Automation Framework using Visual Studio/C# and WordPress

Monday, March 21, 2016

4:43 PM

Author: John Steele

Phase One - Basic Setup of the Automation Framework

This framework is meant to serve as a jumping off point for our clients who don't have anything in place yet.  This will be built in Visual C# using the freely downloadable Visual Studio Community Edition 2015 but most any edition of Visual Studio should work going back to Visual Studio 2012.

* + This configuration will include installing WordPress via Microsoft WebMatrix. The link is here <https://www.microsoft.com/web/wordpress> and will serve as the generic webapp under test or WUT. It installs MySQL layer to SQL Server (I think?) plus the Microsoft Web Platform installer called WebMatrix which I think is a lite version of IIS maybe even IIS Express? See the afore mentioned link for more info.
  + Create two projects in Visual Studio. The 1st will be for the framework and will be of the type "Class Library". I’m naming it WordpressAutomation. The 2nd project will be for the WUT tests and will be of the type "Unit Test Project". I'm naming that project WordpressTests. In the references section of the WordpressTests project right click and add a reference to the 1st project or the WordpressAutomation project.
  + Framework code will go into the WordpressAutomation project. All the test code will go into the WordpressTests project. The idea is a layered approach. Beginning top first the layers are - Tests, Framework, Navigation/utils, and Selenium on  the bottom.
  + The Framework will use Selenium, the WordpressTests should not use Selenium.
  + Now we want to add Selenium to our WordpressAutomation project.
    - Select the project -> References, then select Manage NuGet Packages. In the search type with the Browse tab selected enter 'Selenium'.
    - Choose two packages. The 1st one is **Selenium.WebDriver** *.Net bindings for the Selenium WebDriver API.* The 2nd one is **Selenium.Support** *Support classes for the .Net bindings of the Selenium WebDriver API*. They install quickly and on completion you will see 2 additional references - WebDriver and WebDriver.Support for the WordpressAutomation project.
    - The Solution Explorer view on the right should look similar to the image below:

* + Machine generated alternative text:
    Microsoft Visual Studio 
    Build Debug Team 
    Debug 
    m NuGet WordpressAutomation -G X UnitTestI .cs 
    WordpressAutomation - 
    Edit View Project 
    • System.Core 
    • System.Data 
    • System.XmI 
    Tools Test 
    Any CPU 
    Classl .cs 
    Analyze 
    Window Help 
    start 
    Quick Launch (Ctrl+Q 
    Solution Explorer 
    John R. Steele • 
    NuGet Package Manager: WordpressAutomation 
    Browse 
    Selenium 
    Installed Updates 
    Package source: 
    Selenium.Support 
    nuget.org 
    Include prerelease 
    Selenium.WebDriver 1.05M downloads 
    .NET bindings for the Selenium WebDriver API 
    Selenium.Support 844K downloads 
    Support classes for the .NET bindings of the Selenium WebDriver API 
    Selenium.WebDriver.PhantomJS.Xplatform by Selenium Committers, Eberhard 8eilharz, 8.1K downloads 
    .NET bindings for the Selenium WebDnver API 
    Selenium.WebDriverBackedSelenium 133K downloads 
    Implementation of the Selenium Remote Control (RC) API, using WebDriver technology 
    Selenium.WebDriver.Mono by Selenium Commiters, 3.76K downloads 
    .NET bindings for the Selenium WebDriver API 
    Selenium Chloride by Daniel Charlton, 856 downloads 
    A reagent for Selenium 
    SpecBind.Selenium 4.88K downloads 
    An eRenslon to Spec8ind that allows it to integrate with Selenium. 
    Each package is licensed to you by cwner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages. 
    Do not show this again 
    Output 
    Show output from: Package Manager 
    O v2.i3.o 
    O v2.i3.o 
    v2A5.o.1 
    v2.i3.o 
    v2A5.o 
    VI 1.0.8 
    VI 62.232 
    Installed: 
    Version: 
    2.53.o 
    2.52.o 
    Uninstall 
    U pdate 
    @Options 
    Description 
    Selenium is a set of different software tools each with a different approach to 
    supporting browser automation. These tools are highly flexible, allowing many 
    options for locating and manipulating elements within a browser, and one of 
    its key features is the support for automating multiple browser platforms. This 
    package contains .NET support classes for the Selenium WebDriver API, which 
    includes helper classes for HTML Select elements, waiting for conditions, and 
    Page Object creation. 
    Author(s): 
    Date publis hed: 
    Project U RL: 
    Report Abuse: 
    Tags: 
    2.52.o 
    Selenium Committers 
    http://www.apache.org/Iicenses/LICENSE-2.O 
    Friday, February 12, 2016 (2/12/2016) 
    https://github.com/SeIeniumHQ/seIenium 
    https://www.nuget.org/packages/SeIenium.Support/2.52.O/ 
    ReportAbuse 
    Selenium, browser, WebDriver, automation, suppot 
    Search Solution Explorer (Ctrl+,•) 
    Solution 'WordpressAutomation' (2 proje 
    @ WordpressAutomation 
    Properties 
    References 
    Analyzers 
    Microsoft.CSharp 
    • System 
    • System.Data.DataSetExtensions 
    • System.Drawing 
    • System.Net.Http 
    System .XmI.Linq 
    WebDriver 
    WebDnverSu ort 
    C* Classl 
    packages.config 
    WordpressTests 
    properties 
    References 
    Analyzers 
    Microsoft.Visua[Studio.QuaIit,' 
    WordpressAutomation 
    C* UnitTestI 
    Solution Explorer 
    Team Explorer Class View 
    Properties 
    WordpressTests Project Properties 
    Project File 
    Project Folder 
    WordpressTests.csproj 
    an Auti 
    Attempting to gather dependencies information for package 'Selenium.Support.2.53.ß' with respect 
    Attempting to resolve dependencies for package 'Selenium.Support.2.53.ß' with Dependencyaehavior 
    Resolving actions to install package 'Selenium.Support.2.53.ß' 
    Resolved actions to install package 'Selenium.Support.2.53.ß' 
    to project 
    ' Lowest ' 
    'WordpressAutomation ' 
    targeting 
    Adding package 'Selenium.Support.2.53.ß' to folder an Automated Testing Framework With 
    Added package 'Selenium.Support.2.53.ß' to folder an Automated Testing Framework With ' 
    Added package 'Selenium.Support.2.53.ß' to 'packages. config' 
    Successfully installed 'Selenium.Support 2.53.ß' to WordpressAutomation 
    Output 
    Error List 

* + Now we need to run a simple test just to make sure our configuration is working in preparation for doing real work and real tests. In the Class1.cs file add the following code to the existing namespace and Class1 default template of the **WordpressAutomation** project:
    - **Note**: Keep in mind that knowledge of C# and object oriented programming will help as you look at and study these modules.

namespace WordpressAutomation

{

    public class Class1

    {

        public void Go()

        {

            var driver = new FirefoxDriver();

            driver.Navigate().GoToUrl("<http://google.com>");

        }

    }

}

* + Remember to include the import of "using OpenQA.Selenium.Firefox;" above.
  + In the UnitTest1.cs add the following code to the default template in the **WordpressTests** project:

namespace WordpressTests

{

    [TestClass]

    public class UnitTest1

    {

        [TestMethod]

        public void TestMethod1()

        {

            var c = new Class1();

            c.Go();

        }

    }

}

* + Remember to include the import of "using WordpressAutomation" statement above.
  + Now you should be able to run the above code. Right click somewhere on above Test Class code and select Run Tests from the context menu.
    - You should have a Test Explorer window come into view in Visual Studio with the passing results of this test. If you got a passing result your configuration is good and ready to start writing code for the real framework and tests.

Phase Two - Implement First Real Smoke Test

* + With Phase One completed the idea now is to transform our framework and tests to do the 1st real smoke test. To do that I'm going to show the implementation in the class modules we've already done plus a couple others.
  + In the **WordpressAutomation** project the Class1.cs file is renamed to LoginPage.cs will now have the following code:

* + **NOTE**: There has been code added to this module to allow for timing of the delay of the web page to appear. Remember this is Framework code and as such things like timing should be part of the Framework, and should not be the responsibility for tests in the WordpressTests project. Also note the addition of the LoginAs() method and the LoginCommand class with its Login() method.
  + **Note**: Keep in mind that knowledge of C# and object oriented programming will help as you look at and study these modules.

using OpenQA.Selenium;

using OpenQA.Selenium.Support.UI;

using System;

/// <summary>

/// LoginPage.cs

/// \*\*Framework\*\*

/// Performs the navigation to the landing page for beginning the test. Additionally provides

/// for Login functionality.

/// </summary>

namespace WordpressAutomation

{

    public class LoginPage

    {

        public static void GoTo()

        {

            Driver.Instance.Navigate().GoToUrl("<http://localhost:2329/wp-login.php>");

|  |  |
| --- | --- |
| var wait = new WebDriverWait(Driver.Instance, TimeSpan.FromSeconds(10)); |  |

            wait.Until(d => d.SwitchTo().ActiveElement().GetAttribute("id") == "user\_login");

        }

        public static LoginCommand LoginAs(string userName)

        {

            return new LoginCommand(userName);

        }

    }

    public class LoginCommand

    {

        private readonly string userName;

        private string password;

         public LoginCommand(string userName)

        {

            this.userName = userName;

        }

        public LoginCommand WithPassword(string password)

        {

            this.password = password;

            return this;

        }

        public void Login()

        {

            var loginInput = Driver.Instance.FindElement(By.Id("user\_login"));

            loginInput.SendKeys(userName);

            var passwordInput = Driver.Instance.FindElement(By.Id("user\_pass"));

            passwordInput.SendKeys(password);

            var loginButton = Driver.Instance.FindElement(By.Id("wp-submit"));

            loginButton.Click();

        }

    }

}

A new class "Driver.cs" has been added to the **WordpressAutomation** project and contains the following:

**NOTE**: This sets up the instance of the WebDriver Firefox browser object which belongs to the domain of the Framework and not the Test layer.

using OpenQA.Selenium;

using OpenQA.Selenium.Firefox;

using System;

namespace WordpressAutomation

{

    public class Driver

    {

        public static IWebDriver Instance { get; set; }

        public static void Initialize()

        {

            Instance = new FirefoxDriver();

            Instance.Manage().Timeouts().ImplicitlyWait(TimeSpan.FromSeconds(5));

        }

        public static void Close()

        {

            Instance.Close();

        }

    }

}

Another new class "DashboardPage.cs" has been added to the **WordpressAutomation** project and contains the following:

**NOTE**: This module is for verification that the login was successful by looking for the main header entitled "Dashboard" is present on the landing page.

using OpenQA.Selenium;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WordpressAutomation

{

    public class DashboardPage

    {

        public static bool IsAt

        {

            get

            {

                var h1s = Driver.Instance.FindElements(By.TagName("h1"));

                if (h1s.Count > 0)

                    return h1s[0].Text == "Dashboard";

                return false;

            }

        }

    }

}

Finally in the **WordpressTests** project the UnitTest1.cs file has been renamed to LoginTests.cs has been modified as follows:

**NOTE**: This module represents the Test layer. This is where the QA/Test team will write or modify the tests to be executed on the WUT. Also note the presence of attributes such as [TestMethod], [TestClass] and so forth. They are necessary attributes for the structure of a test file in Visual Studio. Notice how the Framework is doing the test overhead work of initialization, navigation, and test cleanup. Finally change the username "*admin*" and password "test123" with whatever credentials used in the setup of Wordpress app, or create a new user just for the purpose of testing.

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using WordpressAutomation;

namespace WordpressTests

{

    [TestClass]

    public class LoginTests

    {

        [TestInitialize]

        public void Init()

        {

            Driver.Initialize();

        }

        [TestMethod]

        public void Admin\_User\_Can\_Login()

        {

            LoginPage.GoTo();

            LoginPage.LoginAs("admin").WithPassword("test123").Login();

            Assert.IsTrue(DashboardPage.IsAt, "Failed to login.");

        }

        [TestCleanup]

        public void CleanUp()

        {

            Driver.Close();

        }

    }

}

Run the smoke test

With the LoginTests.cs module selected, right click anywhere in the code editor and select Run Tests from the context menu. You should see the Output window of Visual Studio that the test has started, or alternatively if there is some kind of syntax or build error, that will be indicated likewise. If the test is successful the Test Explorer window will appear and the result of Pass or Fail will be indicated therein. **Remember** in the LoginTests.cs to use the username and password credentials supplied at the setup of the WordPress web app if the test continually fails at login.

**NOTE**: For debugging purposes it might be helpful to comment out the "Driver.Close" statement by preceding it with "//" (2 forward slashes) in the [TestCleanup] section of LoginTests.cs. Then the browser will not close on completion of the test.

Visual Studio and particularly Solution Explorer on the left should look similar to the following at this point:

Machine generated alternative text:
WordpressAutomation • Microsoft Visual Studio 
File Edit View Project Build Debug Team 
Debug 
Tools Test Analyze Window 
Help 
Quick Launch (Ctrl•Q) 
Solution Explorer 
John R. Steele 
Any cpu 
DashboardPage.cs 
WordpressTests 
us Ing 
using 
using 
Driver.cs 
System; 
LoginPage.cs 
Start •I - 
LoginTests.cs x 
Word pressTests.LoginTests 
WebDriverWait [from metadata' 
Init0 
Microsoft . Visual Studio. TestTooIs. UnitTesting; 
WordpressAutomat ion ; 
0 41 o 
Search Solution Explorer (Ctrl e;) 
Solution *hrdpressAutomation• (2 projects) 
WordpressAutomation 
Properties 
References 
c• DashboardPage.cs 
Ca Driver. cs 
c• LoginPage.cs 
packages.config 
WordpressTests 
> Properties 
References 
> LoginTests.cs 
11 
17 
100% 
Output 
- namespace WordpressTests 
[TestClass] 
public class LoginTests 
[Testlnitialize] 
public void 
Driver. Initial ize() ; 
[Testmethod] 
public void 
Logi nPage. GOTO( ) ; 
Logi nPage . LoginAs( "admin" ) . . Login( ) ; 
Assert. 15True(DashboardPage .1sAt, "Failed to login. " ) ; 
[TestCIeanup] 
public void CleanUp() 
Driver. Close() 
Solution Explorer 
Properties 
ieam Explorer Class View 
Show output from: Tests 
- Run test started - 
Run test finished: 
Build succeeded 
1 run 

Phase 3 - Creating 2 Smoke Tests

1. Create and Post a Blog and edit a page

2. Edit a page

* + **Note**: Keep in mind that knowledge of C# and object oriented programming will help as you look at and study these modules.

* + Add a new class in the WordpressTests project called "CreatePostTests.cs". This will be the test runner that will create a new blog post, add a title, add a sentence to the body of the post, publish the post, and then verifies the title is what it should be. The code for this class module will be the following:

using Microsoft.VisualStudio.TestTools.UnitTesting;

using WordpressAutomation;

/// <summary>

/// CreatePostTests.cs

/// \*\*Test\*\*

/// Selects the menu to create a new blog, fills in the title and body with some

/// text, publishes the new blog, then selects the link to view the new blog and

/// verifies the title is correct.

/// </summary>

namespace WordpressTests

{

    [TestClass]

    public class CreatePostTests

    {

        [TestInitialize]

        public void Init()

        {

            Driver.Initialize();

        }

        [TestMethod]

        public void Can\_Create\_A\_Basic\_Post()

        {

            LoginPage.GoTo();

            LoginPage.LoginAs("admin").WithPassword("test123").Login();

            NewPostPage.GoTo();

            NewPostPage.CreatePost("This is the test post title").WithBody("Hi this is the body").Publish();

            NewPostPage.GoToNewPost();

            Assert.AreEqual(PostPage.Title, "This is the test post title", "Title did not match new post.");

        }

        [TestCleanup]

        public void CleanUp()

        {

            Driver.Close();

        }

    }

}

* + Add a new class in the WordpressTests project called "PageTests.cs". This will be the test runner that will select the existing Sample Page and then verifies the page is in Edit mode. The code for this class module will be the following:

using Microsoft.VisualStudio.TestTools.UnitTesting;

using WordpressAutomation;

namespace WordpressTests

{

    /// <summary>

    /// Test the Pages functionality, which is similar to Post tests.

    /// </summary>

    [TestClass]

    public class PageTests

    {

        [TestInitialize]

        public void Init()

        {

            Driver.Initialize();

        }

        [TestMethod]

        public void Can\_Edit\_A\_Page()

        {

            LoginPage.GoTo();

            LoginPage.LoginAs("admin").WithPassword("test123").Login();

            ListPostsPage.GoTo(PostType.Page);       // general purpose class this will work for both posts and pages

            ListPostsPage.SelectPost("Sample Page");    // or you could add an index [0] or [1] for the 1st or 2nd post of that page

            //Assert.IsTrue(NewPostPage.IsInEditMode(), "Wasn't in edit mode");

            Assert.AreEqual("Edit Page Add New", NewPostPage.Header(), "Header did not match");

            Assert.AreEqual("Sample Page", NewPostPage.Title, "Title did not match");

        }

        [TestCleanup]

        public void CleanUp()

        {

            // Driver.Close();

        }

    }

}

* + Add the following NewPostPage.cs class module to the WordpressAutomation project. This will be part of the framework and contains methods to support the blog post tests.

using OpenQA.Selenium;

using System;

using System.Threading;

/// <summary>

/// NewPostPage.cs

/// \*\*Framework\*\*

/// Methods to support blog post tests and page tests.

/// </summary>

namespace WordpressAutomation

{

    public class NewPostPage

    {

        public static String Title

        {

            get

            {

                var title = Driver.Instance.FindElement(By.Id("title"));

                if (title != null)

                    return title.GetAttribute("value");

                return string.Empty;

            }

        }

        public static void GoTo()

        {

            var menuPosts = Driver.Instance.FindElement(By.Id("menu-posts"));

            menuPosts.Click();

            var addNew = Driver.Instance.FindElement(By.LinkText("Add New"));

            addNew.Click();

        }

        public static CreatePostCommand CreatePost(string title)

        {

            return new CreatePostCommand(title);

        }

        public static void GoToNewPost()

        {

            var message = Driver.Instance.FindElement(By.Id("message"));    // View the new post just created.

            var newPostLink = message.FindElements(By.TagName("a"))[0];

            newPostLink.Click();

        }

        public static string Header()

        {

                var header = Driver.Instance.FindElement(By.XPath("//div[@id='wpbody-content']/div[@class='wrap']/h1[contains(text(),'Edit Page')]"));

                if (header.Text == "Edit Page Add New")

                    return header.Text;

                else return string.Empty;

         }

    }

    public class CreatePostCommand

    {

        private readonly string title;

        private string body;

        public CreatePostCommand(string title)

        {

            this.title = title;

        }

        public CreatePostCommand WithBody(string body)

        {

            this.body = body;

            return this;

        }

        public void Publish()

        {

            Driver.Instance.FindElement(By.Id("title")).SendKeys(title);

            Driver.Instance.FindElement(By.Id("content")).SendKeys(body);

            Driver.Instance.FindElement(By.Id("publish")).Click();

            Thread.Sleep(2000); // wait some time for the publish to process

        }

    }

}

Add the following PostPage.cs to the WordpressAutomation project as it is also part of the framework.

using OpenQA.Selenium;

using System;

/// <summary>

/// PostPage.cs

/// \*\*Framework\*\*

/// Helper to provide verification that the newly created blog title is accurate.

/// </summary>

namespace WordpressAutomation

{

    public class PostPage

    {

        public static string Title

        {

            get

            {

                var title = Driver.Instance.FindElement(By.ClassName("entry-title"));

                if (title != null)

                    return title.Text;

                return String.Empty;

            }

        }

    }

}

Finally add the ListPostsPage.cs class module to the WordpressAutomation project. Contains methods to support menu selection and Sample Page selection in PageTests.

using OpenQA.Selenium;

/// <summary>

/// Methods to support menu selection and Sample Page selection in PageTests

/// </summary>

namespace WordpressAutomation

{

    public class ListPostsPage

    {

        public static void GoTo(PostType postType)

        {

            switch (postType)

            {

                case PostType.Page:

                    Driver.Instance.FindElement(By.Id("menu-pages")).Click();

                    Driver.Instance.FindElement(By.LinkText("All Pages")).Click();

                    break;

            }

        }

        public static void SelectPost(string title)

        {

            var postLink = Driver.Instance.FindElement(By.LinkText("Sample Page"));

            postLink.Click();

        }

    }

    public enum PostType

    {

        Page

    }

}

Your Visual Studio Solution Explorer should look similar to the following:

Machine generated alternative text:
WordpressAutomation - 
Edit View Project 
Microsoft Visual Studio 
Build Debug Team 
Debug 
Tools Test 
Any CPU 
Analyze Window 
Start • 
Help 
LoginPage.cs 
Quick Launch (Ctrl+Q 
Solution Explorer 
Search Solution Explorer (Ctrl+,•) 
John R. Steele • 
m Driver.cs 
DashboardPage.cs 
PostPage.cs 
NewPostPage.cs 
QVcrdpressTests.PageTests 
ListPostsPage.cs 
LoginTests.cs 
@WordpressTests 
[Suslng 
using 
using 
using 
using 
using 
using 
CreatePostTests.cs* 
Init() 
Mic rosoft . Vis u a IStudio. TestTooIs . Lln itTesting; 
System; 
System. Collections . Generic ; 
System. Ling; 
System. Text; 
System. Threading. Tasks; 
Word pres sAutomation ; 
8 namespace WordpressTests 
11 
15 
18 
24 
100 8', 
Output 
Test the Pages functionality, 
[TestCIass 
public class 
Page Tests 
[Testlnitialize] 
public void Init() 
Driver. Initialize() ; 
[Testmethod] 
public void C) 
Login Page. GoTo() ; 
which 
similar 
to 
Post 
tests . 
- 
- 
- 
PageTests.cs 
page 
Ln8 
x 
Col 77 
Solution 'WordpressAutomation' (2 projects) 
WordpressAuto mation 
Properties 
• References 
DashboardPage.cs 
Driver.cs 
ListPostsPage.cs 
LoginPage.cs 
NewPostPage.cs 
packages.config 
PostPage.cs 
WordpressTests 
Properties 
• References 
Create PostTests.cs 
LoginTests.cs 
Pa geTests.cs 
Login Page . LoginAs ("admin " ) . With Password ("test123") . Login ; 
ListPostsPage . GoTo(PostType. Page) ; 
// general purpose 
Page"); 
// or you could 
"Wasn 't in edit 
Page Add New", UewPostPage.Header()' 
class this will work for both posts 
add an index or C I] for the 1st 
mode" ) ; 
"Header did not match"); 
and pages 
or 2nd post 
of 
that 
Page", 
NewpostPage.TitIe, "Title did not match"); 
Show output from: Tests 
I run 
Error List 
Run test started - 
- Run test finished: 
Output 
Ch 77 
Solution Explorer 
Properties 
INS 
Team Explorer Class View 
WordpressAutomation master 

Phase 4 - Building Out The Framework

During this phase the objective is to refactor the code to minimize duplication. So for example if there are methods in the test runner modules that is continually being duplicated, then they are prime candidates to refactor out of the test runner modules and into a base class that the test runner files inherit from. At first glance in all three of the test runner modules in the WordpressTests project there are [TestInitialize] and [TestCleanup] methods across the board. Therefore there needs to be more a base class where we can inherit these functionalities from and not have to repeat them for each new test suite. The base class that will be added to the WordpressTests project will be WordpressTest.cs and will contain the following:

using Microsoft.VisualStudio.TestTools.UnitTesting;

using WordpressAutomation;

namespace WordpressTests

{

    public class WordpressTest

    {

        /// <summary>

        /// WordpressTest.cs

        /// \*\*Tests\*\*

        /// abstracts away from each individual test module and provides [TestInitialize]

        /// and [TestCleanup] here to inherit from.

        /// </summary>

        [TestInitialize]

        public void Init()

        {

            Driver.Initialize();

            LoginPage.GoTo();

            LoginPage.LoginAs("admin").WithPassword("test123").Login();

        }

        [TestCleanup]

        public void CleanUp()

        {

            Driver.Close();

        }

    }

}

Notice that in the above block of code we have moved the [TestInitialize] and [TestCleanup] sections here. Make sure to remove these sections from the other the other test runner files in the WordpressTests project. One more thing worth noting is that I have also moved the redundant  login statements from the other test runner files to the [TestInitialize] section. These changes make a significant impact on reducing code duplication and it is good to get this done earlier rather than later when changes such as this will have a broader range of impact and be more time consuming to change. Make sure to retest all the test runner tests at this point to make sure they still work and that nothing else got broken from our changes.

Created with Microsoft OneNote 2016.