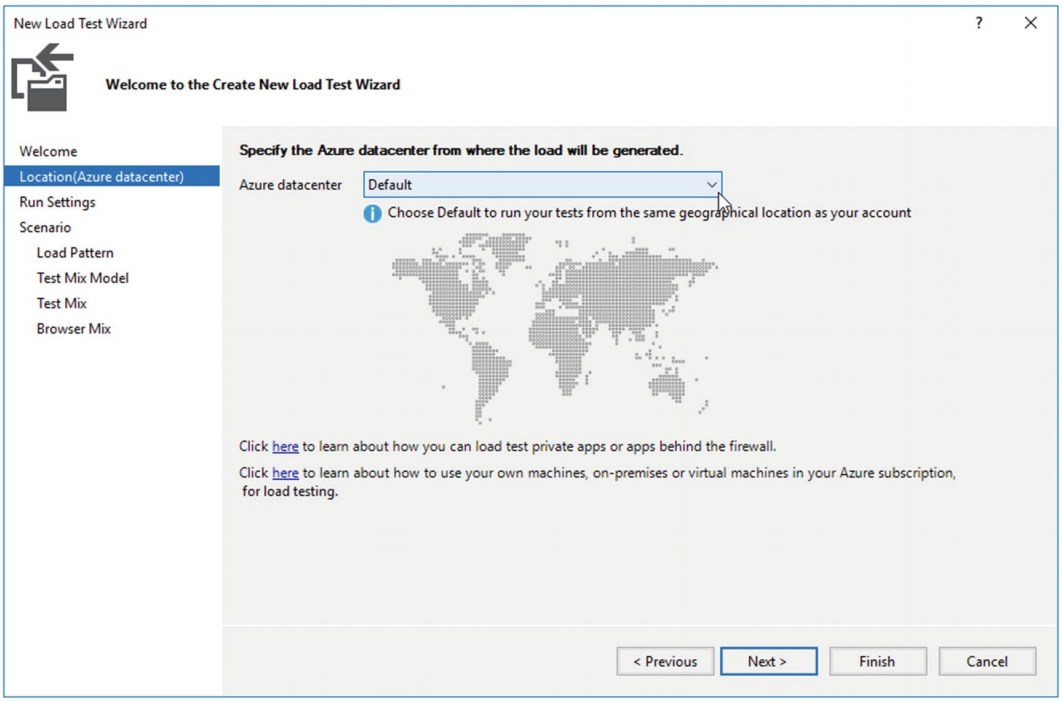

Figure 7-14 Select geo location

You would be able to add different test settings throughout the wizard. You can add Test Mix, Network Mix, and Browser Mix along with other settings. In the wizard, you will find a page that allows you to add the web tests to the load test scenario. Click on the Add button (see Figure 7-15).

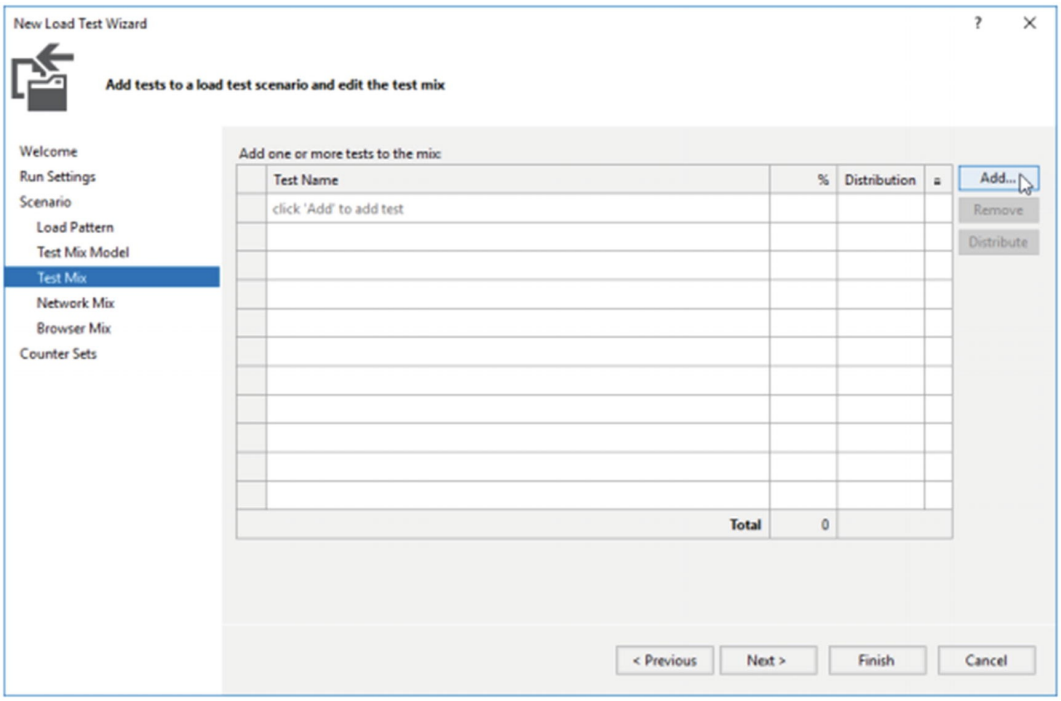

Figure 7-15 Add test mix

After you click on the Add button, the Add Tests pop-up will open. Select Test from the available test box and click on the arrow to put the test into the selected tests box (see Figure 7-16).

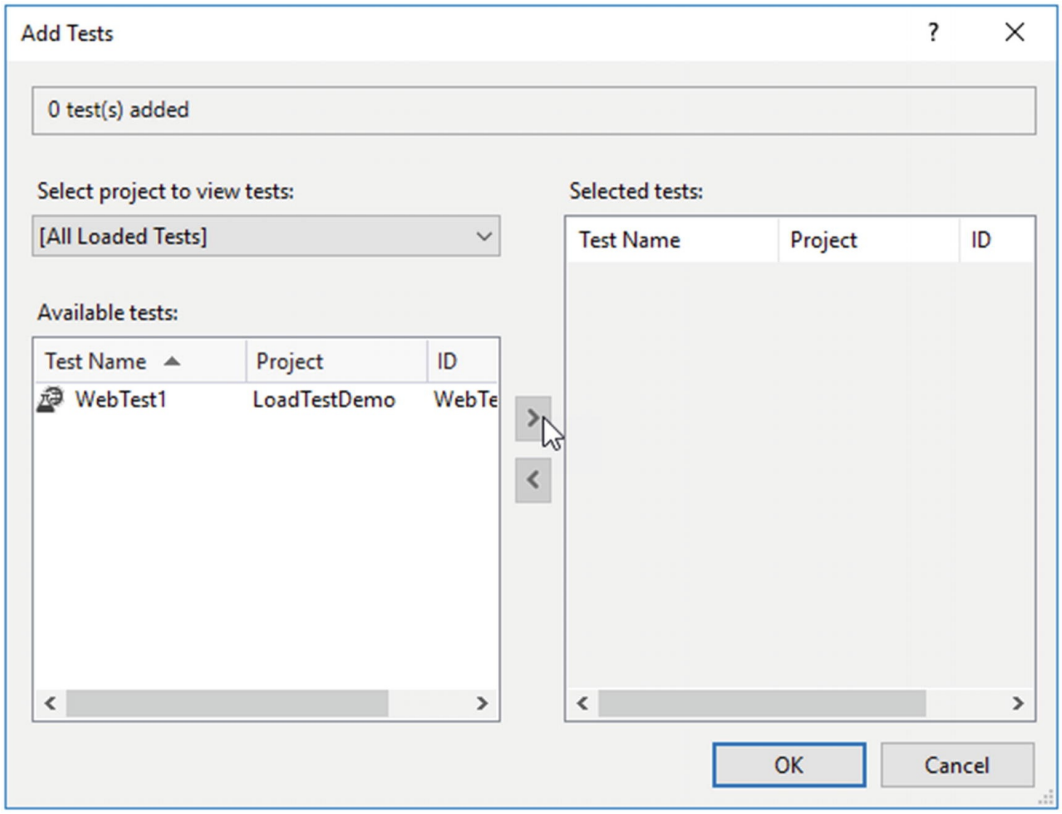

Figure 7-16 Select test from available test

Continue wizard and click on Finish to create the load test. Now we have created a basic load test with a single web test. Let's run the load test by right-clicking on the load test and selecting the Run Load Test (see Figure 7-17).

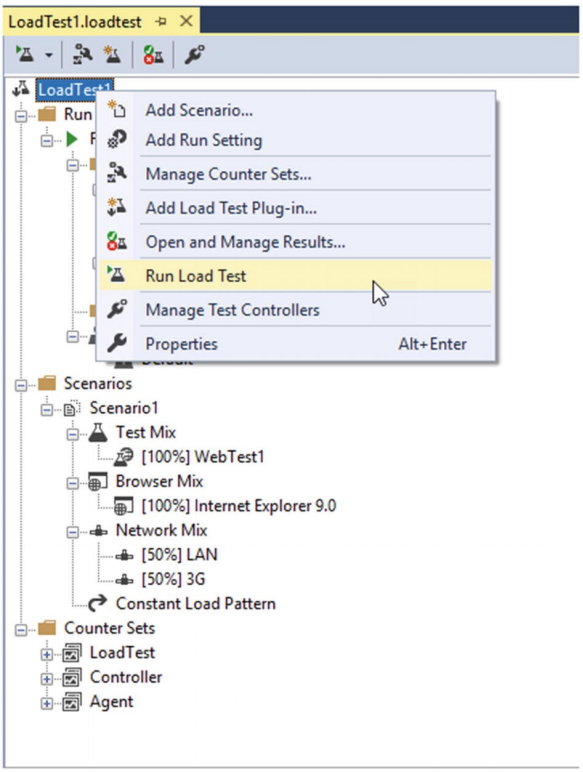

Figure 7-17 Run load test

You can see that load test starts running. You will be able to see a window with all the graphical analysis charts (see Figure 7-18).

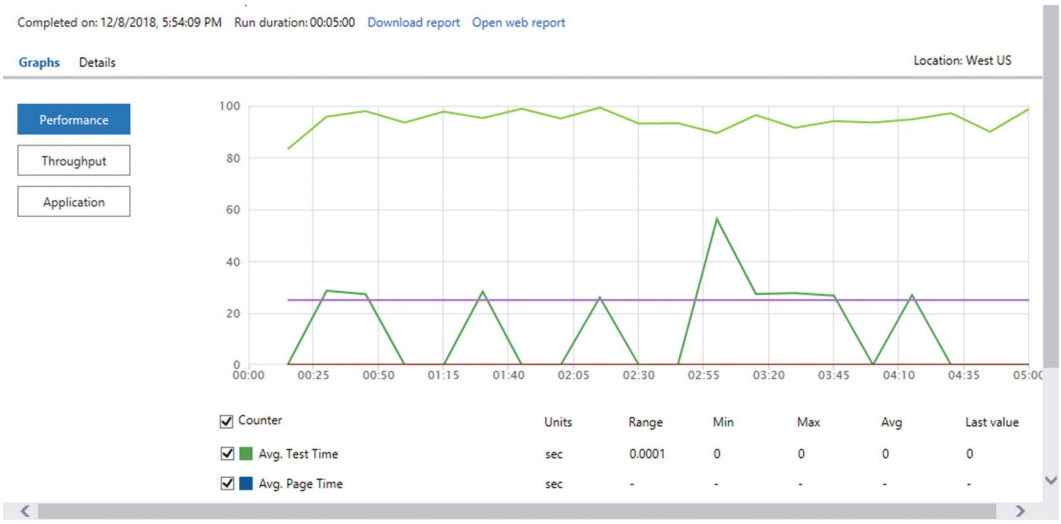

Figure 7-18 Load test running

After the test run is complete, you will be able to see the summary window with all the analyzed data of the load test. The load test that we have run in this lesson actually runs in Azure DevOps, as we have specified it as a cloud-based load test (see Figure 7-19). The load test is created in the Azure DevOps Team Project, which is connected in the Team Explorer of the Visual Studio instance that you are using for this lesson.

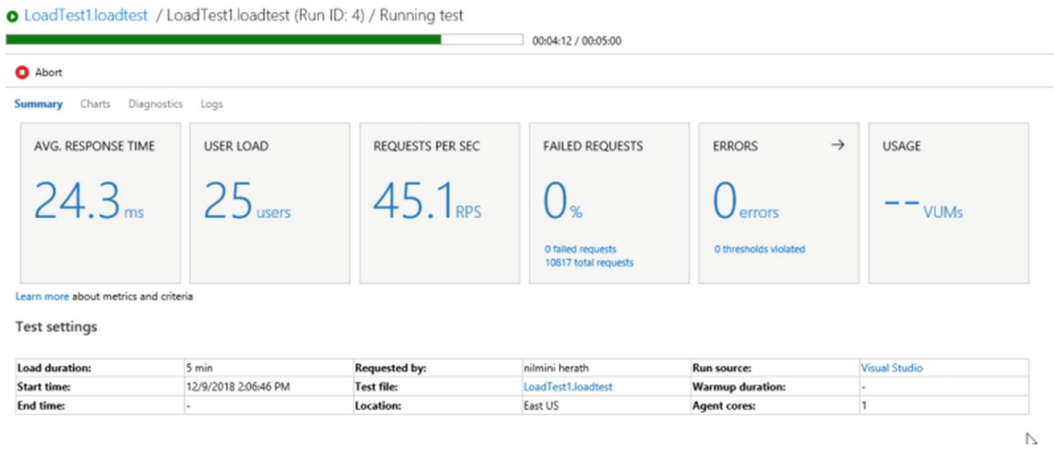

Figure 7-19 Load test running

In this lesson, we have explored how to author a load test using Visual Studio and run it as a cloud load test from the Visual Studio.

Lesson 7.02: Load Testing with Azure DevOps

This lesson will explain how to load test your application using the Azure DevOps URL-based load test option.

Prerequisites: You need to have Visual Studio Enterprise Subscription for the account you have used to log on to your Azure DevOps organization. You are familiar with using Azure DevOps.

To create a URL-based load test, open an Azure DevOps team project. Click on Load Test under Test Plan tab (see Figure 7-20).

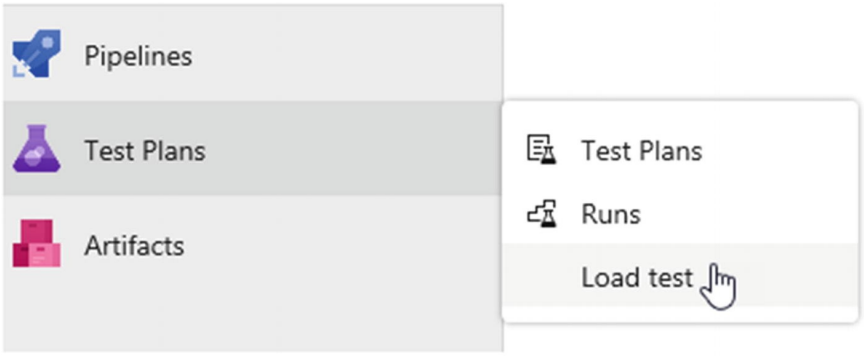

Figure 7-20 Select Load test

Select New ➤ URL-based test to create a new load test (see Figure 7-21).

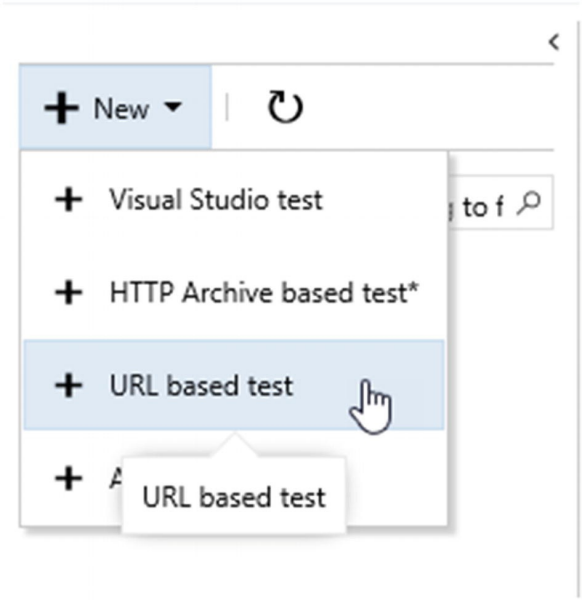

Figure 7-21 Select URL-based test

Add a name for the load test. Enter the URL you are going to test and keep HTTP method as GET. You can add multiple URLs as per your requirements and set different HTTP methods. If you want to send header values and query strings as part of the request, you can find the section where you can add the header values and the query string parameters (see Figure 7-22).

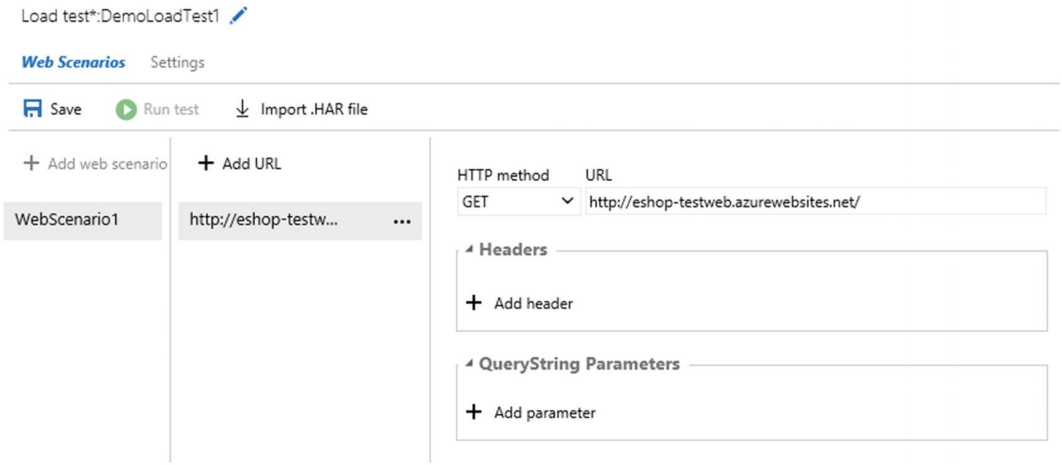

Figure 7-22 Add URL to test

Select the Settings tab. Load test parameters can be set here. You can configure, run duration, number of users, browser mix, geo location, and other settings (see Figure 7-23).

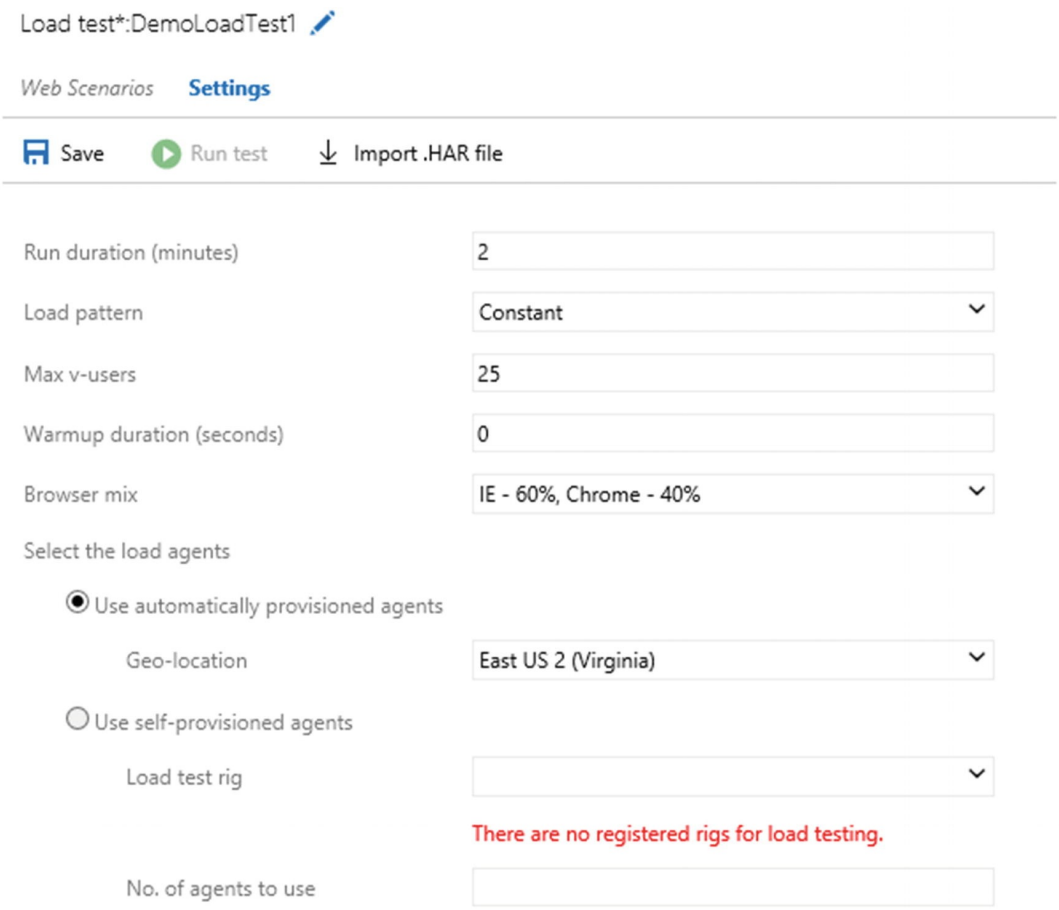

Figure 7-23 Add load test settings

After setting the parameters, save the load test. Then click on the Run Test to start the test execution (see Figure 7-24).

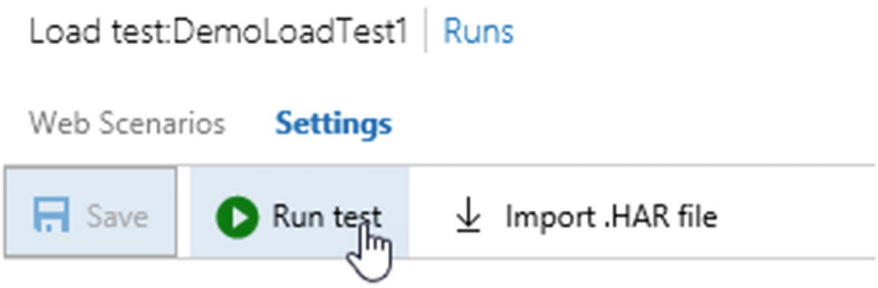

Figure 7-24 Click on Run test

The test will start and you will be able to see the live test run information (see Figure 7-25).

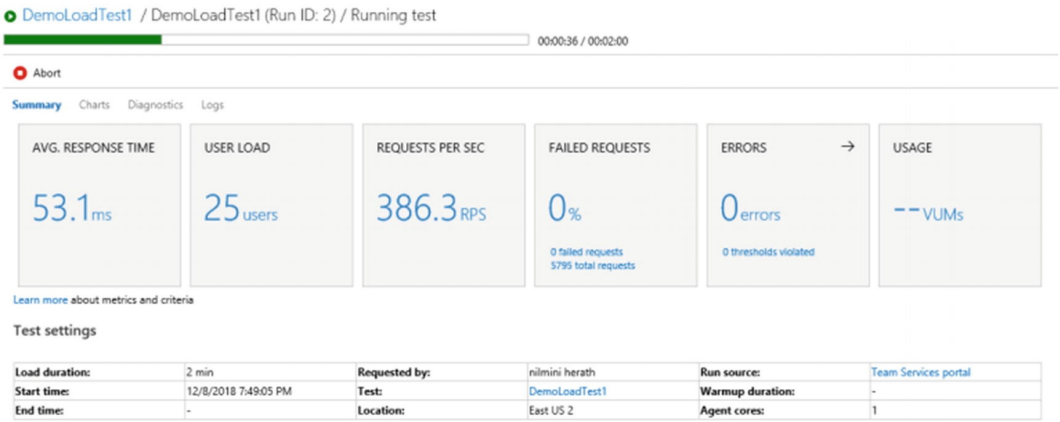

Figure 7-25 Live test run view

After the test run is completed, you can see the summary page with the load test run results (see Figure 7-26) .

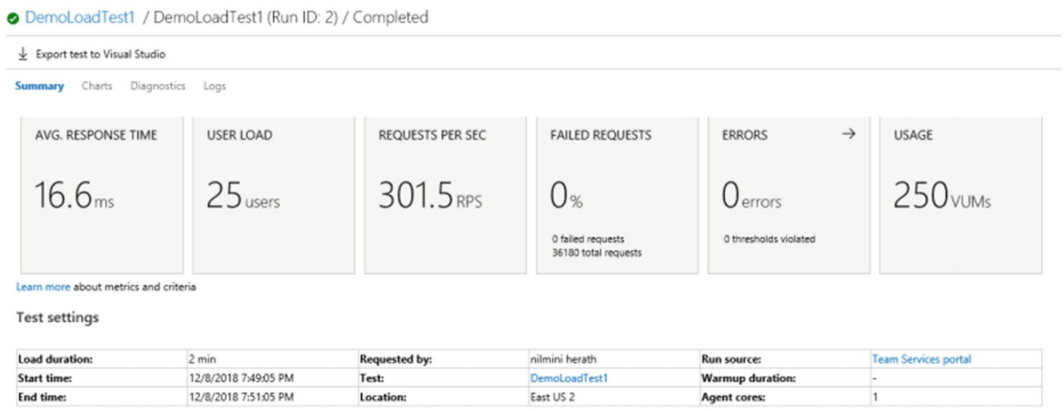

Figure 7-26 Load test run result

Further, you will be able to see the test run results as charts. Move to the Charts tab to see the test results in the charts (see Figure 7-27).

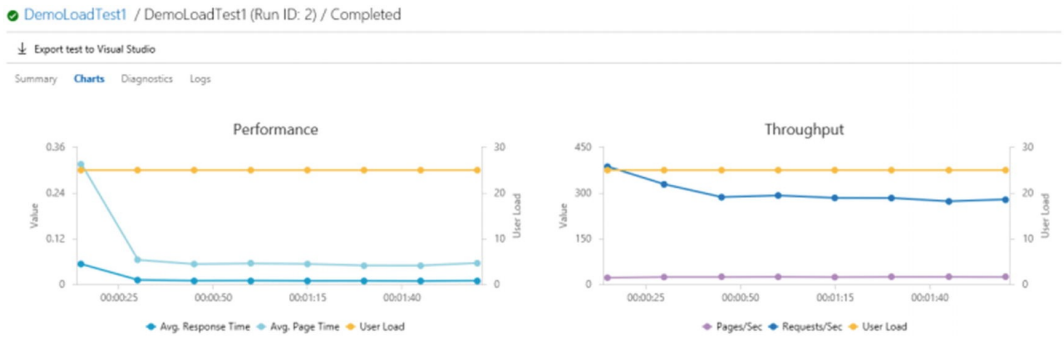

Figure 7-27 Load test result charts

Next to the Charts tab you can see the Diagnostics tab. Move to Diagnostics tab to see details of the test errors (see Figure 7-28).

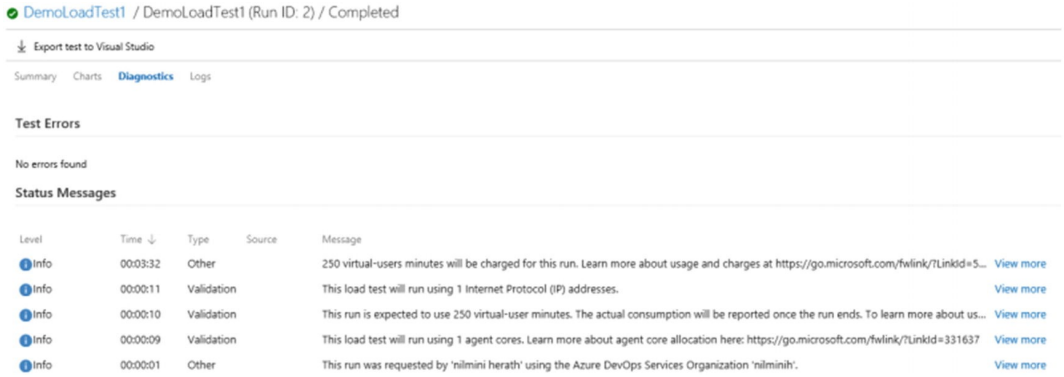

Figure 7-28 Diagnostics page

You can find the log file of the load test run inside the Logs tab (see Figure 7-29).

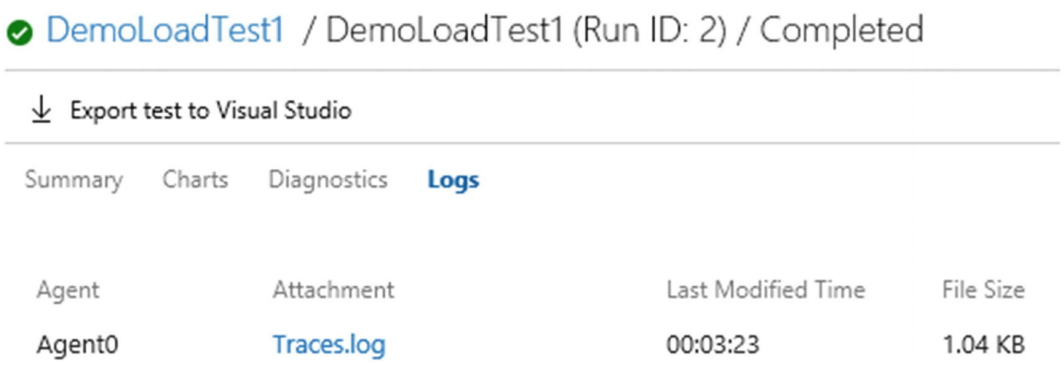

Figure 7-29 Logs file

We have explored how to author and run Cloud-based load tests in Azure DevOps in this lesson.

Lesson 7.03: Load Testing in the Azure Portal

This lesson explains how to do load tests using the Azure portal.

Prerequisites: You need to have an Azure subscription and be familiar with using the Azure portal. You must have a web application running in Azure App services. You need an Azure DevOps organization and be familiar with working with Azure DevOps.

Log in to the Azure portal and select App Services from the side menu (see Figure 7-30).

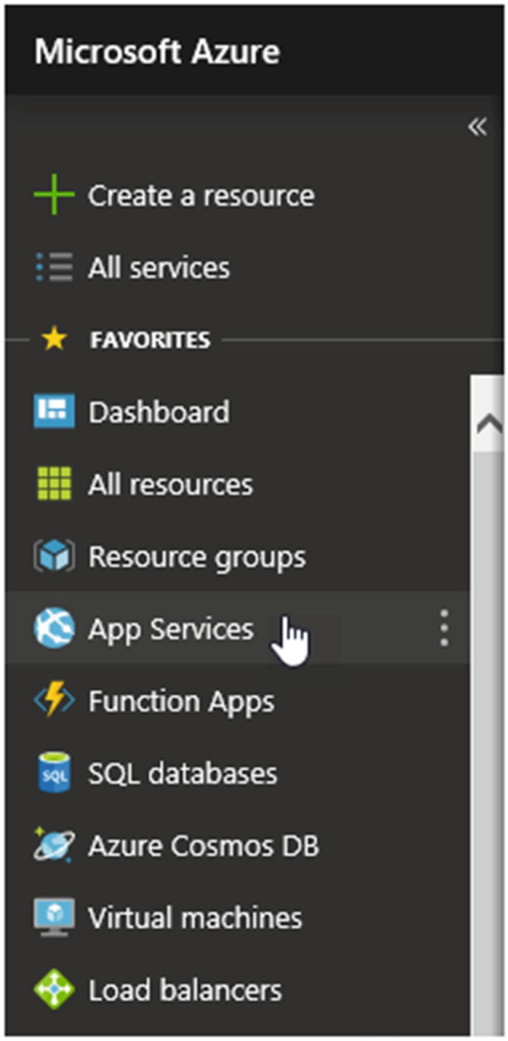

Figure 7-30 Select App services from side menu

Select the web application from the App Services window. If you do not have an application running in Azure App services, create a Visual Studio sample ASP.NET application and deploy it to the Azure app services (see Figure 7-31).

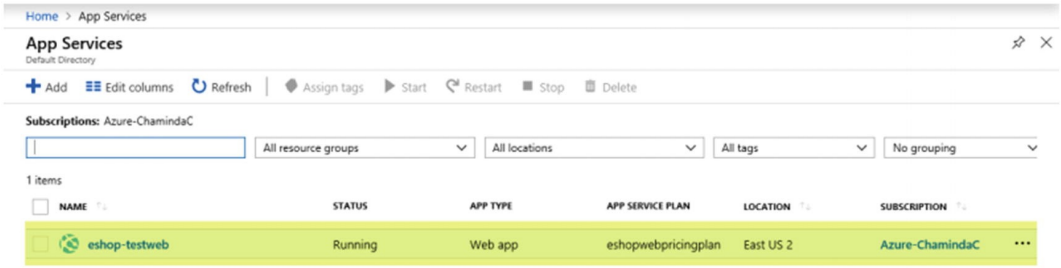

Figure 7-31 Select the web app

Select the Performance test under the Development tools section (see Figure 7-32).

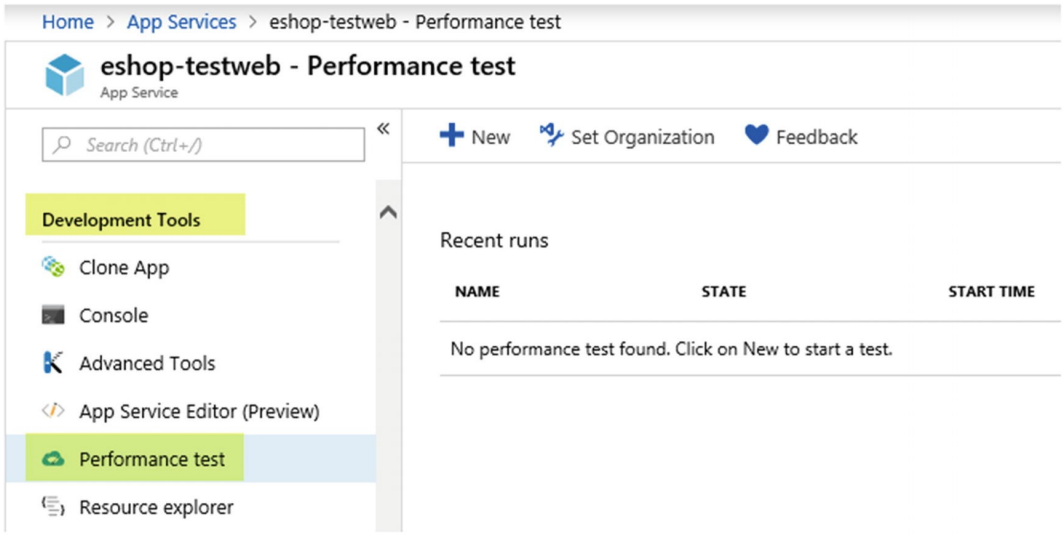

Figure 7-32 Select performance test

Now you need to connect with your Azure DevOps organization to keep the performance test history details. To connect with Azure DevOps, click on Set Organization link (see Figure 7-33).

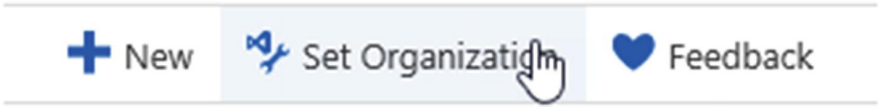

Figure 7-33 Click on Set Organization

If you already have an Azure DevOps organization, select it or you can create a new one (see Figure 7-34).

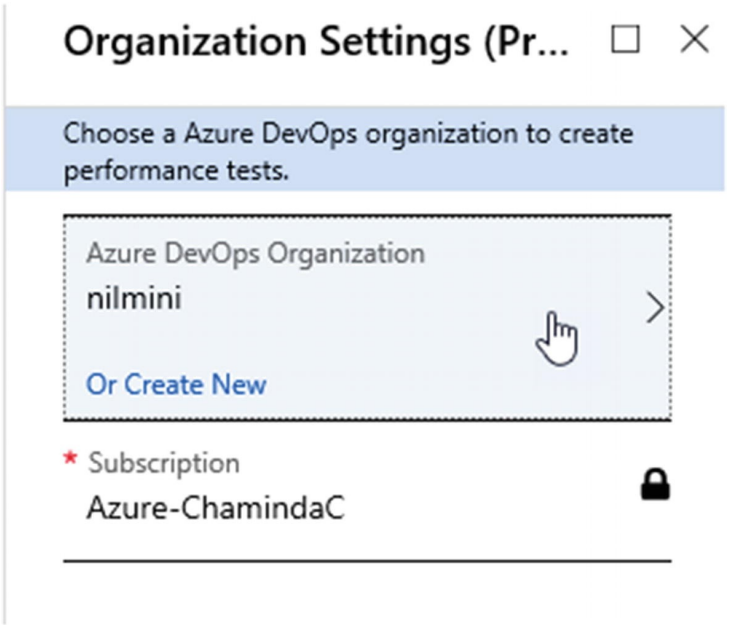

Figure 7-34 Select Azure DevOps subscription

Click on New to create a new performance test (see Figure 7-35).

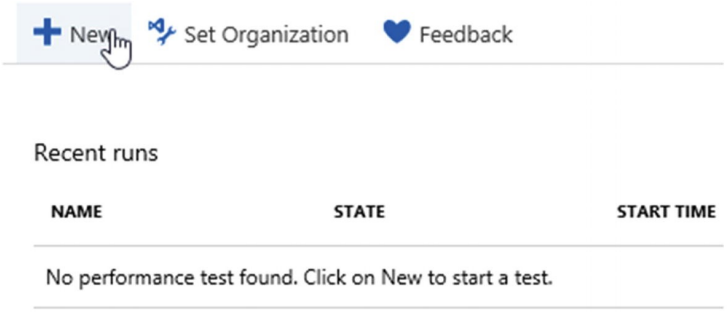

Figure 7-35 Click on New link

You can add performance settings and click on Run Test to run the test (see Figure 7-36). Additionally, you can set multiple URLs for performance testing by selecting Configure Test Using in this step. You have the option to upload a Web Test file authored in Visual Studio to enable multiple URL testing, which we will omit in this lesson for simplicity. You can experiment with creating multiple URLs with performance tests as a further learning exercise.

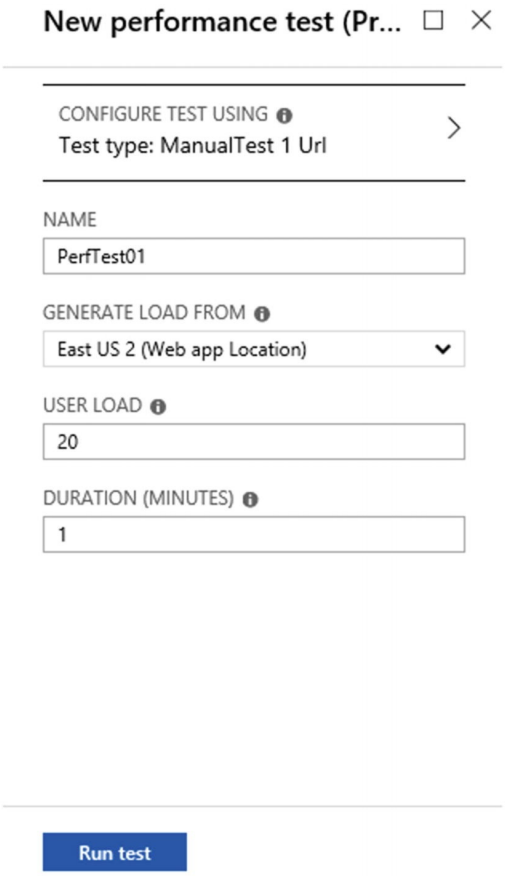

Figure 7-36 Add test run settings

After you click on the Run test, you can see the performance test has triggered as shown in Figure 7-37.

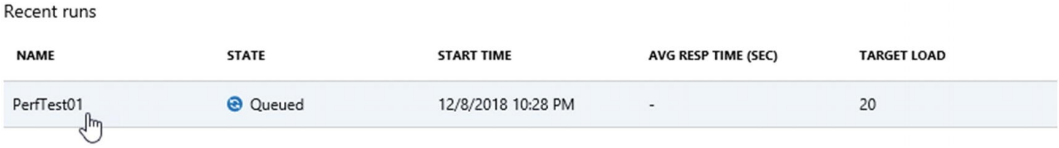

Figure 7-37 Queued performance test

Select the PerfTest01 and you can see live test run progress. After the test run is complete, you can see a detailed report of the test run in the Azure portal (see Figure 7-38).

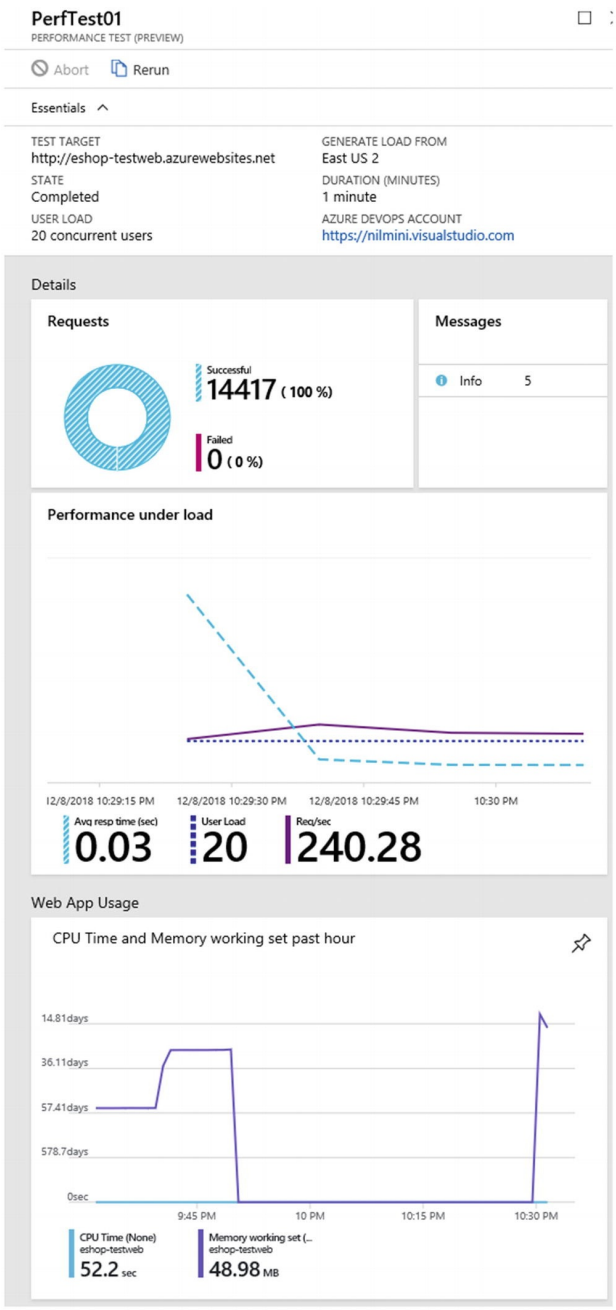

Figure 7-38 Test run result

In this lesson, we have explored load test authoring with the Azure Portal, which is available as a preview feature at the time of writing this book.

Lesson 7.04: Comparing Load Test Results

This lesson will explain how to compare load test run results.

Prerequisites: You need to have an Azure DevOps organization that has been used to practice the previous lessons. You need to have completed previous lessons and have run the load tests as instructed in the lessons.

Go to Azure DevOps and select Load Test under the Test Plan section.

You will be able to see all available load test runs. You can filter the relevant load tests using filters (see Figure 7-39).

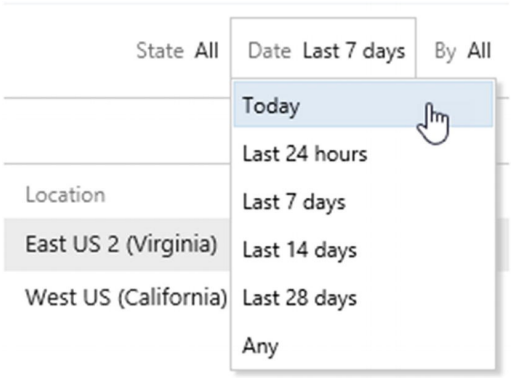

Figure 7-39 Load test filters

You can compare two load tests to verify how application performances have increased or decreased. To compare two tests, select both tests and click Compare two runs link (see Figure 7-40).

Figure 7-40 Compare two tests

Then you can see the comparison details, which you can analyze to identify how the application being tested performs with load over time (see Figure 7-41).

Figure 7-41 Performance comparison

In this lesson, you have learned how to compare load test run results in Azure DevOps.

Summary

In this chapter, we have explored options of authoring and running Cloud-based load tests involving Azure DevOps as the test running facilitator. You can use this knowledge to make sure you are delivering applications to your production environment after doing thorough analysis of the performance of the applications under real traffic and load.

In this book, we have learned wiring functional test automation using Selenium and Visual Studio. You have used C# and Python to author tests and have used few frameworks, such as MAQS and, SpecFlow to make test automation authoring more stable. Getting the tests integrated with CI/CD pipelines will come in handy as you will be able to leverage capabilities of Azure DevOps to achieve real benefits of functional test automations, which will bring your organization one more step toward being a DevOps-enabled organization. Further, the theoretical explanations and hands-on lessons in this book have given you essential knowledge for implementing functional test automations. Finally, we have explored the load testing options as well to help you implement applications that can handle real-world usage needs.

Index

A, B

C

D

E, F

G

H

I, J, K

T

U

V

W

X, Y, Z