

|  |
| --- |
| Dynmc_h_c  Microsoft Dynamics™ 365  Performance Toolkit Getting Started Guide |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication and is subject to change at any time without notice to you. This document and its contents are provided AS IS without warranty of any kind, and should not be interpreted as an offer or commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented. The information in this document represents the current view of Microsoft on the content. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, places, or events is intended or should be interred.

Code snippets, samples, or examples contained in this document are provided AS-IS without warranties of any kind. Do not distribute. The descriptions of other companies’ products in this document, if any, are provided only as a convenience to you. Any such references should not be considered an endorsement of support by Microsoft. Microsoft cannot guarantee their accuracy, and the products may change over time. Also, the descriptions are intended as brief highlights to aid understanding, rather than as thorough coverage. For authoritative descriptions of these products, please consult their respective manufacturers.

Contents

[Dynamics CRM Performance Toolkit Overview 4](#_Toc132109100)

[Dynamics 365 Performance Toolkit Components 5](#_Toc132109101)

[CRM\_Perf\_Benchmark Tool 5](#_Toc132109102)

[Setting Up the Test Environment 6](#_Toc132109103)

[Prerequisites 6](#_Toc132109104)

[Identify Dynamics 365 Organization for Testing 6](#_Toc132109105)

[Administrator Account for Setup and Testing 6](#_Toc132109106)

[Create Dynamics 365 Test Users 6](#_Toc132109107)

[Install Microsoft Visual Studio Enterprise 2019 6](#_Toc132109108)

[Install Microsoft SQL Server 7](#_Toc132109109)

[Install Fiddler 7](#_Toc132109110)

[Dynamics 365 Performance Toolkit installation 7](#_Toc132109111)

[Post Installation 9](#_Toc132109112)

[Create EntityManager Database 9](#_Toc132109113)

[Manually Add Test Users (OPTIONAL) 9](#_Toc132109114)

[Compile the CRM\_Perf\_Benchmark tool 9](#_Toc132109115)

[Populate the EntityManager Database 9](#_Toc132109116)

[Run a WebTest 11](#_Toc132109117)

[Guidelines for configuring your own performance testing 13](#_Toc132109118)

[Creating and running a Load Test 13](#_Toc132109119)

[Advanced Scenarios 20](#_Toc132109120)

[Creating a Custom Web test 20](#_Toc132109121)

[Recording a Custom Web Test that will Create a New CRM Record 20](#_Toc132109122)

[Recording a Custom Web Test that will Update an Existing CRM Record 28](#_Toc132109123)

[Refreshing the EntityManager Database 34](#_Toc132109124)

[Filtering Records in PreWebTest 34](#_Toc132109125)

[Configuring Controller/Agent Setup to Generate load from more than one machine 36](#_Toc132109126)

# Dynamics CRM Performance Toolkit Overview

The Microsoft DynamicsTM 365 Performance Toolkit was created by the Microsoft CRM product team to formalize performance testing of Microsoft Dynamics 365.

The toolkit facilitates load testing the performance of CRM deployments. By carefully planning the required dataset that the deployment needs to support and the workload requirements, the toolkit can be used to test if the scale requirements of a particular deployment can be met. This mechanism can be used in the decisions on a particular deployment solution and avoid costly downtimes at a later stage.

The Performance toolkit contains various tools listed below that can be used in customizing the CRM installation, populating the necessary semantic data for the required scale and conducting the benchmarking tests against the CRM installation. The tools provided in the Performance toolkit are:

1. CRM\_Perf\_Benchmark Tool
2. Data Population Tool
3. ImportCustomization Tool

The Performance Toolkit is distributed in source format under the Open source licensing agreement. Knowledge of Microsoft Visual Studio® and Microsoft Visual C#® is required to use the toolkit. The following are the pre-requisites to conduct the performance tests.

* Microsoft Visual Studio 2015 Enterprise running on a client system used to drive the CRM product. A workstation with 4GB RAM is sufficient to run the benchmarking tests.
* Microsoft Dynamics 365 Organization suitable for performance testing. Ideally, the organization configuration should be similar to the actual production instance.

# Dynamics 365 Performance Toolkit Components

The toolkit consists of following tools:

## CRM\_Perf\_Benchmark Tool

This is a tool that can be run in the Visual Studio Test infrastructure that comprises of several scenarios driven tests that can be used to test the performance of the Microsoft Dynamics CRM deployment under a load simulation.

# Setting Up the Test Environment

## Prerequisites

### Identify Dynamics 365 Organization for Testing

A Dynamics 365 Organization that mimics the production organization should be identified for the Benchmark Engagement. The organization should have customizations and production like dataset created to ensure an accurate load test. Some sample data can be created as part of the engagement if needed.

Records may be created/updated in the organization during the load testing based on the provided use cases. This should be taken into consideration when choosing the test organization such that the testing and resulting changes does not impact other users.

The Microsoft Premier Field Engineer will need to be provided an administrator account that has access to the organization to be used during the engagement. A pool of sequential test users will also need to be created. Details of the administrator and test accounts are listed below.

In some cases, the identified test organization will be moved to a dedicated testing scalegroup to ensure that other online tenants are not impacted by the load test.

### Administrator Account for Setup and Testing

An Administrator account is needed for the setup of the testing harness. This account will be used to install and configure the test harness and execute the Load Tests. As part of the configuration the user will need to access the Dynamics 365 customizations area to identify schema information for the entities to be tested.

### Create Dynamics 365 Test Users

A pool of test users must be created in the test organization used in the Dynamics 365 Benchmark Testing. There should be at least enough test users to simulate the various roles in Dynamics 365 and levels of ownership.

Each of these users will be named with a prefix and an index, for instance we will have test users named crmusr1 to crmusrN where N is the number of test users that the Dynamics 365 deployment needs to support. In the test environment it is assumed all the test users have the same password. If the test users do not have sequential names or a common password additional steps are required during the setup process to ensure the test users are loaded correctly.

Once test users are created they should then be added into Dynamics 365 with appropriate security roles that mimic actual production users.

### Install Microsoft Visual Studio Enterprise 2019

Install Microsoft Visual Studio according to the Visual Studio installation instructions on a dedicated client computer.

The Microsoft Dynamics 365 Performance Toolkit uses Microsoft Visual Studio as the platform for its load tests. This document assumes the user is familiar with using Microsoft Visual Studio.

### Install Microsoft SQL Server

Install a copy of SQL Server on the Load Test workstation or another accessible server. Two databases will be created and used during the load testing of CRM. The Databases are used for storing load test results by Visual Studio and the other for storing CRM input data needed when running tests.

### Install Fiddler

Download and install Fiddler from <http://Fiddlertool.com>. This will be used when recording custom web tests.

## Dynamics 365 Performance Toolkit installation

The toolkit zip file is available for download at

<https://microsoft-my.sharepoint.com/:f:/p/jemorl/EkhHJ7aV1QlJiTVkY6BeEOcBFIeZCKQk8yBov7If7dYTnw?e=GBs2rV>

Toolkit Setup:

1. Download [CRMToolkit](https://microsoft-my.sharepoint.com/:f:/p/jemorl/EkhHJ7aV1QlJiTVkY6BeEOcBFIeZCKQk8yBov7If7dYTnw?e=GBs2rV) zip file to your local workstation
2. Unzip the toolkit to the C: drive.
3. Rename CrmToolkit Folder to include orgname. (Example: CRMToolkitContoso)
4. Run the Powershell installation script.
   1. Launch Windows Powershell ISE under Administration mode
   2. Open the installation powershell script from C:\CRMToolkit\1ToolkitInstall.ps1.
   3. Update the environment details in the script and execute to complete the installation.

**Note:** If test users do not have sequential names or a common password the userbase and userpassword parameters can be left at the default values.



#### Configuration Settings

A configuration setting xml file will be generated at the end of toolkit setup. You can find the configuration setting xml file in <toolkit dir>\Config folder.

A sample of the xml that gets created by the setup process is shown below:

<?xml version="1.0"?>

<configsettings>

<msods value="true" />

<authentication type="onlinefederation" />

<crmonlinedllpath value="C:\CRMToolkitjemorl\CRM\_Perf\_Toolkit\CRMOnline\bin\Debug\CRMOnline.dll, CRMOnline.EntityManagerOnline" />

<ignoretokencheck value="true"/>

<trace value="false"/>

<webservicens value="http://schemas.microsoft.com/crm/2009/WebServices"/>

<crmdomain value="jemorl.onmicrosoft.com"/>

<runas userid="jmorlock@jemorl.onmicrosoft.com" password="Access123"/>

<emsqlcnn value="Application Name=CRM\_Perf\_BenchMark;Server=.;Initial Catalog='EntityManagerjemorl';Integrated Security=sspi"/>

<reportserver value="http://jemorl.crm.dynamics.com"/>

<parsedependentrequests value="false"/>

<bandwidthcapture value="false">

<netcapdir value="C:\Program Files\Support Tools"/>

<bandwidthoutputdir value="C:\CRMToolkit\CRM\_Perf\_Toolkit\BandwidthTestResults"/>

</bandwidthcapture>

<outlooksyncdir value="c:\outlooksyncdata"/>

<exchangepassword value="Access123"/>

<turboformsenabled value="true"/>

<crmservers>

<crmserver>

<serverbaseurl value="https://jemorl.crm.dynamics.com" />

<organizationbaseurl value="https://jemorl.crm.dynamics.com" />

<organization name="jemorl"/>

<organizationserviceurl value="https://jemorl.api.crm.dynamics.com/XrmServices/2011/Organization.svc" />

<discoveryserver value="disco.crm.dynamics.com" />

<sqlcnn value="Application Name=CRM\_Perf\_BenchMark;Server=.;Initial Catalog='jemorl\_MSCRM';Integrated Security=sspi"/>

<configsqlcnn value="Application Name=CRM\_Perf\_BenchMark;Server=.;Initial Catalog='MSCRM\_CONFIG';Integrated Security=sspi"/>

<adxenabled value="False"/>

<passport>

<userbase value="crmusr" /><adminuser value="jmorlock@jemorl.onmicrosoft.com" />

<start value="1" />

<count value="5" />

<outlookstart value="1" />

<outlookcount value="5" />

<passportdomain value="jemorl.onmicrosoft.com" />

<password value="Access123" />

<partner value="port.crm.dynamics.com" />

</passport>

</crmserver>

</crmservers>

</configsettings>

<executemultiple>

<maxThreads>4</maxThreads>

<batchSize>1000</batchSize>

<totalEntities>10000</totalEntities>

</executemultiple>

</configsettings>

## Post Installation

### Create EntityManager Database

1. Open SQL Scripts Folder <toolkit dir>\Binaries\Scripts\SQL.
2. Run EMDB\_Setup.SQL in SQL Management Studio to create the EntityManager Database. The database name will include the organization being tested.

### Manually Add Test Users (OPTIONAL)

**NOTE:** This is only needed if they do not use Sequential Usernames or a Common Password

1. Navigate to the AddRandomUsers folder <toolkit dir>\CRMToolkit\AddRandomUsers
2. Open the userlist.csv file and populate the list of usernames and passwords for the test users.
3. Run the insertusers.sql file in SQL management studio. When EMDBLoader is ran it will load users that match values provided in both the config file and the userlist table.

### Compile the CRM\_Perf\_Benchmark tool

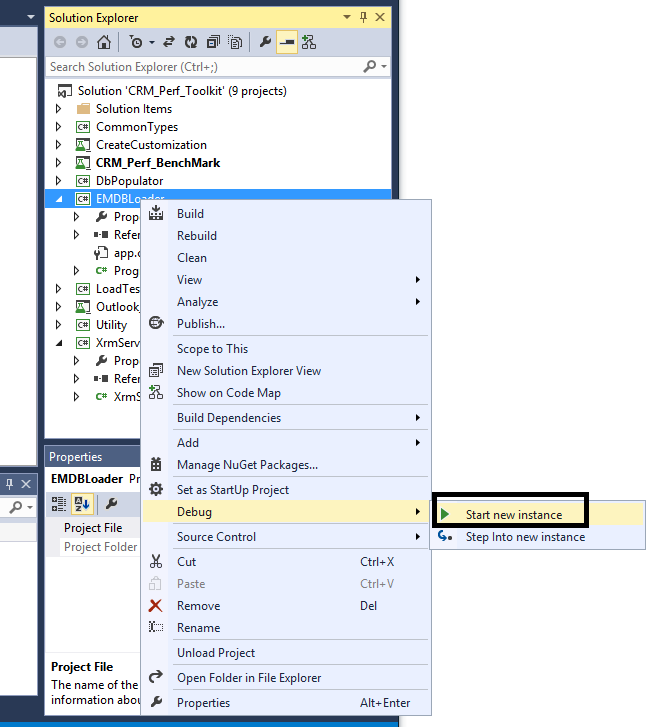
1. Open <toolkit dir>\CRM\_Perf\_Toolkit\CRM\_Perf\_Toolkit.sln in Visual Studio.
2. Click Build Menu and Build Solution.

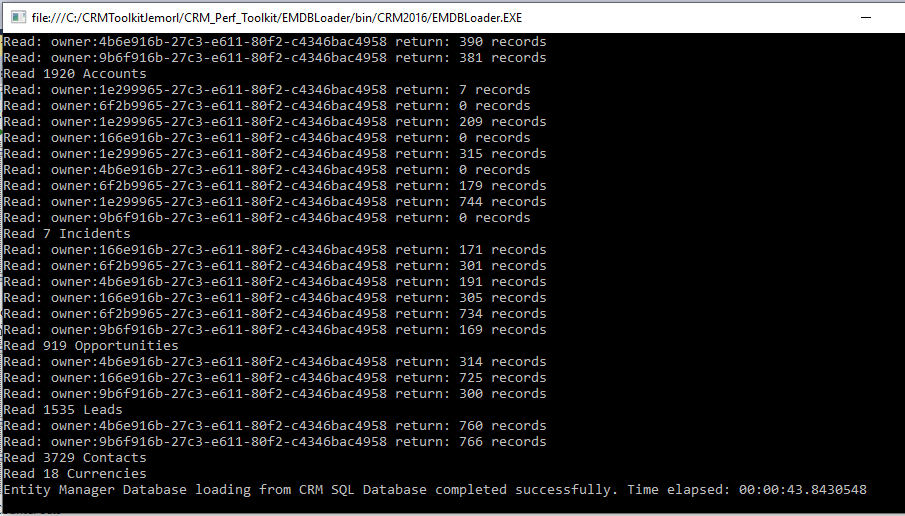
### Populate the EntityManager Database

Dynamics 365 data used by the performance tests is copied to a local database on the Client from the CRM servers. This database that stores this data is called EntityManager database. The EntityManager database is created by EMDBLoader process if it has not been created manually. The copying of the data from the CRM databases to the EntityManager database can be done running the EMDBLoader.exe which is a part of the CRM\_Perf\_Toolkit solution.

Run EMDBLoader to populate the EntityManager database.

* 1. From Visual Studio, right click on EMDBLoader, Debug, Start a new instance.

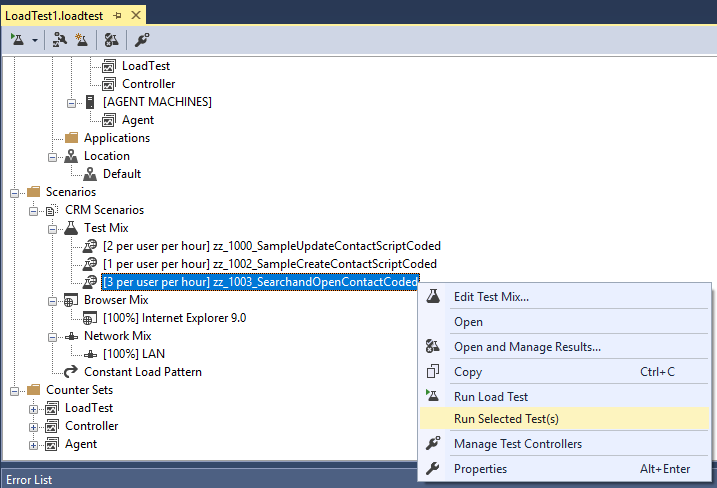




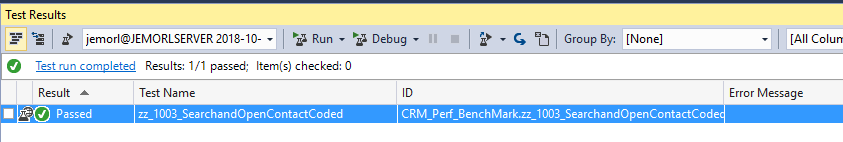
### Run a WebTest

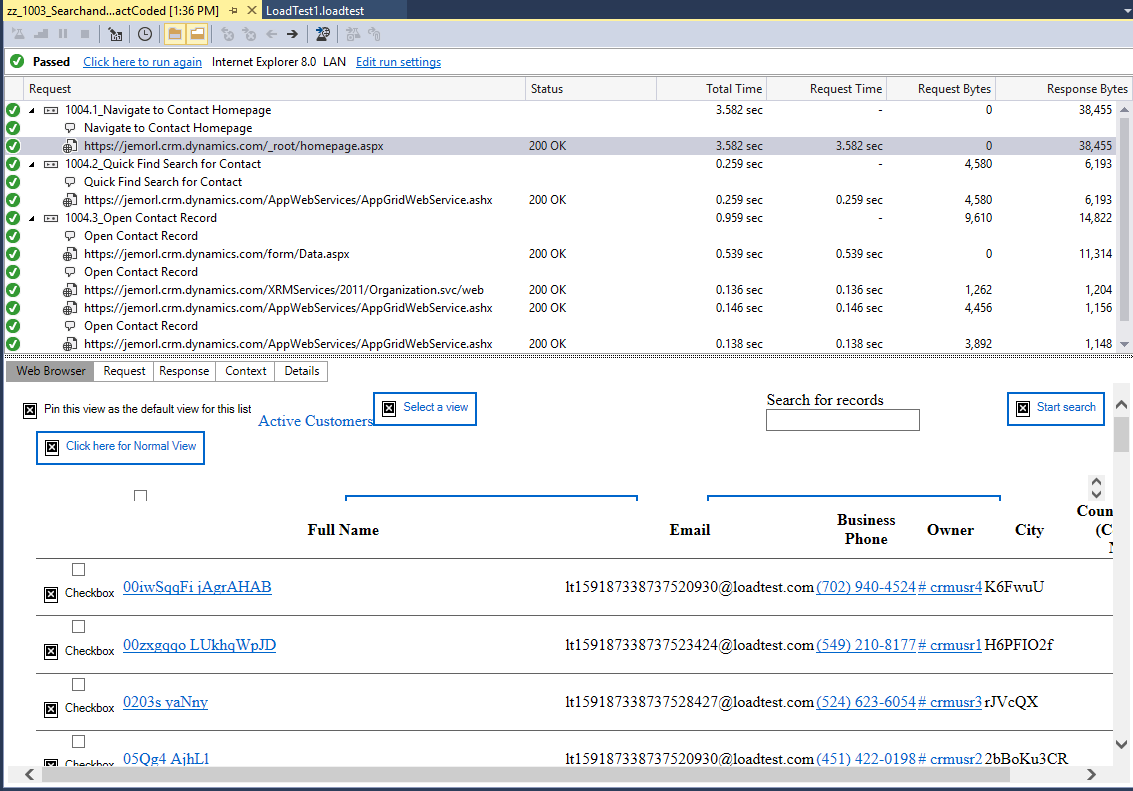
Once EMDBLoader is finished the solution is ready to run some sample tests. Most configuration and setup problems can be diagnosed by running an individual Web test. To run an individual Web test, do the following:

1. From Visual Studio, Open LoadTest1.loadtest file in CRM\_Perf\_Benchmark\ \_SampleTests
2. Within the Test Mix, Right click on zz\_1003\_SearchandOpenContactCoded and Click on Run Selected Test



1. The test result can be viewed by double clicking on the completed test within the Test Results window at the bottom of Visual Studio.

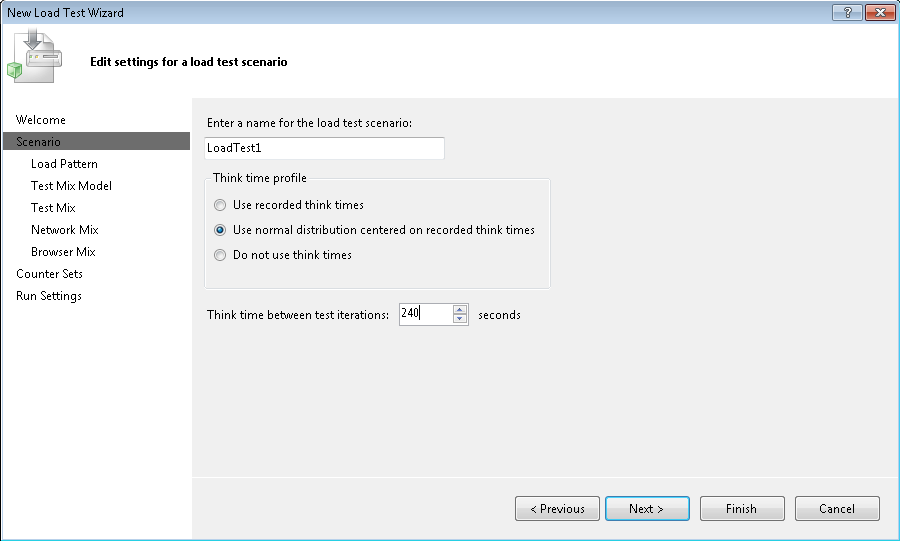




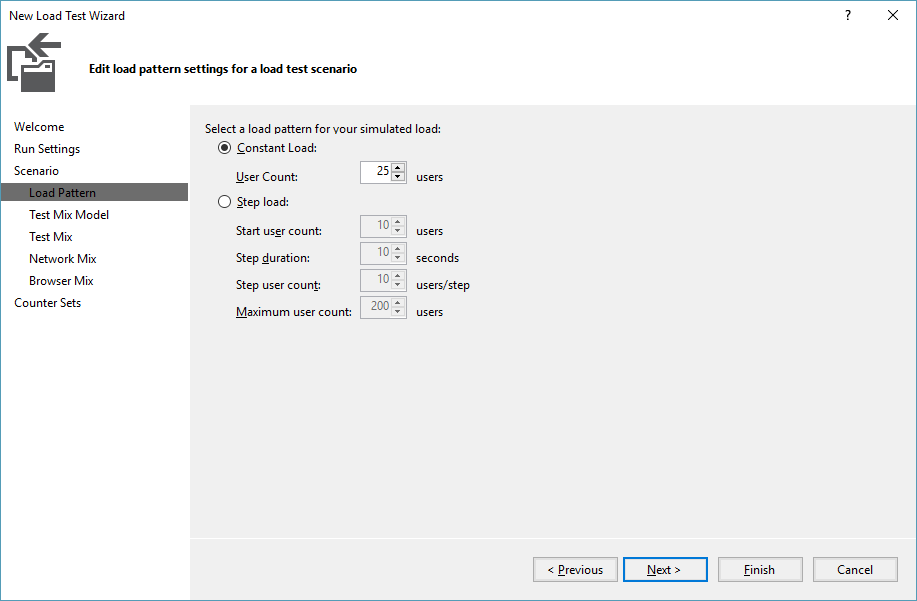
## Guidelines for configuring your own performance testing

### Creating and running a Load Test

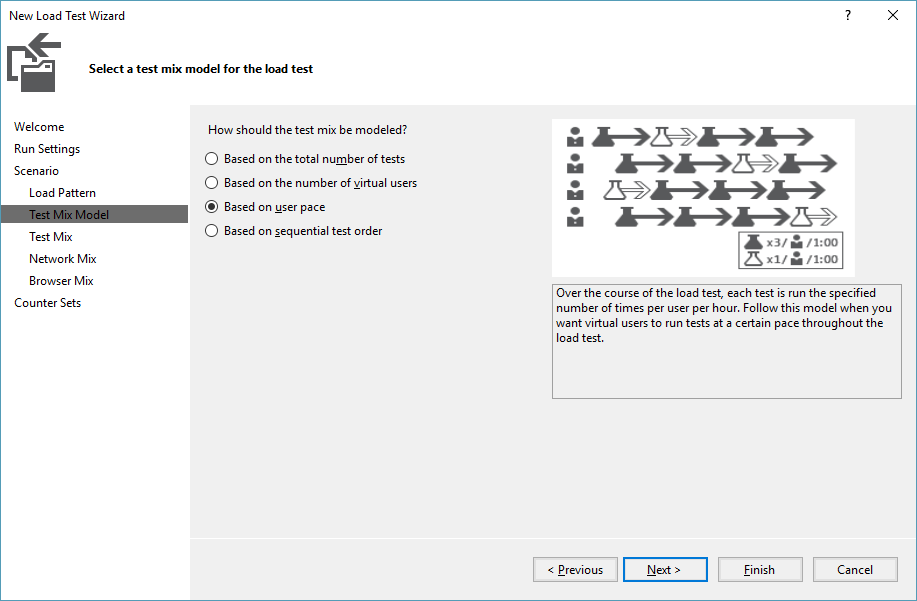
1. Open the Performance Toolkit Solution in Visual Studio.
2. Select New Test from the Test Menu.
3. Select Load Test, Enter a Test Name, and click OK. The New Load Test Wizard will now open.
4. Click Next. Enter a Scenario Name and [Think Time](#_Think_Times). Click Next.



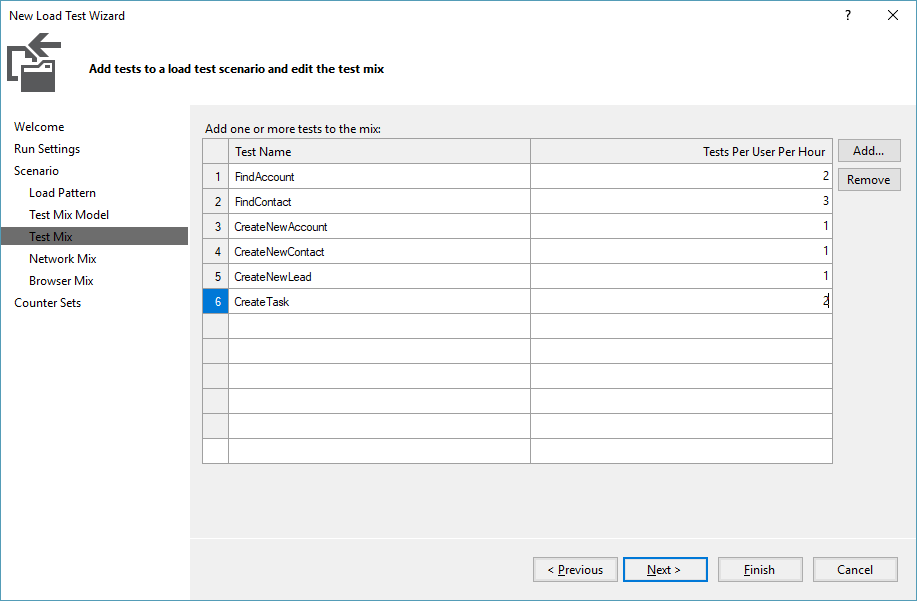
1. Select a Load Pattern and enter number of users that will be tested. A Constant Load pattern will start and finish with the same number of users. The Step Load pattern will start with a small number of users and progressively add more users up to the maximum that was specified. Once the Load Pattern is entered, Click Next.



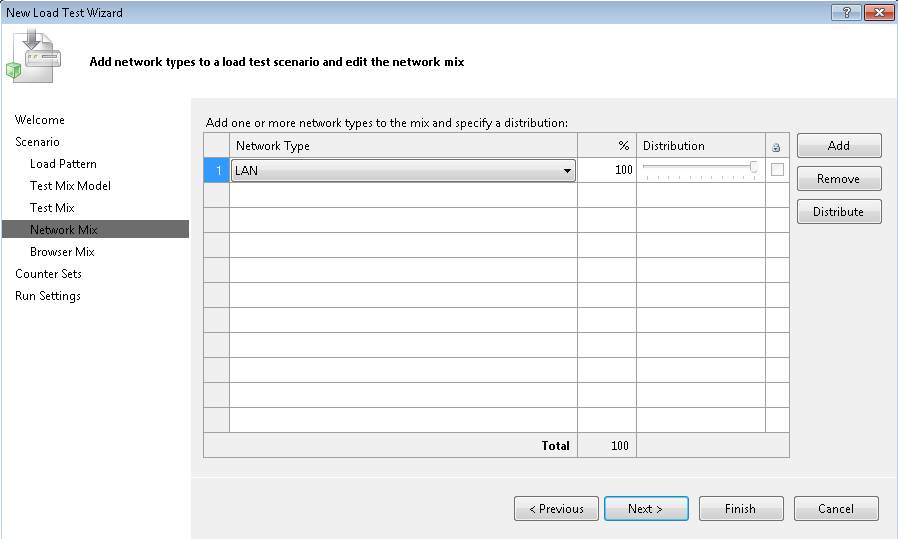
1. Choose the Test Mix Model based on how you want the web tests to be executed. Options are available to choose the web test based on percent distribution, tests per hour, or the sequential order chosen. Click Next



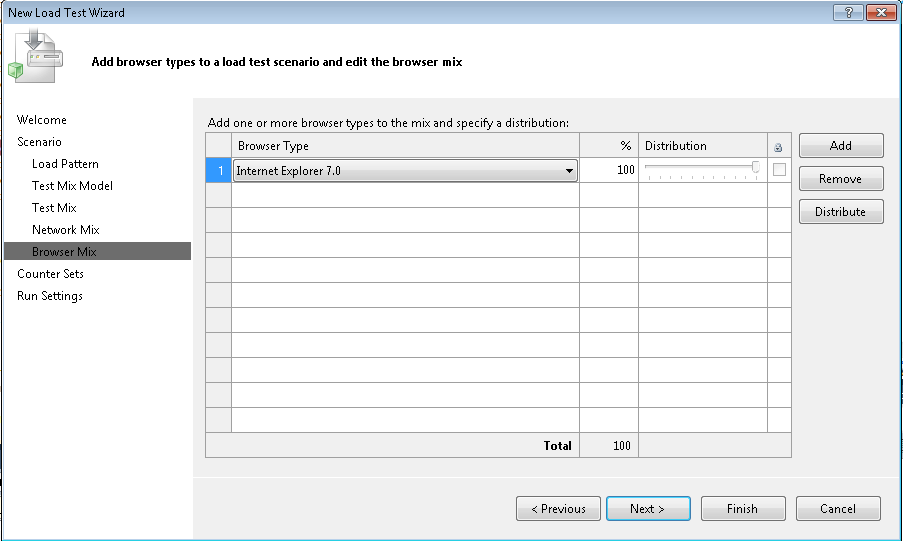
1. Choose the Test Mix by selecting the Web Tests and entering distribution as needed. Click Next



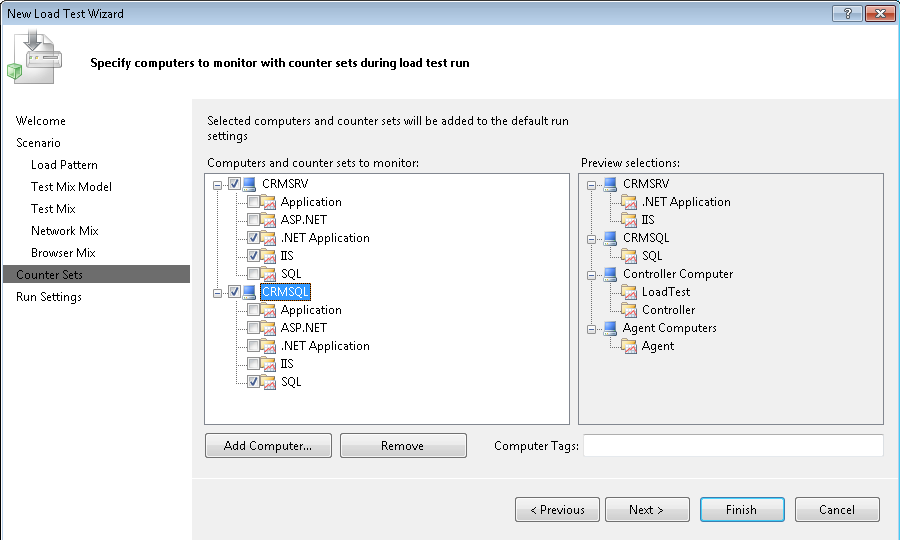
1. The defaults on the Network Mix can be kept unless different network conditions need to be emulated. Click Next.



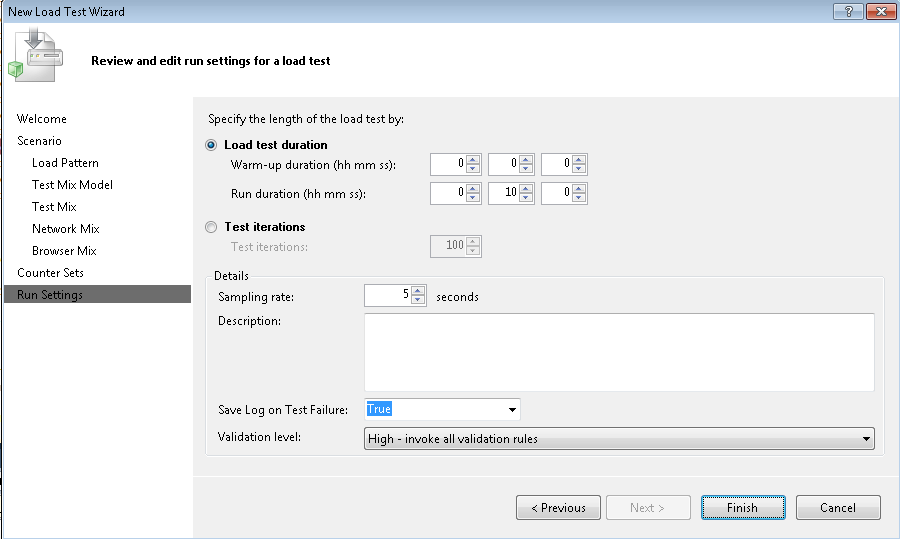
1. The Default Browser Type of Internet Explorer 7.0 can be left. Changing the Browser mix does not have any effect on the outcome of CRM web tests, so the defaults can remain in place. Click Next.



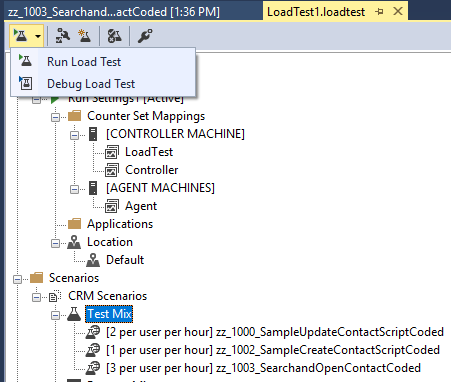
1. We do not have access to online server so this step can be skipped. For OnPrem deployments add any Servers and Performance Counters that you’d like to have monitored during the load test. Click Next.



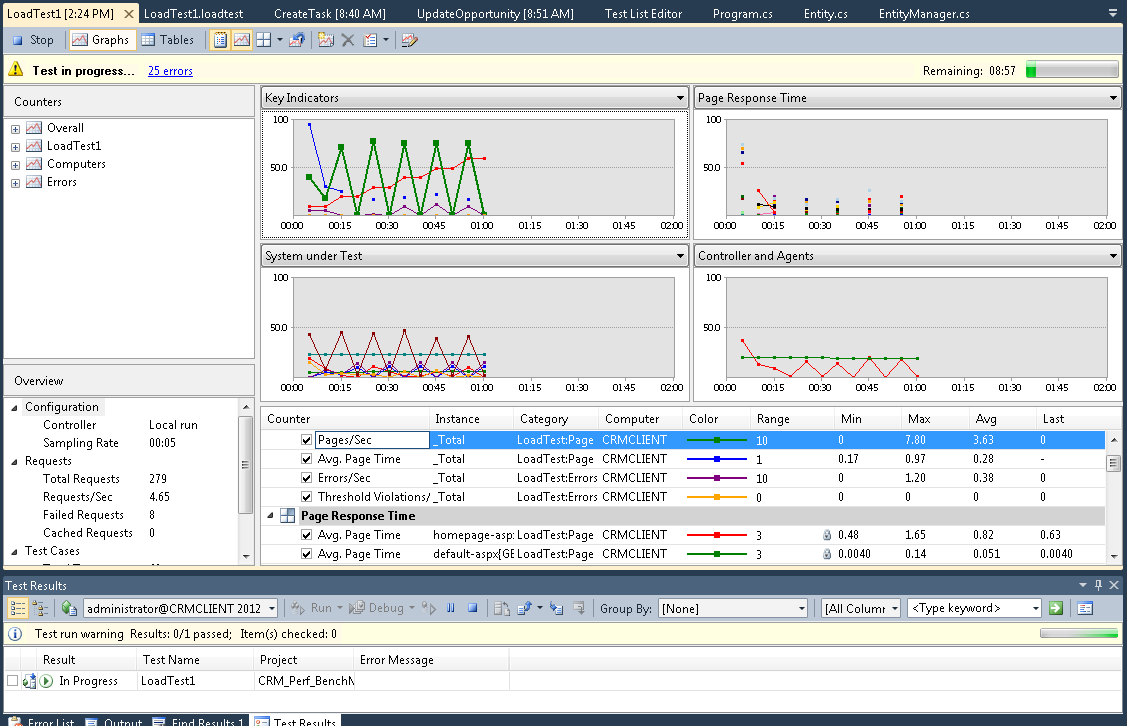
1. Enter the Warm-up and Duration. Warm-up is the time that users will be ramped up until the max user level is met and the test will run for the Run Duration at that full user load. Click Finish.



1. The new load test will be opened in Visual Studio. Click the Run Test button on the toolbar to start the load test.



1. Details of the load test will be displayed in Visual Studio once the testing begins.



Microsoft Visual Studio lets you monitor the progress of the loadtest while it is running. There are two windows, which you can select by clicking either Graphs or Tables at the top of the window. For detailed information about the Test Result window, see the Visual Studio Team System documentation.

The Web tests provided with the toolkit contain explicit transactions around key stages in each business use case scenario. For example, the Web test "CreateNewAccount" contains the transactions "NavAccountHomepage" and "CreateAccount". The performance of these transactions is viewed by clicking Tables and selecting Transactions from the drop-down list in the Load Test Result window. The other tables available in the drop-down list provide different levels of granularity when you view the results.

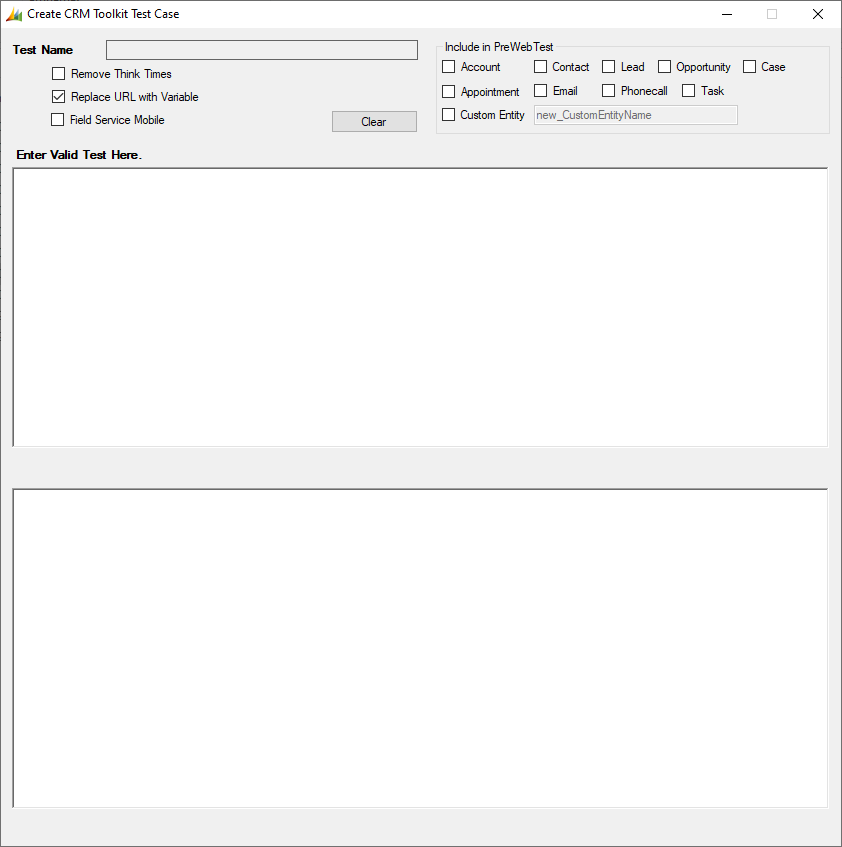
# Advanced Scenarios

## Creating a Custom Web test

This section will provide the general steps for creating custom web tests for Dynamics 365. Recording custom web tests will ensure the test accurately represents the user actions including any changes from customized forms and code.

### Recording a Custom Web Test that will Create a New CRM Record

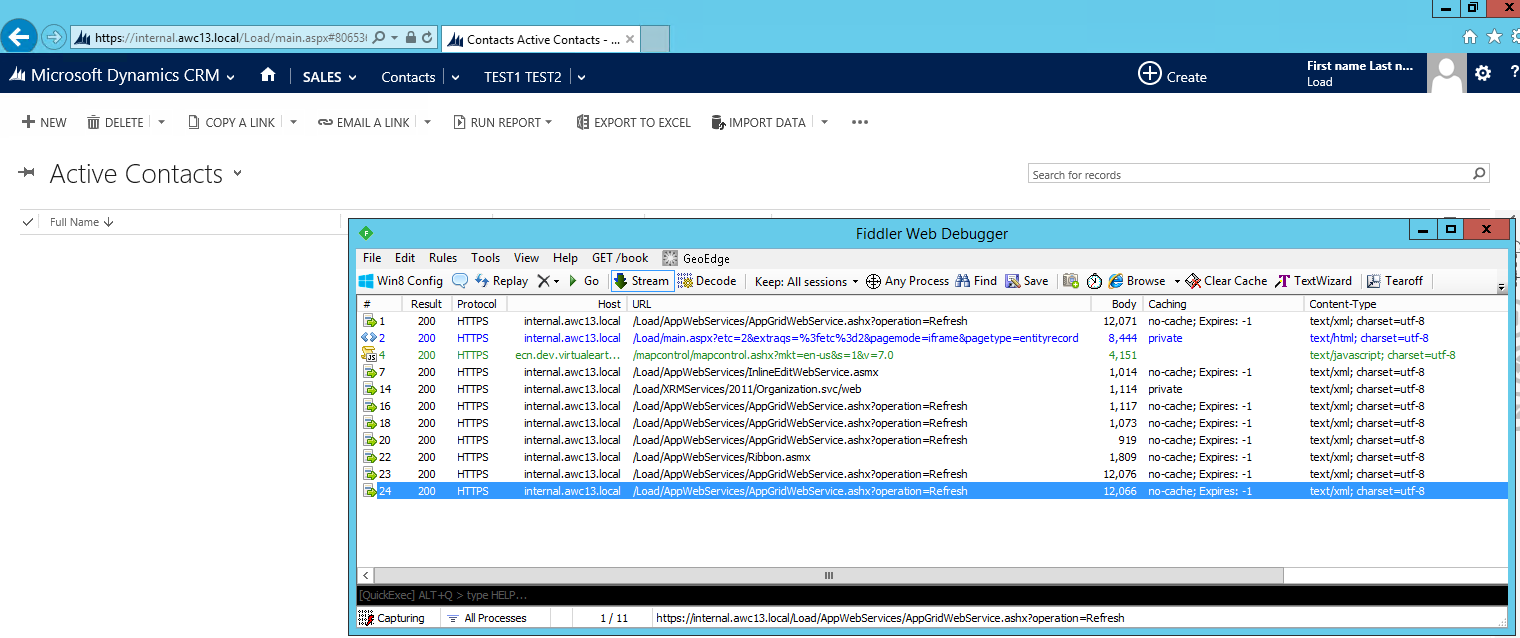
1. Open the Create CRM Toolkit Test Case utility from within the CRMToolkit Folder.



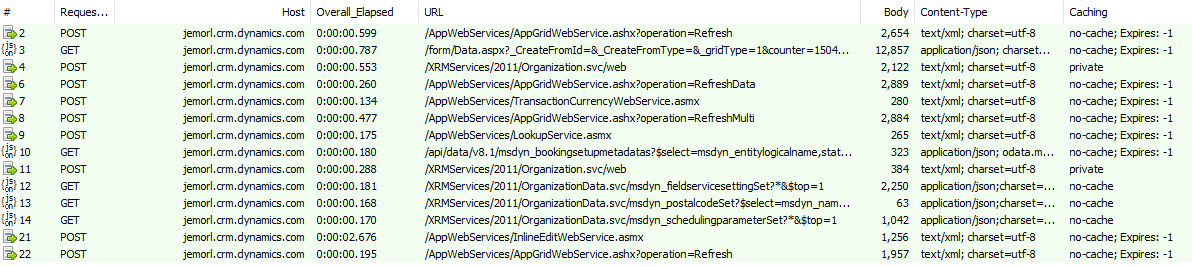
1. Open CRM and use Fiddler to capture the navigation steps for creating a record. It is recommended to complete navigation at least once before starting Fiddler. This will ensure that the static files are cached properly and only dynamics request are captured in the Fiddler trace.

For Example:

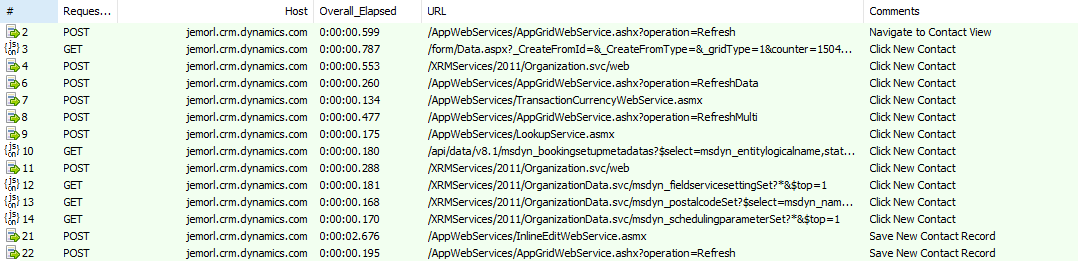
* 1. Open CRM
  2. Open Fiddler
  3. Navigate to the Contact View
  4. Click New
  5. Enter required fields
  6. Click Save and Close
  7. Stop the Fiddler Capture



1. Remove everything except for the .svc, .aspx, .asmx, and .ashx entries with a 200 result. The remaining requests should say private or no-cache in the Caching column signifying they are not static files.

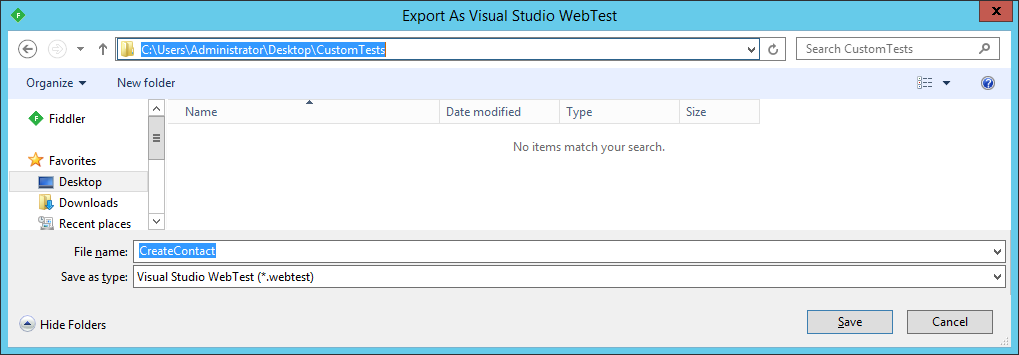


1. Comments should be added in fiddler for each group of requests to identify which action the requests were for. This will be used when adding transactions to the test. To add a commend select the group of requests, right click and select Comment. Add Comments such as (Navigate to Contact View, Click New Contact, Save New Contact). The comments will show up in the comments column.

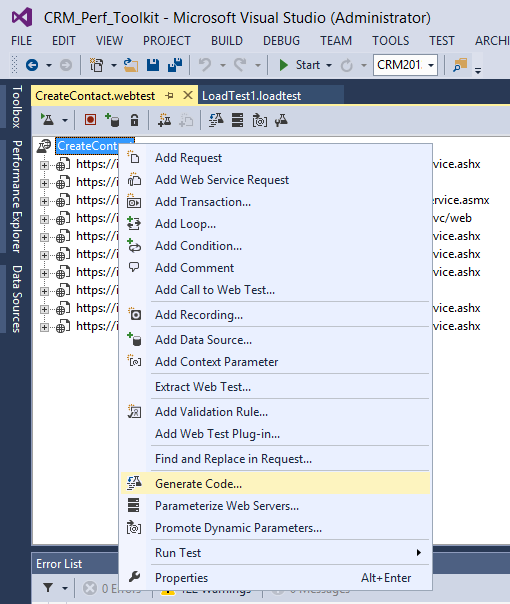


1. Save all output as a Visual Studio Web Test
   1. Click File-Export Sessions-All Sessions
   2. Choose Visual Studio Web Test and Click Next
   3. Provide the test a name, such as 01\_CreateContact.webtest, Change the path to <toolkit dir>\TestScripts and Click Save

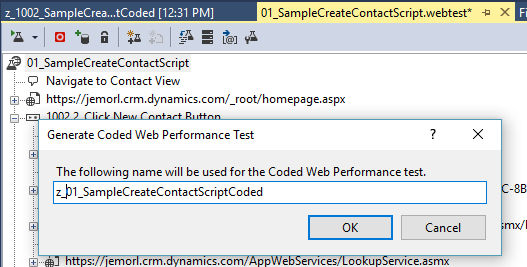
Note: numbering the tests will make them easier to order when reviewing results.



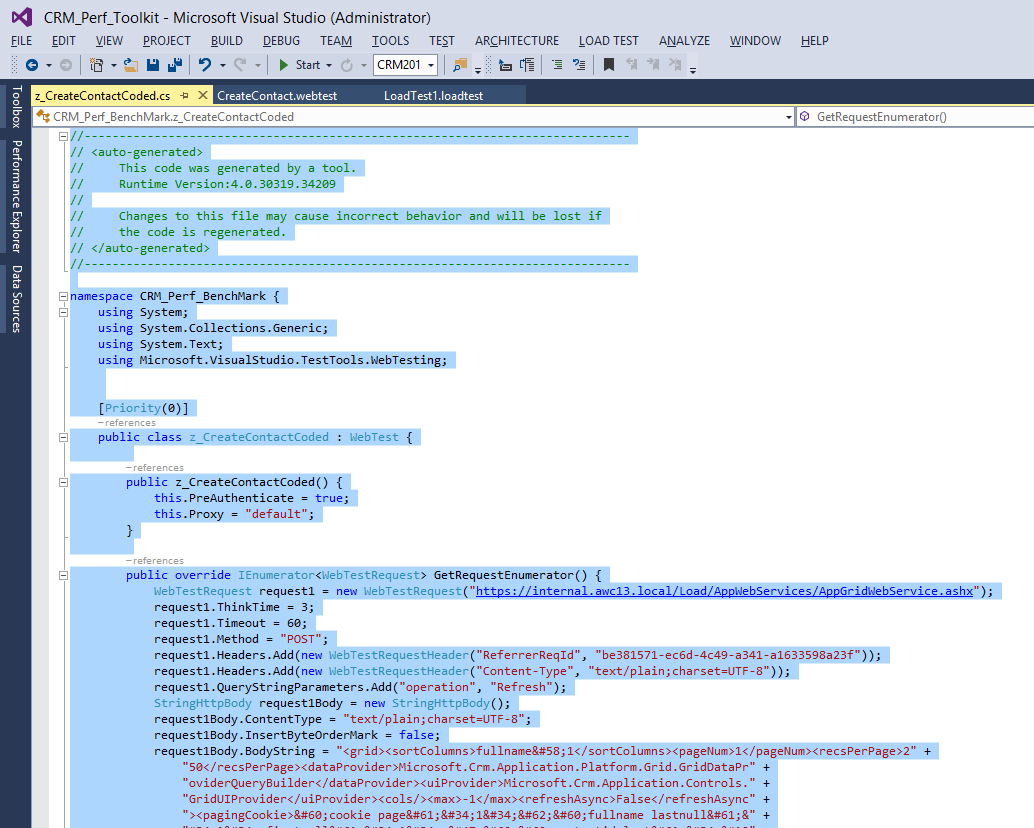
1. Run AddTransactionstoWebTest.exe Utility
   1. Navigate to <toolkit dir>\TestScripts.
   2. Run the AddTransactionstoWebTest.exe Utility. This will add numbering to each comment and transactions to the webtest.
2. Add the Web Test to your Visual Studio project.
   1. Right click on the CRM\_Perf\_Benchmark project, Click Add, New Folder, and Name the folder CustomTests. Proceed to step 2 if you’d like to use an existing folder.
   2. Right Click the new folder, Click Add, Existing Item. Browse to the webtest that was created in step 3, and click Add.
3. Double-click the imported webtest to open it.
4. Right click on the root of the webtest, and select Generate Code



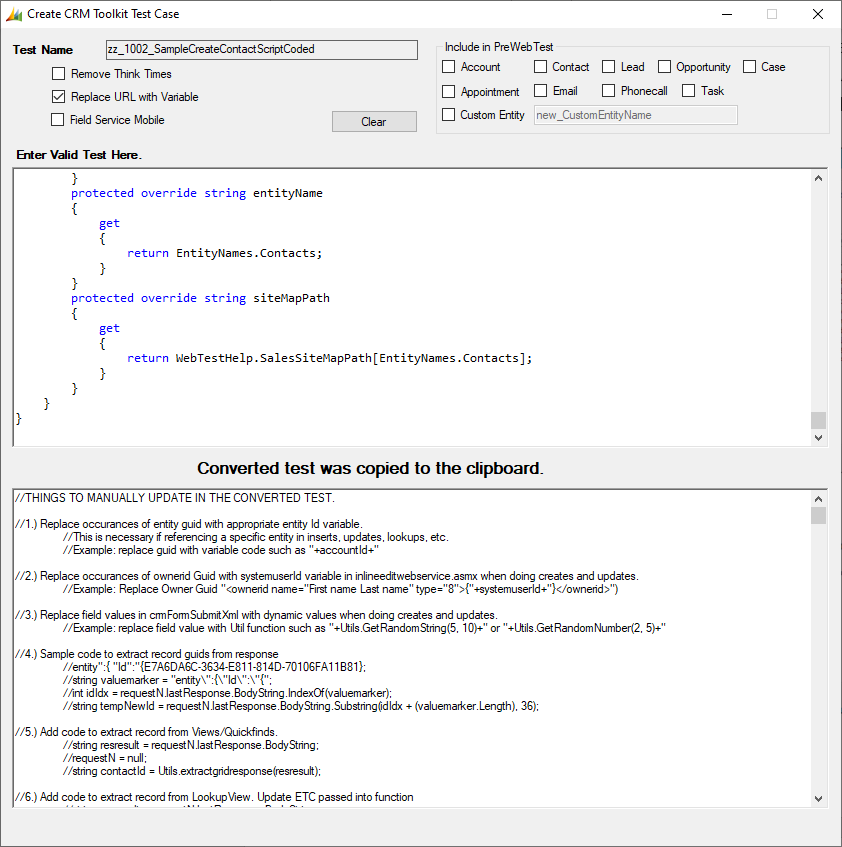
1. Enter a name for the coded web test. Add the z\_ prefex to the custom test name to keep custom tests grouped together. A new coded web test will be created and opened.



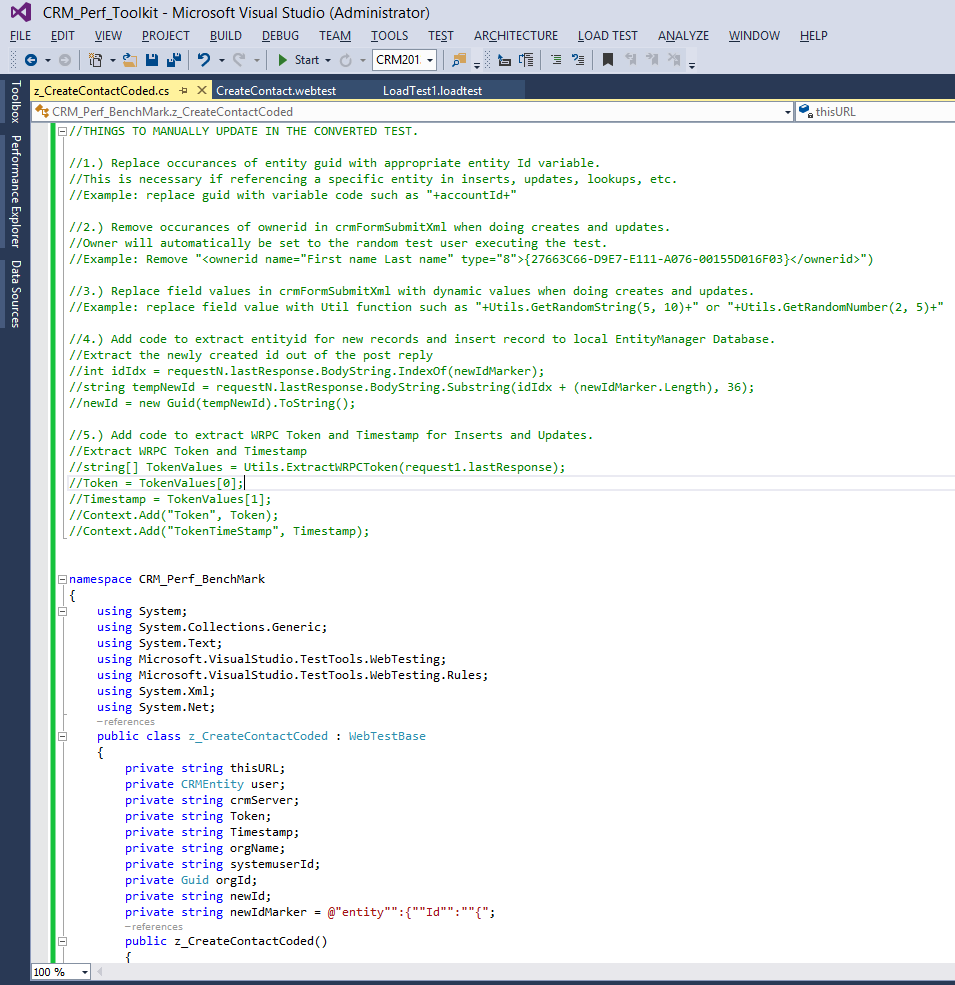
1. Select and copy the code from the coded web test.



1. Maximize the Create CRM Toolkit Test Case utility and paste the code. The code will be converted to format compatible with the performance toolkit and place code back on the clipboard. The conversion process will consist of tasks including creating required variables, creating prewebtest with methods to retrieve required data, replacing request URL’s to variables, change requests from webrequest to crmrequest, remove unnecessary header values and extract ID’s from created records.



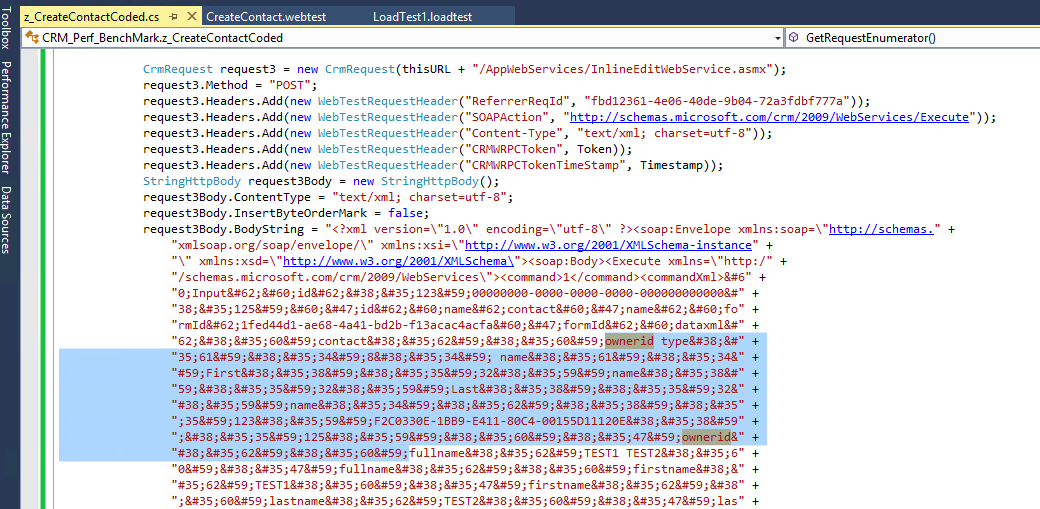
1. Paste the updated code over the original code in Visual Studio.



1. Make manual code changes to updated code.
   1. Review each web request to ensure hardcoded CRM URL’s were properly replaced with the variable thisURL + “.



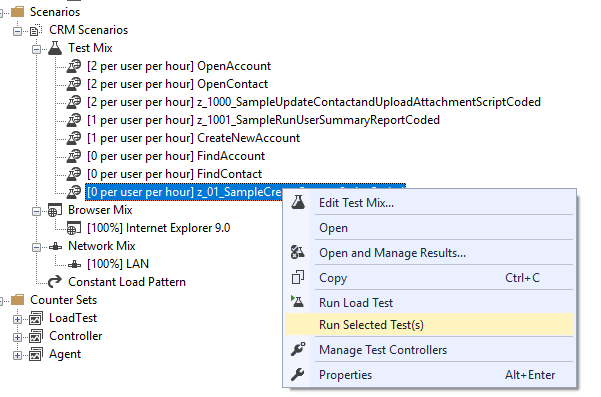
* 1. Navigate to the Inlineeditwebservice.asmx request and replace the ownerid guid with the systemuserId variable.

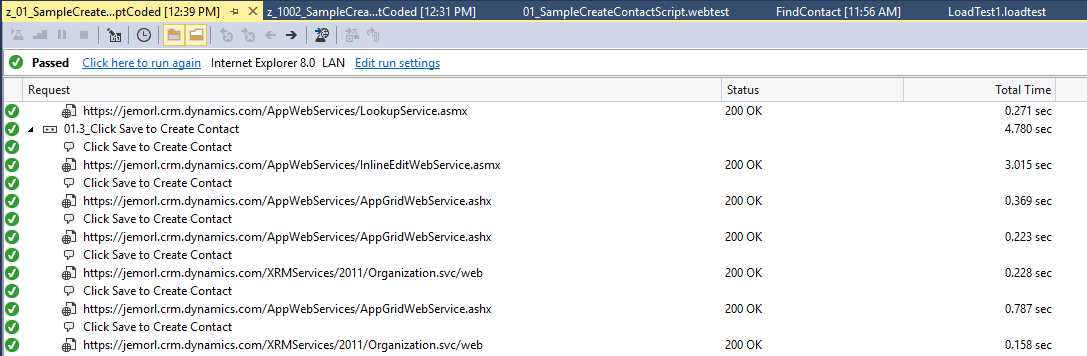


* 1. Optional: Replace variables in Inlineeditwebservice BodyString with functions such as Utils.GenerateRandomString(min,max), or Utils.GenerateRandomNumber(min,max). This is only needed if random variables are desired.



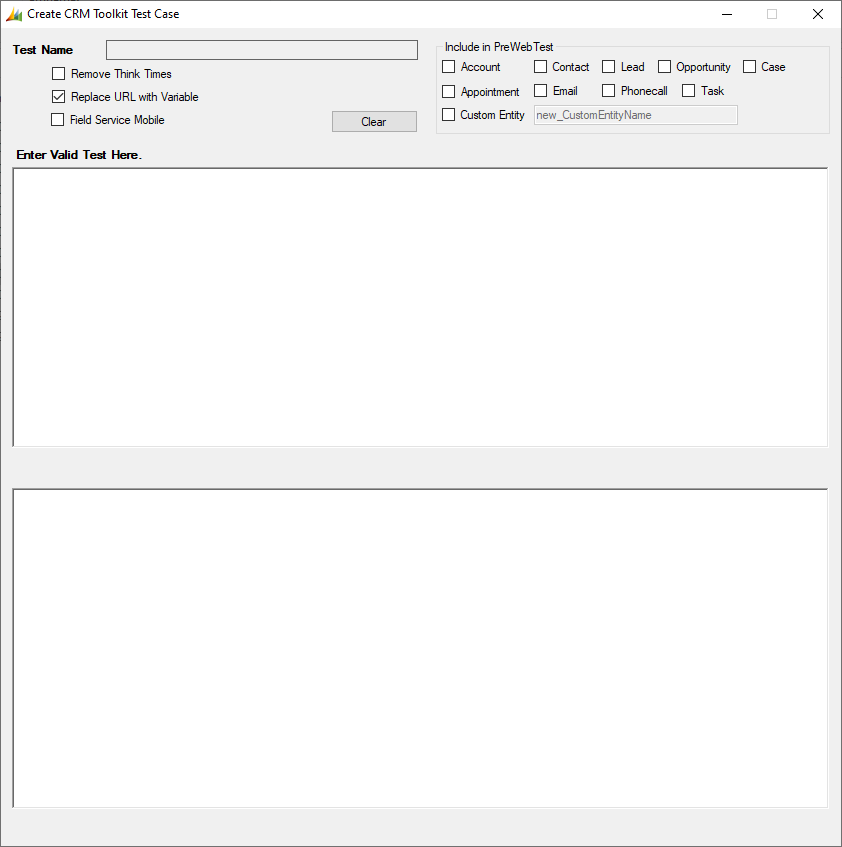
* 1. Save the coded test and click build solution from the build menu.
  2. The test can now be added to a load test and executed to verify it functions properly.





### Recording a Custom Web Test that will Update an Existing CRM Record

1. Open the Create CRM Toolkit Test Case utility provided by PFE and select the correct CRM Version. If applicable, select ADFS, Internal or External. Minimize the window.

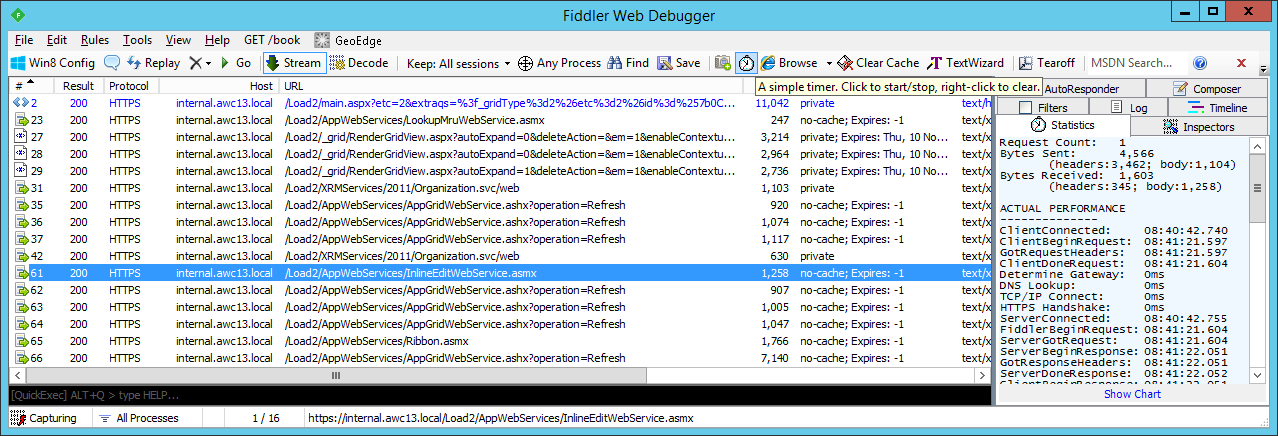


1. Open CRM and use Fiddler to capture the navigation steps for Updating a record. It is recommended to complete navigation at least once before starting Fiddler. This will ensure that the static files are cached properly and only dynamics request are captured in the Fiddler trace.

For Example:

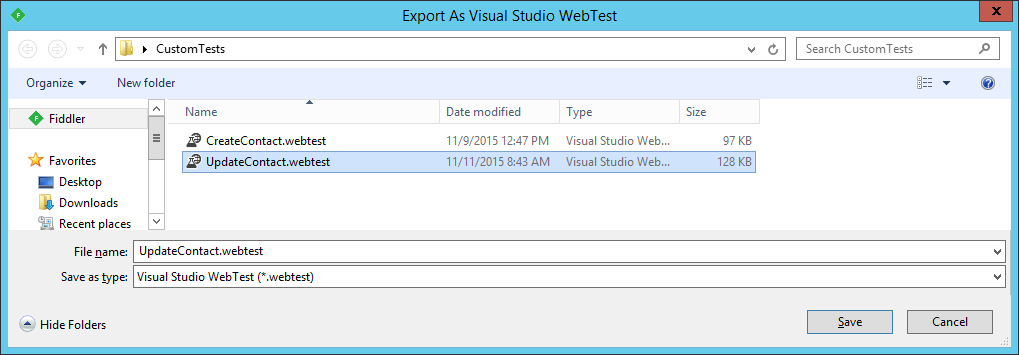
* 1. Open CRM
  2. Open Fiddler
  3. Navigate to the Contact View
  4. Open an Existing Contact
  5. Update one or more attributes
  6. Click Save and Close
  7. Stop the Fiddler Capture

1. Remove everything except for the .svc, .aspx, .asmx, and .ashx entries with a 200 result. The remaining requests should say private or no-cache in the Caching column signifying they are not static files.

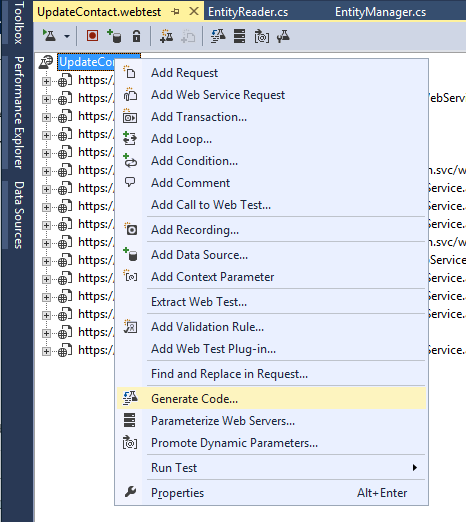


1. Add comments to each group of requests.
2. Save all output as a Visual Studio Web Test
   1. Click File-Export Sessions-All Sessions
   2. Choose Visual Studio Web Test and Click Next
   3. Provide the test a name, such as UpdateContact.webtest, Change the path to <toolkit dir>\TestScripts and Click Save

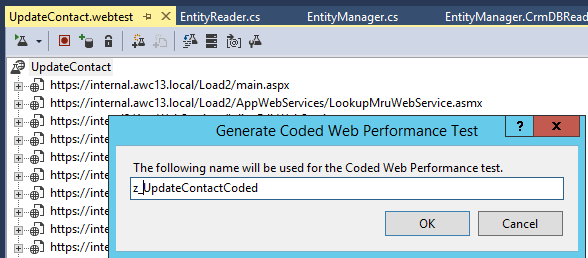
Note: numbering the tests will make them easier to order when reviewing results.



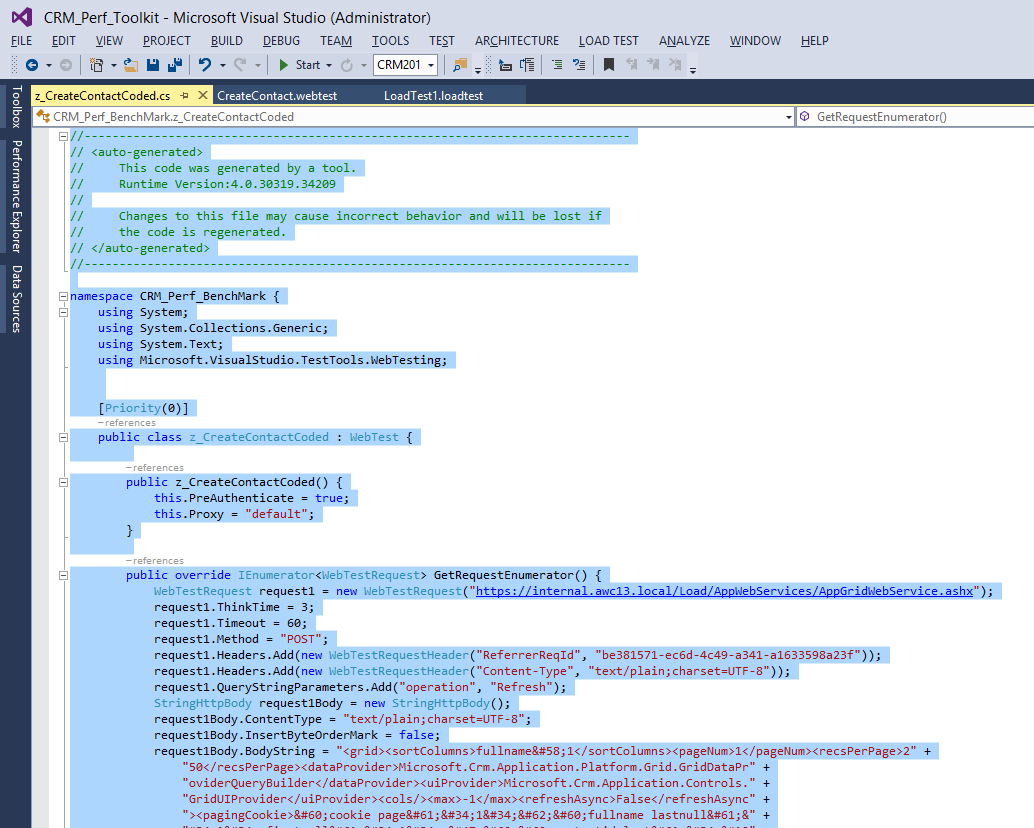
1. Run AddTransactionstoWebTest.exe Utility
   1. Navigate to <toolkit dir>\TestScripts.
   2. Run the AddTransactionstoWebTest.exe Utility. This will add numbering to each comment and transactions to the webtest.
2. Add the Web Test to your Visual Studio project.
   1. Right click on the CRM\_Perf\_Benchmark project, Click Add, New Folder, and Name the folder CustomTests. Proceed to step 2 if you’d like to use an existing folder.
   2. Right Click the new folder, Click Add, Existing Item. Browse to the webtest that was created in step 3, and click Add.
3. Double-click the imported webtest to open it.
4. Right click on the root of the webtest, and select Generate Code



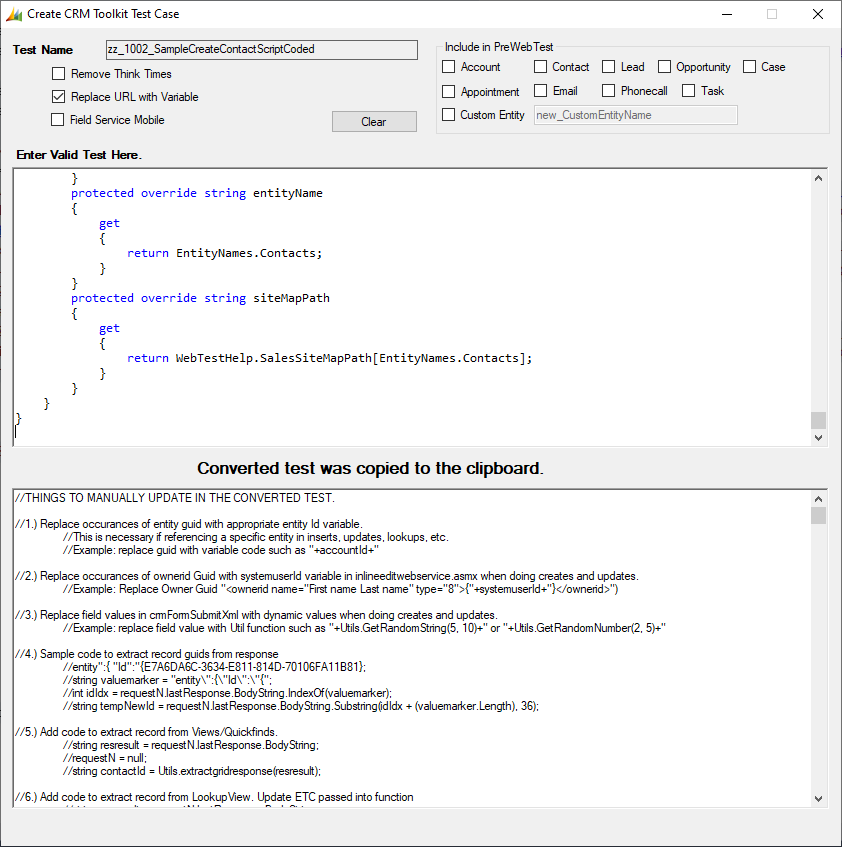
1. Enter a name for the coded web test. Add the z\_ prefex to the custom test name to keep custom tests grouped together. A new coded web test will be created and opened.



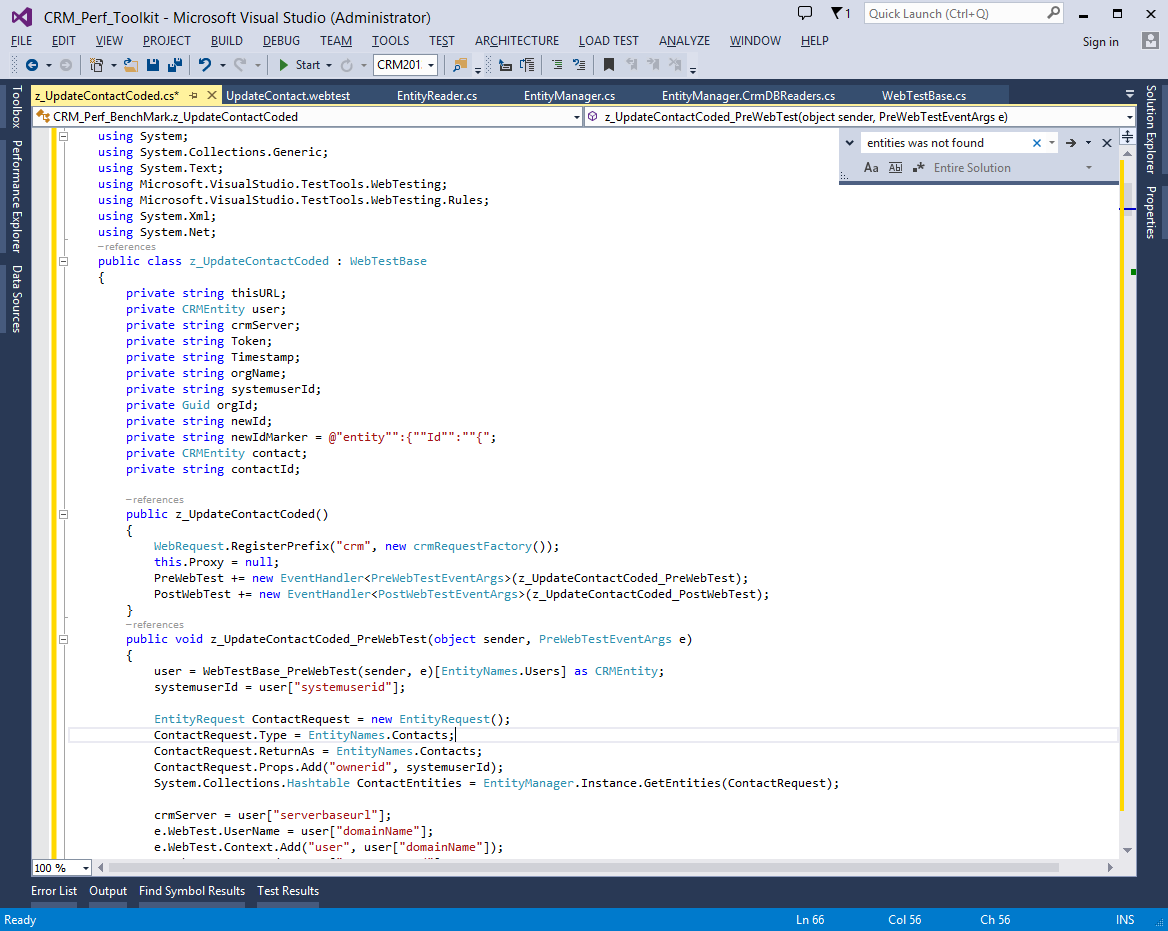
1. Select and copy the code from the coded web test.



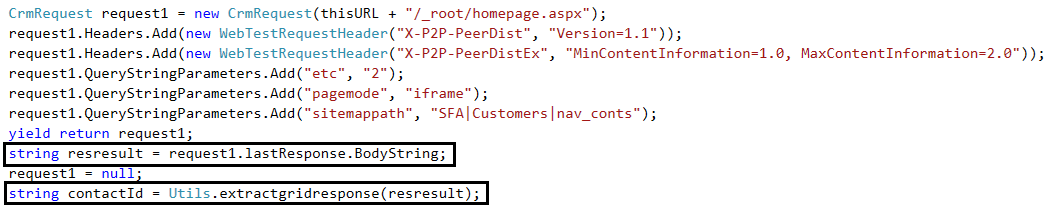
1. Maximize the Create CRM Toolkit Test Case utility and paste the code.



1. Paste the updated code over the original code in Visual Studio.



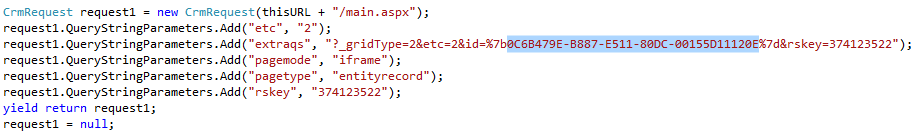
1. Make manual code changes to updated code.
   1. Add Code to homepage request to randomly select contact record from the results. Sample code to extract responses is provided at the top of the converted web test.



* 1. Review each web request to ensure hardcoded CRM URL’s were properly replaced with the variable thisURL + “.



* 1. Navigate to the main.aspx request and replace the hardcoded contactid Guid with the contactId variable.
     1. Find and Select the contactid Guid.



* + 1. Click Ctrl + h to open the find replace window.



* + 1. Replace all occurrences of the Guid with the “+contactId+” variable.

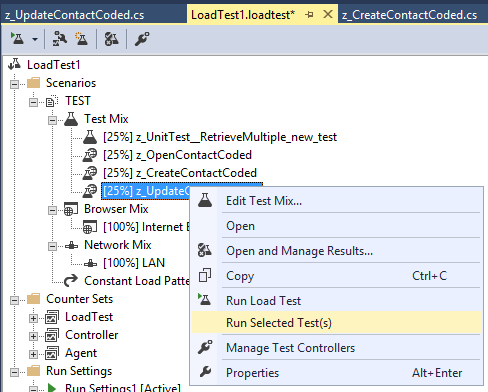


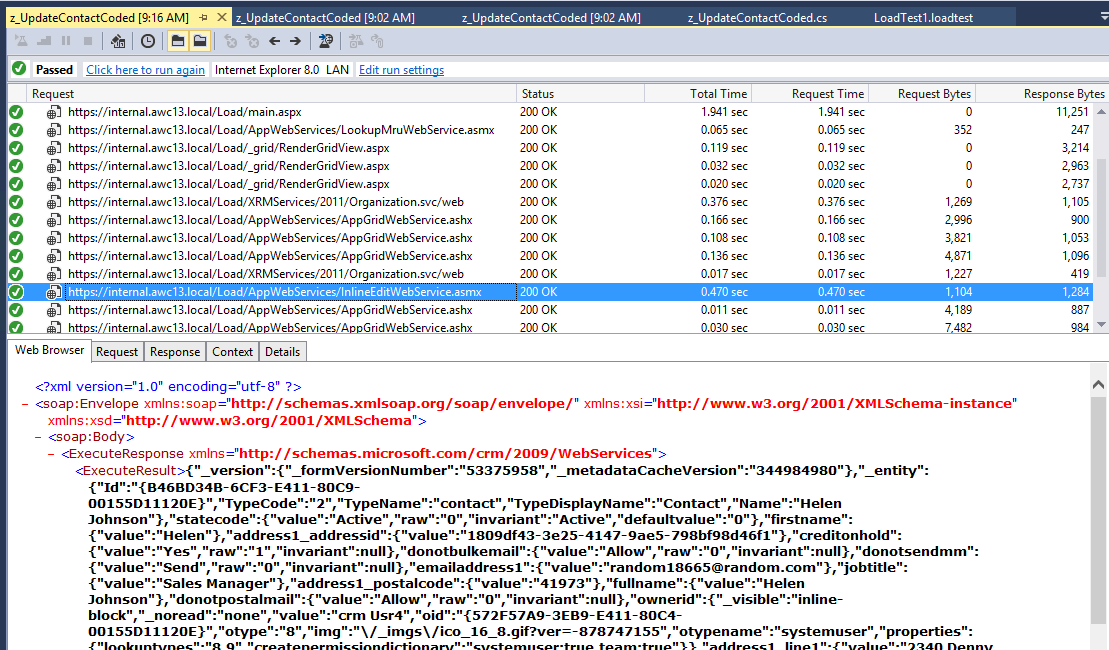


* 1. Optional: Replace variables in Inlineeditwebservice BodyString with functions such as Utils.GenerateRandomString(min,max), or Utils.GenerateRandomNumber(min,max). This is only needed if random field values are desired.



* 1. Save the coded test and click build solution from the build menu.
  2. The test can now be added to a load test and executed to verify it functions properly.





## Refreshing the EntityManager Database

This will be required if new data is added to the CRM database and needs to be pulled to local EntityManager database.

1. EMDBLoader can be started by right-clicking on the EMDBLoader Project, and choosing Debug-Start a new instance. This will clear out the EntityManager Database and pull the latest data from the CRM organization.

## Filtering Records in PreWebTest

During the PreWebTest a random test user and other required records will be retrieved from the local EntityManager database. These will be used to correctly build and execute the web tests. In some scenarios it will be required to filter which user is chosen based on a security role. It may also be required to filter other related records. These steps will show how to retrieve a user with a specific security role.

**NOTE:** If users have multiple security roles it will only store the first role in the EntityManager database.

1. Open the web test that you want to apply the user filtering.
2. Navigate to the PreWebTest method. This is the code that will retrieve the test user.

Add the following code prior to the request that retrieves the user. The first argument in the property is the column that will be filter on and the second is the string value being filtered against.

e.WebTest.Context.Add("UserRole", "salesperson");

user = WebTestBase\_PreWebTest(sender, e)[EntityNames.Users] as CRMEntity;

Filter will be used when retrieving the user record in C:\CRMToolkit\CRM\_Perf\_Toolkit\CRM\_Perf\_BenchMark\WebTests\WebTestBase.cs

**NOTE:** Filters can only be applied to columns that exist in the EntityManager database. For example the “UserRole” column exists in the SystemUser table of the EntityManager database.

1. Now only users with the salesperson role will be selected and executing the web requests.

**Example:**

public void <WebTestName>\_PreWebTest(object sender, PreWebTestEventArgs e)

{

e.WebTest.Context.Add("UserRole", "Salesperson");

user = WebTestBase\_PreWebTest(sender, e)[EntityNames.Users] as CRMEntity;

systemuserId = user["systemuserid"];

**Example 2:**

The following example shows the code to retrieve the contact that matches the defined owner. We add the property directly to the contact EntityRequest.

public void <WebTestName>\_PreWebTest(object sender, PreWebTestEventArgs e)

{

…

EntityRequest ContactRequest = new EntityRequest();

ContactRequest.Type = EntityNames.Contacts;

ContactRequest.ReturnAs = EntityNames.Contacts;

ContactRequest.Props.Add("ownerid", systemuserId);

System.Collections.Hashtable ContactEntities = EntityManager.Instance.GetEntities(ContactRequest);

## Configuring Controller/Agent Setup to Generate load from more than one machine

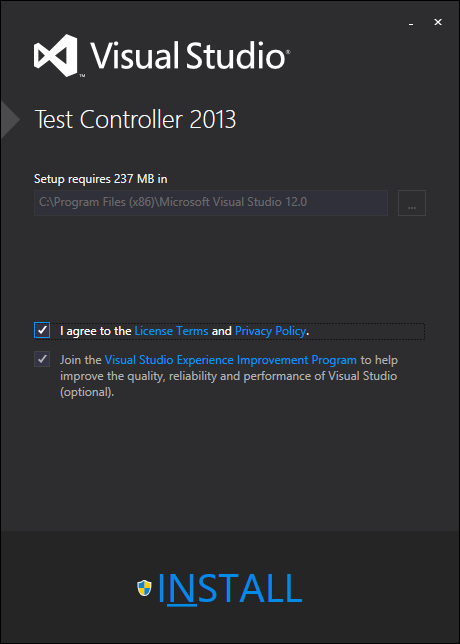
There may be scenarios where load tests need to simulate more traffic than one test machine can handle. To support larger load tests a controller & agent configuration can be used where multiple workstations configured as agents will evenly generate the load. This will also allow load balancing to better distribute the traffic amongst web servers. When tests are completed all result are transferred back to the controller machine and viewable just as if the tests were ran from a single machine.

Below are the steps to install and configure the controller and agents.

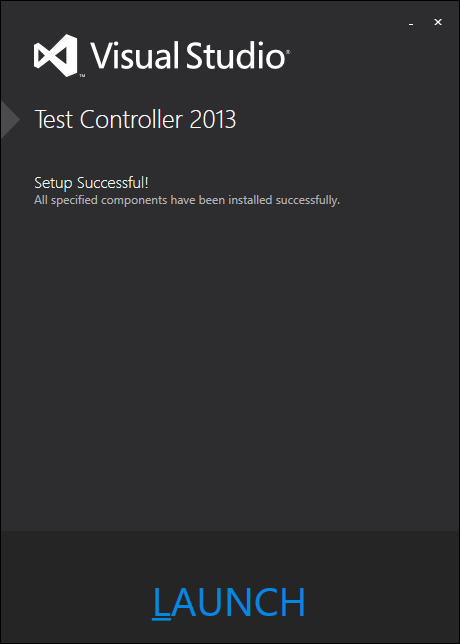
1. Download Agents for Microsoft Visual Studio 2013 (VisualStudioControllerAgentInstaller.zip) from the following site:

<https://microsoft.sharepoint.com/teams/CampusProjectSites089/hahzsakosd/ipdev/Dynamics/Forms/AllItems.aspx?FolderCTID=0x0120008199B7A495361241A7771B2781D93CF3&id=%2Fteams%2FCampusProjectSites089%2Fhahzsakosd%2Fipdev%2FDynamics%2FBenchmark%20Hands%20on%20Lab%2FCRM%2FDynamics%20365%20Benchmark%20Assessment%20%28Online%29>

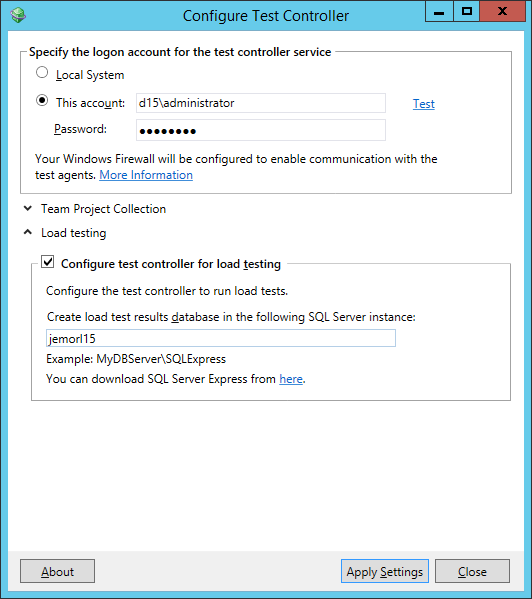
1. Extract or mount the ISO File.
2. Install the Visual Studio Test Controller.
   1. Run the vstcs\_testcontroller application from the TestController folder.
   2. Select the checkbox to agree to the terms and Click INSTALL



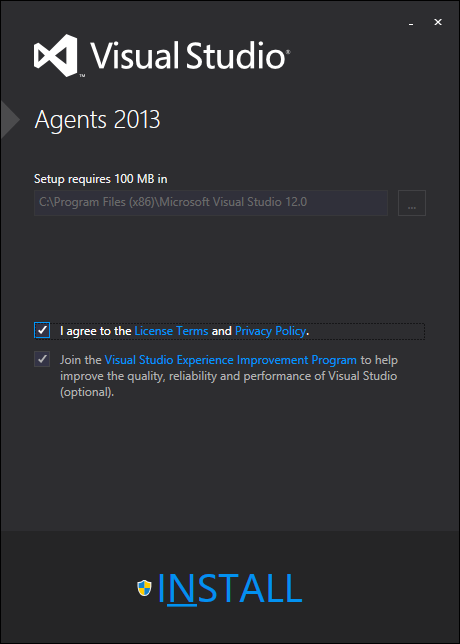
* 1. Once the installation is finished click Launch to configure the controller.



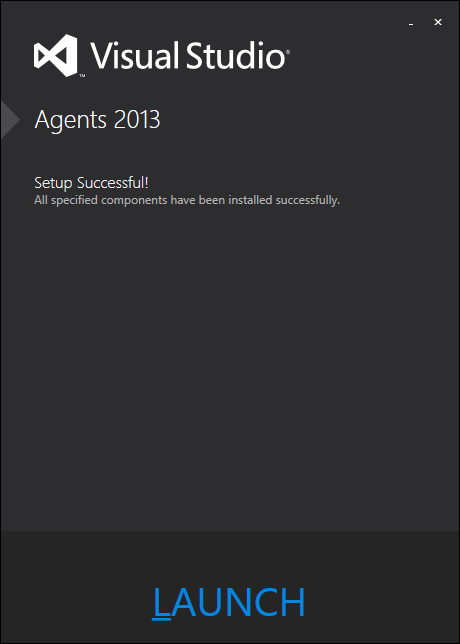
* 1. Enter credentials for the user account running the load tests and the SQL instance where the loadtest2010 database was created. Click Apply Settings and Close to finish configuring the Controller.

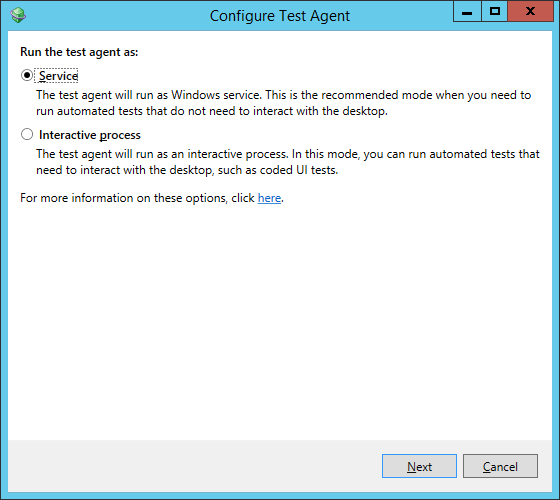


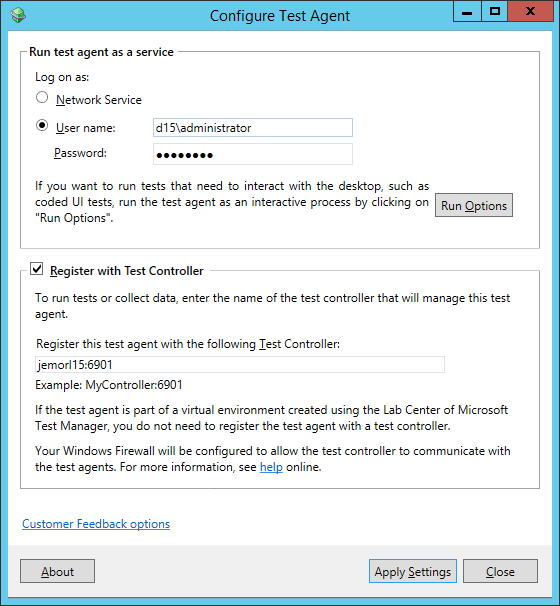
1. Install/Configure the Visual Studio Test Agent on the main controller and other workstations that you’d like to run tests.
   1. Run the vstcs\_testagent application from the TestAgent folder.
   2. Select the checkbox to agree to the terms and Click INSTALL

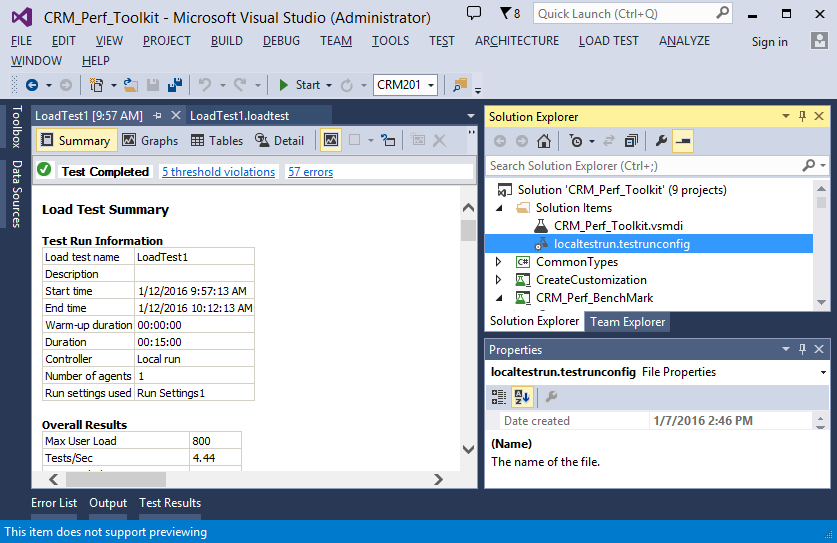


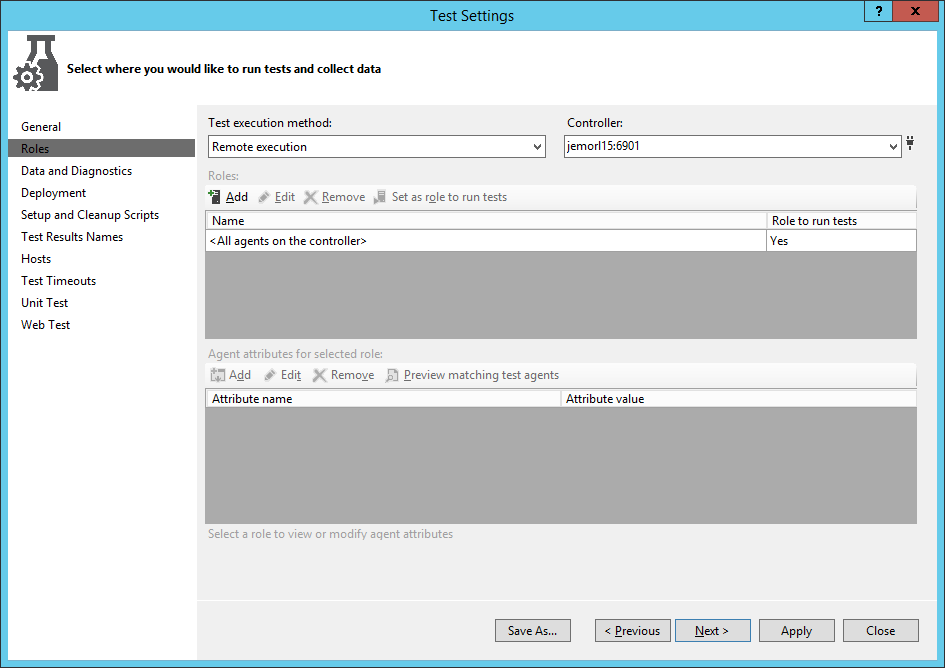
* 1. Once the installation is finished click Launch to configure the agent.



* 1. Choose Service and Click Next.
  2. Enter credentials for the user account running the load tests and name of the machine that the test controller was installed. Click Apply Settings and Close to finish configuring the Agent.



1. Copy Performance Toolkit to C: drive of each agent machine. The SQL connection string in the configsettings.xml file will need to be updated to use the server where the entitymanger database resides. This way each agent will not need a local copy of sql or the databases.
2. Create a new TestRunConfig file.
   1. Expand Solution Items within the CRM\_Perf\_Toolkit Solution and open the localtestrun.testrunconfig file. 
   2. Navigate to the Roles page, change the Test execution method to Remote execution, and select the Controller.



* 1. Click Save As and save the configuration as remotetestrun.testrunconfig.

1. Select Load Test – Select Active Test Settings and choose the remotetestrun.testrunconfig. Now when loadtesting is executed it will use run tests from each workstations where the agent is configured. 