Hands-On Lab

Introduction to Web Performance and Load Testing with Visual Studio Ultimate 2012

Lab version: 11.0.60315.01 Update 2

Last updated: 4/9/2013

* + 1. 

Contents

[Overview 3](#_Toc352920859)

[Exercise 1: Creating Web Performance Tests 3](#_Toc352920860)

[Exercise 2: Load Testing 19](#_Toc352920861)

[Exercise 3: Executing & Analyzing Load Tests 33](#_Toc352920862)

Overview

* 1. In this lab, you will be introduced to the web performance and load testing capabilities provided in Visual Studio Ultimate 2012. You will walk through a scenario using a fictional online storefront where your goal is to model and analyze its performance with a number of simultaneous users. This will involve the definition of web performance tests that represent users browsing and ordering products, the definition of a load test based on the web performance tests, and finally the analysis of the load test results.
  2. **Note:** For the most part, this lab is intended to be an introduction to web performance and load testing in Visual Studio 2012 from the ground up. Improvements and other differences from 2010 are noted where appropriate.

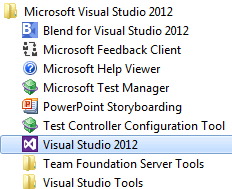
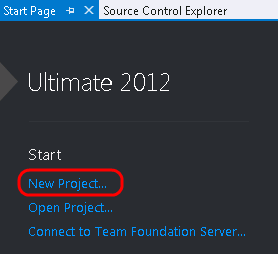
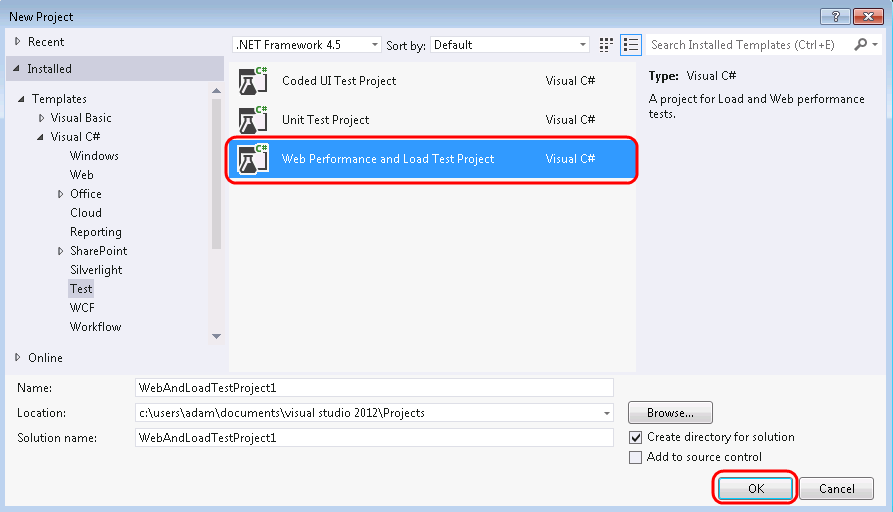
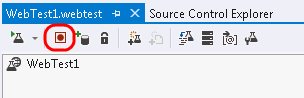
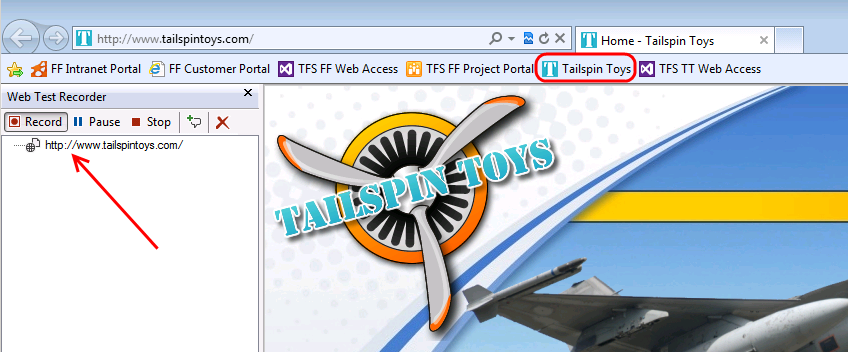
# Prerequisites

* 1. In order to complete this lab you will need the Visual Studio 2012 virtual machine provided by Microsoft. For more information on acquiring and using this virtual machine, please see [this blog post](http://aka.ms/VS11ALMVM). Since this lab makes extensive use of computer resources, it is suggested that you provide as much RAM and CPU as you can provide the virtual machine.

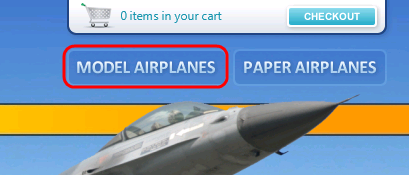
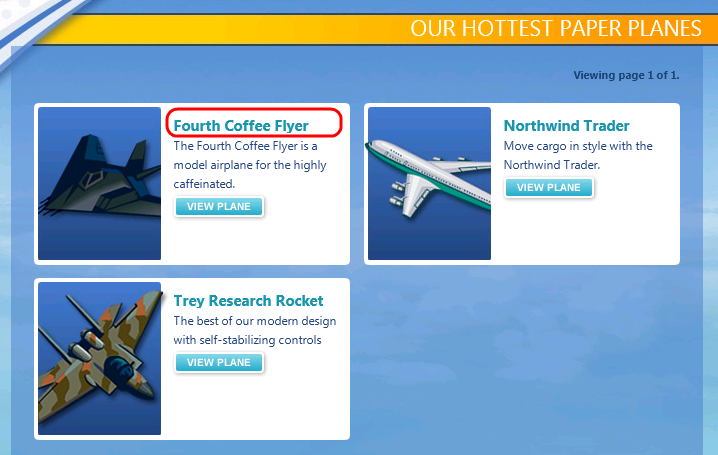
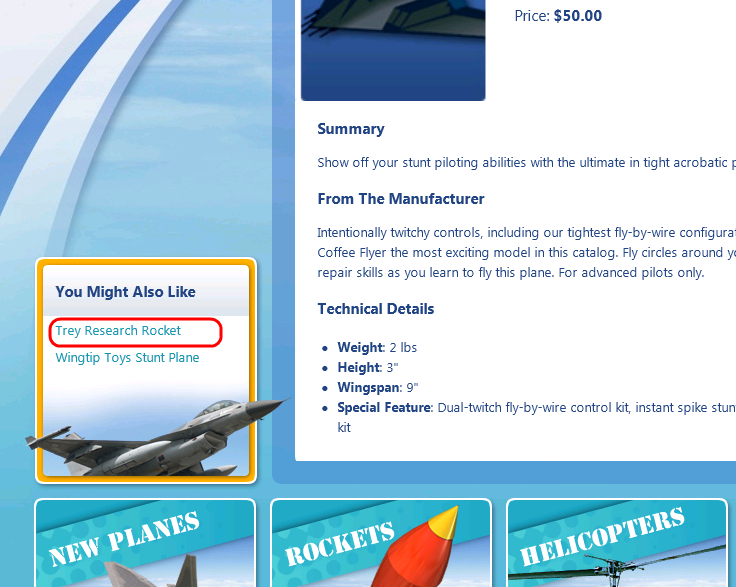
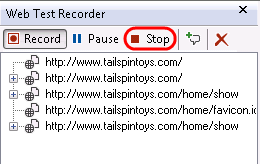
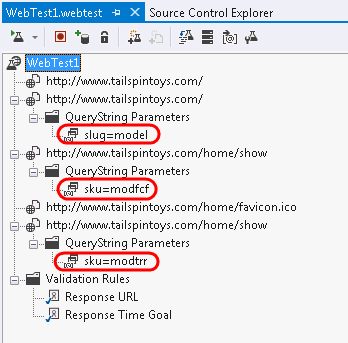
# Exercises

* 1. This hands-on lab includes the following exercises:
  2. Creating Web Performance Tests
  3. Load Testing
  4. Executing & Analyzing Load Tests
  5. Estimated time to complete this lab: **60 minutes**.

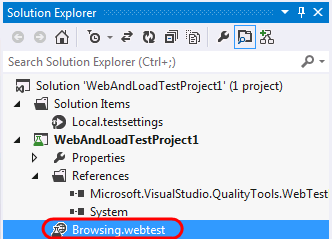
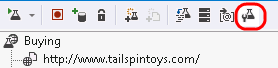
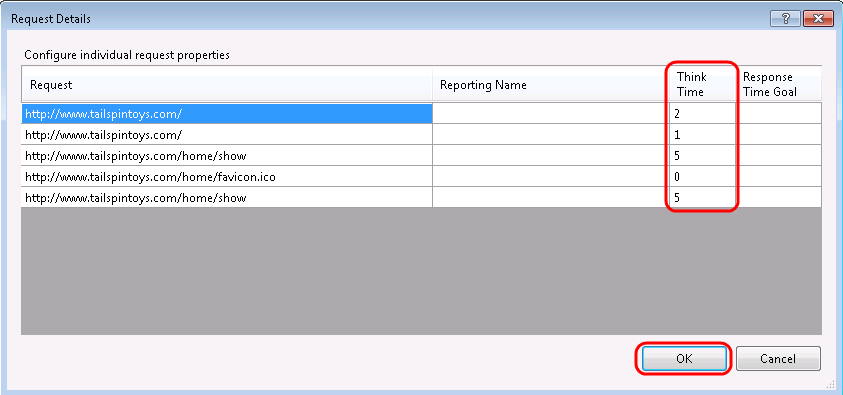
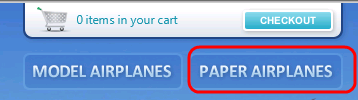
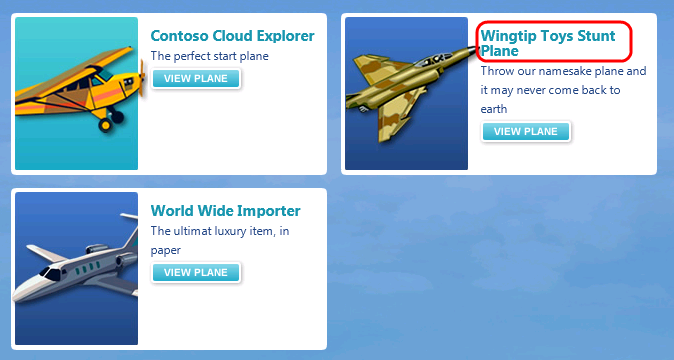
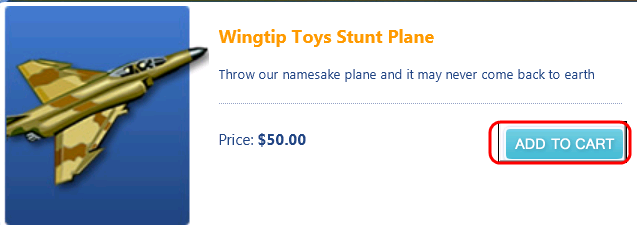
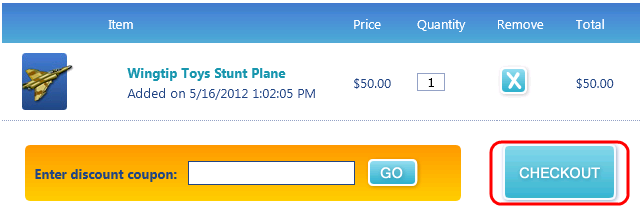
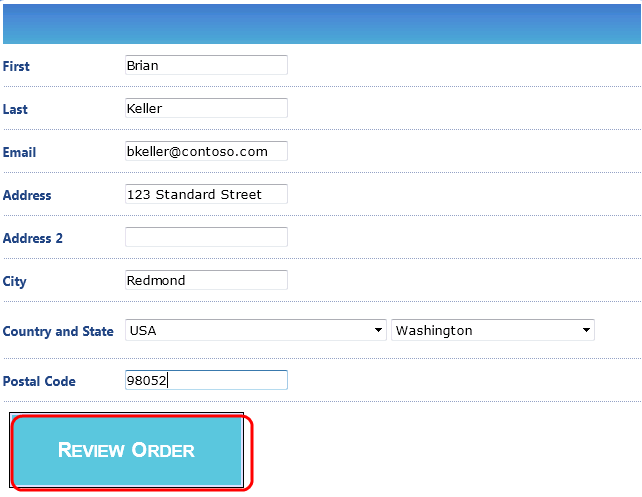
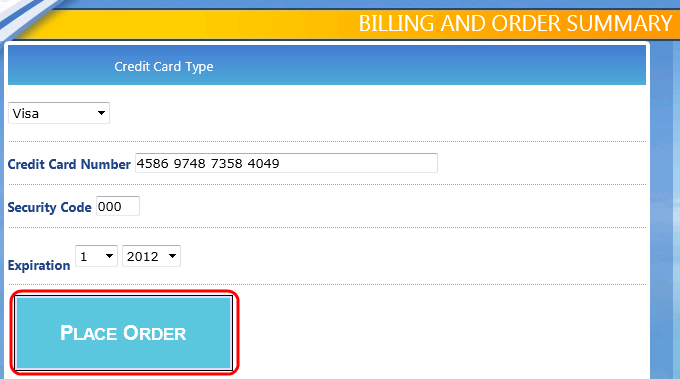
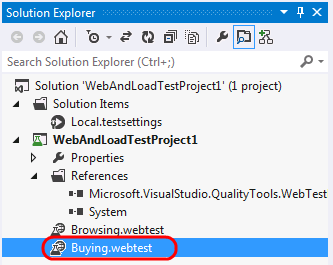
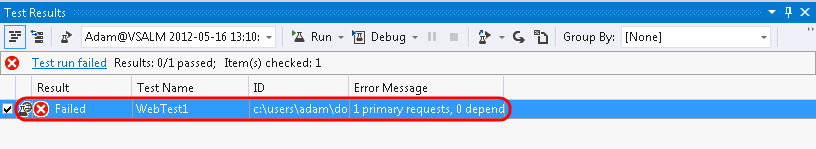
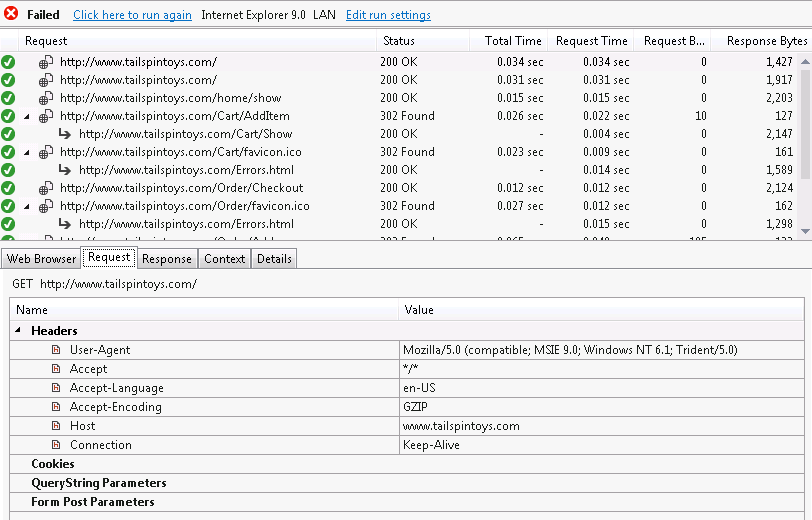
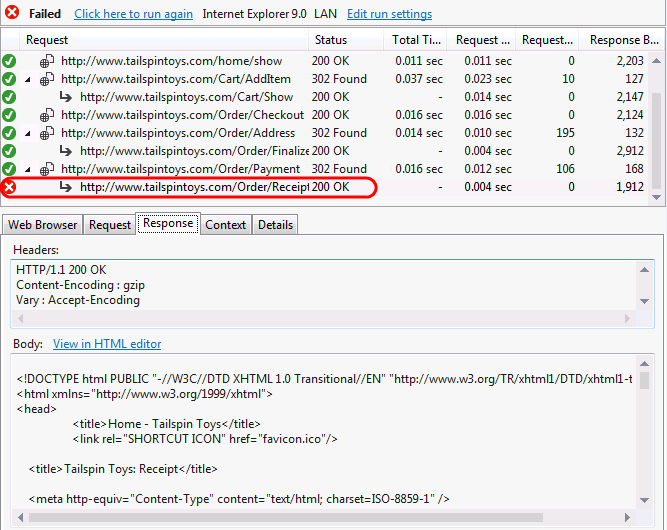
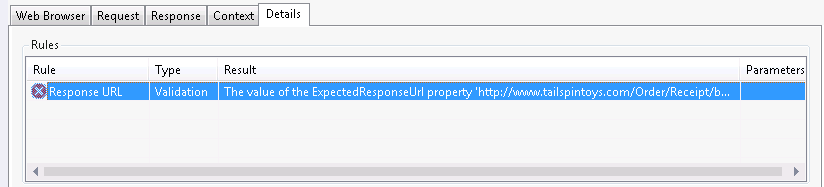
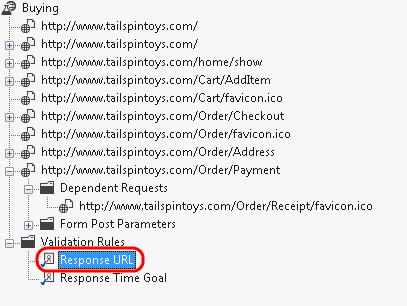
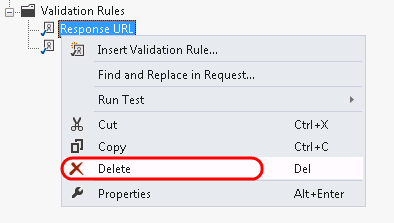
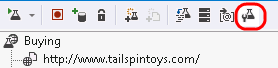
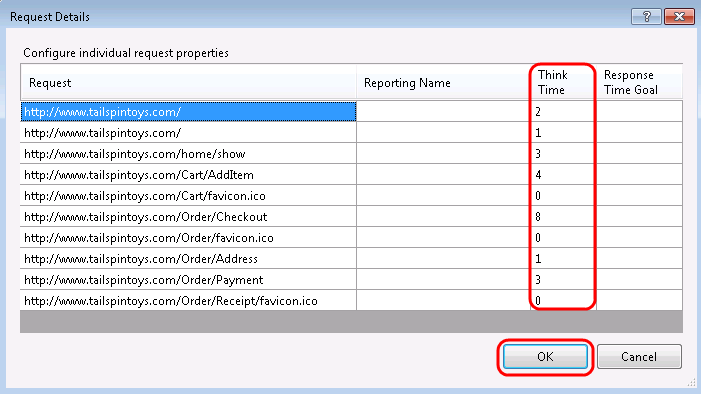
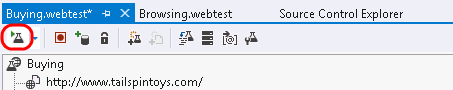
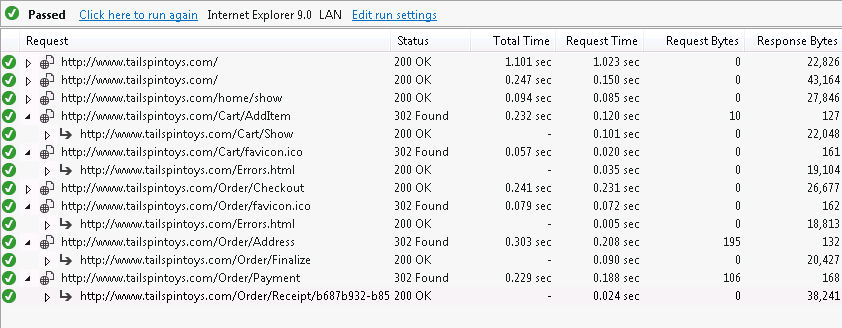
Exercise 1: Creating Web Performance Tests

* 1. In this exercise, you will setup and execute a basic Web performance test. This type of test generates HTTP requests and records expected responses while measuring response times and throughput.
  2. Log in as **Adam**. All user passwords are **P2ssw0rd**.
  3. Launch **Visual Studio 2012** from the shortcut on the taskbar or from **Start | All Programs | Microsoft Visual Studio 2012**.
     1. 
     2. Figure 1
     3. Starting Visual Studio 2012
  4. Click the **New Project…** link on the Start Page.
     1. 
     2. Figure 2
     3. Creating a new project
  5. In the **New Project** window, select the **Web Performance and Load Test Project** template from **Visual C# | Test** and select the **OK** button to create the project.
     1. 
     2. Figure 3
     3. Creating new load test project
     4. **Note:** Existing web performance and load tests that were created in Visual Studio 2010 will continue to run in Visual Studio 2012, but the version of any test controllers must match the version of Visual Studio. For more information about upgrading, please see <http://msdn.microsoft.com/en-us/library/hh704420>.
  6. The new web and load test project is created with default test settings and a single web test definition file named **WebTest1.webtest**. Select the **Add Recording** button at the top of the Web Performance Test Editor to start recording.
     1. 
     2. Figure 4
     3. Starting the test recorder
  7. Internet Explorer and the Web Test Recorder should now be open and in **Record** mode, which you can verify by looking at the toolbar at the top of the recorder window. You can pause, stop, or reset the recording at any time.
     1. 
     2. Figure 5
     3. Web Test Recorder running in Internet Explorer
     4. **Note**: While the Web Test Recorder is in the Record state, requests will be recorded complete with “**think time**”. Keep this in mind as you record scenarios as you think the average user would – pause on product pages, wait a few seconds to make sure your credit card is correct, and so on. We are not just creating this web test to ensure that requests and responses are correct, we will use it to do some load testing later on.
  8. In the next few steps, we will play the part of a customer browsing the site by clicking on a few products and drilling into product details. The goal here is to create a profile that represents the typical browsing scenario for the site for a single user.
  9. In **Internet Explorer**, click on the **Tailspin Toys** button from the favorites bar to launch the Tailspin Toys website. Note that the request was logged in the test recorder.
     1. 
     2. Figure 6

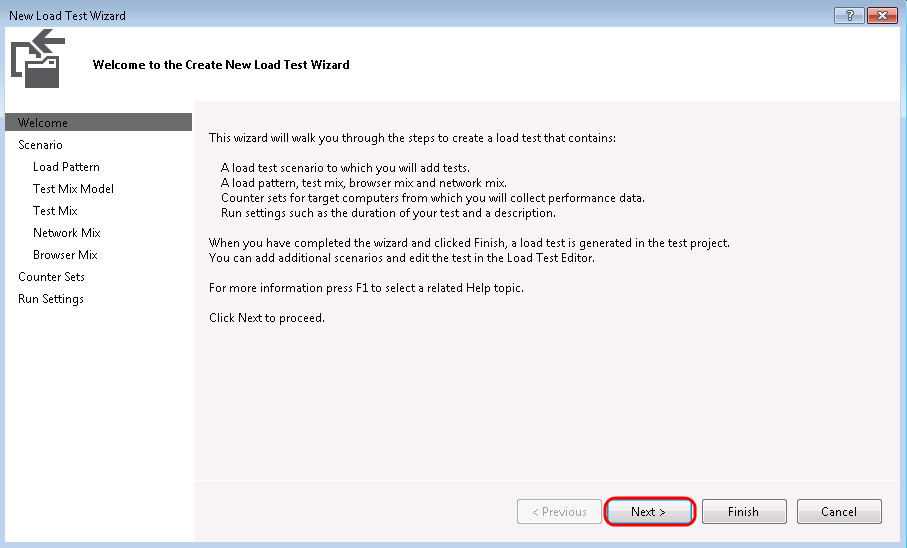
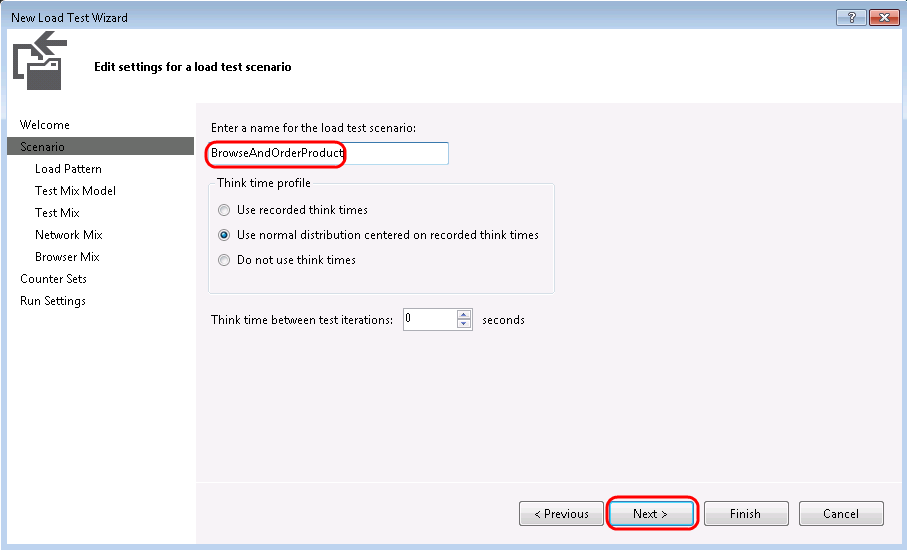
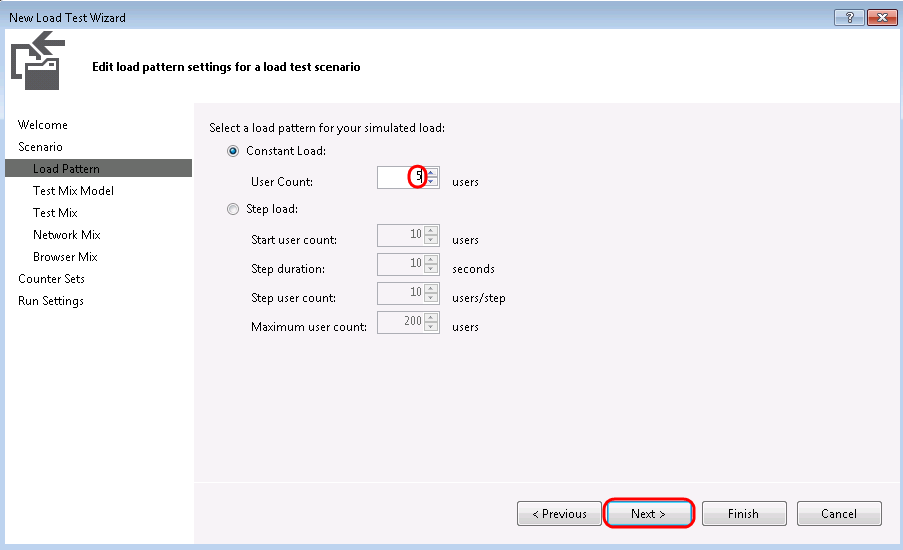
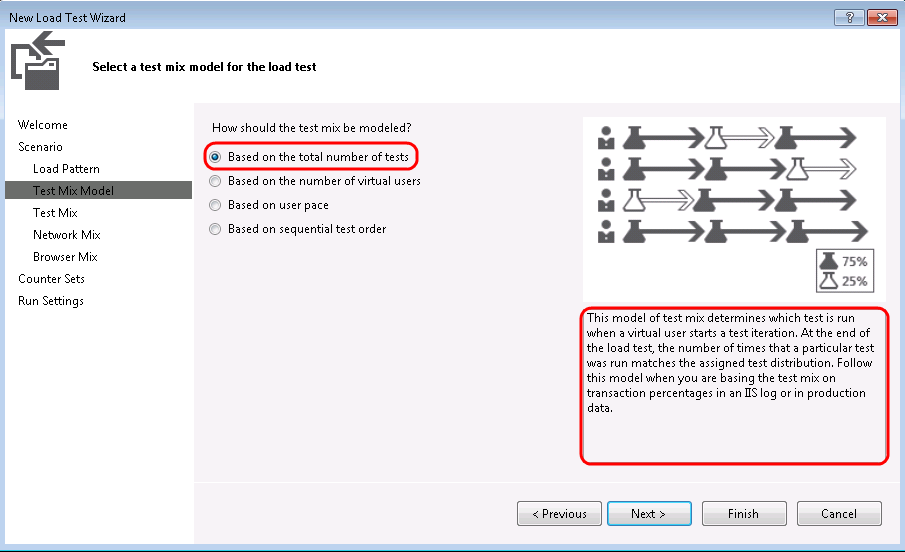
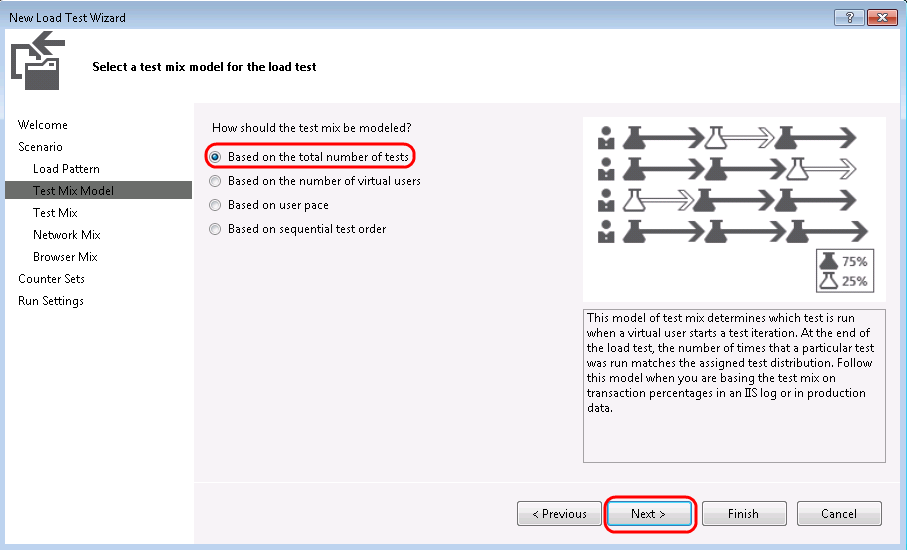
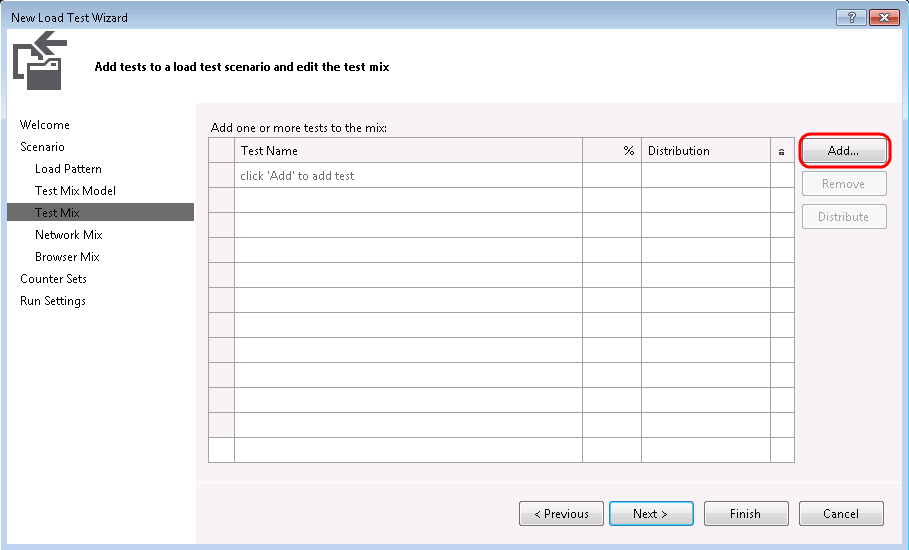
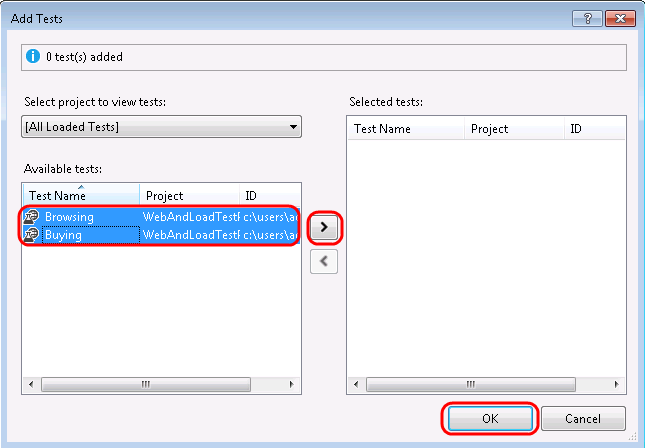
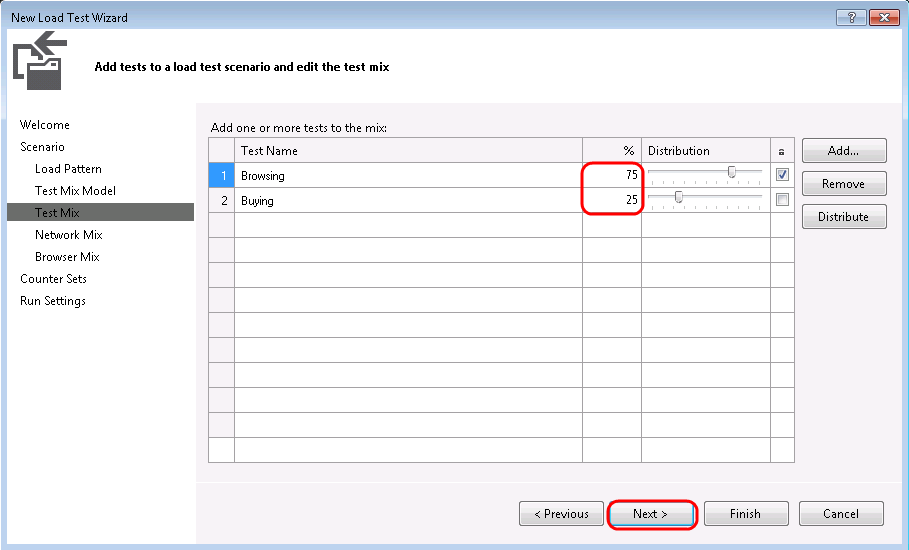
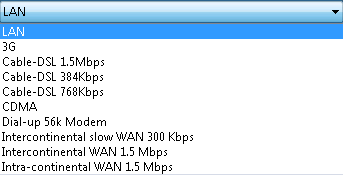
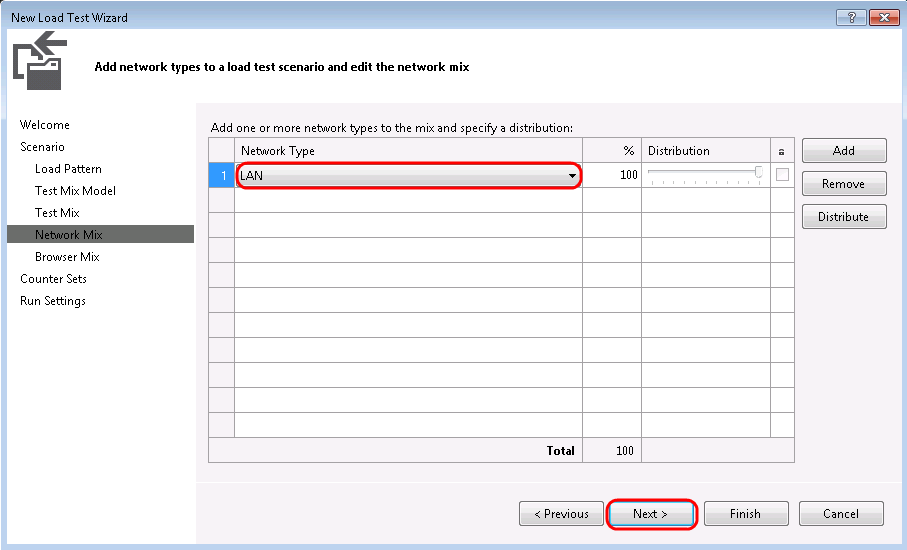
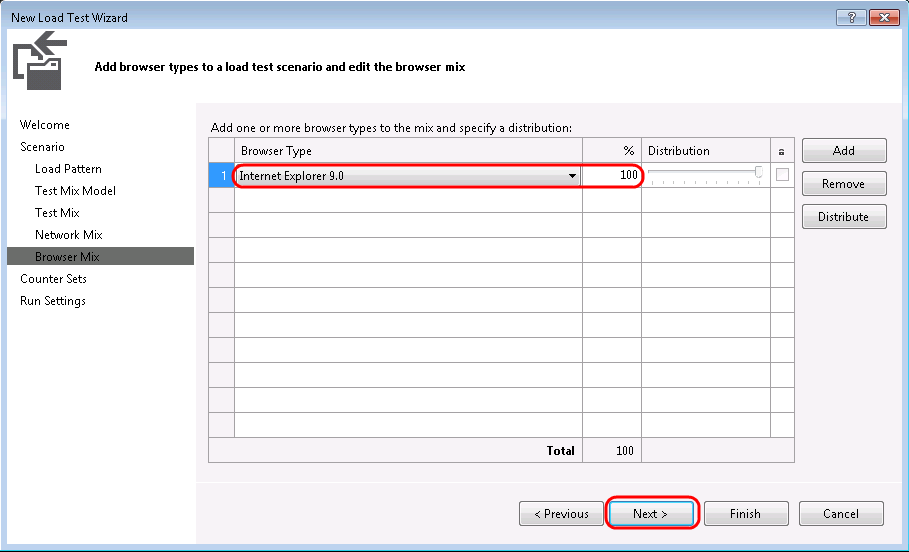
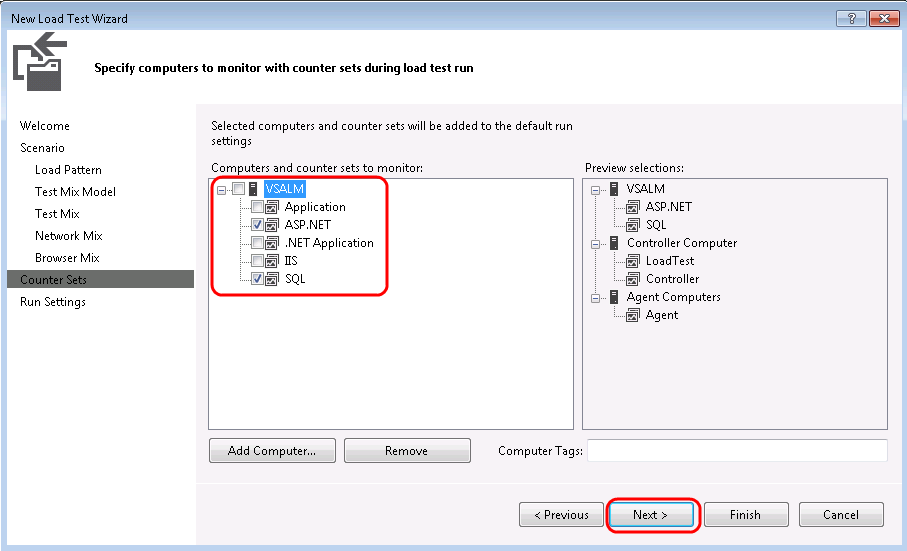
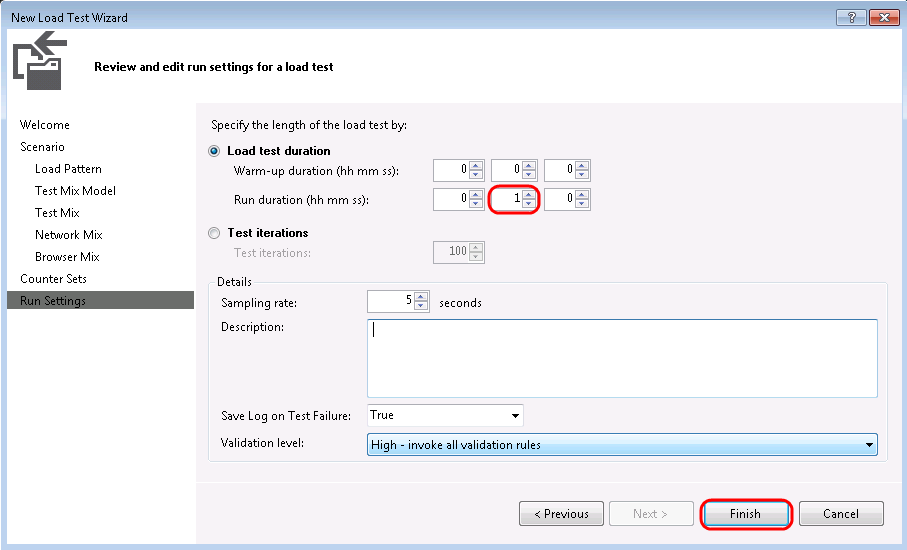
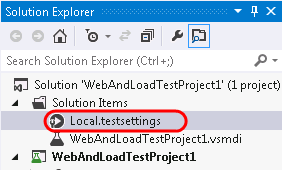
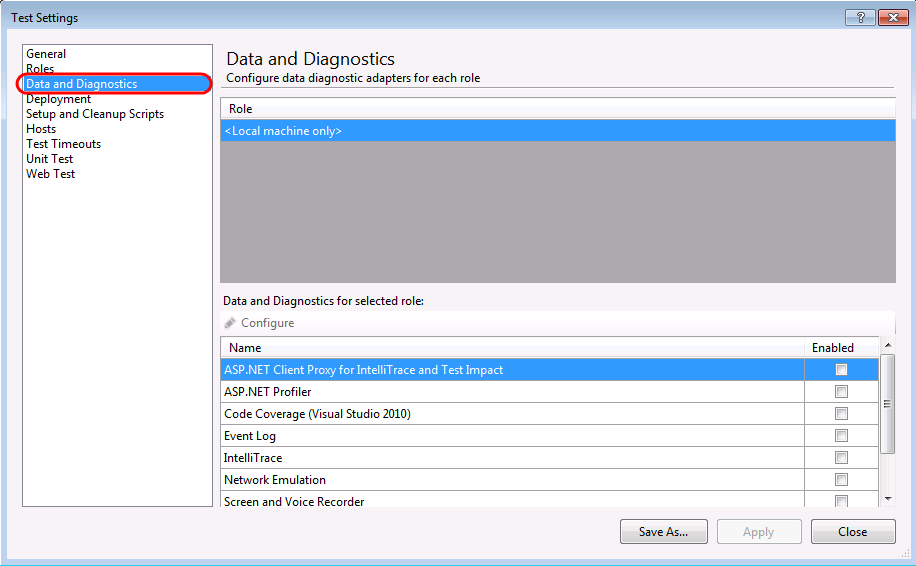
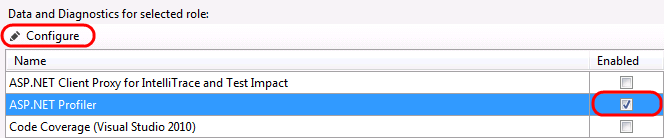
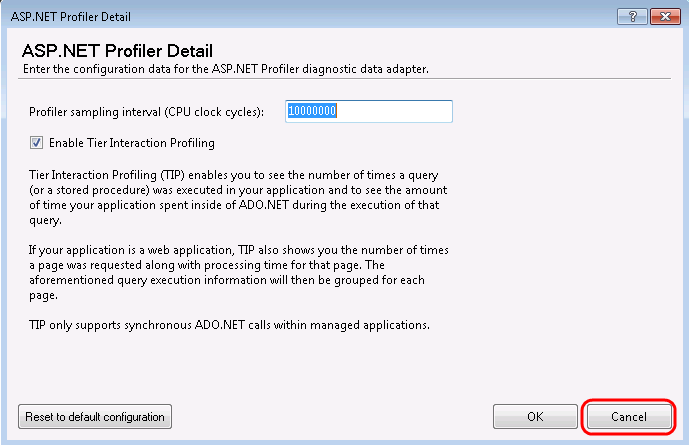
Initial request to Tailspin Toys website

* 1. Select on the **Model Airplanes** button.
     1. 
     2. Figure 7
     3. Selecting Model Airplanes
  2. Select the **Fourth Coffee Flyer** link.
     1. 
     2. Figure 8
     3. Selecting the Fourth Coffee Flyer
  3. Select the **Trey Research Rocket** from the “You Might Also Like” section.
     1. 
     2. Figure 9
     3. Selecting the Try Research Rocket
  4. Select the **Stop** button from the Web Test Recorder toolbar.
     1. 
     2. Figure 10
     3. Stopping the Web Test Recorder
  5. Once the Web Test Recorder stops recording and returns to Visual Studio, two tasks will commence:
     + **Dynamic parameters** that could not be identified and extracted during the recording will be identified. Examples of dynamic parameters include cookie values, ASP.NET view state, query string parameters, and post parameters. Extracting dynamic parameters when they are first used will allow the test to be run at any time in the future and therefore we will be able to utilize them as part of a load test.
     + The recorded web performance test will run and display results.
  6. Take a moment to view the recorded web requests and see that query string parameters were automatically extracted. Visual Studio will also look for and extract hidden HTML fields and form post parameters.
     1. 
     2. Figure 11
     3. Web test recording showing extracted query string parameters
     4. **Note:** There are two types of rules that can be applied to the response to each HTTP request, validation and extraction. **Validation rules** can be used to verify that pages have specific text, tags, attributes, or form fields on them. **Extraction rules** can be used to store results in the test context as name value pairs – and these rules can extract form fields, text, attributes, headers, regular expressions, and hidden fields.
  7. Although it is out of the scope of this lab to dig into other Web performance test features, it is worth pointing a few of them out.

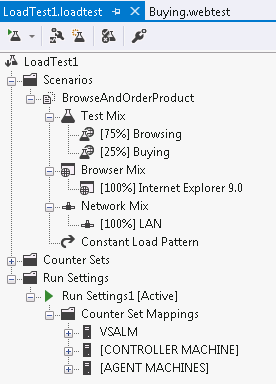
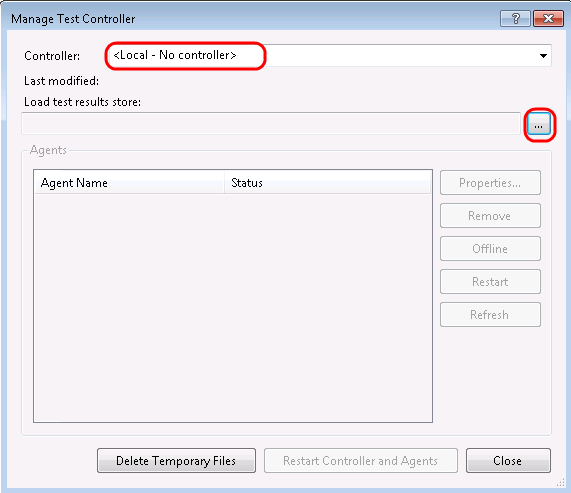
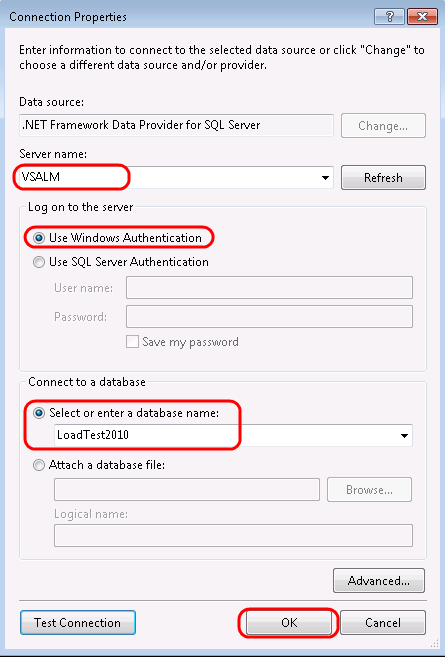
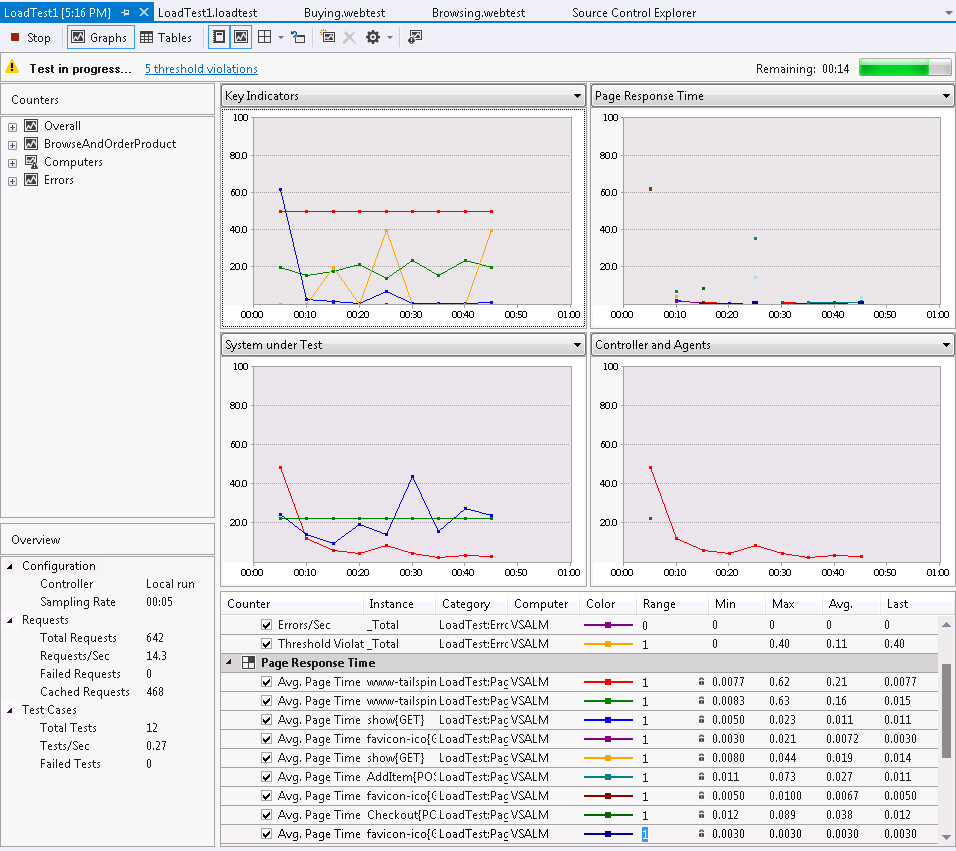
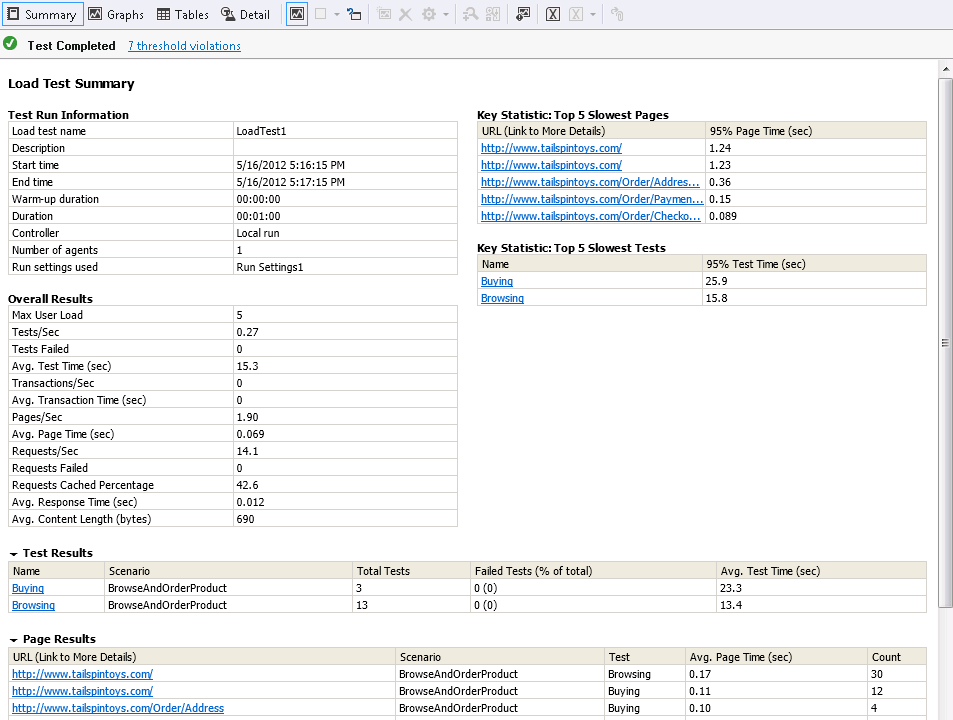
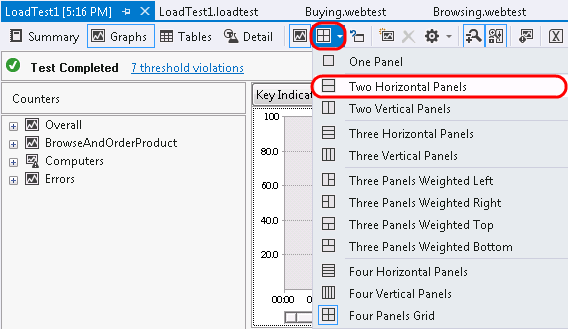
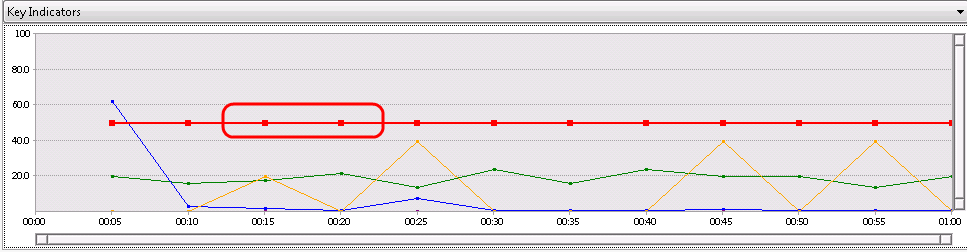
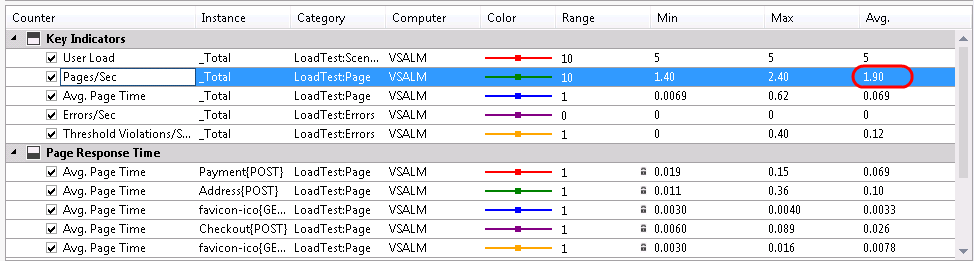
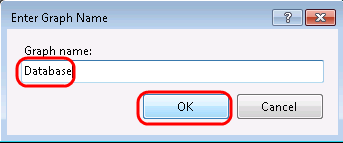
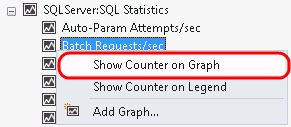
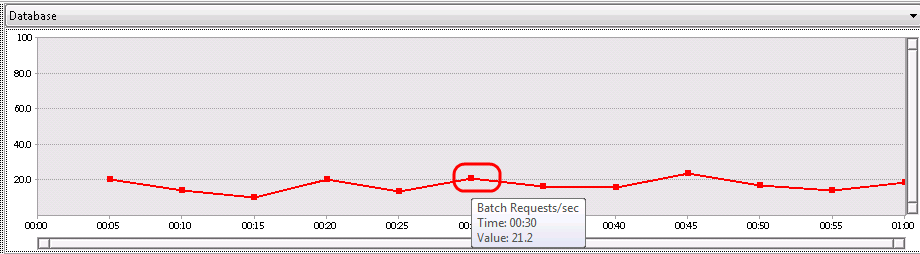
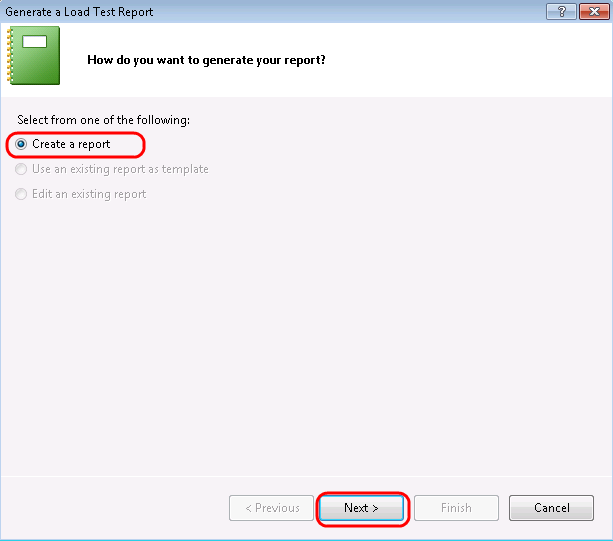
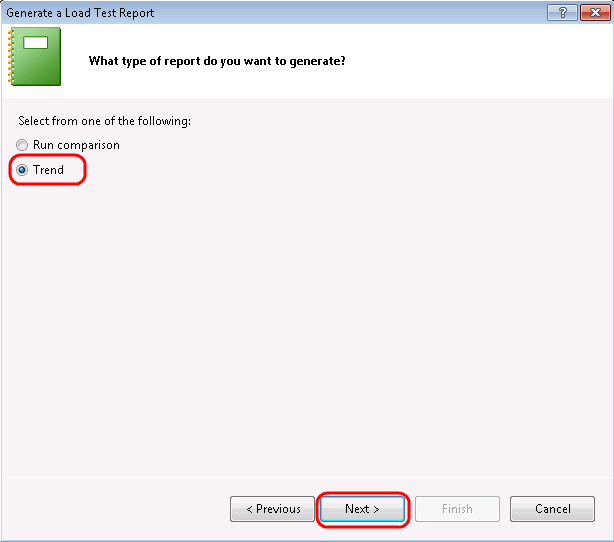
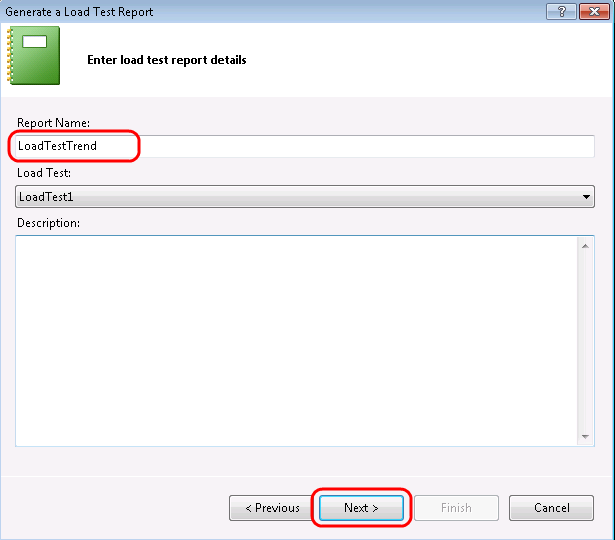
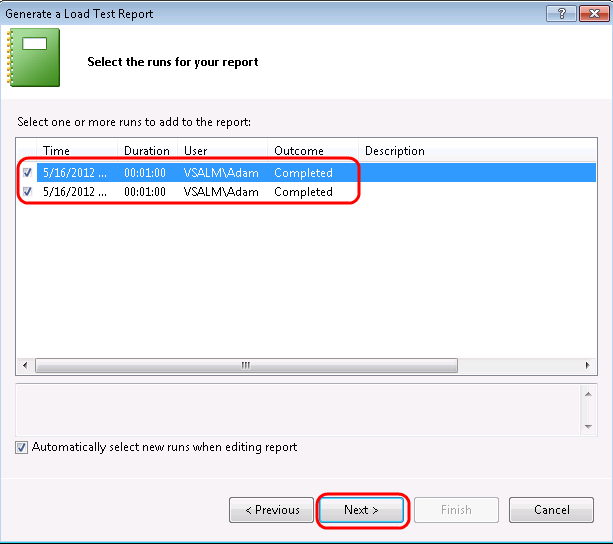
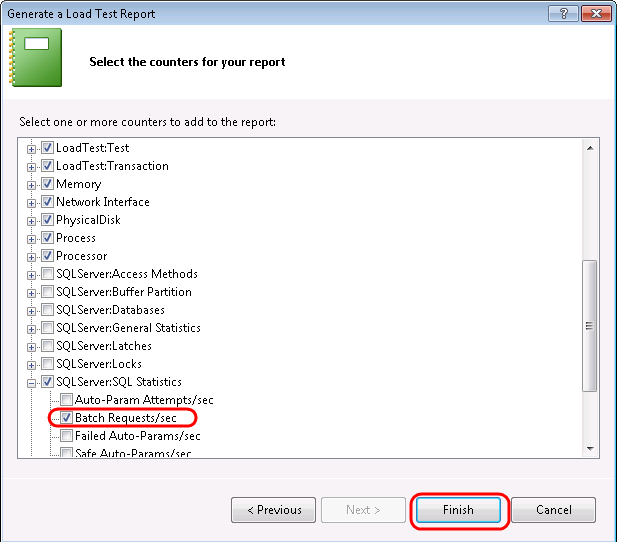
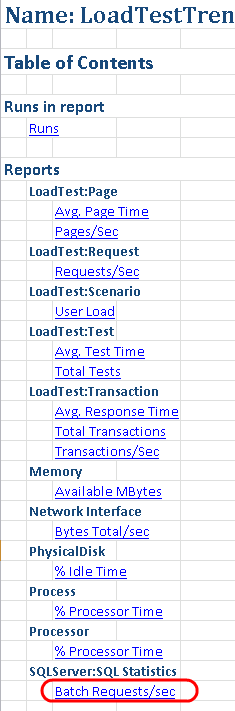
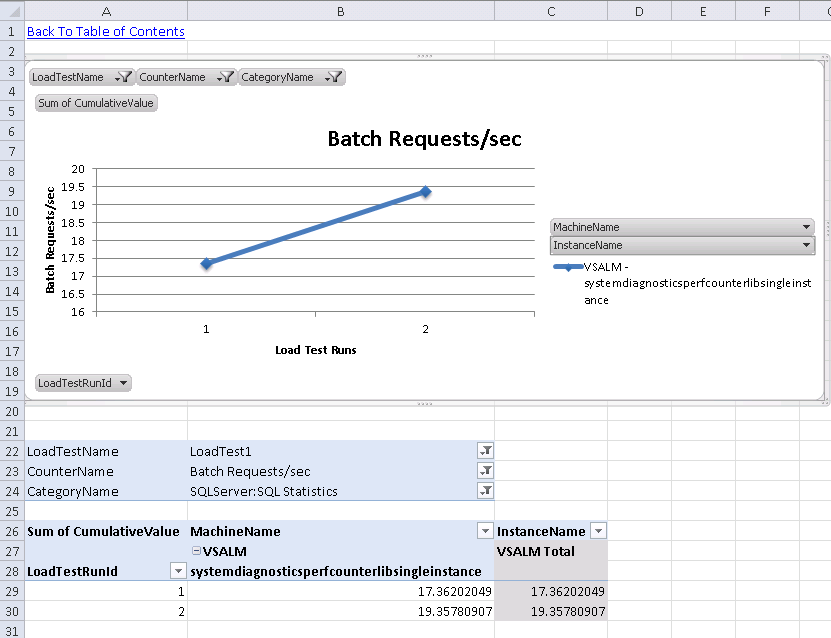
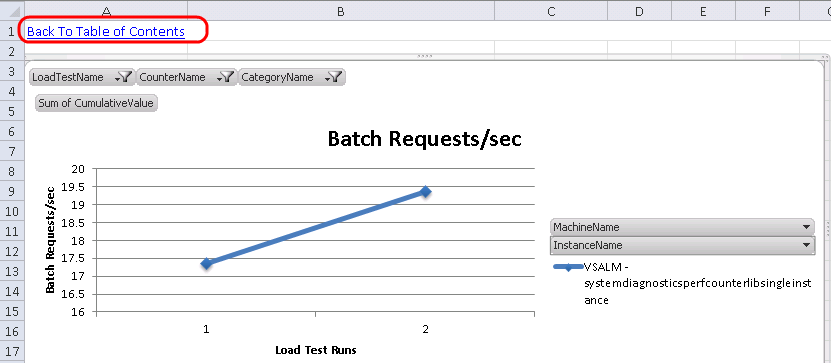
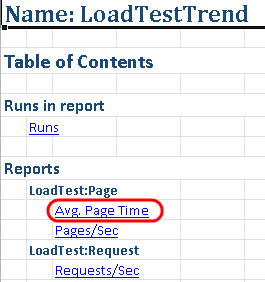
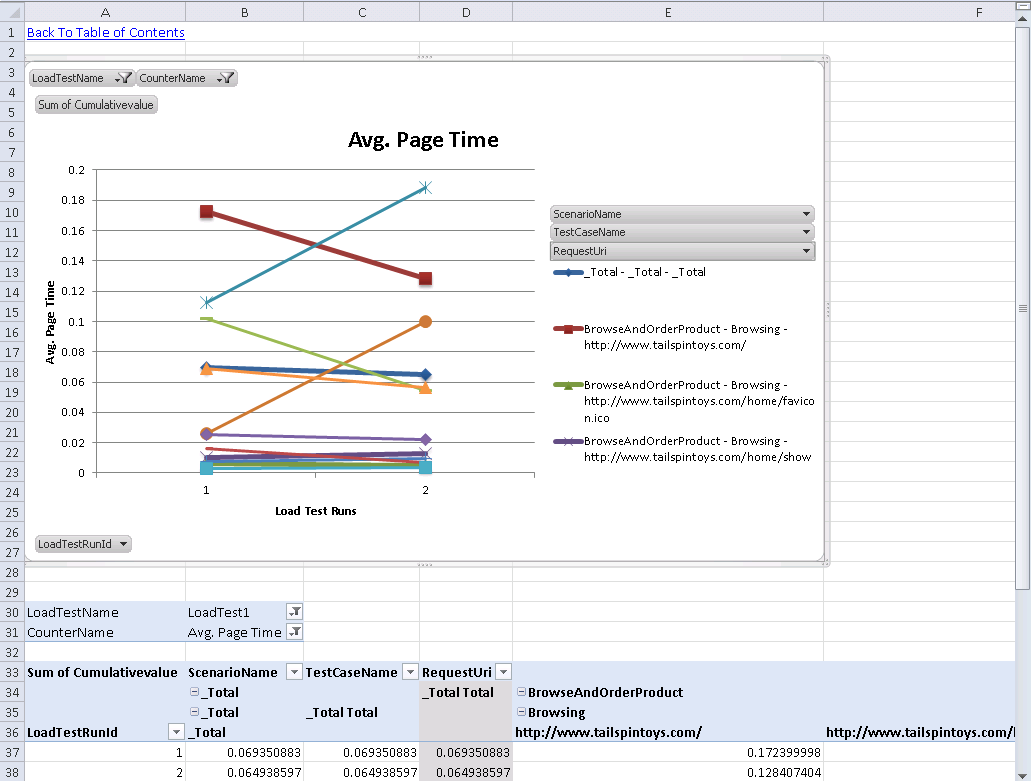
|  |
| --- |
| Other Web Performance Test Features |
| * + Convert a recorded Web performance test into a coded Web performance test   + Add reporting names to clarify identification of Web requests   + Customize test with artificial think times   + Configure the permitted response time for a Web page   + Add a data source to bind to HTTP requests (database, XML, CSV)   + Customize execution by using loops, branching, and transactions   + Configure credentials for test (basic or Integrated Windows) |

* 1. In Solution Explorer, **right-click** on **WebTest1.webtest** and select **Rename** to change the name to “Browsing.webtest”. This will make it easier to keep track of the scenario that we recorded.
     1. 
     2. Figure 12
     3. Updated web test name
  2. Let’s take a look at the recorded think times to make sure they are appropriate. Select the **Set Request Details** button from the toolbar to open the Request Details window.
     1. 
     2. Figure 13
     3. Location of Set Request Details button
  3. The **Request Details** window shows a grid containing all requests along with a reporting name, think time (in seconds), and response time goals (in seconds). For the purpose of this lab, make sure that the sum of all think times are no more than about 15 seconds. This will help ensure that we can get good results when we create a load test later on. Select the **OK** button to continue if you made any changes.
     1. 
     2. Figure 14
     3. Request Details window
  4. Now let’s add in another web test to represent a customer that browses and purchases a product from the Tailspin Toys website. Select **Project | Add Web Performance Test** from the main menu in Visual Studio. We could have created a single web test that included both the browsing and buying scenario, but composing the tests in this fashion will make constructing a realistic load test easier later on.
  5. Navigate to **Tailspin Toys** and select the **Paper Airplanes** button.
     1. 
     2. Figure 15
     3. Selecting Paper Airplanes
  6. Select the **Wingtip Toys Stunt Plane** link.
     1. 
     2. Figure 16
     3. Selecting Wingtip Toys Stunt Plane
  7. Select the **Add To Cart** button.
     1. 
     2. Figure 17
     3. Adding product to cart
  8. Select the **Checkout** button.
     1. 
     2. Figure 18
     3. Check out
  9. Fill out the order form with some test data and then select the **Review Order** button.
     1. 
     2. Figure 19
     3. Entering address information
  10. Select the **Place Order** button.
      1. 
      2. Figure 20
      3. Placing the order
  11. After placing the order, you should see the receipt page. Select the **Stop** button in the Web Test Recorder to return to Visual Studio.
  12. In Solution Explorer, **right-click** on **WebTest1.webtest** and select **Rename** to change the name to “Buying.webtest”.
      1. 
      2. Figure 21
      3. Updated web test name
  13. Load the **Test Results** window and note that the test run that was automatically kicked off ended up failing. **Double-click** on the test run to view the test result details.
      1. 
      2. Figure 22
      3. Viewing test result details
  14. The **Test Result Details** window shows the sequential list of HTTP requests, responses, some Visual Studio test context, and other details. Take a few minutes to familiarize yourself with the information available here.
      1. 
      2. Figure 23
      3. Test Result Details window showing test run results
  15. Scroll down to the request that shows where the web performance test failed and select it. Note that the returned status code is a 200 and the Request and Response tabs look fine.
      1. 
      2. Figure 24
      3. Response to order placement shows failure even though we have HTTP 200
  16. If you click on the **Details** tab, you will see that the test failed because a **Response URL Validation** rule was expecting to see the same response URL that was recorded during the Web Test definition (and that included an order number which is unique).
      1. 
      2. Figure 25
      3. Details tab showing reason for failure
  17. Close the test result details window to return to the **Buying** web test in the editor.
  18. Now that we understand why the web test failed, we can make a modification so that it will succeed and still give us a good test representing a user browsing for and ordering a product. Scroll down to the **Validation Rules** node and location the **Response URL** rule.
      1. 
      2. Figure 26
      3. Location of Response URL validation rule
  19. **Right-click** on the **Response URL** and select the **Delete** option.
      1. 
      2. Figure 27
      3. Deleting the Response URL validation rule
  20. Before we verify that removing the Response URL validation rule fixed the problem, let’s take a look at the recorded think times to make sure they are appropriate. Select the **Set Request Details** button from the toolbar to open the Request Details window.
      1. 
      2. Figure 28
      3. Location of Set Request Details button
  21. For the purpose of this lab, make sure that the sum of all think times are no more than about 30 seconds. This will help ensure that we can get good results when we create a load test later on. Select the **OK** button to continue if you made any changes.
      1. 
      2. Figure 29
      3. Request Details window
  22. **Run** the **Buying** web test again to make sure that it passes.
      1. 
      2. Figure 30
      3. Location of Run Test button
      4. **Note:** By default, web performance tests will run without taking into consideration think times, although it can be configured in the test settings if desired.
  23. The web test results should now show green checkmarks next to each step.
      1. 
      2. Figure 31
      3. All web test requests passed
  24. Close the test results window.

Exercise 2: Load Testing

* 1. In this exercise, you will use the web performance tests that you created in the previous exercise as the basis of a load test. Building one or more web performance tests that accurately reflect a user scenario is critical to the foundation of a useful load test. To create a load test we will define user load, specify the web performance tests to use, the type of network and browser simulation to use, and the performance counters and other metrics that we want to collect for the duration of the test.
  2. Select **Project | Add Load Test** from the main menu in Visual Studio.
  3. In the **New Load Test Wizard**, select the **Next** button to start defining the load test scenario.
     1. 
     2. Figure 32
     3. New Load Test Wizard walks you through common setup options
  4. Enter a name for the scenario like “**BrowseAndOrderProduct**” but leave the default think time profile in place. The default uses the think times of the web performance tests as a median value, with a normal distribution used to generate some variation. The goal is a more realistic generation of load on the web site.
  5. Click the **Next** button to continue on to the Load Pattern definition screen.
     1. 
     2. Figure 33
     3. Starting to add scenario definition for load test
  6. Use the Constant load option (the default) for this load test, but change the **User Count** to **5** users since we are operating within a virtual machine. It is important to keep the simulated user count low enough such that the machine has enough resources to properly run IIS and the load test on the same machine. Depending upon the web site under test, using a step load to ramp up usage of the web site may be more realistic, but it also requires longer test runs.
     1. **Note:** The limit on the number of virtual users that you can use in your load tests has been removed in Visual Studio Ultimate 2012. You no longer have to purchase virtual user licenses to increase the number of virtual users that you can use in your load tests. The Visual Studio Ultimate 2012 Trial license, however, does limit you to 25 virtual users and only allows local tests.
  7. Select the **Next** button to continue on to the Test Mix Model definition screen.
     1. 
     2. Figure 34
     3. Setting up a constant virtual user load to run for duration of load test
  8. Read the description of each test mix model by clicking on it and viewing the description that appears on the right-hand side.
     1. 
     2. Figure 35
     3. Defining test mix model
  9. Let’s say that our current production site give us some indication of the percentage of browsing users that end up making purchases. Select the first option that models the test mix **based on the total number of test**s and then select the **Next** button to continue on to the Test Mix screen.
     1. 
     2. Figure 36
     3. Defining test mix model
  10. Select the **Add** button to load the Add Test window.
      1. 
      2. Figure 37
      3. Adding tests to the mix
  11. Select **both tests**, add them to the test mix, and then select **OK**.
      1. 
      2. Figure 38
      3. Adding tests to the mix
      4. **Note:** Load tests can include a mix of coded UI tests, web performance tests, and even other test types such as unit tests. It is important to note that for coded UI tests, you need one virtual or physical machine per user that you are simulating since it assumes that it has control over the entire user interface.
  12. Let’s say that our production logs tell us that 25% of users browsing the site end up buying something. Change the **Distribution** to reflect this knowledge and then select the **Next** button to continue on to the Network Mix screen.
      1. 
      2. Figure 39
      3. Defining test mix
  13. The **Network Mix** screen allows you to choose one or more network types and specify the distribution of those types across the tests to be executed by the virtual users. Select the **Network Type** dropdown to see the available options.
      1. 
      2. Figure 40
      3. Network Type options
  14. For the purpose of this lab, leave the default Network Type of **LAN** in place and select the **Next** button to continue on to the Browser Mix screen.
      1. 
      2. Figure 41
      3. Defining network mix for load test
      4. **Note:** Network emulation will not work when operating within this virtual machine environment because the URL under test loops back to localhost.
  15. The **Browser Mix** screen allows you to specify one or more browser types and specify the distribution of those types across the tests to be executed by the virtual users. Just like the network mix, this allows us to more realistically model how the users interact with the web site. For the purpose of this exercise, leave the default at 100% Internet Explorer 9.0 and select the **Next** button to continue on to the Counter Sets screen.
      1. 
      2. Figure 42
      3. Defining browser mix for load test
  16. The **Counter Sets** screen allows you to specify the computers and counter sets from which to gather performance counters during the load test. Select the **Add Computer** button and type ‘**VSALM’** for the computer name.
  17. Select the **ASP.NET** and **SQL** counter sets to monitor since we are load testing a website. Note that **Controller Computer** and **Agent Computers** collect some data by default, and that both of these represent the same machine in this case. Once the counter sets have been set, select the **Next** button to continue on to the Run Settings screen.
      1. 
      2. Figure 43
      3. Defining performance counter data to collect
      4. **Note:** It is possible to modify or add counter sets to be used during load tests by working directly with the *.CounterSet* XML files located in the **\Common7\IDE\Templates\LoadTest\CounterSets** directory. The **LoadTest** directory also contains network and browser definitions as well.
  18. The **Run Settings** for a load test allow you to specify how long the test should run using either time duration or a specific number of test iterations. We will use a **time duration**, but change it from **10** minutes to **1** minute for demonstration purposes. The default sampling rate of 5 seconds is fine here, and it is a good choice in general for shorter test runs. If you want to run longer tests, consider sampling less often as it will generate a lot less data to store in the load test database.
  19. Select the **Finish** button to save the load test configuration.
      1. 
      2. Figure 44
      3. Defining load test run settings
  20. It is also possible to configure data diagnostic adapters for each role that is part of the load test. In Solution Explorer, **double-click** on the **Local.testsettings** file to load the Test Settings window.
      1. 
      2. Figure 45
      3. Loading test settings
  21. In the **Test Settings** window, select the **Data and Diagnostics** option to see the available adapters. Options include an ASP.NET Profiler, Event Log, IntelliTrace, Network Emulation, and more. No adapters are selected by default, in part because many of them have a significant impact on the machines under test and can generate a large amount of data to store over the course of long load tests.
      1. 
      2. Figure 46
      3. Data diagnostic adapters
  22. Select the **Enabled** checkbox for the **ASP.NET Profiler** adapter and then select the **Configure** button.
      1. 
      2. Figure 47
      3. Configuring ASP.NET Profiler adapter
  23. The ASP.NET profiler collects performance statistics, .NET memory allocations and other data, and tier interaction data that highlights performance data about synchronous ADO.NET calls made to Microsoft SQL Server. Although out of scope for this lab, keep in mind that this powerful feature is available to help diagnose performance issues. Select the **Cancel** button.
      1. 
      2. Figure 48
      3. ASP.NET Profiler defaults
      4. **Note:** To learn more about the ASP.NET Profiler, see the documentation at:
      5. <http://msdn.microsoft.com/en-us/library/dd504817.aspx>
  24. **Close** the **Test Settings** window without saving changes to the settings.

Exercise 3: Executing & Analyzing Load Tests

* 1. In this exercise, you will execute the load test that you defined in the previous exercise and analyze the results.
  2. Open the load test that you defined in the previous exercise by **double-clicking** on it in **Solution Explorer** if necessary.
     1. 
     2. Figure 49
     3. Load test definition
  3. Select the **Mange Test Controllers** button in the Load Test Editor toolbar.
     1. 
     2. Figure 50
     3. Manage Test Controllers button
  4. Note that the selected **Controller** is set to **<Local – No controller>**. Select the **ellipses** button to setup the connection string to the load test results store.
     1. 
     2. Figure 51
     3. Manage Test Controller window
  5. In the **Connection Properties** window use a Server Name of **VSALM**, use **Windows Authentication**, and use the LoadTest2010 database name. Once you have the database connection properties in place, select the **OK** button to save.
     1. 
     2. Figure 52
     3. Setting connection properties to load test database for test controller
  6. Select the **Close** button to exit out of the Manage Test Controller window.
  7. Start the load test by select the **Run Test** button from the toolbar.
     1. 
     2. Figure 53
     3. Location of Run Test button
  8. Once the load test initializes and starts its 1-minute run, the load test results window will load with the **Graphs** view visible. By default, you should see four panels showing some key statistics, with some key performance counters listed below that. Data is sampled every 5 seconds by default, but that can be changed in the load test settings.
     1. 
     2. Figure 54
     3. Load test results window showing KPIs during load test run
     4. **Note:** Screenshots showing statistics and graphs may vary widely from those that you see during your walkthrough of this lab. This is due primarily to the different hardware that you are running this virtual machine on. In addition, you may see some threshold violations that result from the VM being busy during test. In a real world situation, and especially one in which you want to drive more virtual users, you would probably best be served by using multiple machines during test, not only to generate the load but also for each component of your system as it will be deployed in production.
  9. After the load test run finishes, it will automatically switch to the Summary view. The **Summary** view shows overall aggregate values and other key information about the test. Note that the hyperlinks to specific pages open up even more details in the **Tables** view.
     1. 
     2. Figure 55
     3. Summary view for load test results
  10. Switch to the Graphs view by clicking on the **Graphs** button in the toolbar.
      1. 
      2. Figure 56
      3. Graphs button
  11. Note that you can manipulate the graphs that you view. Select the **panels** drop down control in the toolbar as shown below and select the **Two Horizontal Panels** option.
      1. 
      2. Figure 57
      3. Customizing the graph view
  12. By default the top graph will show **Key Indicators** and the bottom graph will show **Page Response Time**, two very important sets of data to a web application.
      1. 
      2. Figure 58
      3. Graphs showing Key Indicators and Page Response Time
  13. Click on one of the **Key Indicator** graph lines or data points so select it. This will also highlight the counter that it is associated with below the graphs. The screenshot below shows that the red line represents the **User Load** at different points during the load test. It is always equal to **5** as we configured it to be.
      1. 
      2. Figure 59
      3. User Load data selected
  14. Click on the **Pages/Sec** row from the **Key Indicators** section of the counter grid to highlight it in the graph. In the screenshot shown below, we can see that the average number of pages per second over the duration of the test was 1.90.
      1. 
      2. Figure 60
      3. Viewing counter data
  15. It is also possible to create your own graphs using any of the collected performance counters. Select the **Add New Graph** button from the toolbar. 
      1. Figure 61
      2. Add New Graph button
  16. Enter ‘**Database**’ for the graph name and select the **OK** button to create the new blank graph.
      1. 
      2. Figure 62
      3. Creating new graph
  17. Let’s say that we are interested in seeing how the database performed during the load test. In the **Counters** pane on the left-hand side of the Graphs view, expand the Computers, VSALM, and **SQLServer:SQL Statistics** nodes. **Right-click** on ‘**Batch Requests/sec’** and select **Show Counter on Graph**.
      1. 
      2. Figure 63
      3. Adding select counters to the custom graph
      4. **Note:** If the connection to the load test results database times out, you may need to modify the connection by going to **Load Test | Manage Test Controllers**, select the **ellipses** button to the right of the connection string, select the **Advanced** button, and then increase the **Connection Timeout** property. Try increasing to 30 seconds and then re-try the failed operation.
  18. Hold the mouse cursor over any data point on the graph to see the number of batch requests. This only gives us an idea of how busy the database was during the load test, but it is one measure that we can use to start to understand the impact of the web application under test.
      1. 
      2. Figure 64
      3. Custom graph showing Batch Requests per second
      4. **Note:** Many of the counters collected on this virtual machine, such as for SQL Server, must be looked at with the understanding that there are a bunch of applications running in the same environment. For example, databases for SharePoint Services and Team Foundation Server are also running.
  19. Even though the initial load test may result in some numbers that don’t seem to provide a wealth of information, it does provide a good baseline and allows us to make relative measures between test runs to help measure performance impacts of code changes. For example, if we had seen a relatively high level of batch requests per second during our initial load tests, perhaps that could be addressed by adding in some additional caching, and then re-testing to make sure that the request per second goes down.
  20. **Run** the load test one more time so that we have at least two test results to work with, so we can see how to perform some trend analysis.
  21. When the second load test is complete, select the **Create Excel Report** button from the toolbar to load Excel.
      1. 
      2. Figure 65
      3. Create Excel Report button
  22. In the **Generate a Load Test Report** window within Excel, make sure that the **Create a report** option is selected and then select the **Next** button to continue.
      1. 
      2. Figure 66
      3. Creating a new load test report
  23. When prompted for the type of report to generate, select **Trend** followed by the **Next** button.
      1. 
      2. Figure 67
      3. Selecting the Trend report type
  24. For **Report Name**, enter “**LoadTestTrend**” and select the **Next** button to continue.
      1. 
      2. Figure 68
      3. Naming the Trend report
  25. Select at least two load test runs to generate the trend report and then select the **Next** button to continue.
      1. 
      2. Figure 69
      3. Selecting test runs to generate trend report
  26. When prompted for counters to add to the report, note that there will be a number of pre-selected defaults. Leave those defaults in place, and add in the ‘**Batch Requests/sec**’ counter as well. Once you are done selecting counters, select the **Finish** button to generate the report.
      1. 
      2. Figure 70
      3. Selecting performance counters
  27. After the report is generated, a table of contents will be displayed that provides hyperlinks to specific report sheets. Select the **Batch Requests/sec** link.
      1. 
      2. Figure 71
      3. Table of contents for generated trend report
  28. The **Batch Requests/s** graph shows the batch requests per second average over the entire load test, for each test run that you selected for trend analysis.
      1. 
      2. Figure 72
      3. Batch Requests/s graph
  29. Let’s also take a look at the trend showing how specific web pages performed. Select the **Back to Table of Contents** link.
      1. 
      2. Figure 73
      3. Navigating back to table of contents
  30. Select the **Avg. Page Time** link from the table of contents.
      1. 
      2. Figure 74
      3. Navigating to Avg. Page Time graph
  31. The page time is a measure that includes all requests that were made for a web page, so it is a useful performance indicator to measure. Since we did not make any modifications to the application under test, we do not expect to see significant differences between the test runs.
      1. 
      2. Figure 75
      3. Average Page Time graph

To give feedback please write to [VSKitFdbk@Microsoft.com](mailto:VSKitFdbk@Microsoft.com)

Copyright © 2017 by Microsoft Corporation. All rights reserved.